Desarrollo de una plataforma de monitoreo de corredores en tiempo real con relojes inteligentes y Azure

Sara Lammini Rodríguez

Máster Universitario en Ingeniería de Telecomunicación
Smart Cities

Tutor/a de TF
Xavier Saura Mas
Profesor/a responsable de la asignatura
Xavier Saura Mas

12 de junio de 2023

Universitat Oberta de Catalunya

Ficha del Trabajo Final

Título del trabajo:	Desarrollo de una plataforma de monitoreo de corredores en tiempo real con relojes inteligentes y Azure.
Nombre del autor/a:	Sara Lammini Rodríguez
Nombre del Tutor/a de TF:	Xavier Saura Mas
Nombre del/de la PRA:	Xavier Saura Mas
Fecha de entrega:	07/2023
Titulación o programa:	Máster Universitario en Ingeniería de Telecomunicación
Área del Trabajo Final:	Smart Cities
Idioma del trabajo:	Castellano
Palabras clave	Running, Azure, reloj inteligente, constantes vitales, seguridad, ubicación en tiempo real, seguridad, monitoreo

Resumen del Trabajo

A pesar de los grandes beneficios para la salud que ofrece salir a correr, esta práctica deportiva trae consigo algunos riesgos como son la muerte súbita de corredores, la parada cardiaca, las caídas, los accidentes y la exposición a situaciones inseguras, especialmente en el caso de deportistas mujeres. Este Trabajo de Fin de Máster tiene como objetivo aprovechar los avances de los últimos años en el campo de los relojes inteligentes y la tecnología en la nube para desarrollar una herramienta que permita mitigar estos riesgos monitorizando las constantes vitales y la ubicación de los corredores en tiempo real. De esta forma, en caso de detectarse cualquier anomalía, se podría actuar con la mayor brevedad posible. Para ello, se estudiarán los distintos servicios que disponibles en las nubes públicas y se elegirán los más apropiados para el diseño de nuestra solución. Una vez diseñada la solución se procederá a su implantación. Como resultado se obtendrá una plataforma accesible, escalable y sostenible que permitirá atenuar los riesgos asociados a la práctica de running. Se espera que este proyecto tenga un impacto significativo en la seguridad y salud de los corredores, motivando a más personas a practicar este deporte y a llevar un estilo de vida más activo y saludable.

Abstract

Despite the significant health benefits offered by running, this sport practice entails certain risks, including sudden death, cardiac arrest, falls, accidents, and exposure to unsafe situations, particularly for female athletes. The objective of this Master's Thesis is to leverage recent advancements in smartwatch and cloud technology to develop a tool that effectively mitigates these risks by monitoring runners' vital signs and real-time location. By promptly detecting any anomalies, appropriate action can be taken expeditiously. To accomplish this, an in-depth examination of the various services available in public clouds will be conducted to select the most suitable ones for the solution's design. Once the solution is devised, its implementation will be executed, resulting in an accessible, scalable, and sustainable platform that significantly reduces the risks associated with running. This project is anticipated to have a substantial impact on the safety and health of runners, inspiring more individuals to engage in this sport and adopt a more active and healthier lifestyle.

Index

1. Introducción 11
1.1. Contexto y justificación del Trabajo 11
1.2. Objetivos del Trabajo 15
1.3. Impacto en sostenibilidad, ético-social y de diversidad 15
1.3.1. Sostenibilidad 15
1.3.2. Comportamiento ético y responsabilidad social 16
1.3.3. Diversidad y derechos humanos 18
1.4. Enfoque y método seguido 19
1.5. Planificación del trabajo 19
1.6. Breve sumario de productos obtenidos 21
1.7. Breve descripción de otros capítulos de la memoria 21
2. Estado del arte 22
2.1. Tendencias de IoT en el ámbito de la salud 22
2.2. Avances académicos 24
2.3. Avances empresariales 27
3. Diseño e Implementación 37
3.1. Diseño 37
3.2. Implementación 47
3.2.1. *Smartwatch* 47
3.2.2. Gateway 50
3.2.3. Azure Storage account 52
3.2.4. Azure Event Hub 57
3.2.5. Azure Stream Analytics – FromEventHub2PowerBI 62
3.2.6. Power BI 68
3.2.7. Logic App 80
3.2.8. Azure Function 88
3.2.9. Azure Stream Analytics - HeartRateAnomalyDetection 93
4. Resultados 97
4.1. Disposición final de los recursos 97
4.2. Registro histórico 101

4.3. Panel de monitorización el tiempo real 104
4.4. Generación de alertas 114
5. Conclusiones y trabajos futuros 122
5.1. Conclusiones 122
5.2. Trabajos futuros 123
6. Bibliografía 125
7. Anexos 130
7.1. Anexo 1 – Código de la aplicación 130
7.2. Anexo 2 – Código de la function 147

Lista de Figuras

Figura 1: Esquema de la plataforma de monitoreo de corredores en tiempo real con relojes inteligentes y Azure 14
Figura 2: Muerte súbita de un corredore en el medio maratón de Málaga. Periódico La Vanguardia 16
Figura 3: Muerte súbita en el mundo del fútbol. Diario de Sevilla 17
Figura 4: El milagro del maratón de Boston: sufre un paro cardiaco y corredores y espectadores le salvan la vida. Runner's world 17
Figura 5: Hábitos de running de las mujeres corredoras. Runner's world 18
Figura 6: Impacto del miedo a situaciones inseguras en la rutina deportiva de mujeres. Runner's World 19
Figura 7: Diagrama Gantt con la planificación del TFM 21
Figura 8: Arquitectura básica de una aplicación HIoT (25) 23
Figura 9: Ejemplos de sensores wearables (26) 24
Figura 10: Evolución del número de publicaciones en HIoT por año en el periodo 2013-2018 (23) 24
Figura 11: Diagrama de la solución SmartCare (27) 25
Figura 12: Diagrama de bloques para un sistema de detección de caídas (29) 26
Figura 13: Arquitectura de una solución para le monitoreo de entrenamientos en tiempo (28) 26
Figura 14: Notificaciones relacionadas con la frecuencia cardiaca en Apple Watch (30) 27
Figura 15: Notificaciones de ritmo irregular en Apple Watch (30) 28
Figura 16: Detección de caídas en Apple Watch (30) 28
Figura 17: Recordatorios de medicación en Apple Watch (30) 29
Figura 18: Configuración de un dispositivo Garmin para el envío de los datos registrados a la aplicación de Labfront (34) 30
Figura 19: Funcionamiento del servicio de atención médica remota ofrecido por HumanITcare (35) 30
Figura 20: Funcionamiento del hub de Chronolife para la monitorización remota de pacientes (39) 31
Figura 21: Detección de incidentes en un reloj Garmin 32
Figura 22: Plantilla disponible en Azure IoT Central para la monitorización continua de pacientes (42) 33
Figura 23: Funcionamiento de Azure IoT Hub (44) 33
Figura 24: Funcionamiento de Amazon IoT Core (45) 34
Figura 25: Funcionamiento Azure Event Hub (47) 34
Figura 26: Funcionamiento de Amazon Kinesis Data Streams (48) 35
Figura 27: Esquema de una solución IoT basado en Google Cloud IoT y Google Cloud Pub/Sub (50) 35
Figura 28: Esquema de la solución de monitoreo de corredores en tiempo real con relojes inteligentes y Azure 37
Figura 29: Instalación de la extensión de Monkey C en Visual Studio Code 48

Figura 30: Creación del fichero ejecutable PRG a partir de nuestro proyecto en Monkey C 48
Figura 31: Carga del fichero PRG en el reloj Garmin 49
Figura 32: Aplicación GarminToAzure cargada en el reloj 49
Figura 33: Creación de un nuevo grupo de recursos desde el portal de Azure 53
Figura 34: Resource group creado a través del portal de Azure 53
Figura 35: Creación de una Storage account desde el portal de Azure 1. 54
Figura 36: Creación de una Storage account desde el portal de Azure 2 54
Figura 37:Creación de una Storage account desde el portal de Azure 3 55
Figura 38: Creación de la nueva storage account 56
Figura 39: Creación del contenedor garmindatacontainer dentro de la Storage account desde el portal de Azure 56
Figura 40: Creación de un Namespace desde el portal de Azure 57
Figura 41: Namespace ConnectIQNamespace creado en el portal de Azure 58
Figura 42: Propiedades del Namespace ConnectIQNamespace 58
Figura 43: Configuración de Networking para nuestro Namespace ConnectIQNamespace 59
Figura 44: Creación de un Event Hub desde el portal de Azure 59
Figura 45: Vista del nuevo Event Hub en el portal de Azure 60
Figura 46: Creación de la Shared Access Poicy (SAS) para nuestro event Hub 60
Figura 47: Creación del token utilizando Event Hub Signature Generator 61
Figura 48: Activación de la captura en nuestro Event Hub 62
Figura 49: Process Data en Event Hub 63
Figura 50: Creación del Azure Stream Analytics Job para la visualización de datos en tiempo real desde Power BI 63
Figura 51: Creación de un nuevo consumer group 64
Figura 52: Selección de los campos a importar desde nuestro Event Hub 64
Figura 53: Manage Fields en Azure Stream Analytics 65
Figura 54: Configuración del Power BI de salida 65
Figura 55: Configuración del modo de autenticación contra Power BI 66
Figura 56: Inicio de sesión para la autenticación contra Power BI 66
Figura 57: Iniciación del job 67
Figura 58. Nuevo Stream Dataset en PowerBI 67
Figura 59: Creación de un nuevo Dashboard en Power BI 1 68
Figura 60: Creación de un nuevo Dashboard en Power BI 2 68
Figura 61: Añadir un nuevo título en Power BI 69
Figura 62: Añadir nuevo Custom Streaming Data en Power BI 1 69
Figura 63: Añadir nuevo Custom Streaming Data en Power BI 2 70
Figura 64: Visualization design para nuestro title 71
Figura 65: Title details 71
Figura 66: Frecuencia cardiaca registrada por el simulador y el reloj en tiempo real 72
Figura 67: Frecuencia cardiaca registrada por el reloj en tiempo real 72
Figura 68: Frecuencia cardiaca registrada por el simulador en tiempo real 72
Figura 69: Ask a question about your data 1 73

Figura 70: Ask a question about your data 2 73
Figura 71: Creación de un nuevo visual a Power BI 1 73
Figura 72: Creacón de un nuevo visual a Power BI 2 74
Figura 73: Creación de un mapa para la visualización de la ubicación de corredores 74
Figura 74: Vista del mapa 1 75
Figura 75: Vista del dashboard de monitorización 75
Figura 76: Acceso al mapa 1 76
Figura 77: Acceso al mapa 2 76
Figura 78: Adición de un nuevo visual a Power BI 3 77
Figura 79: Adición de un nuevo visual a Power BI 4 77
Figura 80: Dashboard de monitorización 78
Figura 81: Personalización del Dashboard de monitorización 78
Figura 82: Generación de alertas 1 79
Figura 83: Generación de alertas 2 79
Figura 84: Generación de alertas 3 80
Figura 85: Creación de una nueva Logic App 1 80
Figura 86: Creación de una nueva Logic App 2 81
Figura 87: Creación de un nuevo flujo de trabajo 1 82
Figura 88: Creación de un nuevo flujo de trabajo 2 82
Figura 89: Creación de un nuevo flujo de trabajo 3 83
Figura 90: Creación de un nuevo flujo de trabajo 4 83
Figura 91: Creación de un nuevo flujo de trabajo 5 84
Figura 92: Creación de un nuevo flujo de trabajo 6 84
Figura 93: Creación de un nuevo flujo de trabajo 7 85
Figura 94: Creación de un nuevo flujo de trabajo 8 85
Figura 95: Test Logic App 1 86
Figura 96: Test Logic App 2 87
Figura 97: Test Logic App 3 87
Figura 98: Creacion Function App 1 88
Figura 99: Creación Function App 2 89
Figura 100: Creación de una Function 1 89
Figura 101: Creación de una Function 2 90
Figura 102: Código Function 1 91
Figura 103: URL Logic App 91
Figura 104: Comprobación del funcionamiento de la Function 1 92
Figura 105: Comrpobación del funcionamiento de la Function 2 92
Figura 106: Comprobación del funcionamiento de la Function 3 93
Figura 107: Creación de un nuevo Azure Stream Analytics Job 93
Figura 108: Configuración del Input para el Azure Stream Analytics Job 94
Figura 109: Configuración del output para el Azure Stream Analytics Job 94
Figura 110: Configuración de la autenticación contra la Function 1 95
Figura 111: Function Key 95
Figura 112: Test de conexión satisfactorio entre el Azure Stream Analytics Job y la Auzre Function 96

Figura 113: Configuración de la Query para el Azure Stream Analytics 96
Figura 114: Vista de los recursos que forman parte del grupo de recursos TFM – ConnectIQ 98
Figura 115: Vista de las dependencias entre los recursos de TFM-Connect IQ 98
Figura 116: Ubicación de los recursos del grupo de recursos TFM-ConnectIQ 99
Figura 117: Ubicación de los recursos del grupo de recursos TFM-ConnectIQ en mayor tamaño 99
Figura 118: Número de cada tipo de recurso del grupo de recursos TFM-ConnectIQ 100
Figura 119: Análisis de costes para el grupo de recursos. 100
Figura 120: Proceso de guardado de los paquetes recividos en Azure Event Hub 101
Figura 121: Vista de los paquetes almacenados automáticamente en el contenedor garmindatacontainer desde el portal de Azure 102
Figura 122: Vista de los paquetes almacenados automáticamente en el contenedor garmindatacontainer desde el Microsoft Azure Storage Explorer 102
Figura 123: Descarga de los paquetes almacenados en el contenedor 103
Figura 124: Datos almacenados en el contenedor 103
Figura 125: panel de monitorización de corredores en tiempo real final desarrollado en Power BI 104
Figura 126: Evolución de la frecuencia cardiaca a lo largo del tiempo cuando hay dos dispositivos conectados a la plataforma 105
Figura 127: Evolución de la frecuencia cardiaca a lo largo del tiempo cuando únicamente el reloj real se encuentra conectado a la plataforma 105
Figura 128: Evolución de la frecuencia cardiaca a lo largo del tiempo cuando únicamente el reloj real se encuentra conectado el simulador a la plataforma. Valores de frecuencia cardiaca comprendidos entre 50 y 70 pulsaciones por minuto. 106
Figura 129: Evolución de la frecuencia cardiaca a lo largo del tiempo cuando únicamente el reloj real se encuentra conectado el simulador a la plataforma. Valores de frecuencia cardiaca comprendidos entre 180 y 250 pulsaciones por minuto. 106
Figura 130: Código para el registro o simulación de datos de frecuencia cardíaca 107
Figura 131: tabla de coordenadas registradas por los dispositivos en tiempo real 107
Figura 132: visualización de la ubicación de los corredores en tiempo real sobre un mapa 108
Figura 133: Ubicación de los dispositivos en tiempo real sobre el mapa 108
Figura 134: Código para el registro o simulación de coordenadas 109
Figura 135: Ubicación del simulador sobre el mapa 1 cuando las coordenadas se sitúan en Italia 109
Figura 136: Modificación del código para situar el simulador en Turquía 109
Figura 137: ubicación del simulador sobre el mapa 1 cuando las coordenadas se sitúan en Turquía 110
Figura 138: Opciones de interacción con el mapa 110
Figura 139: Vista del mapa con estilo Aerial 111
Figura 140: Vista mapa estilo grayscale 111
Figura 141: Zoom sobre el mapa 112
Figura 142: Frecuencia cardiaca máxima registrada por el simulador y por el reloj 112

Figura 143: Frecuencia cardiaca máxima registrada cuando un dispositivo no se encuentra conectado....113
Figura 144: Número de dispositivos conectados....113
Figura 145: Peticiones, mensajes y thorughput del Azure Event Hub garmineventhub....114
Figura 146: Mensajes y Throughput en el Azure Event Hub garmineventhub....114
Figura 147: hearRateAnomalyDetection Overview....115
Figura 148: Test del correcto funcioamiento de la query del HeartRateAnomalyDetection Stream Analytics Job....115
Figura 149: Resultado del test de la query cuando no se producen pulsaciones anómalas....116
Figura 150: Modificación del código de nuestra aplicación para la simulación de las pulsaciones deseadas....116
Figura 151: Resultado del test del Azure Stream Analytics Job cuando se generan pulsaciones anómalas....117
Figura 152: Overview del job HeartRateAnomalyDetection cuando se generan pulsaciones anómalas....117
Figura 153: Gráfica de ejecuciones de la function como resultado de pulsaciones anómalas....118
Figura 154: Tiempo de ejecución de la function....118
Figura 155: Registro de triggers de la Logic App....119
Figura 156: Registro de ejecución de la Logic App....119
Figura 157: Métricas de ejecución de la Logic App....120
Figura 158: Correos electrónicos generados por la Logic App....120
Figura 159: Correo electrónico de alerta....121
Figura 160: Estructura de los ficheros de los que se compone nuestra solución....130
Figura 161: launcher_icon.png....131
Figura 162: monkey.png....131
Figura 163: Layout de la aplicación....131

1. Introducción

Este capítulo introductorio consta de siete secciones. En la primera sección se expone la motivación detrás de este Trabajo de Fin de Máster (TFM) y el contexto que lo envuelve. La segunda sección establece el objetivo principal y los objetivos secundarios que se persiguen en la elaboración del presente TFM. La tercera sección presenta el impacto que la solución propuesta puede tener desde los siguientes puntos de vista: sostenibilidad, comportamiento ético y responsabilidad social y diversidad y derechos humanos. En la cuarta sección se presenta el enfoque y la metodología seguida en el desarrollo de este trabajo. La quinta sección presenta al lector la planificación a seguir durante todo el cuatrimestre para lograr los objetivos establecidos previamente. La sexta sección presenta el sumario de los productos. Por último, en la séptima sección se realiza una descripción de los capítulos restantes de la memoria.

1.1. Contexto y justificación del Trabajo

En esta sección, se aborda la motivación detrás de la realización del presente TFM, así como el contexto en el que se desarrolla, con el objetivo de entender la necesidad que se pretende cubrir con este proyecto y su relevancia. Para ello, se va a comenzar examinando algunos de los beneficios del running que están convirtiendo a este deporte en una actividad cada vez más popular. Posteriormente, se procederá a describir algunos de los riesgos a los que se enfrentan los corredores, tanto desde el punto de vista de la salud como de la seguridad. Por último, se concluirá esta sección analizando cómo la tecnología puede ser utilizada para mitigar estos riesgos.

Como se adelantaba, correr es un deporte que está ganando gran popularidad en todo el mundo debido a múltiples factores, entre ellos:

- Presenta múltiples beneficios para la salud: mejora la salud cardiovascular, fortalece músculos y huesos, mejora el sistema inmune, reduce del riesgo de enfermedades crónicas, mejora la salud mental y favorece la pérdida de peso, entre otros (1–3).

- Es un deporte flexible y accesible, puede realizarse a cualquier hora y en cualquier lugar (1,2).

- Es económico, basta con contar con un calzado adecuado (1,2).

- Es inclusivo, puede ser practicado por personas de todas las edades edad y condición física (1,2).

- Fomenta las relaciones sociales y crea un sentimiento de comunidad. Correr con amigos, familiares, compañeros del trabajo o unirse a un club de running puede ser una gran oportunidad para crear nuevos vínculos y conocer personas con tus mismos intereses. Participar en carreras y otros eventos deportivos también es una gran oportunidad para ampliar el círculo social, mantener la motivación y compartir experiencias (1).

No obstante, a pesar de todas las ventajas que puede ofrecer salir a correr, lo cierto es que esta práctica deportiva trae consigo algunos riesgos. Dese el punto de vista de la salud, algunos de los riesgos a considerar son:

- Muerte súbita. Estudios realizados en España y Francia sitúan al running como el tercer deporte con un mayor número de casos de muerte súbita, por detrás del ciclismo y el fútbol (4). En la mayoría de los casos, estas muertes se deben a enfermedades cardiacas subyacentes no detectadas (5). Según el estudio conducido por el doctor Benito Moretín, jefe del Servicio de Patología del Instituto Vasco de Medicina Legal y profesor asociado del Departamento de Especialidades Médico-Quirúrgicas de la UPV/EHU, es de vital importancia que las personas deportistas sean monitorizadas con el fin de detectar posibles cardiopatías desconocidas que puedan aumentar el riesgo de sufrir una muerte súbita (6).

- Paro cardíaco. La mayor parte de los paros cardiacos tienen lugar cuando el sistema eléctrico de un corazón enfermo no funciona correctamente(7). Este mal funcionamiento provoca un ritmo cardíaco anormal como taquicardia o fibrilación ventriculares. Algunos paros cardiacos también pueden ser causados por bradicardia, una desaceleración extrema del ritmo cardiaco. Los paros cardiacos tienen consecuencias fatales si no se actúa de inmediato (7). Es por ello por lo que contar con una plataforma que permita monitorizar a atletas en tiempo real tiene una importancia crucial.

- Lesiones. Cuando los corredores realizan esta práctica deportiva, pueden producir fuerzas de hasta tres veces el peso corporal en cada paso, lo que frecuentemente se asocia con lesiones por sobrecarga en las extremidades inferiores. La incidencia anual de lesiones relacionadas con la carrera puede afectar hasta el 85% de todas las poblaciones de corredores (8). Algunas de las lesiones más comunes son: fascitis plantar, tendinitis aquilea, esguince de tobillo, tendinitis rotuliana, periostitis tibial, distensión o rotura de los isquiotibiales y dolor lumbar (9).

Desde el punto de vista de la seguridad, algunos de los riesgos a destacar son:

- Riesgo de los corredores de ser atacados. En los últimos años, se han llevado a cabo numerosas encuestas entre corredores para recopilar información sobre sus experiencias en relación con el acoso durante la práctica del running. Una encuesta de 2017 realizada por Runner’s World reporta que el 43% de las mujeres y el 4% de los hombres han sufrido algún tipo de acoso mientras corrían (10). Otro estudio realizado dos años más tarde indicaba que el 67% de las mujeres han sentido miedo de ser

atacadas en alguna ocasión mientras corrían (10). Un sondeo más reciente llevado a cabo en 2021 reportaba que el 60% de las mujeres habían sufrido acosos de distinta índole mientras realizaban esta práctica deportiva (11). Dentro de las técnicas y recomendaciones ampliamente difundidas en internet, es común observar que gran cantidad de corredores manifiestan sentirse más seguros cuando informan a alguien de donde están o portan algún dispositivo de seguridad que les permita solicitar asistencia en caso de necesitarla (10,12,13).

- Riesgo de sufrir accidentes o caídas. Algunos ejemplos comunes de accidentes entre corredores son las colisiones con vehículos o bicicletas y los tropiezos y caídas debidos a superficies irregulares, especialmente en terrenos montañosos (14). En estos escenarios conocer la ubicación exacta de un corredor a través de la monitorización continua puede ser crucial para enviar la ayuda necesaria.

Existen diversas tecnologías a nuestro alcance que pueden ser utilizadas para construir soluciones innovadoras que nos permitan atenuar estos riesgos.

Por un lado, en los últimos años se han producido grandes avances en el campo del *Internet of Things* (IoT) aplicado al ámbito de la salud. La implementación del monitoreo continuo de pacientes ofrece una solución efectiva para reducir el riesgo de admisiones hospitalarias, así como para controlar enfermedades crónicas de manera más efectiva y mejorar los resultados de los pacientes (15).

Por otro lado, se han producido grandes avances tecnológicos en el mundo de los relojes inteligentes que ahora incorporan utilidades como GPS, monitores de actividad física o sensores que permiten monitorizar la salud de los usuarios (sensores para la monitorización cardiaca, para medir la saturación del oxígeno en sangre, etc.) (16). Además, los relojes inteligentes actuales cuentan con sistemas operativos más avanzados que facilitan la integración con los teléfonos móviles y la conectividad con otros dispositivos y servicios como la nube. Los relojes inteligentes gozan cada vez de más popularidad entre corredores, hasta el punto de que han dejado de ser un accesorio reservado únicamente para el deporte de alto rendimiento (17).

Este TFM se enmarca en los avances y tendencias tecnológicas actuales, con el propósito de desarrollar una solución que permita mitigar algunos de los riesgos analizados previamente en esta sección. Para ello, se pretende desarrollar una plataforma que monitorice continuamente las constantes vitales y la ubicación de los corredores. De esta forma, en el caso de que un deportista sufra una anomalía cardíaca, el sistema generará una alerta con la ubicación precisa del corredor para poder enviar la ayuda necesaria y actuar con la mayor brevedad posible. Si un corredor experimentara cualquier tipo de ataque o una caída, conoceríamos en todo momento su ubicación actual.

Por último, al recolectar continuamente los datos de constantes vitales de los corredores, podemos estudiar esta información en busca de posibles patologías cardiacas subyacentes no detectadas, lo que permite reducir los riesgos de muerte cardiaca en deportistas.

La figura 1 muestra el esquema de esta plataforma de monitoreo de corredores en tiempo real. Como se puede observar, los relojes inteligentes, también conocidos como *smartwatches*, de los deportistas se conectan y envían sus datos a la nube de Microsoft. Una vez que los datos llegan a Azure, se utilizan de dos formas distintas. En primer lugar, se representan en un panel de monitorización construido sobre Power BI, lo que permite visualizar los datos de forma inmediata. Por otro lado, los datos se procesan en tiempo real para identificar posibles anomalías en las pulsaciones. En caso de detectarse una anomalía, se genera un correo electrónico de alerta que incluye el identificador del dispositivo, las pulsaciones cardiacas registradas y la ubicación del corredor. Esta alerta permite enviar ayuda de manera rápida y eficiente.

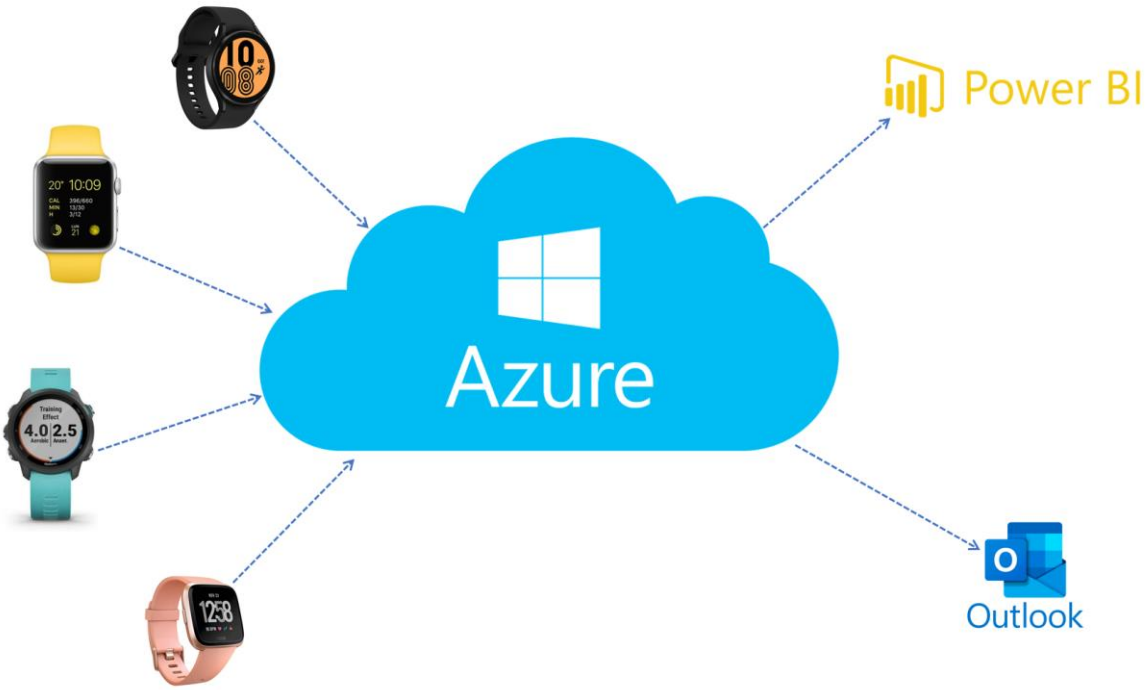

Figura 1: Esquema de la plataforma de monitoreo de corredores en tiempo real con relojes inteligentes y Azure

Con todo esto, se concluye que la motivación detrás de este TFM es aplicar la tecnología más avanzada para contribuir a la seguridad y bienestar de los corredores.

1.2. Objetivos del Trabajo

El objetivo principal de este Trabajo de Fin de Máster (TFM) consiste en desarrollar una plataforma innovadora para el monitoreo en tiempo real de corredores a partir de sus relojes inteligentes y los servicios ofertados por Azure, la nube pública de Microsoft. Con esta plataforma se busca contribuir a la seguridad y bienestar de los corredores. Para ello, se han establecido los siguientes objetivos secundarios:

1. Realizar un análisis exhaustivo de los riesgos sanitarios y de seguridad asociados al deporte de carrera con el objetivo de identificar los factores de mayor riesgo para los corredores.
2. Investigar y evaluar las tecnologías y servicios ofrecidos por Azure para desarrollar una plataforma que permita mitigar los riesgos identificados, incluyendo la monitorización continua de las constantes vitales y la localización de los corredores.
3. Diseñar e implementar una solución integral seleccionando cuidadosamente las tecnologías y servicios más adecuados para nuestro proyecto, asegurando la escalabilidad, fiabilidad y seguridad de la plataforma.
4. Generar alertas en tiempo real para los corredores y autoridades locales, con el fin de mejorar la seguridad y la salud de los corredores.

1.3. Impacto en sostenibilidad, ético-social y de diversidad

En esta sección se va a identificar los impactos positivos y/o negativos del TFM en las tres dimensiones de la competencia transversal UOC “Compromiso ético y global”: sostenibilidad, comportamiento ético y responsabilidad social y diversidad y derechos humanos.

1.3.1. Sostenibilidad

La solución propuesta conlleva inherentemente un impacto positivo en términos de sostenibilidad.

En primer lugar, se debe tener en cuenta que esta herramienta ha sido implementada sobre la nube, tecnología que puede emplearse para mitigar parte de los desafíos en sostenibilidad existentes:

- La escalabilidad del *cloud* permite ajustar de forma dinámica los recursos necesarios para satisfacer las cargas de trabajo existentes en cada momento. De esta forma, es

posible realizar un uso más eficiente de los recursos, reduciendo el desperdicio tecnológico, el consumo innecesario de energía y la huella de carbono.

- Los centros de datos de los hiperescalares son más eficientes energéticamente que los centros de datos privados de las empresas. Por lo tanto, al desarrollar plataforma sobre a nube de Microsoft se realiza un consumo energético más eficiente que si se desplegase sobre otra infraestructura.

En segundo lugar, la monitorización continua de deportistas tiene también un impacto positivo en términos de la presión sobre el sistema nacional de salud puesto que permite la detección temprana de patologías y problemas de salud potencialmente graves.

1.3.2. Comportamiento ético y responsabilidad social

Con respecto a la dimensión del comportamiento ético y de responsabilidad social, se concluye que la solución desarrollada en el presente TFM tiene un impacto social positivo en lo que a la salud se refiere.

La muerte súbita de una persona joven, deportista y que aparentemente goza de buena salud es un acontecimiento trágico que conmociona a la sociedad y a las familias (18). Los casos de muerte súbita en deportistas jóvenes ocupan a menudo los titulares de la prensa deportiva (19). Ejemplos de esto se muestran en las figuras 2 y 3. Con el fin de prevenir y reducir el riesgo de muerte súbita durante la práctica deportiva, es de vital importancia detectar posibles cardiopatías no identificadas. Por ello, esta plataforma de monitorización continua de las constantes vitales de deportistas puede ser crucial para identificar patologías subyacentes y reducir el riesgo por muerte súbita.


ATLETISMO

Muere un corredor en el medio maratón de Málaga

• El hombre de 58 años, de nacionalidad noruega, falleció tras llegar a meta

Varios miembros de Policía Local, Protección Civil y Cruz Roja aguardan la llegada del juez para el levantamiento del cuerpo del corredor fallecido (Daniel Pérez / EFE)

Figura 2: Muerte súbita de un corredore en el medio maratón de Málaga. Periódico La Vanguardia

Figura 3: Muerte súbita en el mundo del fútbol. Diario de Sevilla

Con respecto a los casos de parada cardiaca, se ha visto cómo actuar de forma temprana es primordial para la supervivencia de los pacientes (20). Un ejemplo de esto es la noticia recogida en la figura 4 en la que los corredores y espectadores del maratón de Boston logran salvar la vida de Meghan Roth. Contar con una plataforma que permita monitorizar en tiempo real las constantes vitales y la ubicación de los deportistas puede salvar la vida de muchos atletas.

Figura 4: El milagro del maratón de Boston: sufre un paro cardiaco y corredores y espectadores le salvan la vida. Runner's world

Además, esta herramienta tiene también un impacto positivo en la reducción de las desigualdades sociales. Mediante la realización de un reconocimiento médico deportivo es posible evitar la muerte súbita de deportistas y detectar otras patologías que puedan ser agravadas como consecuencia de la actividad deportiva (21). No obstante, estos reconocimientos médicos no suelen ser gratuitos ni ser accesibles en todos los lugares del mundo. A modo de referencia, según la sede electrónica del ayuntamiento de Madrid, estos reconocimientos pueden rondar entre los 14 y 27 euros para los adultos de entre 27 y 64 años en función del número de pruebas realizadas (22). Gracias a esta herramienta de monitorización continua, es posible realizar un estudio de las constantes vitales de los deportistas y detectar posibles patologías subyacentes sin la necesidad de acudir a una clínica u hospital para la realización de un reconocimiento médico.

Por otro lado, es importante tener en cuenta el impacto ético de la herramienta. Es necesario garantizar que se respete la privacidad de los corredores y se cumplan las normas de protección de datos personales vigentes en cada país. Por ello, se deben establecer medidas claras y transparentes para informar a los usuarios sobre qué información está siendo recopilada, como va a ser utilizada y quién tendrá acceso a ella.

1.3.3. Diversidad y derechos humanos

El último pilar por tratar en este apartado es el de diversidad y derechos humanos. El presente TFM tiene un impacto positivo en términos de igualdad de género.

Se ha visto en la primera sección que los corredores, en especial las mujeres, se ven envueltos en situaciones inseguras. Son innumerables las encuestas que abordan este tema. Si se toma como ejemplo la encuesta realizada a 2000 mujeres atletas por Runner's World en 2021, el 60% de las mujeres corredoras han sufrido algún tipo de acoso mientras corría, el 25% reportan haber experimentado acoso verbal y el 6% denuncian haberse sentido amenazadas hasta el punto de temer por sus vidas (11). Como se observa en la figura 5, esto se traduce en que el 34% de las mujeres solo corran cuando es de día y que el 54% corran a cualquier hora pero evitando ciertas zonas cuando es de noche (11).

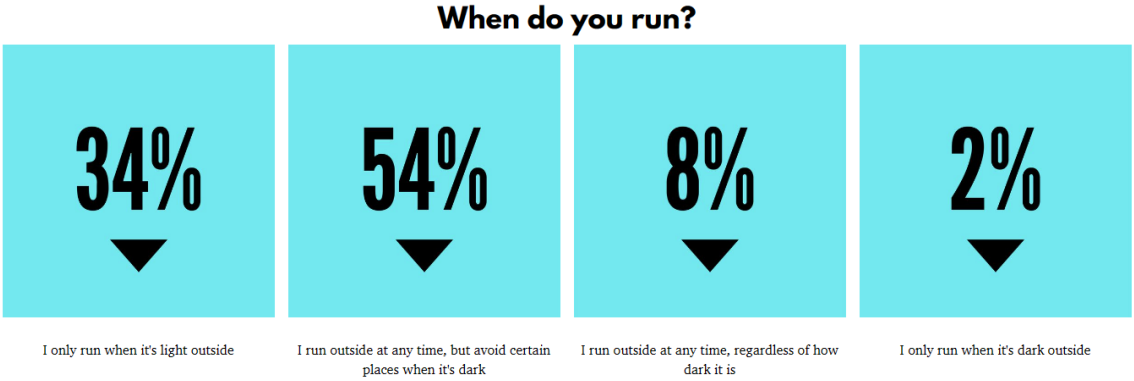

Figura 5: Hábitos de running de las mujeres corredoras. Runner's world

En la figura 6 se recoge el impacto de situaciones inseguras en los hábitos deportivos de las mujeres: el 11% de las mujeres dejan de correr por un tiempo, el 40% comparten su ruta con alguien y el 47% de las mujeres llevan su teléfono con ellas cuando salen a correr. Por todo esto, se concluye que contar una solución que permita conocer la posición real de las corredoras y su ubicación en tiempo real para poder intervenir en caso de peligro es vital para su seguridad y para recuperar la confianza de salir a correr.

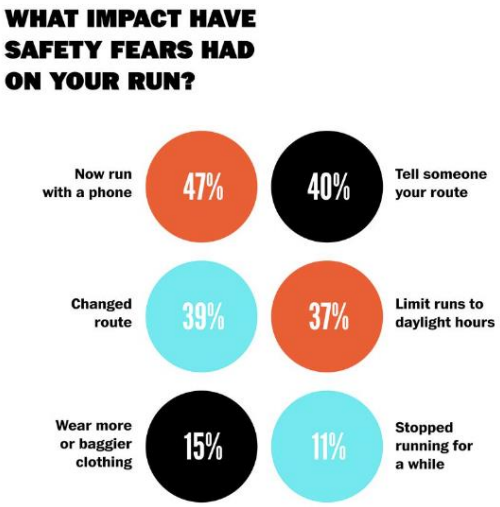

Figura 6: Impacto del miedo a situaciones inseguras en la rutina deportiva de mujeres. Runner's World

1.4. Enfoque y método seguido

Para la realización de este TFM se ha optado por desarrollar una innovadora plataforma para el monitoreo continuo de corredores a partir de relojes inteligentes y servicios disponibles en Azure. El enfoque es crear una herramienta innovadora a partir de los últimos avances tecnológicos en el mundo de *cloud* y de los relojes inteligentes para atenuar los riesgos a los que se enfrentan los corredores.

1.5. Planificación del trabajo

En este quinto apartado se describen los recursos necesarios para realizar el proyecto, las tareas a realizar y una planificación temporal de cada tarea a desarrollar.

1. Recursos necesarios

Los recursos necesarios para la realización del presente Trabajo de Fin de Máster (TFM) son los siguientes:

- Un reloj inteligente con tecnología BLE (Bluetooth Low Enery) que mida las constantes vitales y disponga de GPS

- Un teléfono móvil que actúe a modo de gateway para poder conectar nuestro reloj a Azure
- Una suscripción de Azure para poder desarrollar la plataforma de monitorización en tiempo real.
- Cuenta de Power BI para poder representar los datos en tiempo real
- Un ordenador
- Acceso a Internet
- Word para el desarrollo de la memoria

2. Tareas que realizar

Las tareas para realizar para el desarrollo del presente Trabajo de Fin de Máster se pueden agrupar en 5 grandes bloques:

- Fase 1 – Definición. Este primer bloque se corresponde con la primer PEC de la asignatura. En esta fase se llevarán a cabo las siguientes tareas:
  - Definición del tema y alcance del proyecto
  - Definición de los objetivos del proyecto
  - Planificación temporal
- Fase 2 – Estado del Arte. Este segundo bloque se corresponde con la segunda PEC de la asignatura.
  - Estudio de los proyectos y técnicas ya existentes en el mundo
- Fase 3 – Diseño, Implementación y Análisis. Esta tercera fase se corresponde con la tercera PEC de la asignatura y constará de las fases siguientes:
  - Estudio de los posibles servicios disponibles en Azure y selección de los más apropiados para el desarrollo de la solución
  - Diseño de la solución a partir de los servicios seleccionados
  - Implementación de la solución
  - Análisis de los resultados obtenidos
- Fase 4 – Memoria. Este bloque, aunque se corresponde con la cuarta PEC se desarrolla a lo largo de todo el cuatrimestre.
- Fase 5 – Defensa.
  - Presentación del trabajo del TFM
  - Respuesta a las preguntas del tribunal

3. Planificación temporal

Una vez se han enumerado las tareas a realizar en el desarrollo de este TFM, se expone en el diagrama de Gantt presentado en la figura 6 cuál será la planificación temporal para cada una de estas tareas:

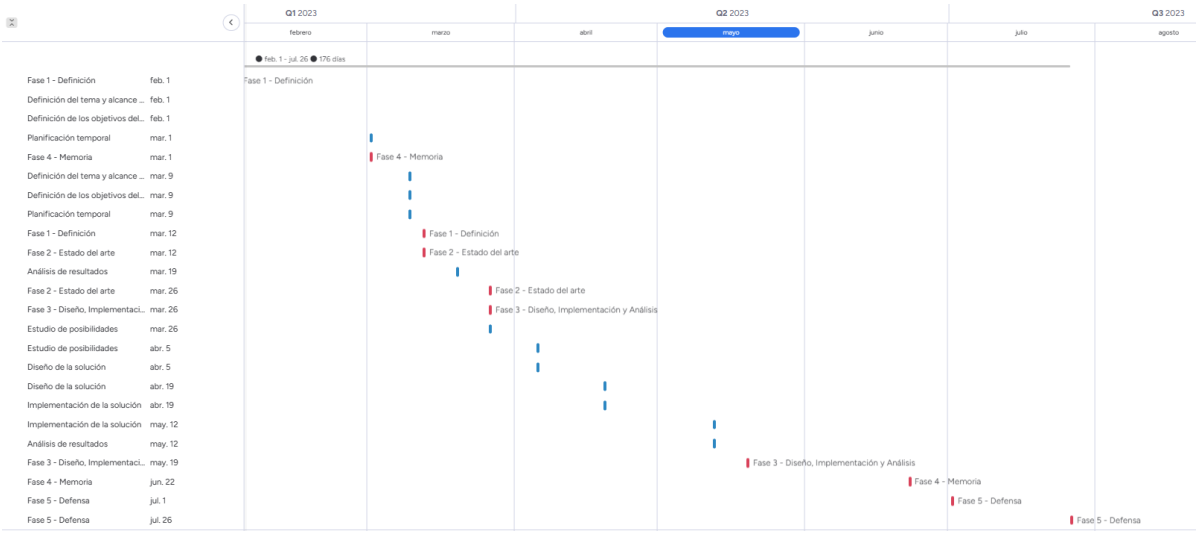

Figura 7: Diagrama Gantt con la planificación del TFM

1.6. Breve sumario de productos obtenidos

Como resultado de este Trabajo de Fin de Máster obtendremos principalmente 3 productos:

- Prototipo de una plataforma para el monitoreo continuo de corredores basada en relojes inteligentes y Azure.
- La presente memoria
- Presentación para la defensa del TFM

1.7. Breve descripción de otros capítulos de la memoria

La presente memoria se compone de los capítulos siguientes:

- **Introducción.** En este capítulo se contextualiza y justifica el Trabajo de Fin de Máter, se establecen los objetivos que se persiguen, se analiza el impacto en sostenibilidad, ético-social y de diversidad, se describe el enfoque y el método seguido, se realiza la planificación del trabajo, se realiza un breve sumario de los productos obtenidos y se describe el resto de la memoria.
- **Estado del Arte.** En este capítulo se presentan otros estudios y proyectos similares y se estudian las tendencias tecnológicas en el ámbito del presente TFM.
- **Diseño e Implementación.** En este capítulo se procede a describir el diseño de la solución y los pasos seguidos para su implementación.
- **Resultados.** En este capítulo se presentan y analizan los resultados obtenidos.
- **Conclusiones y trabajos futuros.** En este último capítulo de la memoria se tratarán las conclusiones finales del trabajo y se valorarán posibles líneas futuras.

2. Estado del arte

En este segundo capítulo de la memoria se aborda el estado del arte de la monitorización continúa utilizando tecnologías como el *cloud* y los relojes inteligentes o *smartwatches*, con el fin de obtener una idea global de la situación actual.

Para ello, en primer lugar, comenzaremos introduciendo las tendencias en cuanto a la incorporación de tecnologías IoT (*Internet of things* o internet de las cosas) en el ámbito de la salud. En segundo lugar, presentaremos algunos de los avances que se están llevando a cabo en el mundo académico. Para finalizar, procederemos a describir las líneas estratégicas y avances en los que están trabajando las organizaciones empresariales dedicadas al desarrollo de tecnologías relacionadas con el ámbito de trabajo de este TFM: relojes inteligentes y *cloud*. Siguiendo este esquema, el capítulo ha sido dividido en 3 secciones: 2.1. Tendencias de IoT en el ámbito de la salud, 2.2 Avances académicos y 2.3. Avances empresariales.

2.1. Tendencias de IoT en el ámbito de la salud

En los últimos años, ha habido un avance significativo en la innovación en el ámbito de la salud gracias al uso de dispositivos conectados. El IoT está revolucionando tanto la medicina como la salud en general (23). Según los resultados de la última investigación llevada a cabo por Microsoft, denominada "*IoT Signals for Healthcare*", en la que se encuestó a diversas organizaciones del sector sanitario, el 85% del ecosistema considera que la tecnología IoT es esencial para el éxito de sus operaciones (24). Además, se ha constatado que el 78% de estas organizaciones tiene previsto aumentar su inversión en tecnologías IoT en los próximos años (24).

Este crecimiento continuo en el uso del Internet de las cosas en el ámbito de la salud, también conocido como HIoT (*Healthcare Internet of Things*), se debe a los numerosos beneficios que mejoran la calidad de vida de millones de personas en todo el mundo, como la prevención de situaciones de riesgo, la mayor eficiencia de los tratamientos médicos, la mejora en la calidad de servicio y el soporte en la toma de decisiones, entre otros (23).

Aunque cada aplicación HIoT tiene una topología diferente en función de las necesidades que pretenda cubrir, muchas de las soluciones desarrolladas hoy en día siguen la arquitectura básica planteada en la figura 8, donde se distinguen tres componentes fundamentales (25):

- *Publisher* o publicador: se trata de una red de sensores y otros dispositivos médicos conectados que bien de forma individual o bien de forma conjunta registran información de los pacientes, como la frecuencia cardiaca, temperatura o presión sanguínea. El publicador se encarga de enviar esta información al *broker* (25).

- *Broker*: responsable del procesamiento y almacenamiento de los datos en la nube (25).

- *Suscriber* o suscriptor: monitoriza los datos de los pacientes. Este suscriptor puede ser personal de un hospital, un cuidador o un asegurador médico, entre otros (25).

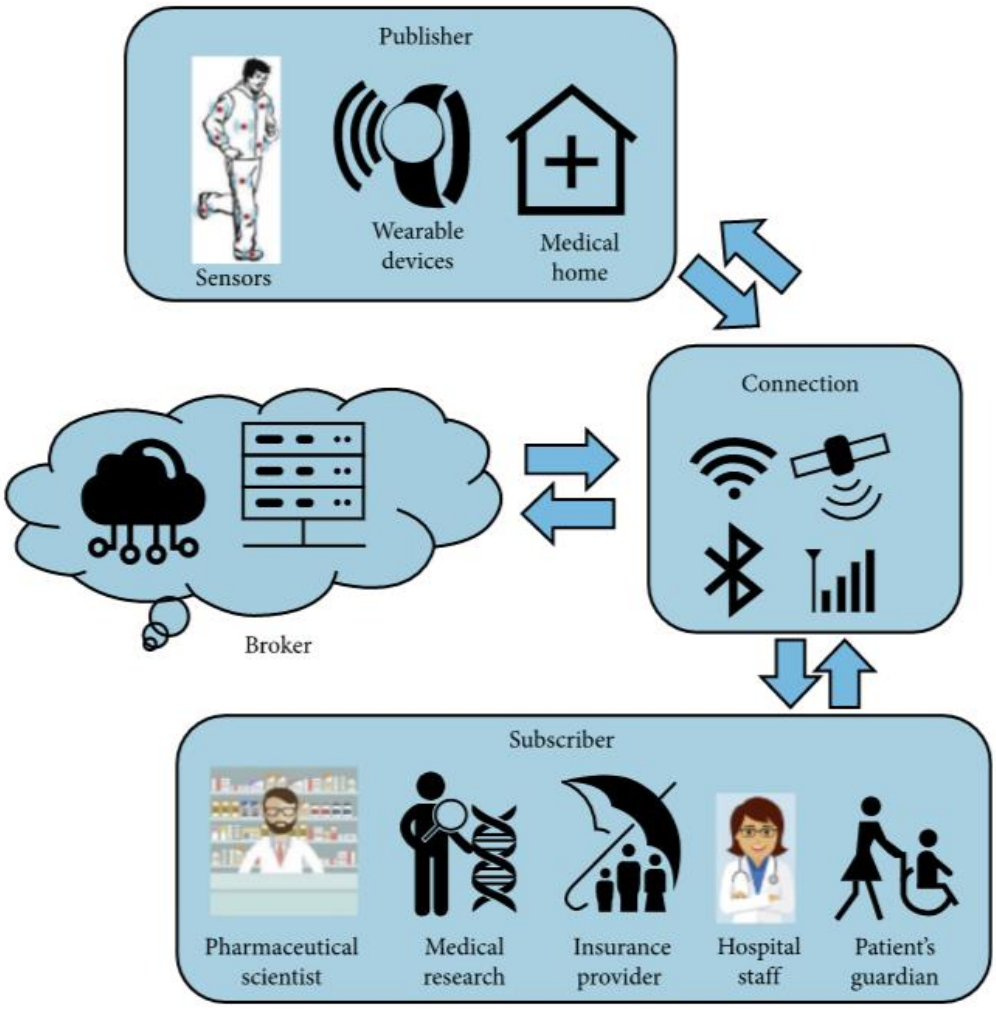

Figura 8: Arquitectura básica de una aplicación HIoT (25)

Es importante destacar cuando hablamos de publicadores la creciente importancia que están cobrando los dispositivos portátiles más conocidos como *wearables*. Esta creciente popularidad se debe, entre otras cosas, a su bajo coste y al hecho de ser no invasivos. Ejemplos de estos dispositivos se muestran en la figura 9: sensores de detección de movimiento, sensores de presión, sensores de temperatura, sensores ECG (electrocardiograma), monitorización de los niveles de glucosa, plantillas inteligentes, monitoreo fetal inalámbrico y ropa inteligente (26).

Figura 9: Ejemplos de sensores wearables (26)

2.2. Avances académicos

Si se tienen en cuenta los rápidos avances tecnológicos en el ámbito de los *smartwatch*es que están revolucionando el mundo del fitness y la salud (27) y el hecho de que estos dispositivos inteligentes no dejen de ganar popularidad (28), no es de extrañar que cada vez sean más las líneas de investigación dedicadas a desarrollar nuevas soluciones en el ámbito de la monitorización de individuos. En la figura 10, se muestra la evolución exponencial en el número de publicaciones en HIoT por año desde el 2013 al 2018.

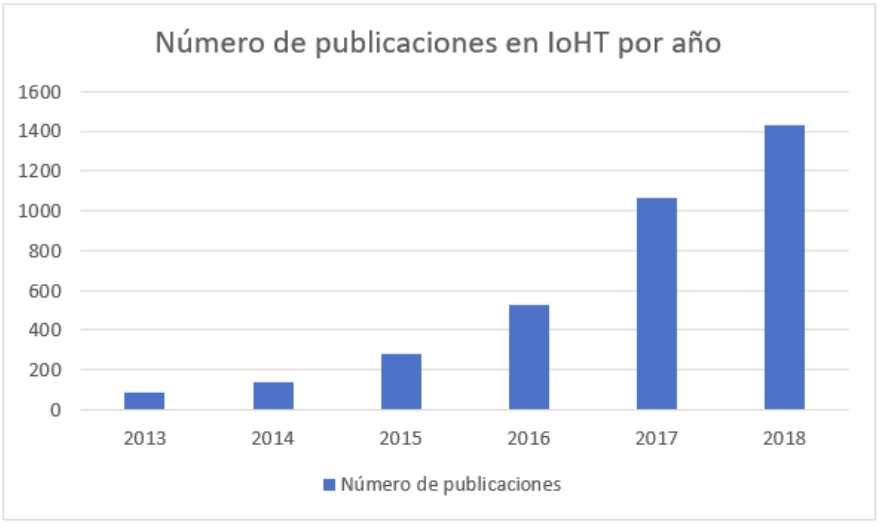

Figura 10: Evolución del número de publicaciones en HIoT por año en el periodo 2013-2018 (23)

Una investigación reciente es "*SmartCare: Detecting Heart Failure and Diabetes Using Smartwatch"* (27). Este estudio comienza señalando que los estilos de vida actuales están derivando en un aumento de enfermedades como fallos cardíacos y diabetes que demandan una revolución del sistema de salud. Estas enfermedades son normalmente asintomáticas en las fases tempranas, por lo tanto, el desarrollo de soluciones inteligentes que monitoricen automáticamente el estado del corazón de los usuarios sin intervención humana es crucial. Por ello, el estudio se centra en proponer la solución SmartCare cuya arquitectura se recoge en la figura 11. SmartCare permite la detección automática de fallos cardíacos y diabetes a partir de los datos recogidos con relojes inteligentes utilizando algoritmos de Random Forest y Regresión Logística. Ambos algoritmos son métodos de aprendizaje automático empleados para la resolución de problemas de clasificación y regresión. El estudio concluye que es posible detectar el fallo cardiaco y la diabetes con un valor F1 de 0.72 para ambos casos (27).

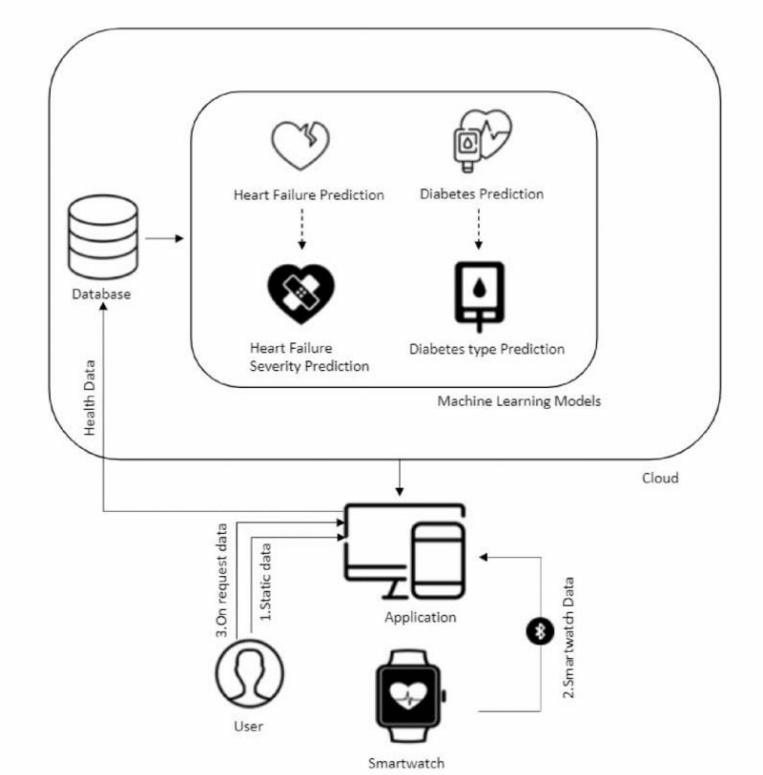

Figura 11: Diagrama de la solución SmartCare (27)

Un segundo estudio es "*Deep learning based fall detection using Smartwatches for healthcare applications"* (29), donde los investigadores buscan desarrollar un sistema basado en los datos recogidos por los relojes inteligentes con el fin de detectar posibles caídas de los usuarios. Para ello, han desarrollado una aplicación móvil que recoge los datos de los sensores de aceleración y giroscopio de los *smartwatche*s y los transfiere a la nube. Una vez los datos han sido enviados a la nube, proceden a aplicar algoritmos de *deep learning* para detectar si el usuario ha sufrido una caída. El esquema de bloques de la solución concreta puede encontrarse en la figura 12. El sistema logra una precisión del 97,25% en *leave-one-subject-out cross-validation* (29). Este método, conocido como

validación cruzada dejando uno fuera en español, divide las muestras en dos sub-bloques: sub-bloque de validación y sub-bloque de entrenamiento. De esta forma se entrenan tantos modelos como números de muestras.

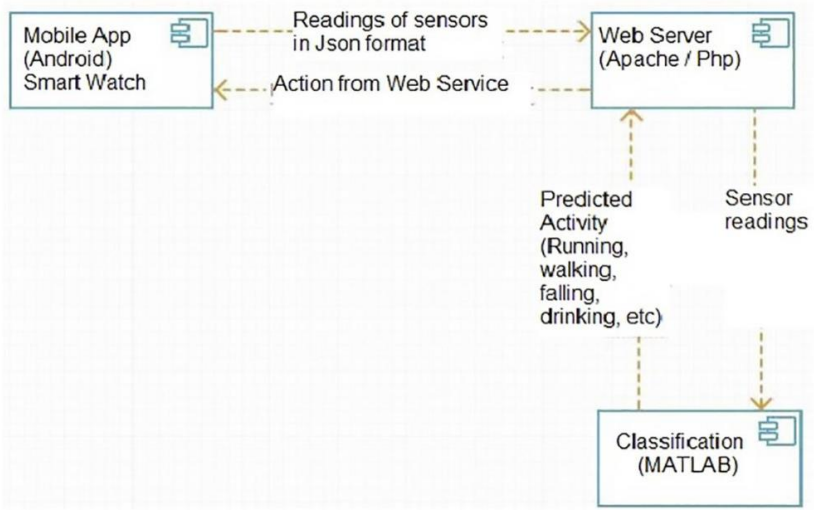

Figura 12: Diagrama de bloques para un sistema de detección de caídas (29)

Un tercer estudio en el que se propone explotar los datos obtenidos con relojes iteligentes haciendo uso del *cloud* es "*Real-time running workouts monitoring using Cloud-Edge computing"* (28). En este artículo publicado en la revista Neural Computing and Applications, los autores Florin Pop y Maria-Ruxanda Avram presentan una solución que gracias a la tecnología *cloud* logar establecer una conectividad *end-to-end* entre atletas y entrenadores. El esquema de la arquitectura de esta solución puede encontrarse en la figura 13. Se trata de un modelo altamente escalable que recoge todos los datos de actividad de los deportistas y se los envía a los entrenadores. Los entrenadores pueden analizar el rendimiento del entrenamiento y mandar su *feedback* a los usuarios en tiempo real (28).

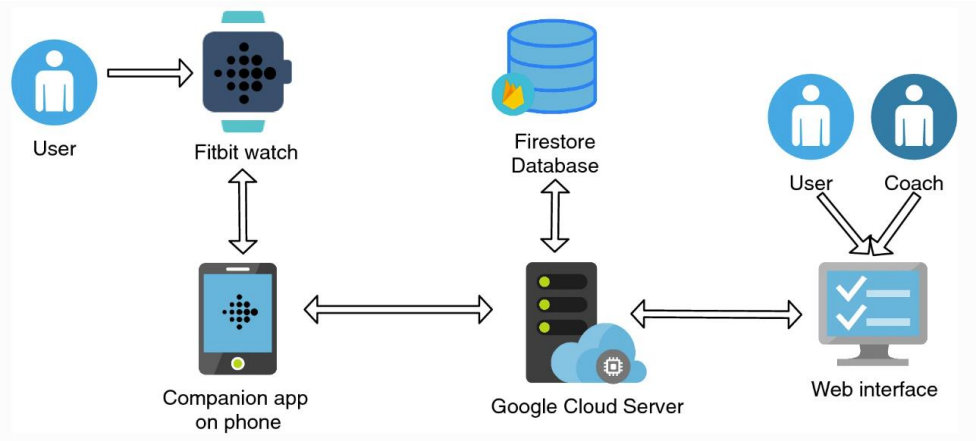

Figura 13: Arquitectura de una solución para le monitoreo de entrenamientos en tiempo (28)

2.3. Avances empresariales

Los avances no se limitan únicamente al mundo académico. En el contexto empresarial también se está apostando por desarrollar soluciones tecnológicas que permitan la explotación de datos recolectados a partir de dispositivos inteligentes para monitorear la salud de pacientes y deportistas.

Un caso a destacar es el de Apple. Esta compañía está apostando por desarrollar aplicaciones que convierten su reloj inteligente, el Apple Watch, en un dispositivo que permite a los pacientes y deportistas llevar el control de su salud cardiovascular, actividad física y medicación, entre otras (30). Entre los avances realizados por la compañía americana en los últimos años conviene destacar:

- Notificaciones relacionadas con la frecuencia cardiaca. Como se presenta en la figura 14, el reloj notifica a los usuarios cuando sus pulsaciones cardíacas están inusualmente altas o bajas (30).

Figura 14: Notificaciones relacionadas con la frecuencia cardiaca en Apple Watch (30)

- Notificaciones de ritmo irregular. En la figura 15 se muestra como el reloj notifica al usuario de cuando se producen ritmos irregulares (30).

Figura 15: Notificaciones de ritmo irregular en Apple Watch (30)

- Detección de caídas. El reloj es capaz de detectar caídas fuertes, como se observa en la figura 16 (30).

Figura 16: Detección de caídas en Apple Watch (30)

- Medicación. El reloj puede ser configurado para mandar recordatorios y ver las medicinas activas, como se recoge en la figura 17 (30).

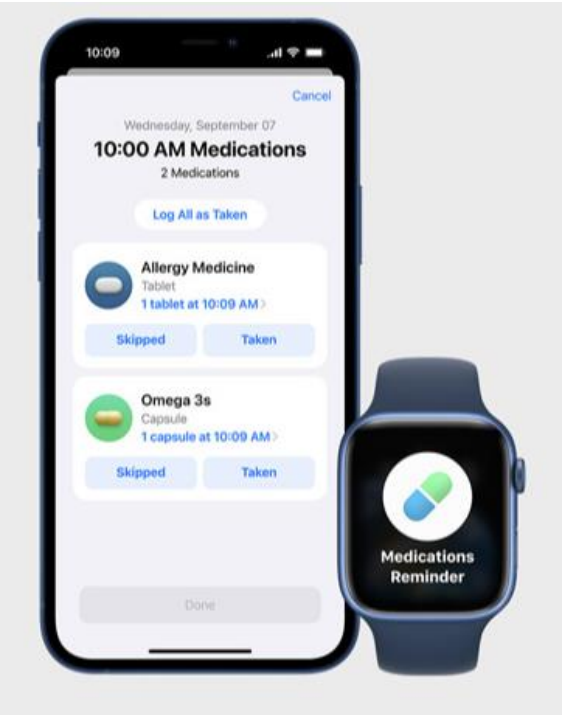

Figura 17: Recordatorios de medicación en Apple Watch (30)

Otro caso de gran interés es el de Garmin, especialmente en el marco de este TFM. Garmin Health ofrece soluciones empresariales personalizadas que aprovechan los datos obtenidos a través de los sensores de su extensa gama de dispositivos inteligentes portátiles con el fin de desarrollar aplicaciones innovadoras para los mercados de monitoreo de pacientes, bienestar corporativo y salud poblacional (31). Son múltiples las colaboraciones que Garmin Health está estableciendo con otras empresas del sector con el objetivo de llevar el mundo de la monitorización continua y de la salud digital a partir de sus dispositivos inteligentes al siguiente nivel.

En 2022, la compañía publicó una colaboración con Labfront, empresa de biomarcadores digitales especializada en el análisis de datos de salud que está revolucionando el mundo de la investigación académica con su plataforma de recolección y análisis de biomarcadores sin código (32). La colaboración busca acelerar la adopción de tecnologías portátiles, como relojes inteligentes, en el mundo de la investigación académica. El objetivo fundamental es el de expandir los beneficios de la colecta y análisis de los datos recogidos mediante los sensores de los dispositivos Garmin a un mayor número de investigadores (33). En la figura 18 se muestra como ya es posible añadir los datos registrados por un dispositivo Garmin a un proyecto de investigación, conectando la aplicación de Labfront con Garmin IQ Connect (34).

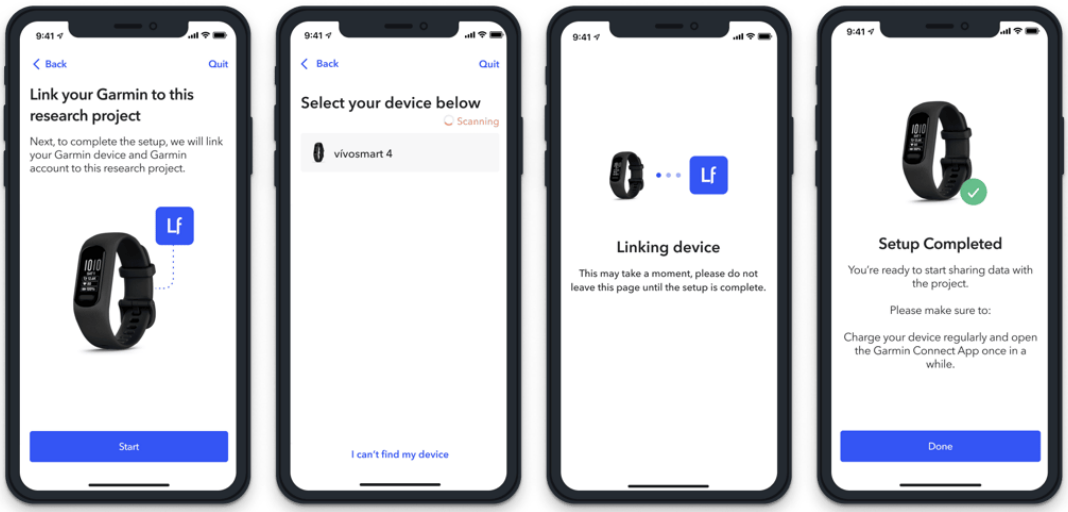

Figura 18: Configuración de un dispositivo Garmin para el envío de los datos registrados a la aplicación de Labfront (34)

Más recientemente, el 23 de enero de 2023 la empresa estableció una colaboración con HumanITcare para impulsar el acceso a la salud conectada. HumanITcare es una empresa dedicada a proporcionar atención médica remota personalizada gracias a la inteligencia artificial (35). En la figura 19 se representa un ejemplo del funcionamiento del servicio proporcionado por HumanITCare. Jön Watzke, director senior Garmin Health recalca con esta colaboración la importancia de apostar por soluciones de salud digital remota para llegar al máximo número de personas y afirma que la integración de los datos fisiológicos recogidos con los relojes Garmin le dará a HumanITCare el poder de mejorar sus soluciones con métricas de salud premium, mejorando los casos de usos clínicos y en el cuidado de la salud (36).

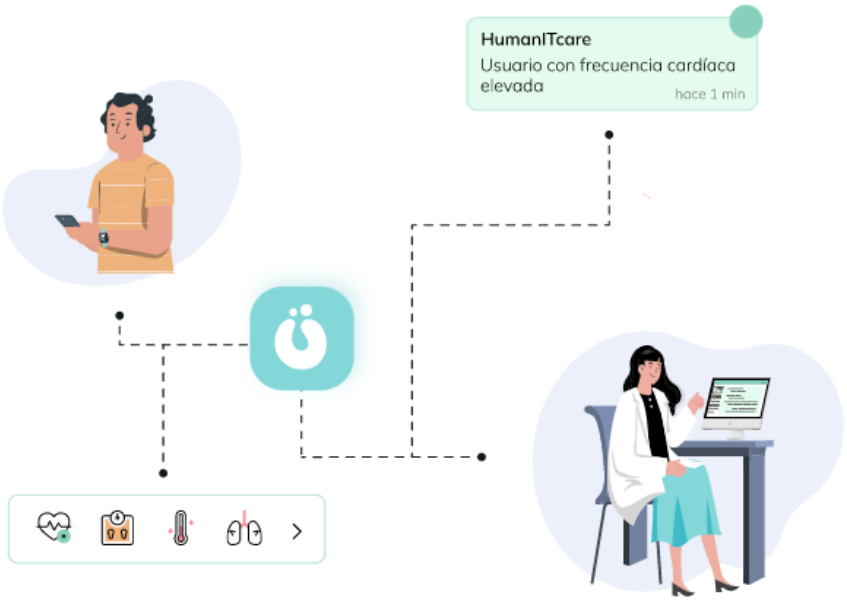

Figura 19: Funcionamiento del servicio de atención médica remota ofrecido por HumanITcare (35)

Otra colaboración realmente interesante de cara a aprovechar al máximo la información recogida por los sensores de Garmin y potenciar los servicios de salud digitales es la establecida con Chronolife. Esta empresa tiene la misión de facilitar a los profesionales del sector sanitario la información necesaria para la intervención temprana de pacientes y garantizar la atención continua que llene los vacíos existentes entre el hospital y el hogar (37). El objetivo de esta colaboración es enriquecer las condiciones del monitoreo continuo de pacientes en Europa y en Estados Unidos. Los datos fisiológicos y de actividad recogida haciendo uso de los sensores presentes en los dispositivos inteligentes de Garmin se integraran automáticamente con el data *hub* de Chronolife, de esta la información de los pacientes se encuentra centralizada en una única plataforma (38). En la figura 20 se presenta como las distintas fuentes de datos se integran con el *hub* de Chronolife y como el personal autorizado puede acceder a estos datos (39).

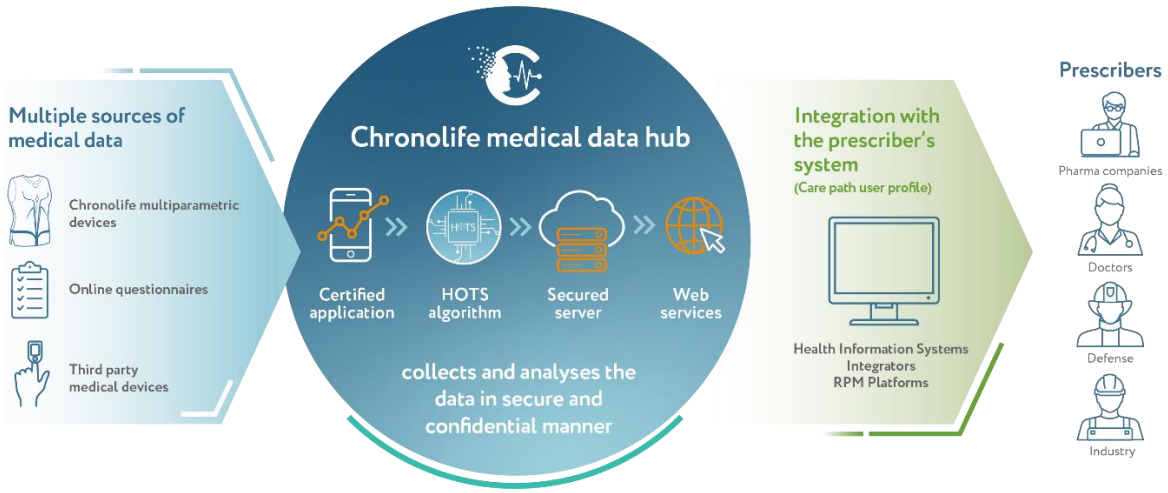

Figura 20: Funcionamiento del hub de Chronolife para la monitorización remota de pacientes (39)

Por otro lado, al igual que Apple, Garmin también trabaja en el desarrollo de aplicaciones con el fin de introducir nuevas soluciones de monitorización de deportistas y pacientes a partir de los datos recogidos por sus dispositivos inteligentes.

Un ejemplo de esto es la funcionalidad de detección de incidentes. Esta función permite establecer una serie contactos de emergencia a los que se enviará la ubicación en caso de detectarse un incidente grave. Un ejemplo de esta funcionalidad se muestra en la figura 21. No obstante, es importante tener en cuenta que ni la aplicación de Garmin *Connect* ni los dispositivos Garmin tienen la capacidad de contactar con servicios de emergencia en el caso de un incidente. Por tanto, no puede utilizarse como método principal para obtener ayuda en caso de emergencia (40). Con el fin de solucionar esta limitación, en este TFM se ha desarrollado una solución que permite el envío de un correo electrónico a emergencias en caso de detectarse una anomalía para poder mandar ayuda lo antes posible.

Figura 21: Detección de incidentes en un reloj Garmin

Otras empresas que también están apostando por la inversión en IoT en general y en el ámbito de la salud en particular son los principales proveedores de cloud: Microsoft, Amazon Web Services (AWS) y Google. En la tabla siguiente se recogen servicios ofrecidos por cada uno de estos proveedores que pueden ser empleados para el desarrollo de aplicaciones IoT en el ámbito de la monitorización continua.

Microsoft	AWS	Google Cloud
Azure IoT Central	-	-
Azure IoT Hub	AWS IoT Core	Cloud IoT Core
Azure Event Hub	Amazon Kinesis	Google Cloud Pub/Sub

Tabla 1: Oferta de servicios para la implementación de soluciones IoT en los principales proveedores de cloud

El primero de ellos, Azure IoT Central, es una plataforma de aplicaciones de IoT totalmente administrada que simplifica la creación y administración de soluciones de IoT. Proporciona una interfaz sencilla e intuitiva para la configuración, monitoreo y administración de dispositivos IoT. Este servicio es de especial utilidad para usuarios que buscan una solución de bajo código para la construcción y administración de aplicaciones de IoT sin tener que preocuparse de la infraestructura subyacente (41). Según Microsoft, la supervisión continua de pacientes es crucial a la hora reducir la probabilidad de readmisiones, la administración eficaz de enfermedades crónicas y la mejora de resultados de los pacientes (15). En línea con su enfoque en la monitorización remota y continua de pacientes, Microsoft ha desarrollado una plantilla específica para la supervisión continua de pacientes en Azure IoT Central. Un ejemplo de esta plantilla se recoge en la figura 22 (42).

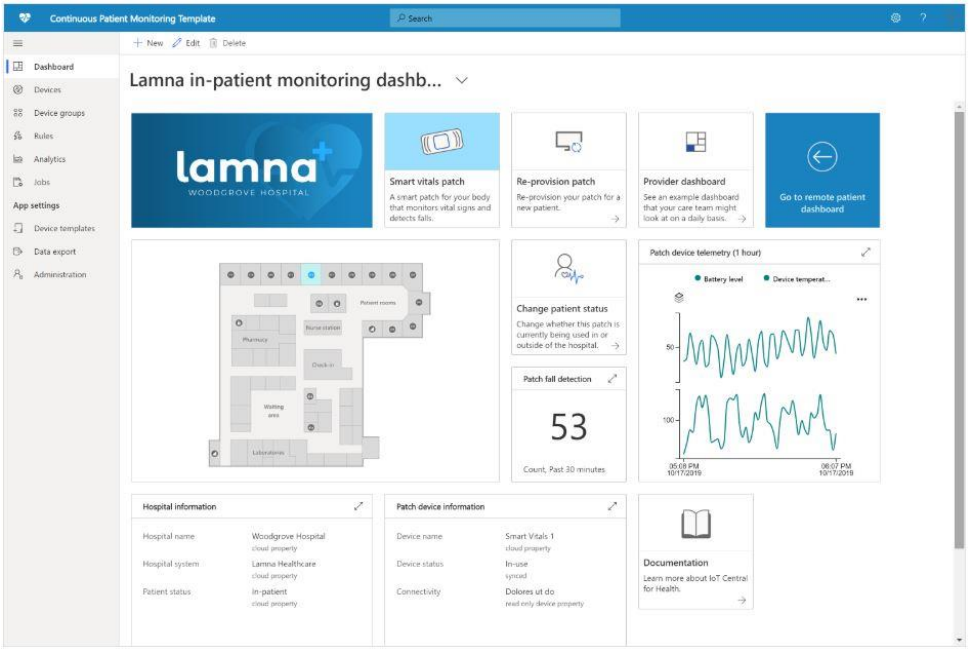

Figura 22: Plantilla disponible en Azure IoT Central para la monitorización continua de pacientes (42)

Como se recoge en la tabla 1, ni AWS ni Google ofrecen un servicio equivalente a Azure IoT Central en términos de funcionalidad y enfoque centrado en la aplicación.

El segundo servicio que destacar es Azure IoT Hub, solución que permite la conectividad y administración segura de dispositivos IoT. Este servicio proporciona capacidades de mensajería bidireccional, identidad por dispositivo y varias características específicas de IoT (43). A diferencia de Azure IoT Central, esta solución es adecuada para usuarios que desean construir soluciones de IoT personalizadas y tener un mayor grado de control sobre la infraestructura e integración con otros servicios. En la figura 23 se muestra cómo una amplia variedad de dispositivos IoT pueden conectarse a un Azure IoT Hub para su monitoreo, control y generación de información en la nube (44).

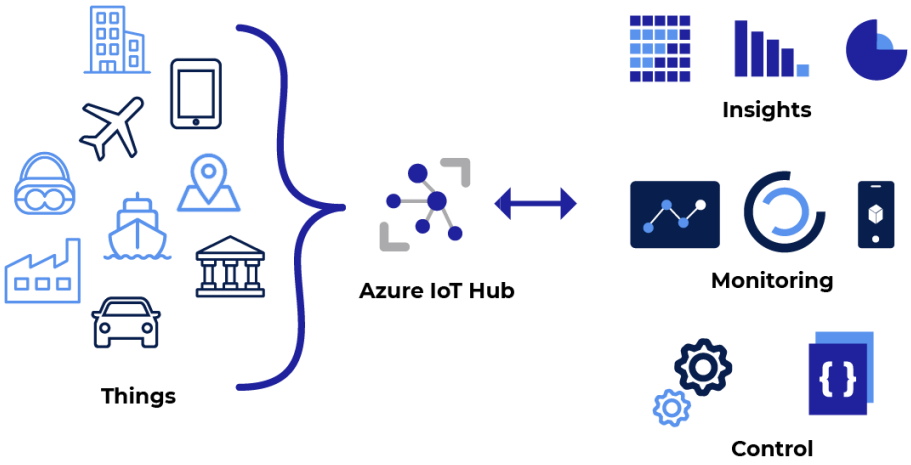

Figura 23: Funcionamiento de Azure IoT Hub (44)

Como se recoge en la tabla 1, tanto AWS como Google ofrecen alternativas similares a Azure IoT Hub. Por un lado, AWS ofrece AWS IoT Core, un servicio que al igual que Azure IoT Hub permite la comunicación bidireccional segura entre dispositivos IoT y aplicaciones en la nube, proporciona autenticación y autorización por dispositivo y ofrece características específicas de IoT (45). El funcionamiento de este servicio se recoge en la figura 25.

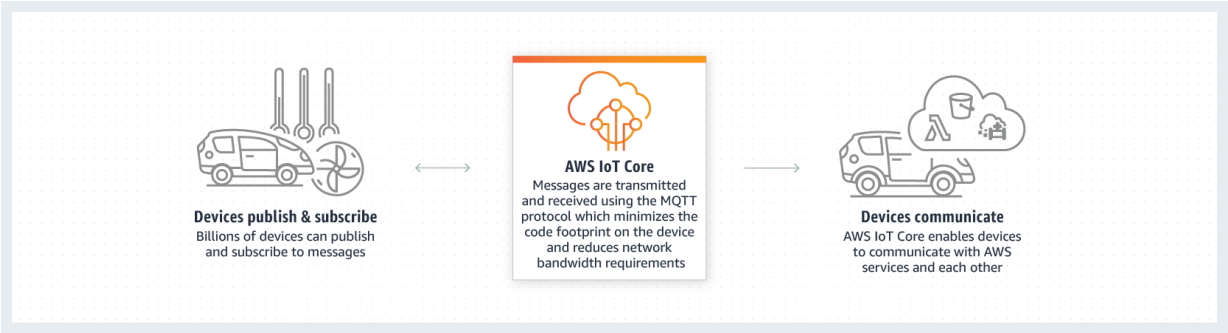

Figura 24: Funcionamiento de Amazon IoT Core (45)

Por otro lado, Google ofrece Google Cloud IoT que permite la conexión y administración de dispositivos médicos de forma segura y escalable, lo que facilita la recopilación de datos en tiempo real y su procesamiento en la nube para su posterior análisis (51). Al igual que Azure IoT Hub y AWS IoT Core, Google Cloud IoT Core proporciona comunicación bidireccional, autenticación y autorización por dispositivo y características específicas de IoT. En la figura 27 se muestra el esquema de una solución que emplea Google Cloud IoT Core.

El tercer servicio recogido en la tabla es Azure Event Hub, servicio de ingesta de datos en tiempo real sencillo y escalable que puede recibir y procesar millones de eventos por segundo de múltiples fuentes (46). Esta solución es ideal para aplicaciones que requieren el procesamiento de grandes volúmenes de eventos en tiempo real, como análisis de datos, telemetría de dispositivos IoT y monitoreo de aplicaciones. A diferencia de Azure IoT Hub y como se muestra en la figura 25, la comunicación entre los dispositivos y la nube es unidireccional (47).

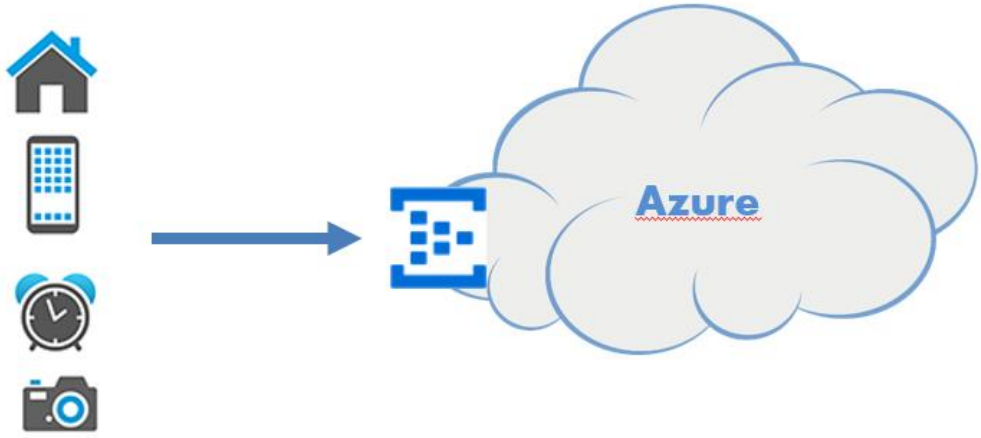

Figura 25: Funcionamiento Azure Event Hub (47)

Como se muestra en la tabla 1, tanto AWS como Google Cloud ofrecen servicios con funcionalidades muy similares a Azure Event Hub. En el caso de AWS, el servicio equivalente recibe el nombre de Amazon Kinesis Data Streams. Este servicio, al igual que Azure Event Hub, permite la ingesta y el procesamiento en tiempo real de grandes volúmenes de datos de eventos procedentes de múltiples fuentes. Como se muestra en la figura 26, este servicio se integra con otros servicios de AWS para el análisis y procesamiento de eventos.

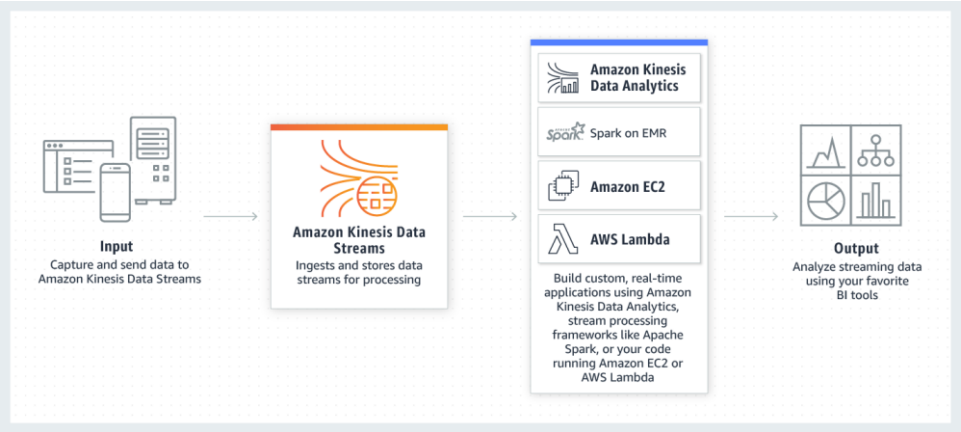

Figura 26: Funcionamiento de Amazon Kinesis Data Streams (48)

En el caso de Google Cloud, el servicio equivalente recibe el nombre de Google Pub/Sub. Esta solución proporciona una infraestructura de mensajería en tiempo real y escalable para la ingesta y distribución de eventos entre servicios independientes, sistemas y aplicaciones. Al igual que Amazon Kinesis Data Streams y Azure Event Hub, este servicio permite la ingesta masiva de eventos y se integra con otros servicios de Google Cloud para el análisis y procesamiento de eventos (49). En la figura 27 se muestra el esquema de una solución IoT basado en Google Cloud y Google Cloud Pub/Sub (50).

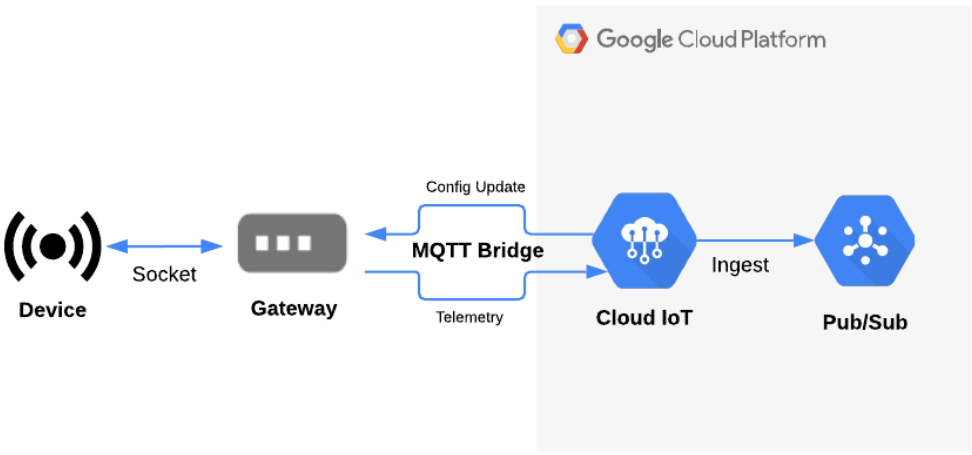

Figura 27: Esquema de una solución IoT basado en Google Cloud IoT y Google Cloud Pub/Sub (50)

Tras la lectura de este capítulo, es evidente que existe un interés creciente en la implementación de soluciones de IoT que permitan la monitorización de individuos. Los grupos de investigación, así como las empresas dedicadas al desarrollo de dispositivos portátiles de IoT, como los relojes inteligentes, y los grandes proveedores de servicios *cloud* están invirtiendo en esta tendencia.

Por consiguiente, este Trabajo de Fin de Máster se centra en aprovechar los últimos avances en la tecnología de relojes inteligentes y las oportunidades que brinda la tecnología *cloud*, con el objetivo de desarrollar una plataforma de monitoreo continuo de las constantes vitales y la ubicación en tiempo real de los corredores. De esta manera, se busca mitigar algunos de los riesgos que se mencionaron en el capítulo introductorio de esta memoria.

3. Diseño e Implementación

El presente capítulo de la memoria aborda el diseño de la solución de monitorización continua de deportistas, así como los pasos seguidos para la implementación de dicha solución. Con el objetivo de estructurar adecuadamente el contenido del presente capítulo, este ha sido dividido en dos subcapítulos, el primero dedicado exclusivamente al diseño de la solución y el segundo centrado en la fase de implementación.

3.1. Diseño

En esta sección, se procederá a describir en detalle el diseño de la solución, especificando los distintos servicios y recursos que la conforman. Para ello, nos vamos a apoyar en el diagrama de bloques recogido en la figura 28. En esta figura 28 se ilustran los diferentes elementos que integran nuestra plataforma para la monitorización continua de corredores: (1) *Smartwatch*, (2) *gateway*, (3) Azure Storage account, (4) Azure Event Hub, (5) Azure Stream Analytics, (6) Power BI, (7) Azure Function, (8) Azure Logic App y (9) Outlook.

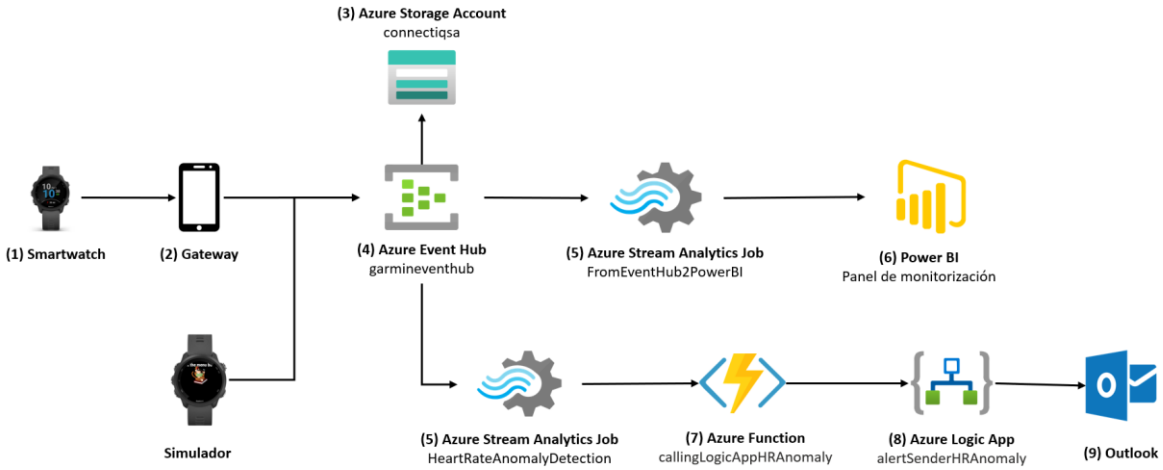

Figura 28: Esquema de la solución de monitoreo de corredores en tiempo real con relojes inteligentes y Azure

Al examinar detenidamente la figura 28, podemos identificar que nuestra solución puede dividirse en cuatro funcionalidades distintas:

- **Conexión de nuestros relojes inteligentes a la nube**. Esta funcionalidad recoge los componentes (1) *smartwatch*, (2) *gateway* y (4) Azure Event Hub.

- **Generación del registro histórico de los datos enviados** por nuestros dispositivos conectados para posteriores análisis y auditorías. Esta segunda funcionalidad involucra a los componentes (3) Azure Storage Account y (4) Azure Event Hub.

- **Generación del panel de monitorización de corredores en tiempo real**. Agrupa los componentes (5) Azure Stream Analytics job y (6) Power BI.

- **Creación de alertas en tiempo real** cuando alguno de los dispositivos conectados registra pulsaciones anómalas. Los recursos que conforman esta funcionalidad son: (5) Azure Stream Analytics job, (7) Azure *Function*, (8) Azure Logic App y (9) Outlook.

A continuación, se procederá a describir cada uno de estos elementos que integran la solución presentada en el diagrama de bloques de la figura 28. El propósito de esta descripción es permitir al lector comprender el papel desempeñado por cada uno de estos elementos, así como la razón detrás de su elección.

(1) *Smartwatch*

El primer componente del diagrama es un *smartwatch* o reloj inteligente. Todas las pruebas realizadas en la elaboración del presente TFM han sido desarrolladas a partir de un reloj Garmin Forerunner 245. No obstante, es conveniente destacar que esta solución es extensible a otros modelos y otras marcas de relojes, siempre y cuando se trate de dispositivos BLE (*Bluetooth Low Energy*).

Es importante señalar que estos dispositivos BLE no tienen la capacidad comunicarse directamente con el *cloud*. Por lo tanto, van a precisar de un *gateway* para el envío de datos a la nube (15), como se muestra en la figura 28 de este capítulo.

En esta figura 28 es posible observar también que conectado al Azure Event Hub, además de un *smartwatch*, hay un simulador. Este simulador ha sido diseñado e implementado utilizando el lenguaje de programación Monkey C persiguiendo dos objetivos fundamentalmente:

- La generación de los valores de frecuencia cardiaca, latitud y longitud necesarios para probar el correcto funcionamiento de la solución.

- El estudio del comportamiento del sistema cuando se tiene más de un dispositivo conectado a la plataforma de monitorización.

Monkey C es un lenguaje de programación orientado a objetos y basado en eventos que ha sido desarrollado por Garmin con el propósito específico de crear aplicaciones para sus dispositivos *wearables*, tales como relojes inteligentes y dispositivos de fitness (51).

(2) Gateway

En la fase de diseño de este *gateway*, se ha optado por utilizar un teléfono móvil como dispositivo central. Este teléfono móvil se encarga de recopilar los datos BLE enviados por el reloj inteligente y transmitirlos al Azure Event Hub. Para implementar esta funcionalidad, se ha desarrollado una aplicación en Monkey C.

La aplicación desarrollada desempeña dos funciones principales: la recolección de datos provenientes del reloj inteligente y su posterior envío al Azure Event Hub. Además, como se mencionaba anteriormente, esta aplicación ofrece la capacidad de simular datos para llevar a cabo pruebas y evaluaciones exhaustivas del sistema según sea necesario.

(3) Azure Storage Account

Azure Storage Account es un servicio de almacenamiento en la nube de Microsoft Azure altamente escalable, disponible y de bajo coste, diseñado específicamente para almacenar grandes volúmenes de datos no estructurados (52).

Una de las ventajas clave de Azure Storage Account es su integración con Azure Events Hub Capture, que permite almacenar de manera automática los datos recibidos por el Event Hub en tiempo real. De esta manera, se simplifica el proceso de captura y almacenamiento de datos sin necesidad de desarrollar soluciones personalizadas (53).

Otra opción de almacenamiento que también ofrece integración directa con Azure Events Hub Capture es Azure Data Lake Storage (53). Azure Data Lake Storage es una solución de almacenamiento escalable y distribuida que puede manejar grandes volúmenes de datos estructurados y no estructurados. Resulta una herramienta especialmente útil para la realización de análisis avanzados o procesamientos de big data sobre los datos almacenados (54). En la tabla 2 se recoge la comparación ente ambos servicios.

	Azure Storage Account	**Azure Data Lake Storage**
Uso	• Almacenamiento de grandes cantidades de datos	• Almacenamiento y análisis de *big data*
Ventajas	• Gran variedad de servicios de almacenamiento	• Diseñado específicamente para el análisis de *big data*
Desventajas	• No optimizado para el análisis de *big data*	• Menos versátil que Azure Storage Account

Tabla 2: Comparación entre Azure Storage Account y Azure Data Lake Storage

Dado que el objetivo principal de la solución diseñada es simplemente almacenar los datos brutos sin procesamiento o análisis avanzados, se ha tomado la decisión de utilizar una Azure Storage Account en lugar de Azure Data Lake Storage. Se concluye por tanto que Azure Storage Account es la opción más rentable, simple y adecuada para la creación de un registro histórico a partir de los datos recibidos en Azure Event Hub. Este registro permite el análisis y auditoría de los datos enviados por los dispositivos conectados a nuestra solución si es necesario.

(4) Azure Event Hub

Azure Event Hub un servicio de Azure que permite la ingesta de grandes cantidades de datos de eventos en tiempo real. Se trata de una plataforma de streaming de eventos escalable y altamente disponible, capaz de procesar y almacenar millones de eventos por segundo (55).

Una vez recibidos los datos enviados por los dispositivos conectados, Azure Event Hub puede integrarse fácilmente con otros servicios en Azure con el fin de procesar y analizar los datos recibidos en tiempo real. Por ejemplo, permite la integración con Azure Functions, Azure Stream Analytics o Azure HDInsights, entre otros. Por otro lado, como se mencionaba en la sección anterior, este servicio permite también la integración con servicios de almacenamiento en la nube como Azure Blob Storage o Azure Data Lake Storage para el procesamiento por microlotes o para retener los datos capturados a largo plazo (56). Por último, Azure Event Hub ofrece la posibilidad de integración con servicios externos como Apache Spark (55).

En el diseño de la solución se han considerado otros servicios de Azure que también podrían ser utilizados para la conectividad de los relojes inteligentes a la nube de Microsoft, como Azure IoT Hub o Azure IoT Central. La tabla 3 presenta una comparativa entre los tres servicios para comprender mejor su uso, ventajas y desventajas.

Como se comentaba previamente, Azure Event Hub es un servicio escalable, flexible y de bajo coste especialmente útil para la ingesta de grandes cantidades de datos. No obstante, a diferencia de Auzre IoT Hub y Azure IoT Central, este servicio no ofrece comunicación bidireccional entre los dispositivos y la nube, siendo posible únicamente el envío de datos desde los dispositivos hacia Azure Event Hub.

Azure IoT Hub es un servicio de Azure que ha sido diseñado específicamente para la conectividad de dispositivos IoT y la ingesta de datos (43). Una de sus ventajas en comparación con Azure Event Hub es que permite la comunicación bidireccional con los dispositivos IoT, además de contar con características adicionales para la gestión de dispositivos, como autenticación, autorización y actualización de firmware (57). Sin embargo, una desventaja respecto a Azure Event Hub es que puede resultar en un costo más elevado, especialmente si se utilizan las características adicionales de gestión de dispositivos. Además, las tareas de configuración y mantenimiento de Azure IoT Hub pueden ser más complejas debido a la gestión de dispositivos.

Por último Azure IoT Central es una plataforma de aplicaciones IoT como servicio que simplifica la creación de aplicaciones de IoT (41). Al igual que Azure IoT Hub, este servicio proporciona características adicionales para la gestión de dispositivos. Una de las principales ventajas de este servicio con respecto a los dos anteriores son las plantillas predefinidas para soluciones de IoT comunes, lo que permite acelerar el tiempo de desarrollo de estas soluciones (58). Entre sus desventajas, cabe destacar que puede generar costos más altos y que es un servicio poco personalizable.

	Azure Event Hub	**Azure IoT Hub**	**Azure IoT Central**
Uso	• Ingesta de grandes cantidades de datos en tiempo real • Orientado para la transmisión de datos en tiempo real	• Diseñado específicamente para la conectividad de dispositivos IoT y la ingesta de datos	• Plataforma de aplicaciones IoT como servicio que simplifica la creación de soluciones IoT
Ventajas	• Escalable • Flexible • Bajo coste	• Incluye características específicas de IoT como la administración de dispositivos o la comunicación bidireccional • Compatibilidad con una gran variedad de protocolos IoT	• Proporciona una interfaz de usuario gráfica para la creación y gestión de soluciones IoT • Proporciona plantillas predefinidas para soluciones IoT comunes • Incluye características específicas de IoT como la administración de dispositivos o la comunicación bidireccional
Desventajas	• No incluye características específicas de IoT como la administración de dispositivos o la comunicación bidireccional	• Coste más elevado que Azure Event Hub • Configuración y mantenimiento más complejo que Azure Event Hub	• Solución poco personalizable • Puede ser más costoso que las dos opciones anteriores

Tabla 3: Comparación entre Azure Event Hub, Azure IoT Hub y Azure IoT Central

En el contexto del diseño de nuestra aplicación, dado que la ingesta de grandes cantidades de datos y el monitoreo en tiempo real son las principales preocupaciones, Azure Event Hub es la solución más simple y económica. Además, es importante tener en cuenta que no todos los dispositivos Garmin admiten comunicación bidireccional y que la implementación de esta comunicación bidireccional y de las características de gestión de dispositivos ofrecidas por Azure IoT Hub y Azure Event Hub es compleja.

Concluimos, por tanto, que la solución planteada hará uso de Azure Event Hub para recibir los datos enviados por nuestros dispositivos en tiempo real de forma eficiente y escalable.

(5) Azure Stream Analytics

Azure Stream Analytics es un servicio de Azure diseñado para analizar y procesar grandes cantidades de datos con latencias de submilisegundos. Por un lado, Azure Stream Analytics nos da la posibilidad de procesar datos en tiempo real, conectando nuestro recurso a servicios como Azure IoT Hub o Azure Events Hub. Por otro lado, también tenemos la posibilidad de utilizar este servicio para la ingesta y procesado de datos históricos, integrándolo por ejemplo con Azure Blob Storage (59).

La salida de este servicio puede ser enrutada a sistemas de almacenamiento como Azure Blob Storage, Azure SQL Database, Azure Data Lake Storage y Azure Cosmos DB o Power BI para la visualización de los datos en tiempo real. Este servicio también permite ejecutar análisis por lotes con servicios como Azure Synapse Analytics o HDInsight (59).

Como podemos observar en el diagrama de bloques de la solución presentado en la figura 28, vamos a contar con dos instancias de este servicio:

- FromEventHub2PowerBI nos permite representar en tiempo real los datos recibidos por el Event Hub en un panel de monitorización construido sobre Power BI. Este panel proporciona una visualización atractiva e interactiva en tiempo real de los datos. Esto va a permitir monitorear las constantes vitales y la ubicación de los corredores de forma intuitiva con el fin de poder tomar decisiones en base a datos actualizados.

- HeartRateAnomalyDetection es un componente clave en la detección de anomalías cardíacas en tiempo real. Esta instancia tiene como objetivo identificar cuando para un usuario determinado y en una ventana de tiempo específica se supera un cierto valor medio de frecuencia cardiaca. Si esta frecuencia cardiaca media supera el valor umbral establecido, este servicio va a llamar a un Azure *Function* con el objetivo de iniciar la lógica de generación de alertas.

Por un lado, existen tres opciones para la visualización de datos en tiempo real a través de *streaming datasets* en Power BI: API, Azure Stream Analytics y Pubnub. No obstante, la única opción que se integra directamente con Azure Event Hubs y se puede configurar sin necesidad de programar código adicional es Azure Stream Analytics. Por esta razón, se ha seleccionado este servicio para la representación en tiempo real de los datos recibidos por Azure Event Hub.

Por otro lado, en relación a la detección de anomalías cardiacas en tiempo real, también se ha decidido optar por Azure Stream Analytics por ser el servicio que permite ingerir, procesar y analizar datos directamente procedentes de Azure Event Hub (60).

(6) Power BI

Microsoft Power BI es una colección de servicios de software, aplicaciones y conectores que trabajan de forma conjunta para convertir datos procedentes de múltiples fuentes en información interactiva, coherente y atractiva visualmente (61). Además, Power BI permite la representación de datos y actualización de paneles en tiempo real gracias a la integración con servicios de transmisión de datos en tiempo real como Azure Stream Analytics o Power BI streaming (62).

En el contexto de nuestra solución, se va a construir un panel de visualización en tiempo real sobre Power BI que va a permitir la creación de una vista atractiva e interactiva de los datos de frecuencia cardiaca, ubicación y user ID de los corredores para poder monitorizar todos estos parámetros constantemente.

(7) Azure Function

Azure Stream Anlytics admite las siguientes salidas o *outputs*: Azure Data Explorer, Azure Function, Azure Synapse Analytics, Blob Storage/ADLS Gen2, Cosmos DB, Data Lake Storage Gen 1, Event Hub, PostgreSQL database, Power BI, Service Bus queue, Service Bus Topic, SQL Database y Table Storage (59). Por lo tanto, a la hora de desencadenar la lógica para el envío de un mensaje en tiempo real las opciones a considerar son: Azure Service Bus Queue, Azure Service Bus Topic y Azure Function. En la tabla 4 se presenta una comparación entre los tres servicios.

Azure Function es una solución *serverless* ofrecida por Microsoft que permite la ejecución en la nube de pequeños fragmentos de código conocidos como funciones. Dada su naturaleza serverless, Azure Functions permite a los desarrolladores centrarse en el código de sus aplicaciones sin tener que preocuparse por la implementación y el mantenimiento de la infraestructura subyacente (63). Azure Functions proporciona los servicios de computación que sean necesarios para satisfacer las necesidades de la aplicación (64).

La ejecución de estas funciones es desencadenada en respuesta a un evento específico generado por una variedad de *triggers*, como por ejemplo solicitudes HTTP, mensajes en colas o cambios en una base de datos (65).

	Azure Functions	Azure Service Bus Queue	Azure Service Bus Topic
Uso	• Servicio *serverless* que permite la ejecución de código personalizado en respuesta a eventos.	• Servicio de mensajería en la nube que permite la comunicación entre aplicaciones y servicios de manera confiable y segura utilizando colas	• Servicio de mensajería en la nube que permite la comunicación entre aplicaciones y servicios utilizando el patrón de publicación y suscripción.
Ventajas	• Flexibilidad para agregar lógica personalizada • Escalabilidad automática granular basada en el número de eventos entrantes • Fácil integración con otros servicios de Azure como Logic Apps	• Entrega confiable y duradera de mensajes	• Entrega confiable y duradera de mensajes • Soporte para patrones de mensajería uno a muchos • Capacidad para filtrar mensajes según criterios específicos.
Desventajas	• Puede ser más costoso que Sercice Bus si se ejecuta con alta frecuencia o durante largos peridos de tiempo	• Menos opciones de desencadenantes • Se pueden producir retrasos en el procesamiento de mensajes en caso de errores • No ofrece la escalabilidad automática granular que ofrece Azure Functions.	• Configuración y administración compleja • Menos opciones de desencadenantes • Se pueden producir retrasos en el procesamiento de mensajes en caso de errores. • No ofrece la escalabilidad automática granular que ofrece Azure Functions

Tabla 4: Comparativa entre Azure Functions, Azure Service Bus Queue y Azure Service Bus Topic

Por otro lado, Azure Service Bus Queue es un servicio de mensajería en la nube que permite la comunicación entre aplicaciones y servicios de manera confiable y segura utilizando colas mientras que Azure Service Bus Topic utiliza el patrón de publicación y suscripción (66).

Entre las ventajas de Azure Functions destacamos la flexibilidad a la hora de agregar lógica personalizada, la escalabilidad automática y la fácil integración con otros servicios de Azure como Logic Apps. Entre las ventajas de Azure Service Bus destacamos la entrega confiable y duradera de mensajes.

Con respecto a las desventajas, es cierto que Azure Functions puede tener un coste superior si se ejecuta con alta frecuencia o durante largos periodos de tiempo. De cara a nuestra solución esta desventaja del servicio no debería ser un gran inconveniente dado que esta función solo se va a ejecutar cuando se detecten pulsaciones anómalas, algo que normalmente sucede en situaciones esporádicas. Además, el periodo de ejecución de esta función es de milisegundos. La escalabilidad granular de Azure Functions garantiza la escalabilidad del sistema si varios usuarios registran pulsaciones anómalas al mismo tiempo. Las desventajas de Azure Service Bus sí son determinantes en el diseño de la solución. Un retraso en el procesamiento de un mensaje puede ser fatal en una situación crítica como es la detección de pulsaciones anómalas.

Teniendo en cuanta todo lo expuesto con anterioridad, se ha decidido diseñar la presente solución utilizando Azure Functions. Esta Azure Function va a ejecutarse cuando Azure Stream Analytics identifique que la frecuencia cardiaca media de un deportista determinado en la ventana de tiempo especificada supera el valor umbral establecido. Cuando esta Azure Function se ejecuta, desencadena una Logic App, enviándole como parámetros la ubicación del corredor, su device ID y la frecuencia cardiaca media que desencadenó la llamada a la función.

(8) Azure Logic App

De cara al diseño de la funcionalidad de envío de mensajes de alerta solicitando ayuda cuando se registran valores de pulsaciones anómalas, se han considerado principalmente los dos servicios que se muestran en la tabla 5: Azure Logic Apps y Send Grid.

Las Azure Logic Apps son un servicio de Microsoft Azure que nos permite crear y ejecutar flujos de trabajo automatizados con poco o sin código. Al tratarse de un servicio totalmente administrado por Azure podemos centrarnos en la funcionalidad y lógica de nuestras soluciones, sin tener que preocuparnos por las tareas de alojamiento, escalado, mantenimiento, administración o monitoreo (67).

Este servicio de Microsoft ofrece varios conectores preconstruidos que facilitan la integración y la comunicación entre servicios y sistemas externos como Office365, Salesforce, Dropbox o Twitter (68).

	Azure Logic Apps	SendGrid
Uso	• Servicio de integración en la nube que permite la creación de flujos de trabajo automatizados	• Servicio de correo electrónico en la nube que proporciona una API para enviar correos electrónicos
Ventajas	• Integración nativa con Azure • Diseñador visual que permite crear flujos en Azure con poco o sin código • Conectores predefinidos para una gran variedad de servicios y aplicaciones	• Especializado en correo electrónico • Escalable • Proporciona APIs y bibliotecas en varios lenguajes de programación
Desventajas	• Aunque Azure Logic App incluye conectores para enviar correos electrónicos no es un servicio dedicado al correo electrónico	• Es un servicio de pago por lo que ese coste se suma al uso de la solución en Azure • La integración de SendGrid con la solución de Azure requiere configuración adicional

Tabla 5: Comparativa entre Azure Logic Apps y Send Grid

Por otro lado, SendGrid es un servicio de correo electrónico en la nube que proporciona una API para el envío de correos electrónicos. Además, este servicio ofrece plantillas personalizables y herramientas de análisis de correo electrónico (69).

Como se muestra en la tabla 4, entre las ventajas de Logic Apps se destaca la integración nativa con Azure, el diseñador visual que permite la creación de flujos de trabajo de forma sencilla y la gran variedad de conectores que ofrece. Entre las desventajas se señala que no es un servicio especializado en correo electrónico, a contrario de lo que sucede con SendGrid. Sin embargo, este servicio presenta algunos inconvenientes como el hecho de incrementar el coste de la solución o la complejidad de configuración.

Haciendo balance de las ventajas y desventajas de cada uno de estos servicios se ha decidido priorizar el coste y la sencillez de configuración. Por lo tanto, para el diseño de la

funcionalidad de envío de correos electrónicos se ha elegido el uso de Azure Logic Apps. Esta Azure Logic App ha sido implementada para automatizar el envío de correos electrónicos a través de Outlook. Cuando el Azure Stream Analytics job detecta que se han superado las pulsaciones medias umbrales en la ventana de tiempo específica, va a invocar a la *function* que a su vez va a desencadenar la Logic App. Está Logic App va a enviar un correo electrónico solicitando ayuda, indicando el dispositivo que ha generado la alerta, su frecuencia cardiaca media en el momento de generación de la alerta y la ubicación, con el objetivo de poder actuar con la mayor celeridad posible.

(9) Microsoft Outlook

Microsoft Outlook es una aplicación de gestión de correo electrónico, contactos, calendario y tareas que ha sido desarrollada por Microsoft (70). Microsoft Outlook es la aplicación utilizada para el envío de los correos electrónicos de alerta.

3.2. Implementación

En esta segunda sección explicaremos los pasos seguidos para implementar todos y cada uno de los componentes de la solución planteada en la figura 28.

3.2.1. *Smartwatch*

En el subapartado anterior, “Diseño de la solución”, se ha destacado que los dispositivos BLE no tienen la capacidad de conectarse directamente a Azure, lo cual implica la necesidad de utilizar un *gateway*. Para abordar este desafío, se ha desarrollado una aplicación en Monkey C que permite el envío de mensajes desde el reloj al Azure Event Hub, empleando un teléfono móvil como *gateway*. Como se ha mencionado previamente, Monkey C es un lenguaje de programación orientado a objetos utilizado para el desarrollo de aplicaciones en dispositivos Garmin. En la sección siguiente (2) Gateway se describe detalladamente el funcionamiento de esta aplicación.

Para el desarrollo de la solución se ha hecho uso de Visual Studio Code, también conocido como VS Code. Visual Studio Code es un editor de código fuente que brinda a los desarrolladores la capacidad de escribir, depurar y administrar su código de manera eficiente. Esta herramienta permite la instalación de múltiples extensiones, lo que amplía las funcionalidades del editor de código. Estas extensiones pueden proporcionar características adicionales, como soporte para lenguajes de programación específicos, integración con servicios en la nube o autocompletado de código, entre otras opciones (71). Para la realización del presente TFM se ha instalado la extensión Monkey C, como se muestra en la figura 29.

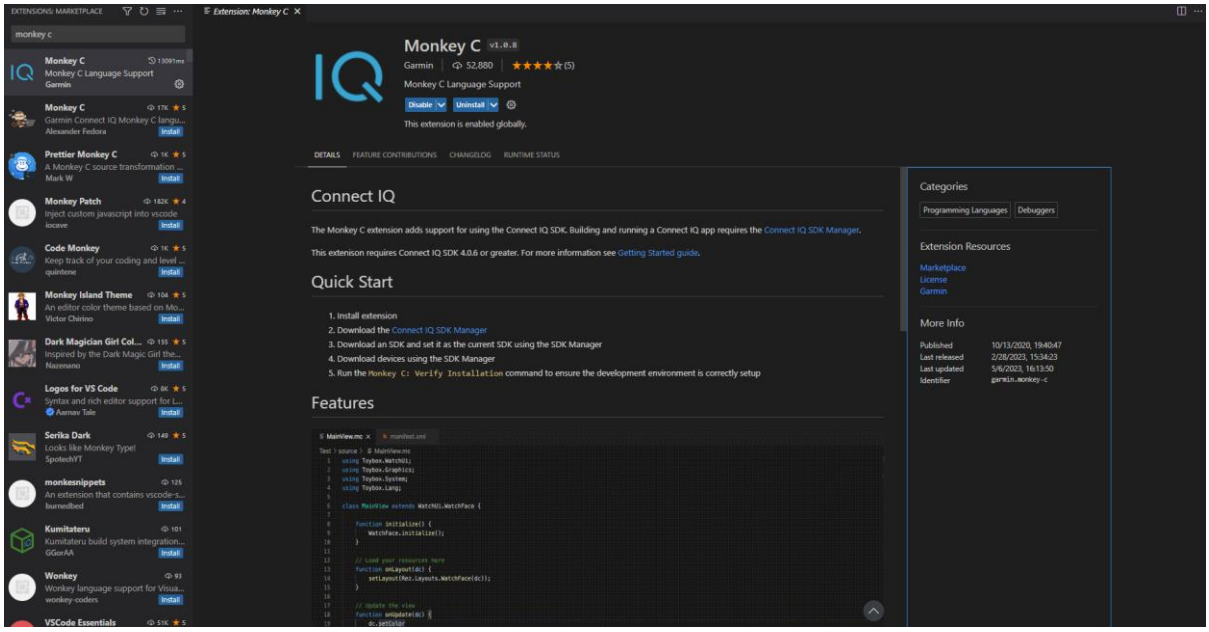

Figura 29: Instalación de la extensión de Monkey C en Visual Studio Code

La extensión Monkey C, proporciona un *wizard* o asistente que permite a los desarrolladores la instalación de aplicaciones lateralmente, es decir, sin utilizar la tienda oficial de Garmin IQ. Este *wizard* permite la creación de un fichero ejecutable PRG a partir de un proyecto dado. Para ello, dentro de Visual Studio Code, es necesario utilizar la combinación de teclas *Ctrl + Shift + P* para abrir la paleta de comandos. Una vez abierta la paleta de comandos, se debe ingresar "*Build for Device*" y seleccionar la opción "*Monkey C: Build for Device*", como se muestra en la figura 30.

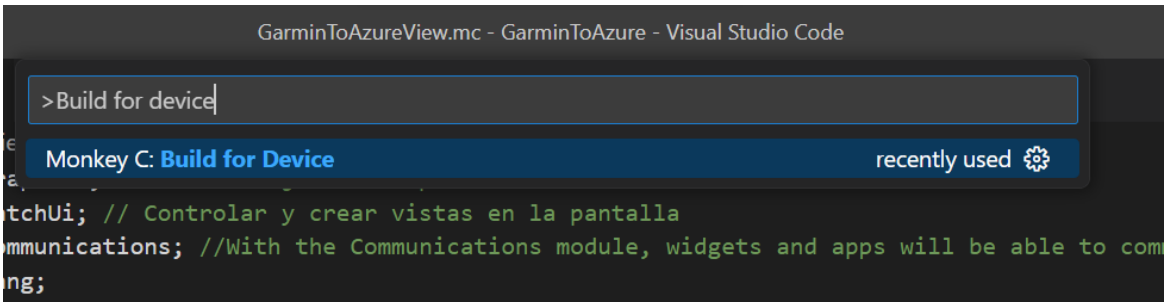

Figura 30: Creación del fichero ejecutable PRG a partir de nuestro proyecto en Monkey C

Una vez se ha generado el archivo PRG, se debe conectar el reloj al ordenador con el fin de copiar el ejecutable generado dentro del directorio GARMIN/APPS del dispositivo tal y como se muestra en la figura 31.

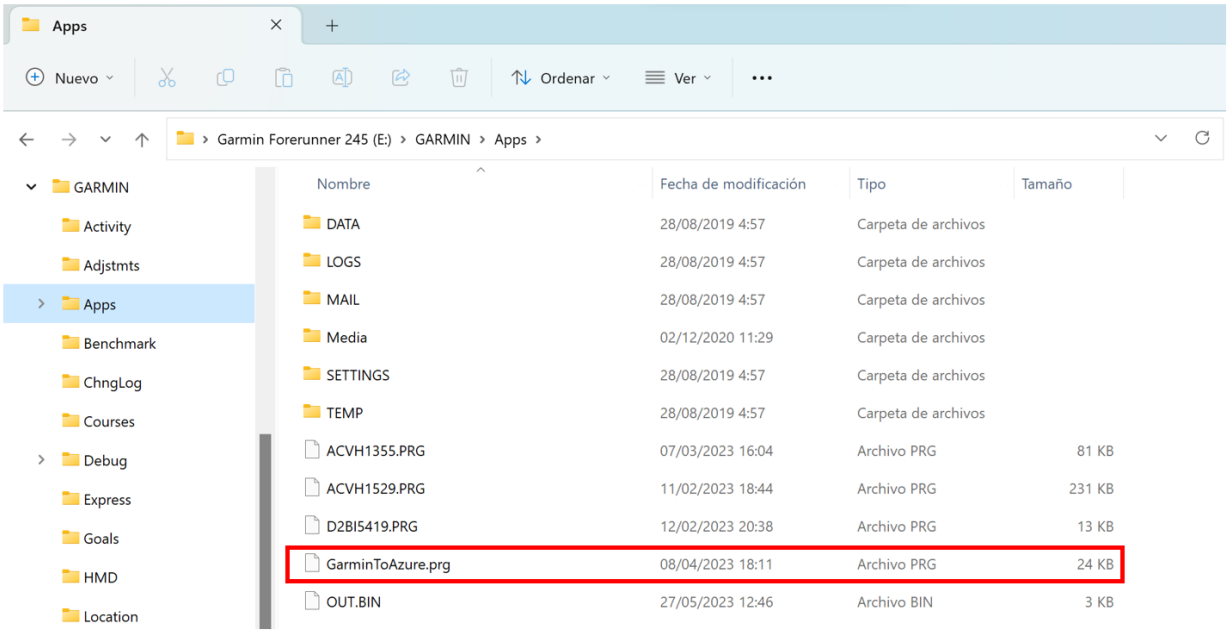

Figura 31: Carga del fichero PRG en el reloj Garmin

Una vez que el archivo PRG ha sido copiado satisfactoriamente en el directorio correspondiente, es momento de desconectar el dispositivo del ordenador. Al ir al menú aplicaciones del dispositivo, se puede verificar, tal como se muestra en la figura 32, que la aplicación GarminToAzure se ha cargado correctamente y está lista para ser utilizada.

Figura 32: Aplicación GarminToAzure cargada en el reloj

3.2.2. Gateway

Como se adelantaba previamente, para poder conectar un reloj BLE a Azure, se va a utilizar un teléfono móvil como *gateway*. La aplicación implementada para poder enviar los datos generados por el reloj al Azure Event Hub a través de este Gateway puede encontrarse en el Anexo 1. Como se puede observar en el Anexo 1, el comportamiento de la aplicación viene modelado principalmente por cuatro ficheros:

- **GarminToAzureApp.mc.** Este fichero define una aplicación llamada "GarminToAzureApp" que utiliza la API de posicionamiento de Garmin para rastrear la ubicación del dispositivo y enviarla a un servicio de Azure. La aplicación define una clase "GarminToAzureApp" que extiende "Application.AppBase" de la biblioteca Toybox. La clase Application.AppBase proporciona una estructura básica y funcionalidad común para las aplicaciones desarrolladas en Monkey C. GarminToAzureApp contiene los siguientes métodos:

    - initialize() se encarga de llamar al método initialize() de la clase padre AppBase. Este método es llamado en el momento de inicialización de la aplicación.

    - on*Start*(state as Dictionary?) es llamada con el inicio de la aplicación y habilita el seguimiento continuo de la ubicación mediante la invocación de Position.enableLocationEvents(Position.LOCATION_CONTINUOUS, method(:onPosition)).

    - onStop(state as Dictionary?) es llamada cuando la aplicación se detiene y deshabilita el seguimiento de la ubicación mediante la invocación de Position.enableLocationEvents(Position.LOCATION_DISABLE, method(:onPosition)).

    - onPosition(info as Position.Info) se llama cada vez que cambia la ubicación del dispositivo.

    - getInitialView() se llama cuando se inicia la aplicación y define la vista de la aplicación. La función llama a la vista creada en el fichero GarminToAzureView.mc y a GarminToAzureDelegate para manejar eventos de entrada en la vista.

- **GarminToAzureDelegate.mc.** Este fichero contiene una clase llamada GarminToAzureDelegate que hereda de la clase WatchUi.BehaviorDelegate. La clase BehaviorDelegate es una clase base que se utiliza para definir el comportamiento de un menú en un reloj Garmin. La clase GarminToAzureDelegate tiene dos métodos:

- initialize(). Este método se llama al inicio de la aplicación y se utiliza para realizar cualquier inicialización necesaria.

- onMenu(). Este método se llama cuando se presiona el botón de menú en el reloj Garmin y muestra el menú principal de la aplicación utilizando la función "WatchUi.pushView". La función "WatchUi.pushView" toma tres argumentos: la vista que se va a mostrar, el delegado que maneja los eventos de entrada en la vista y la dirección en la que se desliza la vista.

- **GarminToAzureMenuDelegate.mc.** Define una clase llamada GarminToAzureMenuDelegate, que hereda de la clase WatchUi.MenuInputDelegate de la biblioteca Toybox. La clase WatchUi.MenuInputDelegate define el comportamiento de la aplicación en respuesta a eventos de entrada en el menú principal de la aplicación.

  - initialize() se llama cuando la el menú es creado.

  - onMenuItem(item as Symbol) se llama cuando se selecciona un elemento del menú.

- **GarminToAzureView.mc.** Este fichero implementa la lógica necesaria para enviar datos desde los relojes al Azure Event Hub. También implementa el código necesario para recoger los datos de los sensores o generar los valores que se consideren oportunos para probar el funcionamiento de la aplicación. Para ello, se comienza definiendo la clase GarminToAzureView que hereda la clase WatchUI.View. La clase WatchUi.View define métodos y propiedades que son necesarios para trabajar con la pantalla y los botones del dispositivo, así como para gestionar la interacción del usuario. Esta clase se compone de los siguientes métodos:

  - initialize(). La función "initialize" establece una conexión segura con Azure Event Hub, utilizando una clave de autenticación y un identificador único del dispositivo Garmin. Para ello, la función obtiene la configuración y el identificador único del dispositivo. Luego, define la URL y la clave SAS para la conexión al Azure Event Hub. A continuación, utiliza la clave SAS para cifrar la URL y generar una firma de acceso compartido (SAS) utilizando el algoritmo HMAC (Hash-Based Message Authentication Code). La clave SAS se utiliza para cifrar la URL y generar una firma de acceso compartido (SAS) utilizando el algoritmo HMAC (Hash-Based Message Authentication Code). Por último, configura un temporizador que llamará a la función "timerCallback" periódicamente para enviar los datos del dispositivo Garmin al servicio en la nube.

- timerCallBack(). La función timerCallback() se utiliza para recopilar y enviar información de los sensores de un dispositivo a una API REST mediante una solicitud POST. Recopila información de los sensores del dispositivo, como el acelerómetro, el ritmo cardíaco, la altitud, la cadencia, el rumbo, el magnetómetro, la energía, la presión, la velocidad y la temperatura. También da la opción al usuario de generar los valores deseados en lugar de utilizar los recopilados por los sensores. De esta forma es posible realizar las simulaciones que se consideren pertinentes para estudiar el comportamiento de la plataforma de monitorización de corredores en tiempo real. A continuación, construye una solicitud POST con esta información y la envía a la API REST.

- onReceive(). El propósito de esta función es actualizar el estado de la aplicación en función de la respuesta de las solicitudes HTTP POST enviada al servicio API REST. La función utiliza una estructura de control *switch* para asignar un valor a la variable status en función del código de respuesta HTTP devuelto por el servidor REST API.

- onLayout(). Carga los recursos necesarios y establece la interfaz de usuario principal de la aplicación.

- onUpdate(). Se encarga de actualizar la interfaz de usuario (UI) en el reloj.

- setPosition(). Solicita una actualización de la interfaz de usuario mediante WatchUi.requestUpdate().

3.2.3. Azure Storage account

Para desplegar cualquier recurso en Azure, es imprescindible crear previamente un grupo de recursos o *resource group*. Un grupo de recursos es un contenedor en el que se almacenan todos los recursos que comparten el mismo ciclo de vida. De esta forma, se facilitan las tareas de implementación, actualización y administración de dichos recursos (72).

Existen varias opciones a la hora de crear un grupo de recursos: desde el portal de Azure (72), desde la cli (Command Line Interface) de Azure con el comando az group *create* (73), mediante Azure PowerShell con New-AZResourceGroup (74) o con python (75).

Para crear el grupo de recursos desde el portal, el usuario debe dirigirse a *resource groups* y seleccionar *create*. A continuación, se abrirá la ventana que se muestra en la figura 33, donde se debe indicar la suscripción, el nombre y la región sobre la que se desea desplegar el grupo de recursos.

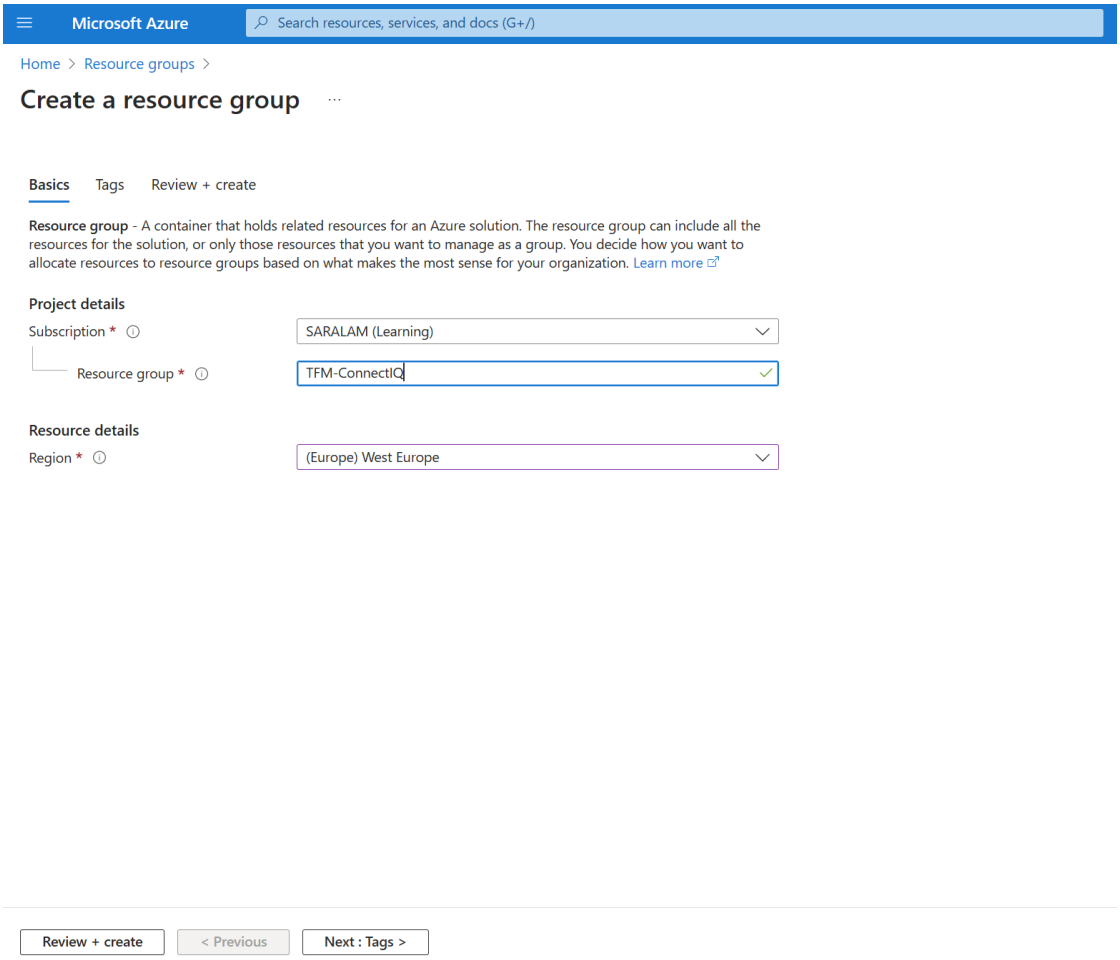

Figura 33: Creación de un nuevo grupo de recursos desde el portal de Azure

Tras completar los campos mencionados anteriormente se debe hace clic en el botón *Review + create.* Yendo nuevamente a *Resource Groups*, puede comprobarse como el nuevo grupo de recursos ha sido creado correctamente.

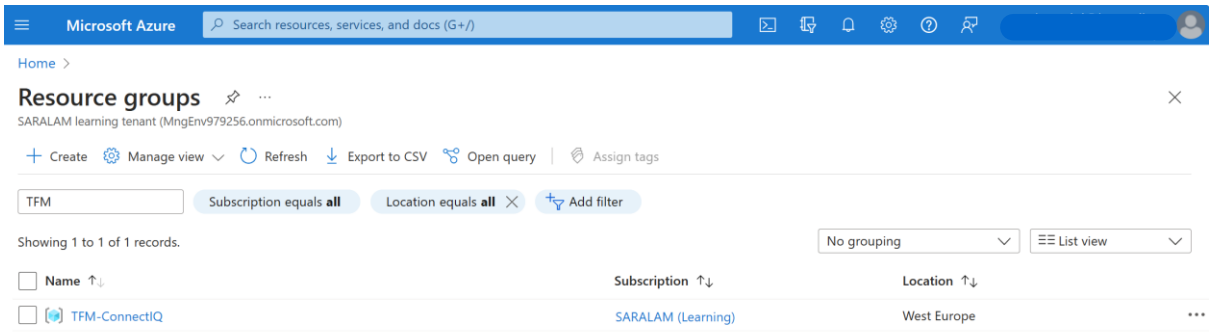

Figura 34: Resource group creado a través del portal de Azure

Una vez confirmada la correcta creación del grupo de recursos, es momento de crear la *storage account*. Existen varias opciones para la creación de una *storage account*: desde el portal de Azure, con Powershell haciendo uso del comando *New-AzStorageAccount*, desde la CLI con el comando *az storage account create*, con Bicep o con una plantilla ARM (52). Si se decide optar por la creación de la *storage account* desde el portal de Azure, se debe en primer lugar buscar el servicio *storage accounts* como se muestra en la figura 35.

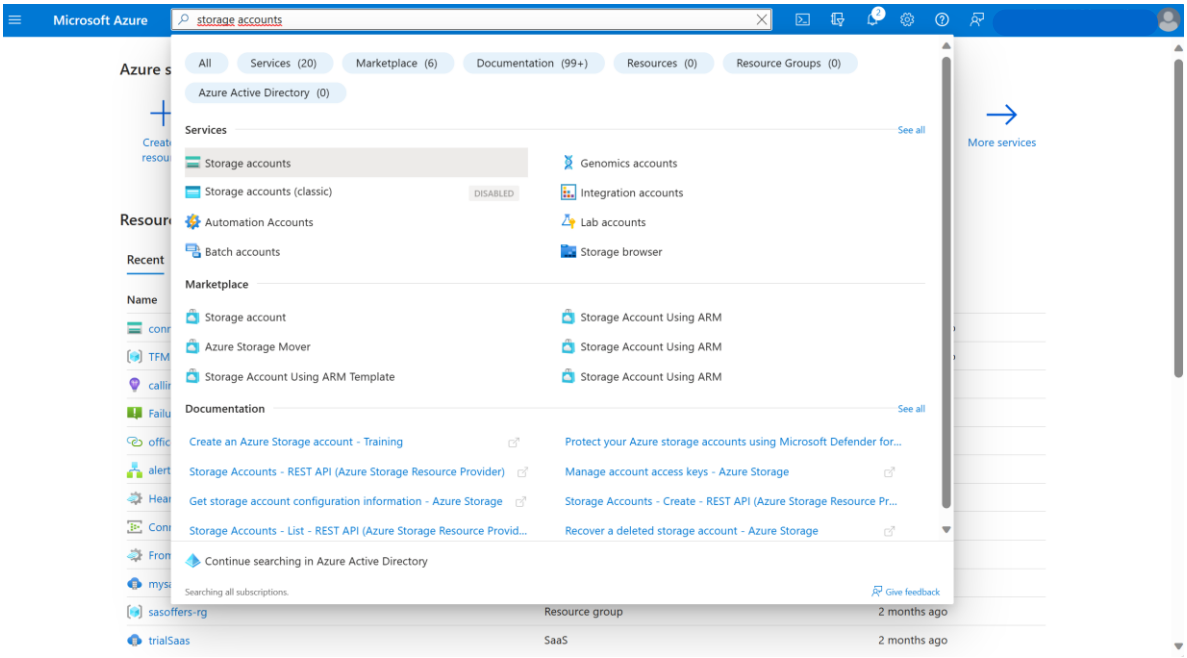

Figura 35: Creación de una Storage account desde el portal de Azure 1.

Tras seleccionar *storage accounts*, se abrirá la vista presentada en la figura 36. A continuación, debe hacerse clic en + *Create* para comenzar con la creación de la nueva *storage account.*

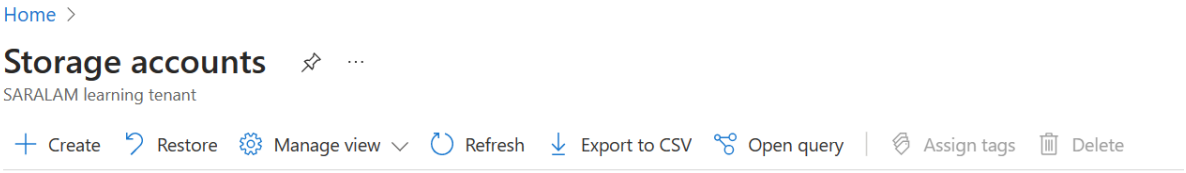

Figura 36: Creación de una Storage account desde el portal de Azure 2

Una vez se ha seleccionado + *Create* el usuario debe especificar las propiedades básicas de la nueva cuenta de almacenamiento. En concreto, los campos a rellenar obligatoriamente son los que se muestran en la figura 37:

- Suscripción: ha de indicarse la suscripción sobre la que se desea desplegar la *storage account*.

- Grupo de recursos: ha de especificarse el grupo de recursos sobre el que desplegar la *storage account*.

- Nombre de la storage account: debe indicarse el nombre que se desea proporcionar a la *storage account*.

- Rendimiento: debe elegirse entre *Standard* o *Premium*. Para este caso concreto se ha decidido optar por la opción recomendada por Microsoft para la mayor parte de escenarios, la opción *Standard.*

- Redundancia: debe escogerse entre los siguientes tipos de redundancia: *locally-redundant, geo-redundant, zone-redundant* o *geo-zone-redundtant*. Dado que estamos trabajando en un prototipo, se ha decidido optar por la opción *locally-redundant* por se la más económica.

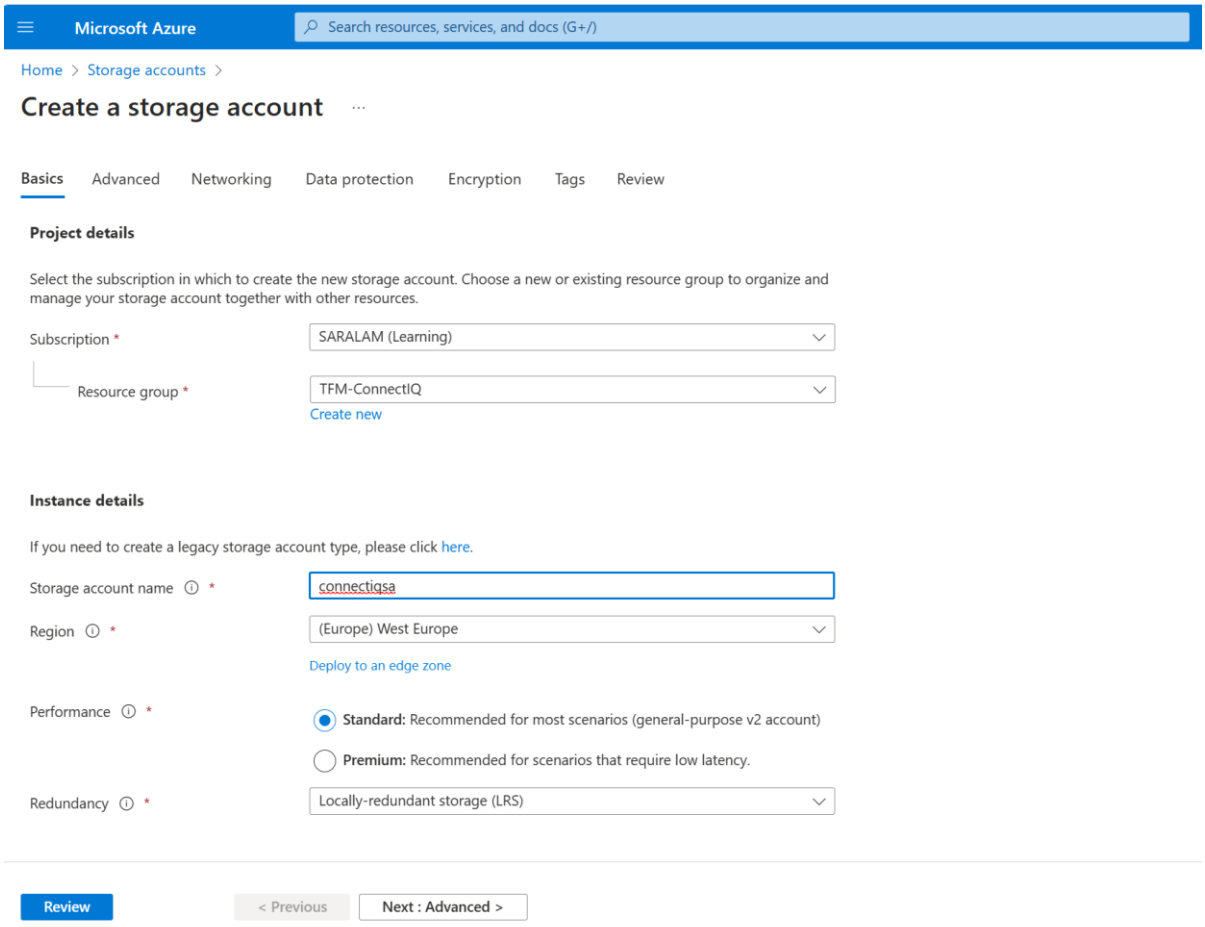

Figura 37:Creación de una Storage account desde el portal de Azure 3

En la figura 38 se muestra como la *storage account* ha sido creada satisfactoriamente.

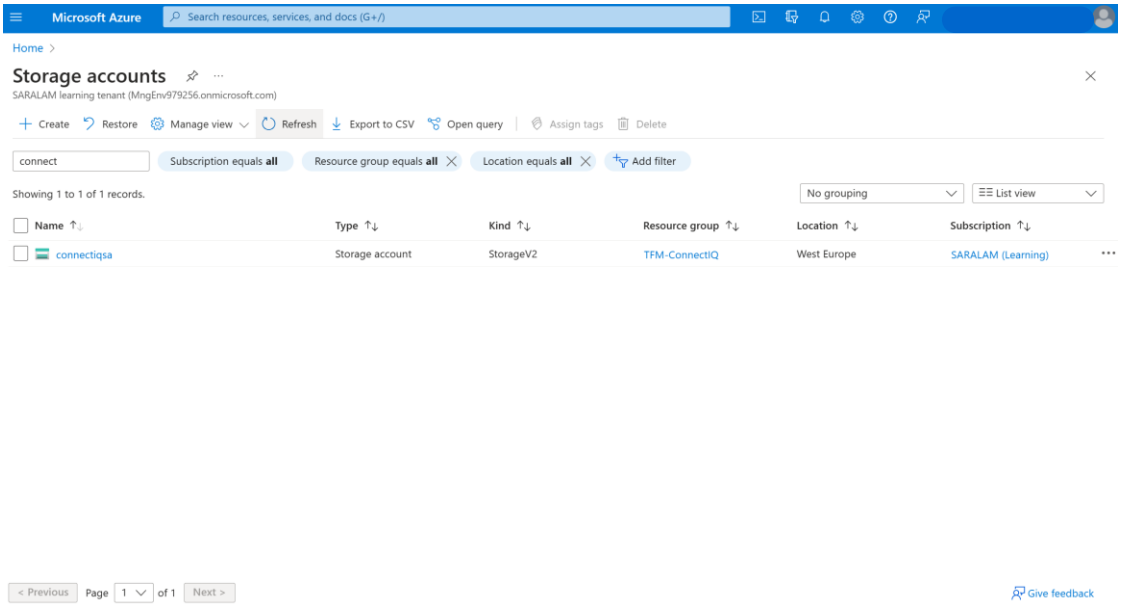

Figura 38: Creación de la nueva storage account

Tras la correcta creación de la *storage account*, el siguiente paso consiste en crear un contenedor donde almacenar los datos. Para ello, en el menú de la izquierda presentado en la figura 38 debe seleccionarse *Containers*. Una vez dentro de la pestaña *Containers*, debe hacerse click en + *Container*. Tal y como se muestra en la figura 39, esto abrirá una nueva ventana para la creación del nuevo contenedor en la que se debe indicar el nombre deseado y seleccionar *Create.*

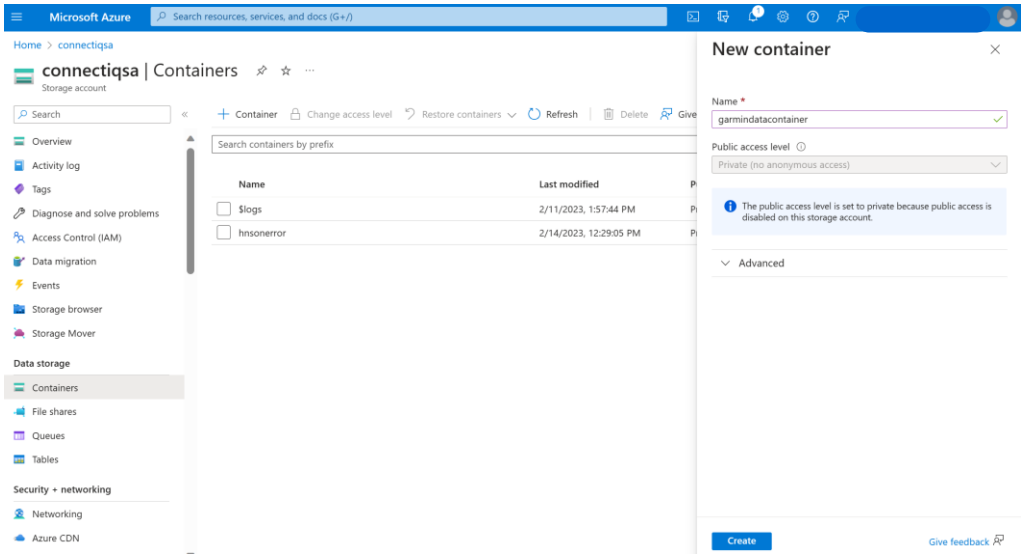

Figura 39: Creación del contenedor garmindatacontainer dentro de la Storage account desde el portal de Azure

3.2.4. Azure Event Hub

Existen varias alternativas para la creación de un nuevo Azure Event Hub: desde el portal de Azure (76), Azure CLI (77), Azure Powershell (78), Bicep (79) o Plantillas ARM (80). Independientemente del método por el que se decida optar, el primer paso consiste en crear un espacio de nombres o *namespace*. Un espacio de nombres de Event Hub es un contenedor lógico para un conjunto de recursos de Event Hub relacionados como *hubs* de eventos, grupos de consumidores y políticas de seguridad (81).

Para la creación del *namespace* desde el portal de Azure, el procedimiento es similar al seguido para la creación de un *resource group* o de una *storage account*. En primer lugar, se debe ir a Event Hubs y seleccionar *Create.* A continuación, como se muestra en la figura 40, se abrirá la página para la creación del espacio de en la que se deben rellenar los campos siguientes:

- Suscripción sobre la que se desea crear el espacio de nombres
- Grupo de recursos al que debe pertenecer el espacio de nombres
- Nombre del espacio de nombres
- Ubicación
- El plan de tarifa
- Las unidades de rendimiento

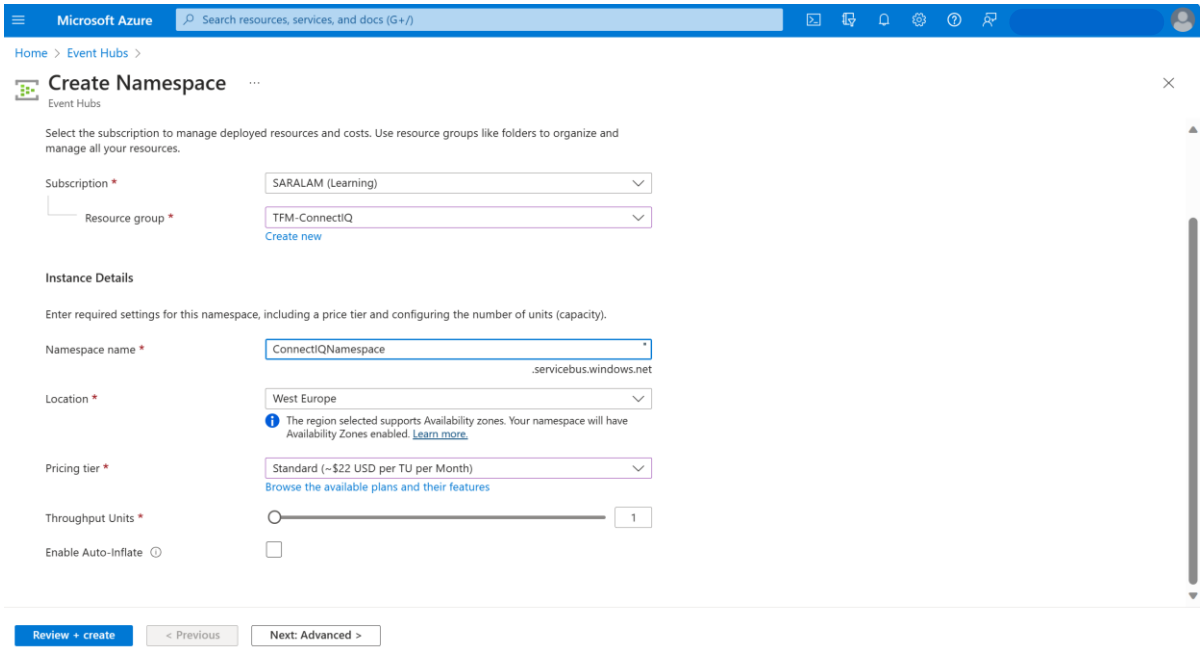

Figura 40: Creación de un Namespace desde el portal de Azure

Tras seleccionar *Review + Create* se muestra en la figura 41 como el espacio de nombres se crea satisfactoriamente.

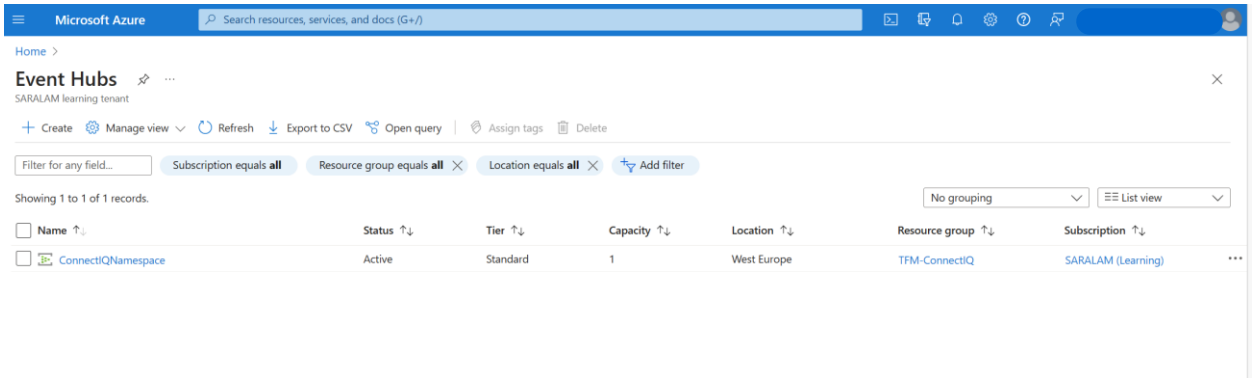

Figura 41: Namespace ConnectIQNamespace creado en el portal de Azure

En la figura 42 se muestran las propiedades principales del *namespace ConnectIQNamespace* recién creado.

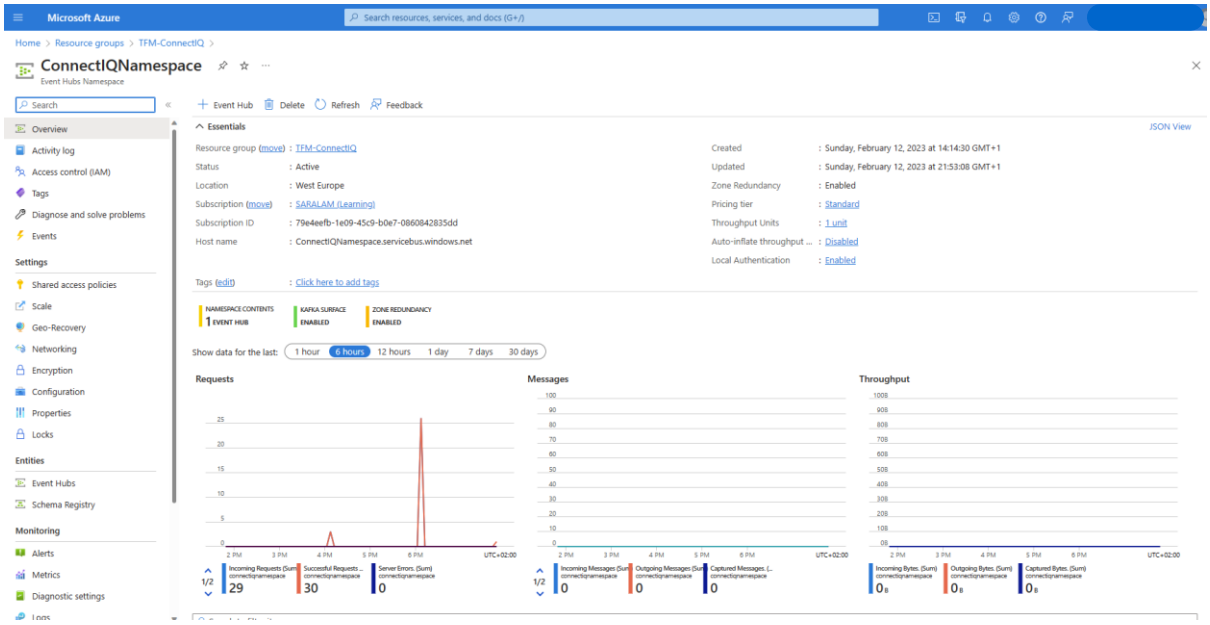

Figura 42: Propiedades del Namespace ConnectIQNamespace

Dado que se desconoce la red a través de la cual se van a conectar los dispositivos (datos del teléfono, Wifi del hogar, del trabajo, …), se debe modificar la configuración de red del nuevo espacio de nombre. Para ello, debe seleccionarse la pestaña *Networking* en el menú izquierdo de la figura 42. Como se plasma en la figura 43, se debe modificar la configuración de red y seleccionando *All networks*.

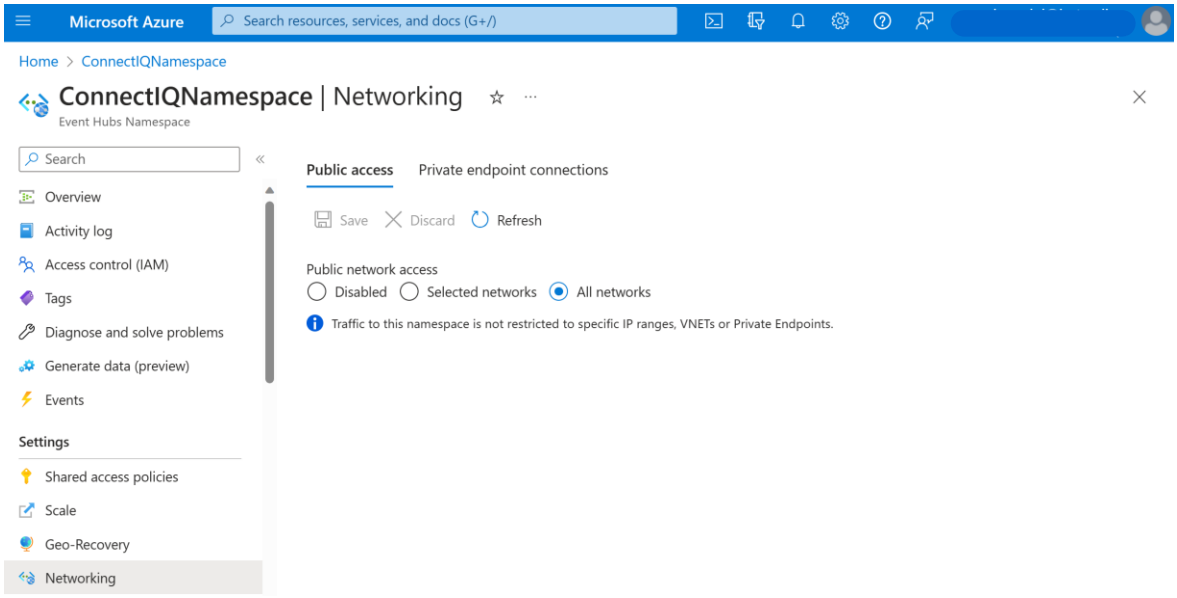

Figura 43: Configuración de Networking para nuestro Namespace ConnectIQNamespace

Después de haber creado el espacio de nombres, el siguiente paso consiste en crear el Event Hub. Para ello, es necesario ir a la pestaña Event Hub en el menú izquierdo de la figura 42 y seleccionar + *Event Hub*. A continuación, tal y como se recoge en la figura 44, se abrirá la ventana destinada a la creación del Event Hub.

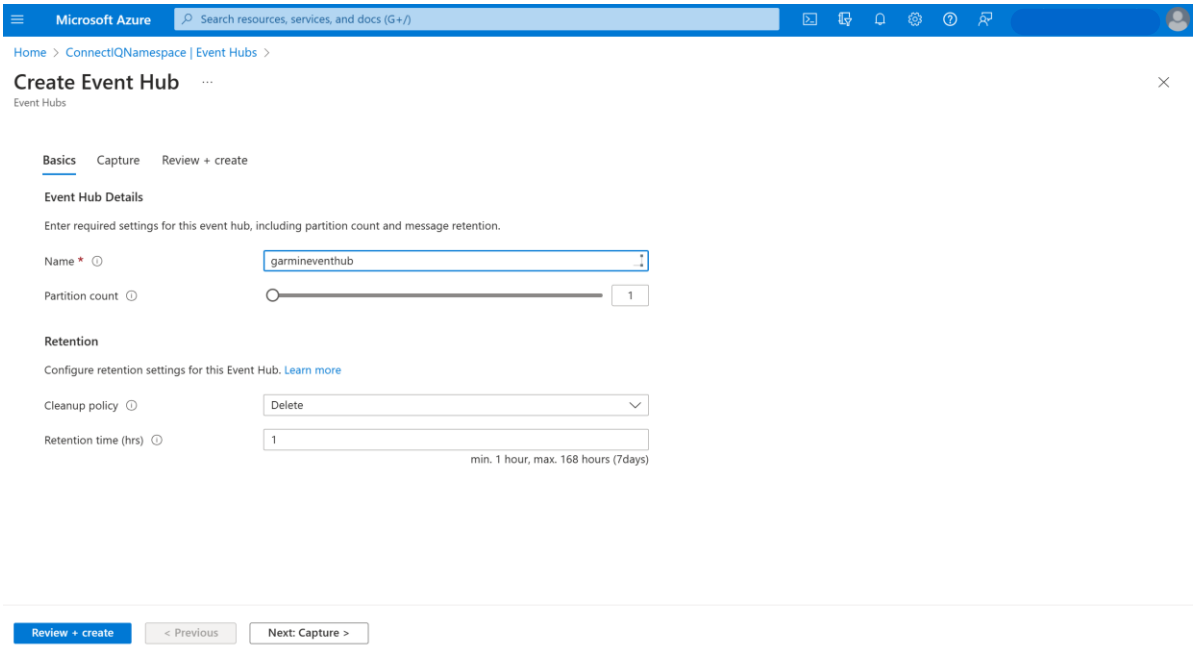

Figura 44: Creación de un Event Hub desde el portal de Azure

En la figura 45 se muestra como el Event Hub garmineventhub ha sido creado satisfactoriamente.

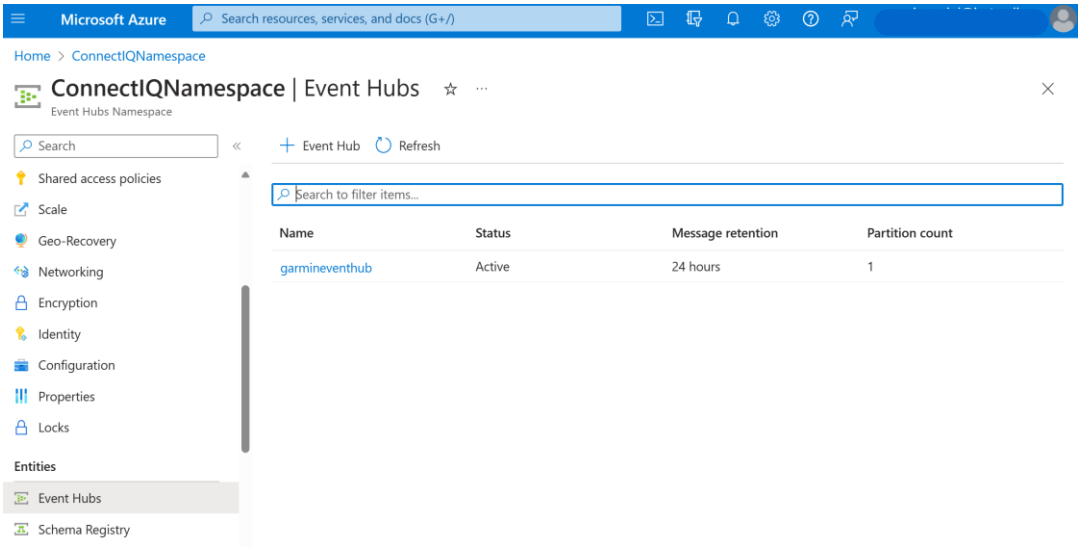

Figura 45: Vista del nuevo Event Hub en el portal de Azure

Una vez que el Azure Event Hub ha sido creado exitosamente, es el momento de establecer una política de acceso compartido o SAS (Shared Access Signature). El SAS proporciona una capa de seguridad para el acceso a los recursos de un espacio de nombres o Event Hub, en base a las reglas de autorización configuradas (82). En el contexto de la solución planteada en este TFM, este SAS será utilizado para permitir que los relojes inteligentes puedan enviar mensajes al Azure Event Hub. Para crear este SAS, denominado garmin, dentro del Event Hub, es necesario acceder a la pestaña "Shared Access policies" y hacer clic en "+ Add", tal como se muestra en la figura 46.

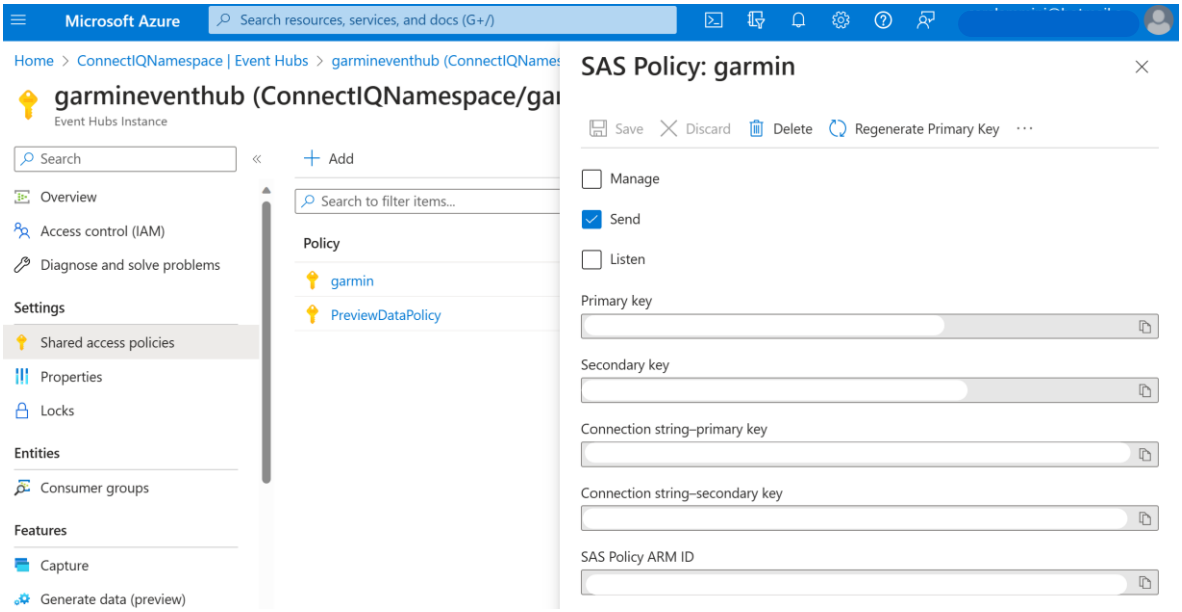

Figura 46: Creación de la Shared Access Poicy (SAS) para nuestro event Hub

A partir de la *primary* o *secondary key* es posible utilizar el Event Hub Signature Generator para crear el *token*. Este *token* permite realizar la autenticación y la autorización contra el Event Hub desde la aplicación (83). Como se observa en la figura 47, los parámetros a especificar son:

- *Namespace*: nombre del espacio de nombres
- *Hub Name*: nombre del Event Hub
- *Publisher*: identificador único del dispositivo que se va a conectar al Event Hub.
- *Sender Key Name*: nombre de las SAS generada
- *Sender Key*: clave primaria o secundaria
- *Token TTL* en minutos: duración del token, en nuestro caso hemos especificado 1 año.

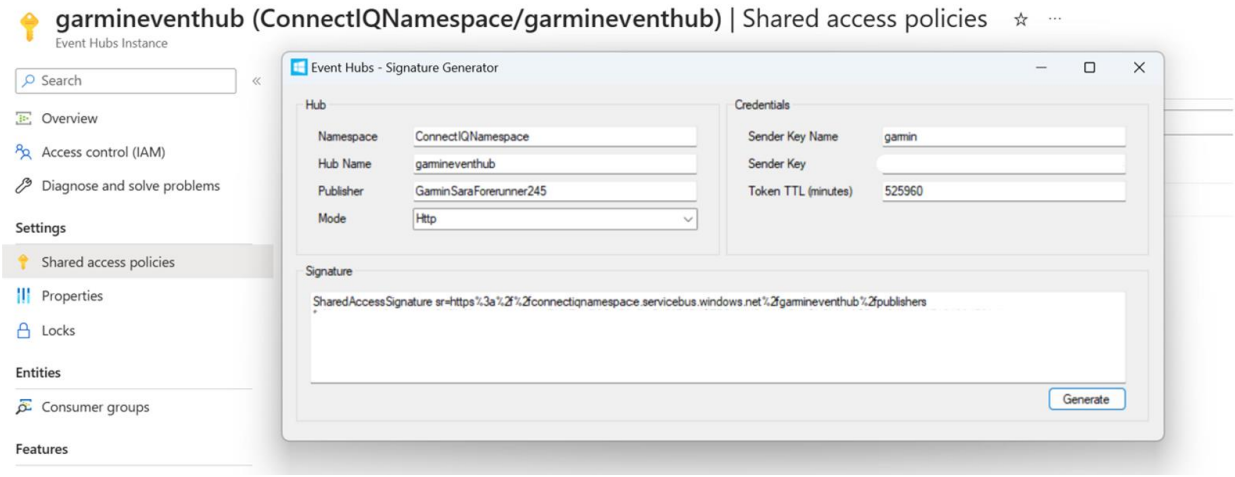

Figura 47: Creación del token utilizando Event Hub Signature Generator

Por último, se debe configurar el Event Hub de forma que comience a capturar los mensajes enviados por los relojes inteligentes conectados. Para ello, se debe seleccionar *Capture* en el menú de la izquierda de la figura 47 y cambiar el modo de captura a *On*. A continuación, tal como se refleja en la figura 48, se debe indicar la ventana de tiempo y el tamaño de captura deseados, así como el contenedor en el que se desea almacenar los mensajes recibidos.

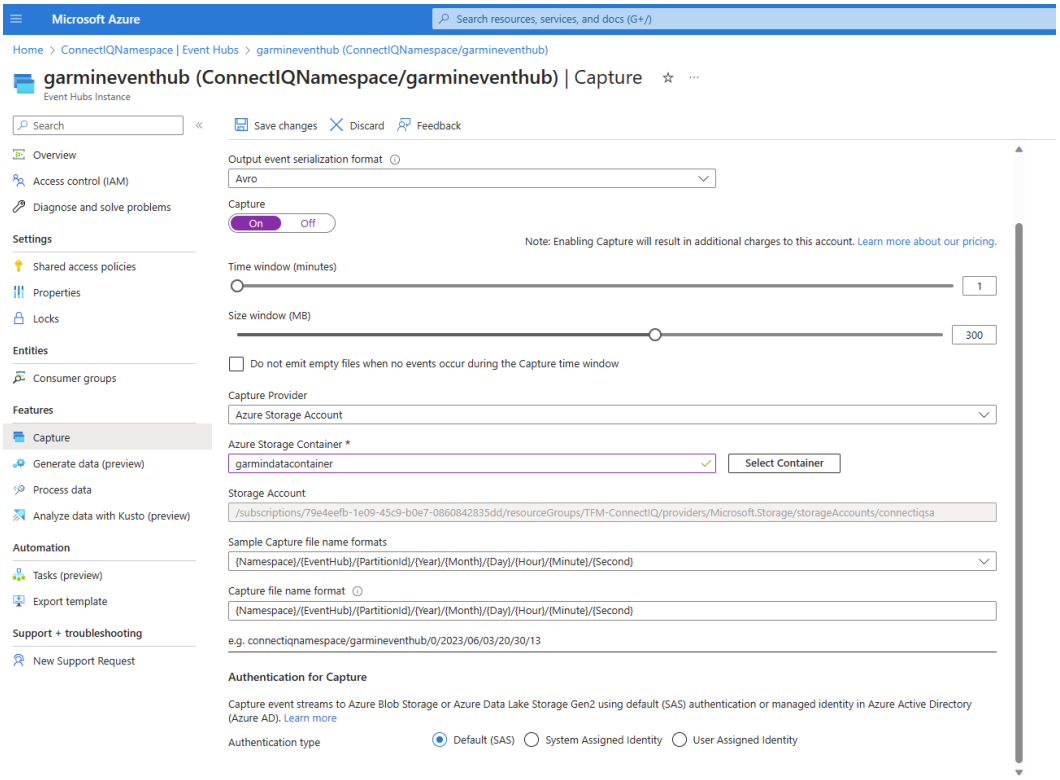

Figura 48: Activación de la captura en nuestro Event Hub

3.2.5. Azure Stream Analytics – FromEventHub2PowerBI

Como se reflejaba en la figura 28, para la implementación de la solución diseñada es preciso crear dos Azure Stream Analytics jobs. En esta sección se presentan los pasos seguidos para implementar el FromEventHub2PowerBI job, responsable de la representación en tiempo real de los datos recibidos en el Azure Event Hub sobre un panel de monitorización en PowerBI. Más adelante, concretamente en el apartado 3.2.9. Azure Stream Analytics Job – HeartRateAnomalyDetection se describirán los pasos seguidos para implementar el job, encargado de detectar anomalías en el ritmo cardíaco y generar alertas.

Para la creación del job FromEventHub2PowerBi parte del Azure Event Hub. Para ello, se va a selecciona la pestaña process data como se muestra en la figura 49. Puede comprobarse como entre las opciones disponibles se encuentra *Build new real-time data dashboard with Power Bi*.

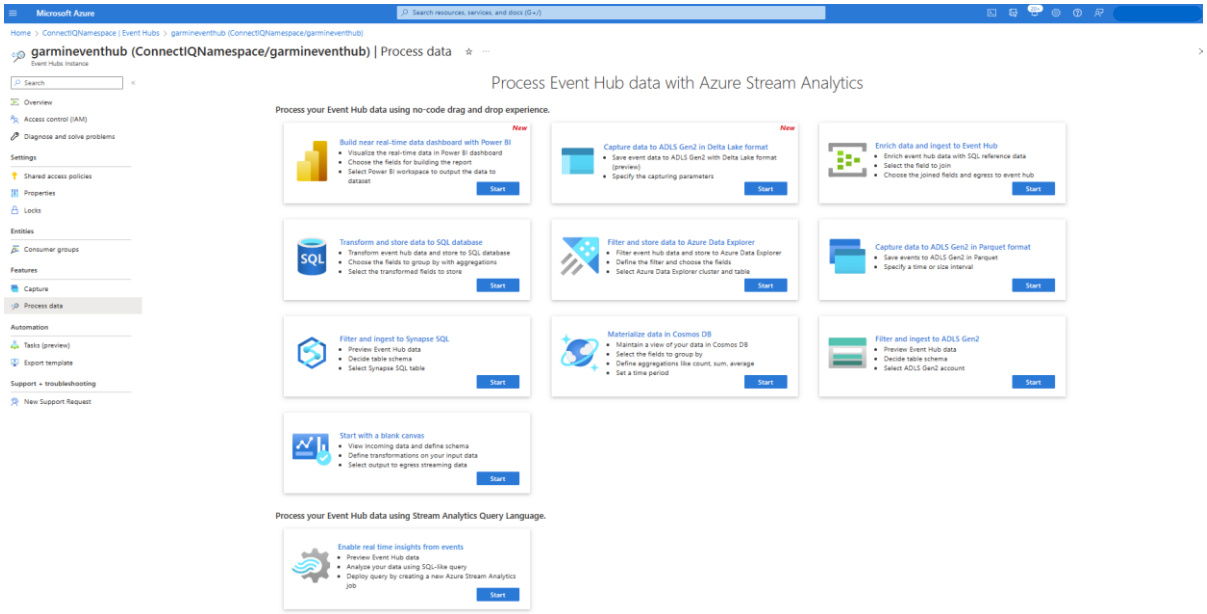

Figura 49: Process Data en Event Hub

Una vez se selecciona *start* en la opción *Build new real-time data dashboard with Power Bi*, como se refleja en la figura 50, aparece una ventana para la creación de un nuevo Stream Analytics job. En esta ventaba debe especificarse el nombre del job, en este caso FromEventHub2PowerBI.

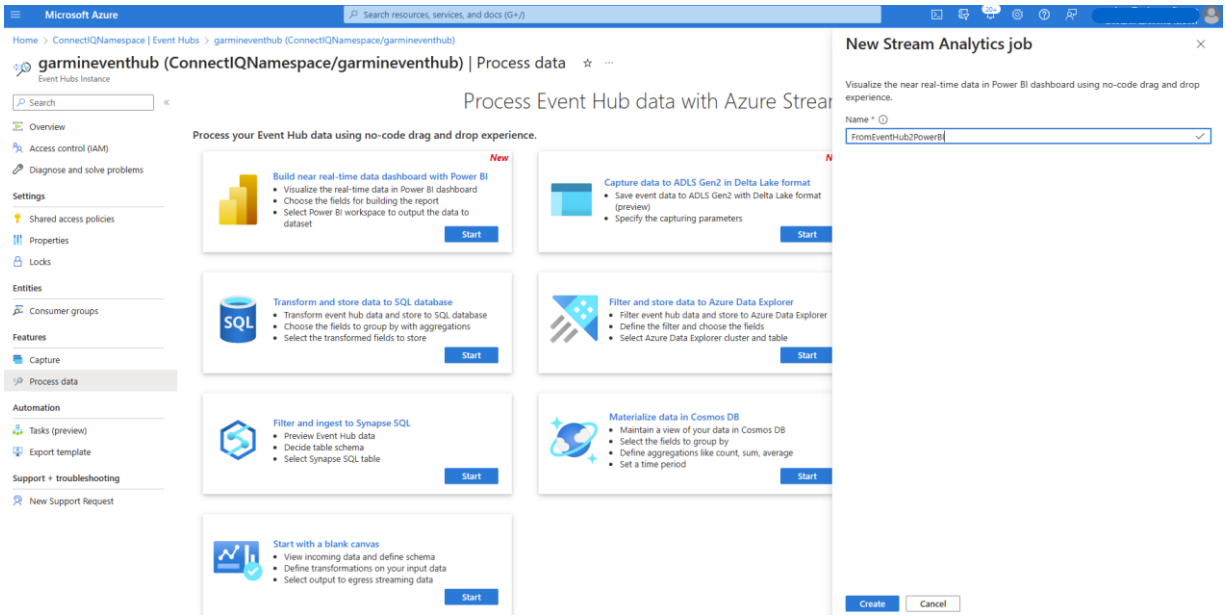

Figura 50: Creación del Azure Stream Analytics Job para la visualización de datos en tiempo real desde Power BI

El siguiente paso se muestra es la figura 51 y consiste en la creación de un nuevo *consumer group*. Un *consumer group* es un conjunto de aplicaciones que consumen eventos desde un Event Hub.

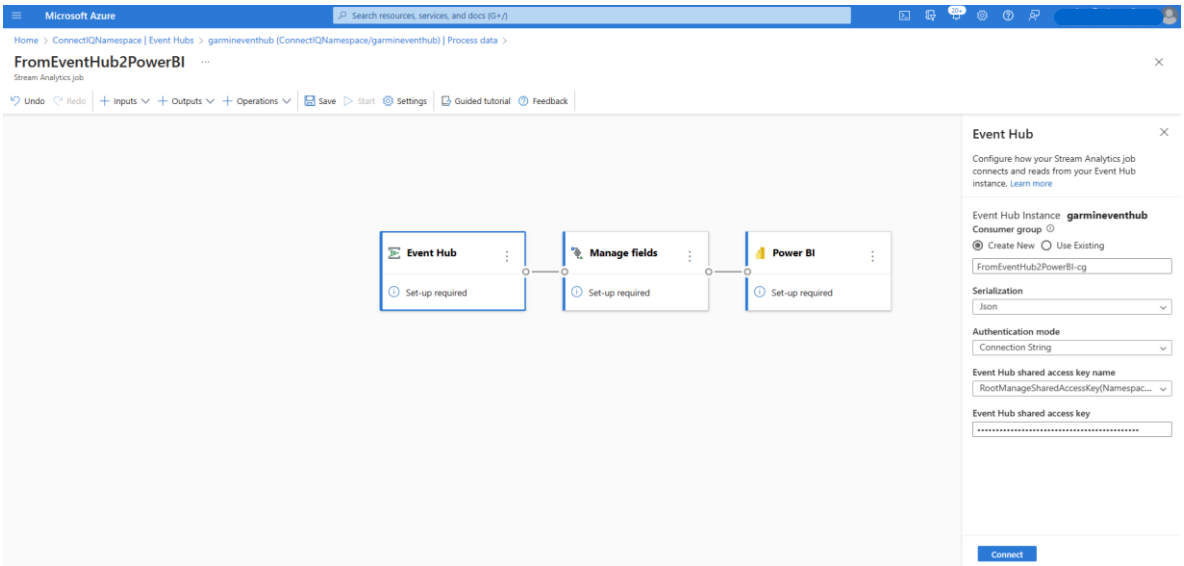

Figura 51: Creación de un nuevo consumer group

A continuación, debe hacerse click en el botón *connect* de la figura 51. Puede observarse en la figura 52 como se obtiene una confirmación de que el test de conexión ha sido satisfactorio. Por lo tanto, ya se está en disposición de seleccionar los campos a importar desde el Event Hub al Stream Analytics job, como se muestra en la figura 52.

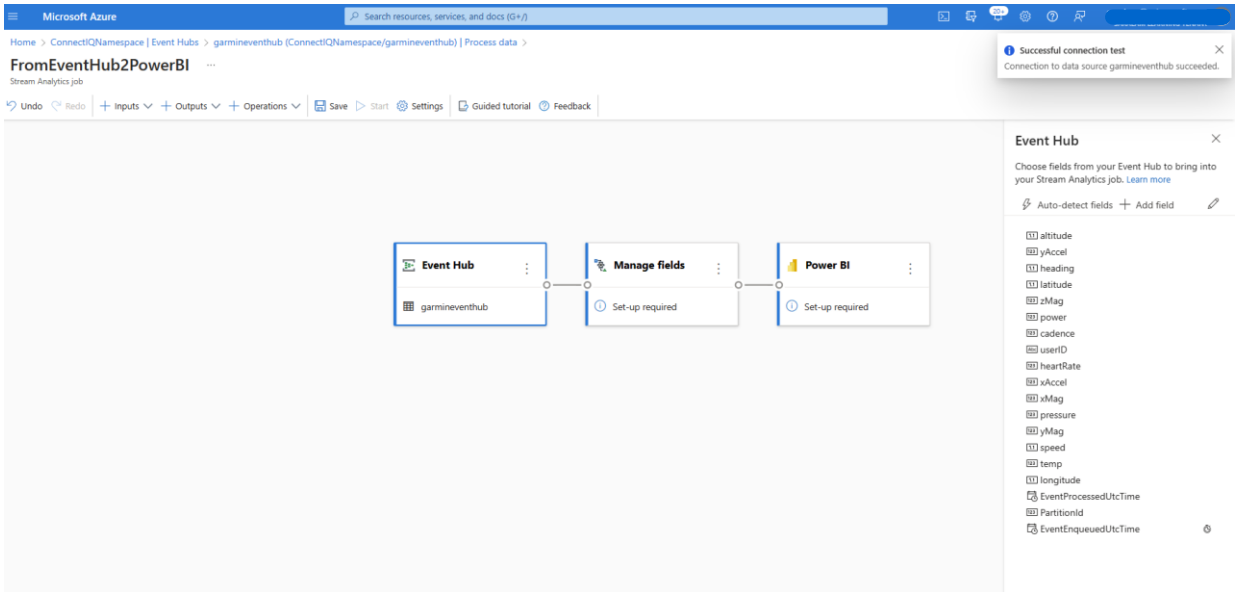

Figura 52: Selección de los campos a importar desde nuestro Event Hub

Después, es momento de añadir en el bloque *Manage fields* los campos deseados, tal y como se representa en la figura 53.

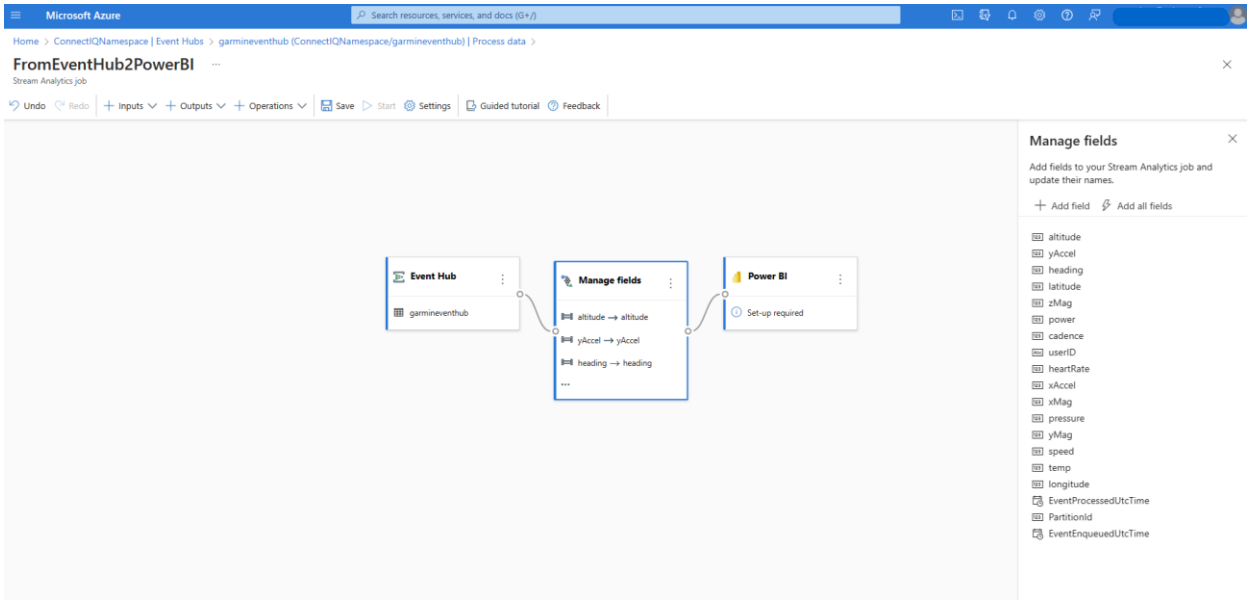

Figura 53: Manage Fields en Azure Stream Analytics

Para finalizar, se selecciona el último bloque, Power BI. Como se captura en la figura 54, debe indicarse el ID del Workspace, el modo de autenticación deseado y los nombres del *dataset* y la *tabla*.

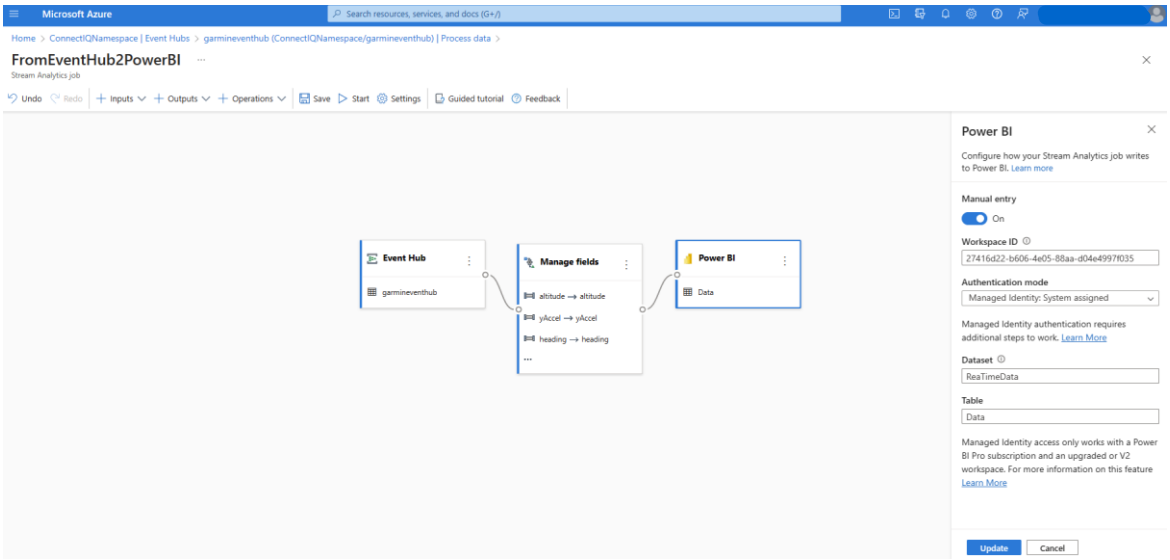

Figura 54: Configuración del Power BI de salida

Dado que la licencia de Power BI que utilizada no se encuentra en la misma organización o tenant, es necesario que cambiar el modo de autenticación de *Managed Identity* a *User Token*, tal y como se plasma en la figura 55.

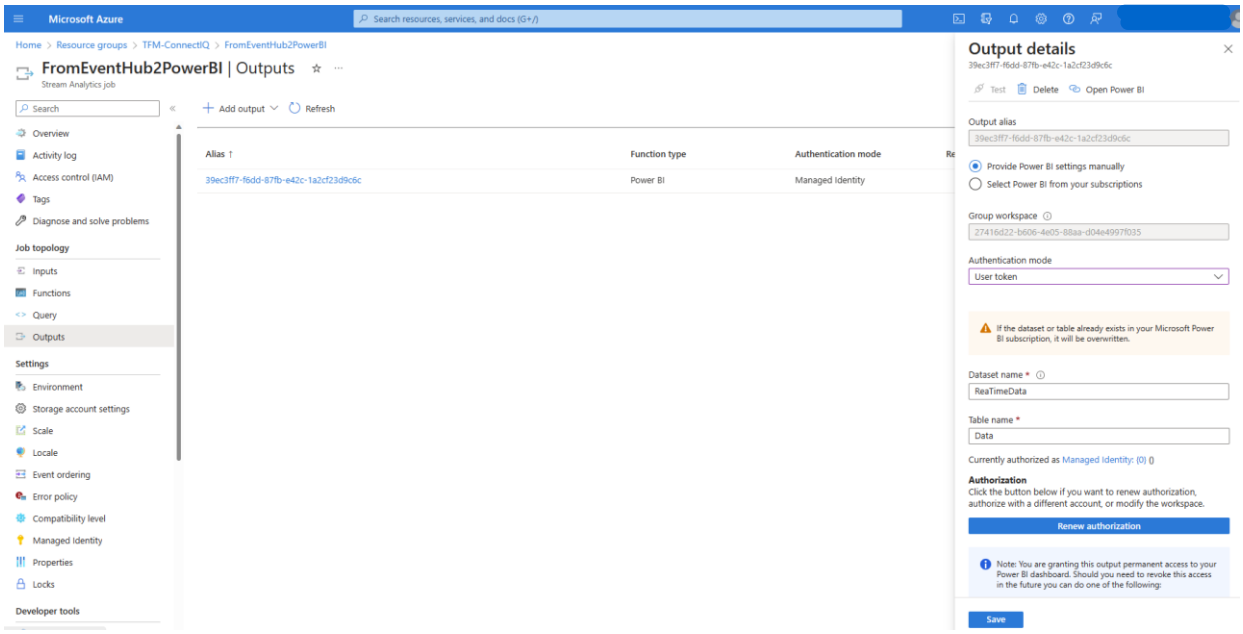

Figura 55: Configuración del modo de autenticación contra Power BI

Una vez realizada esta modificación, debe seleccionarse *Renew authorization* e iniciar sesión con el usuario pertinente, tal y como se refleja en la figura 56.

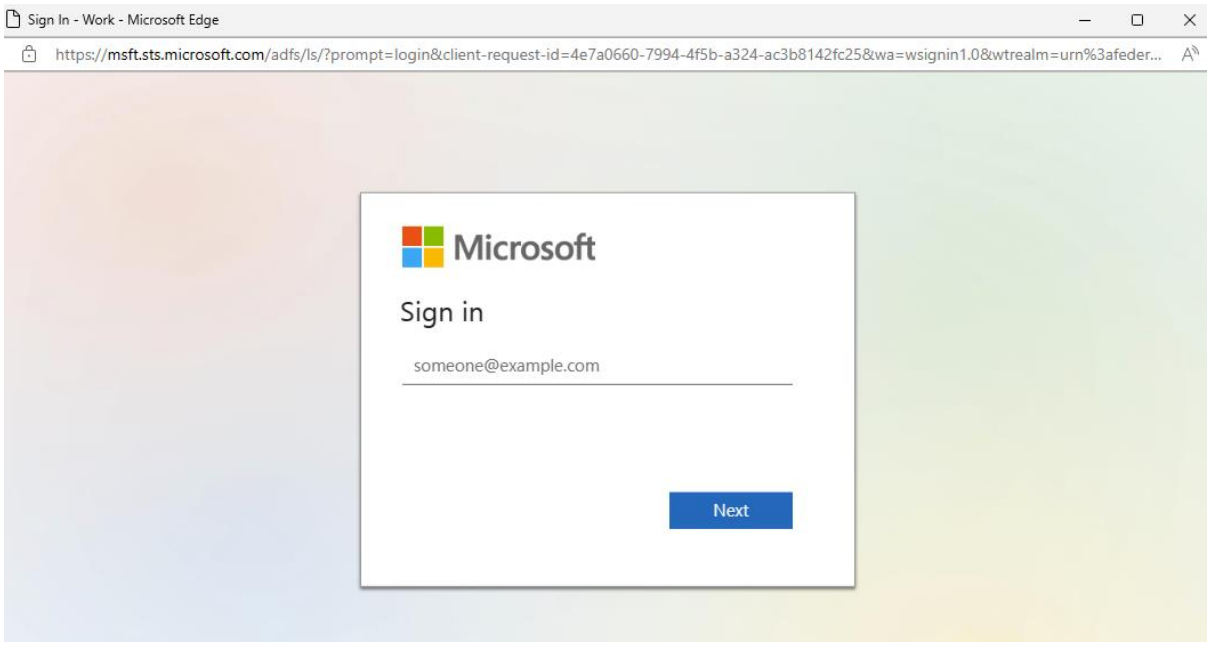

Figura 56: Inicio de sesión para la autenticación contra Power BI

Por último, para iniciar el job es necesario ir al recurso FromEventHub2PowerBI recién creado y seleccionar start, tal y como se muestra en la figura 57.

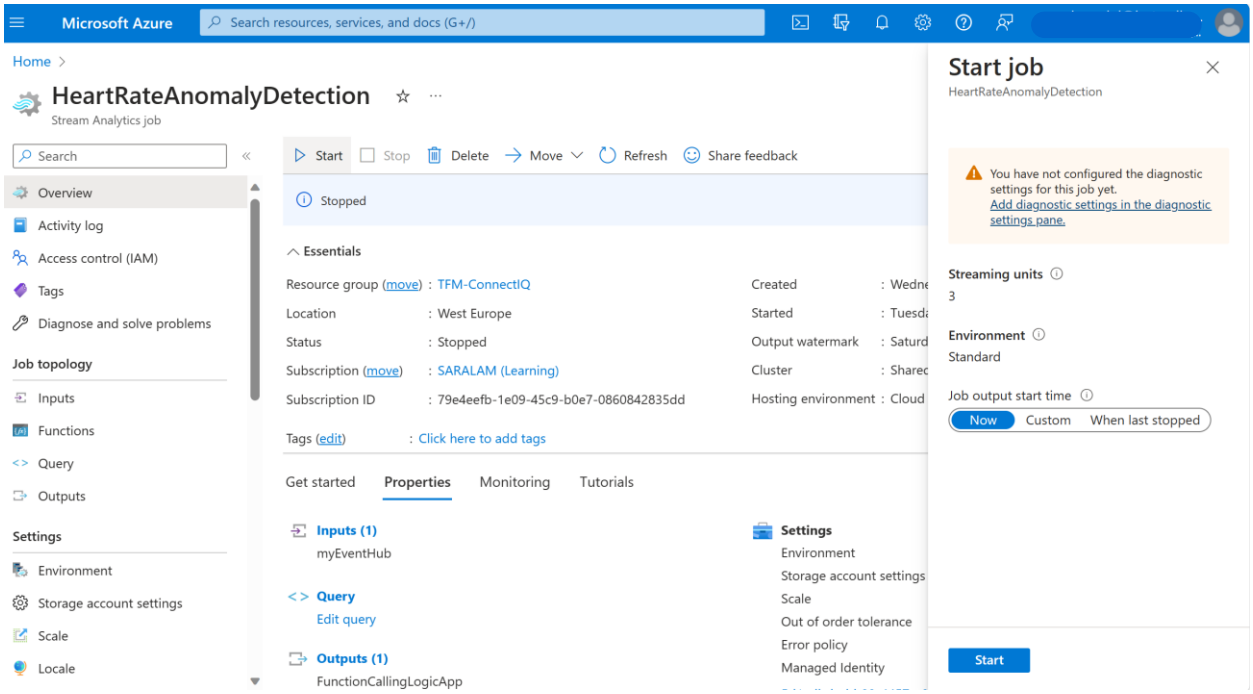

Figura 57: Iniciación del job

Tras iniciar el job, puede comprobarse desde Power BI como se ha creado un nuevo *dataset* con el nombre especificado a partir de los datos que se reciben en el Azure Event Hub.

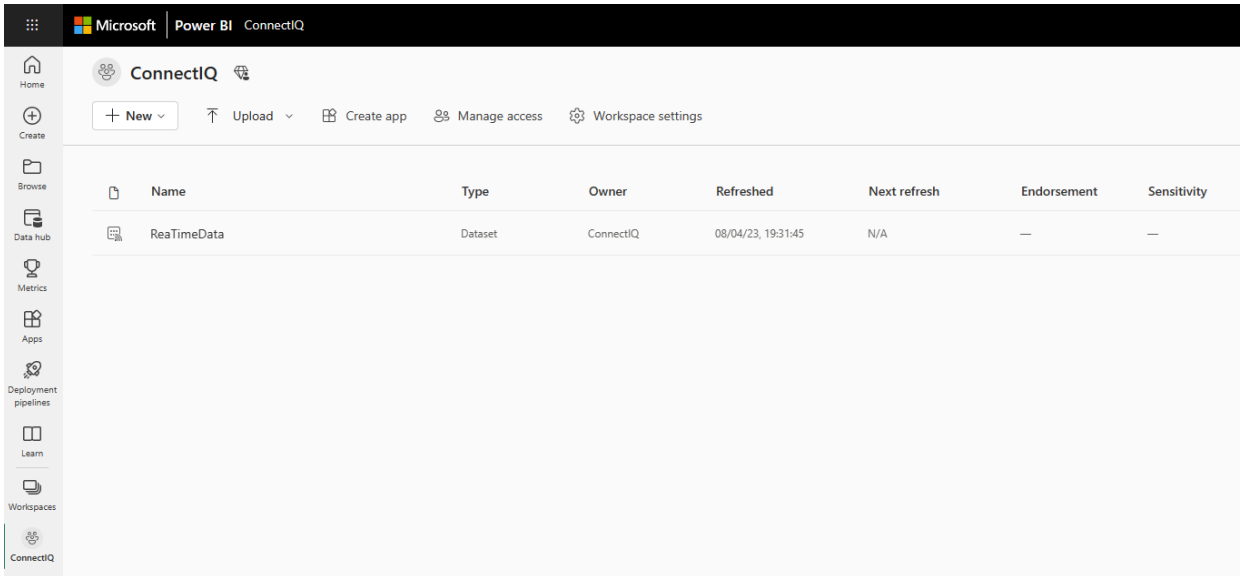

Figura 58. Nuevo Stream Dataset en PowerBI

3.2.6. Power BI

En esta sección se describen los pasos seguidos para crear un panel de monitorización en tiempo real con la información enviada por los relojes inteligentes conectados a la plataforma. Para ello, se debe seleccionar desde PowerBI *New -> Dashboard*, tal y como se recoge en la figura 59.

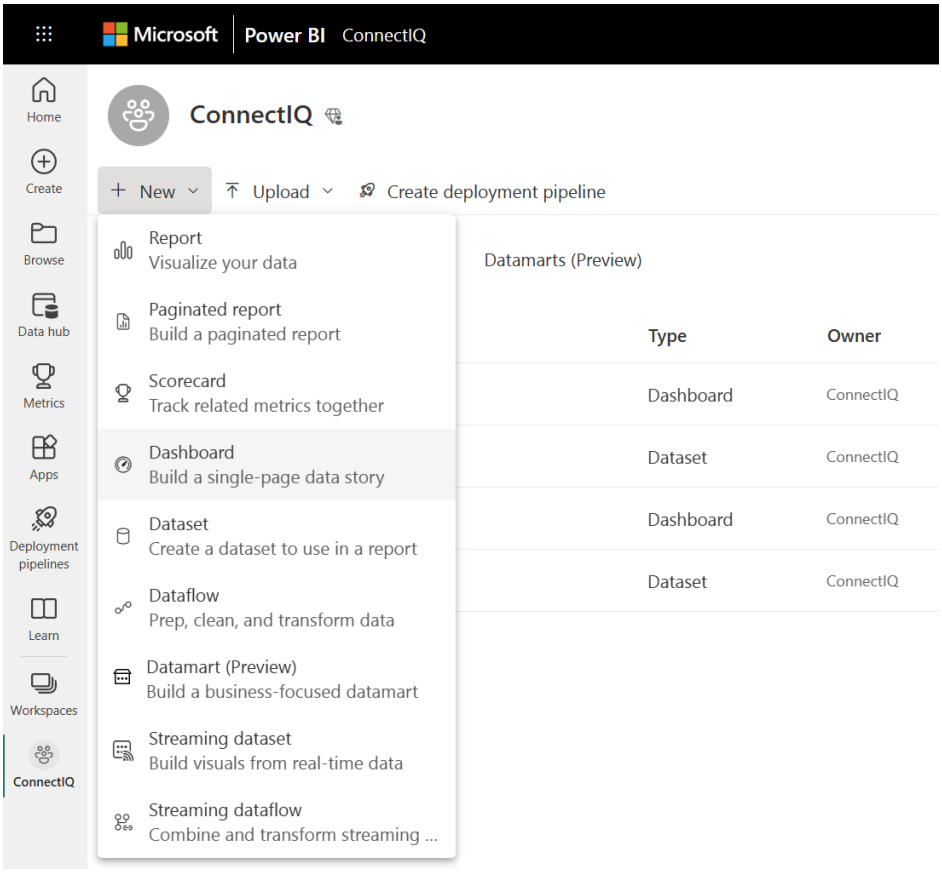

Figura 59: Creación de un nuevo Dashboard en Power BI 1

A continuación, una vez se abre la ventana de la figura 60, es momento de introducir el nombre del nuevo panel y seleccionar *create*.

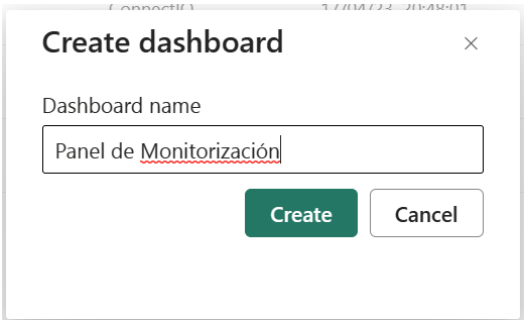

Figura 60: Creación de un nuevo Dashboard en Power BI 2

Una vez en el *Dashboard*, para añadir un nuevo título se necesario hacer clic en *Edit -> Add Title*, como puede verse en la figura 61.

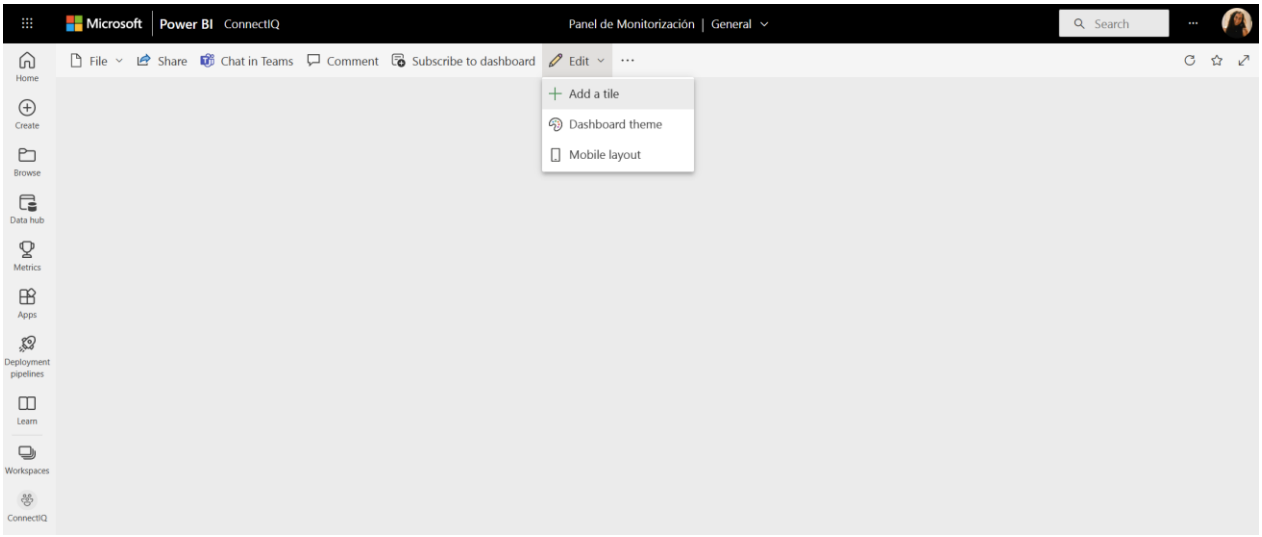

Figura 61: Añadir un nuevo título en Power BI

A continuación, se debe seleccionar en la nueva ventana de la figura 62 la opción *Custom Streaming Data* y hacer clic en el botón *Next*.

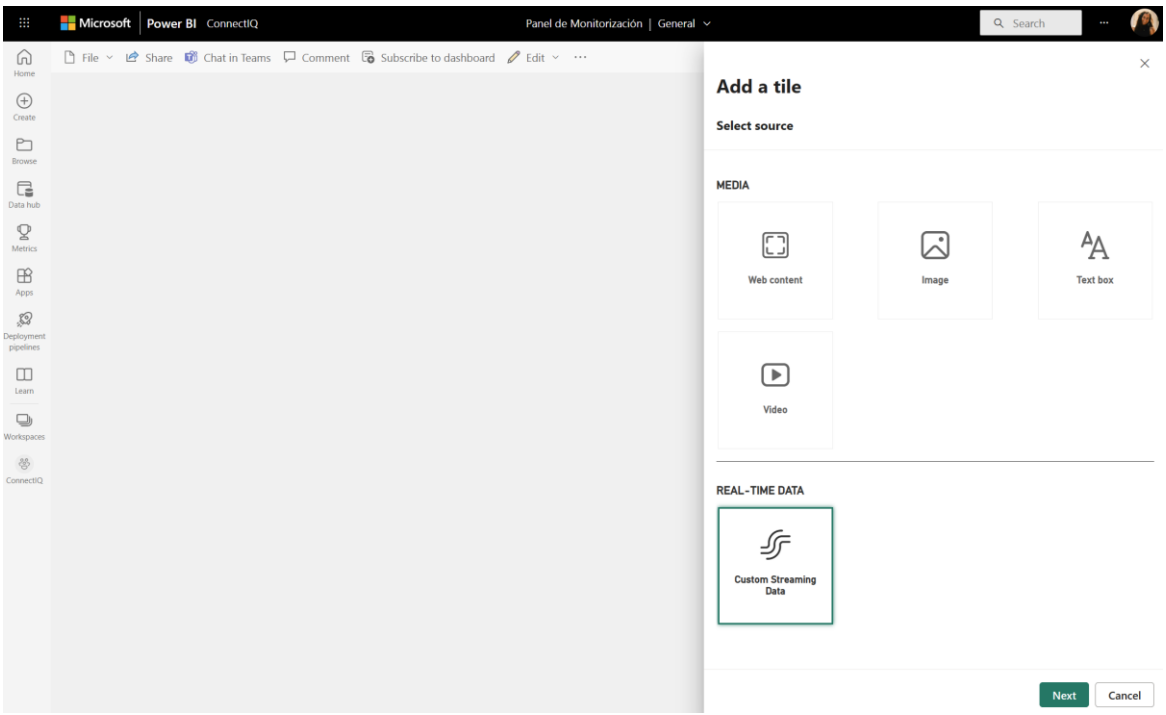

Figura 62: Añadir nuevo Custom Streaming Data en Power BI 1

Después es momento seleccionar el *streaming dataset* procedente de Azure Stream Analytics con los datos enviados por los relojes inteligentes conectados en tiempo real y volver a seleccionar *Next*. Este paso se muestra en la figura 63.

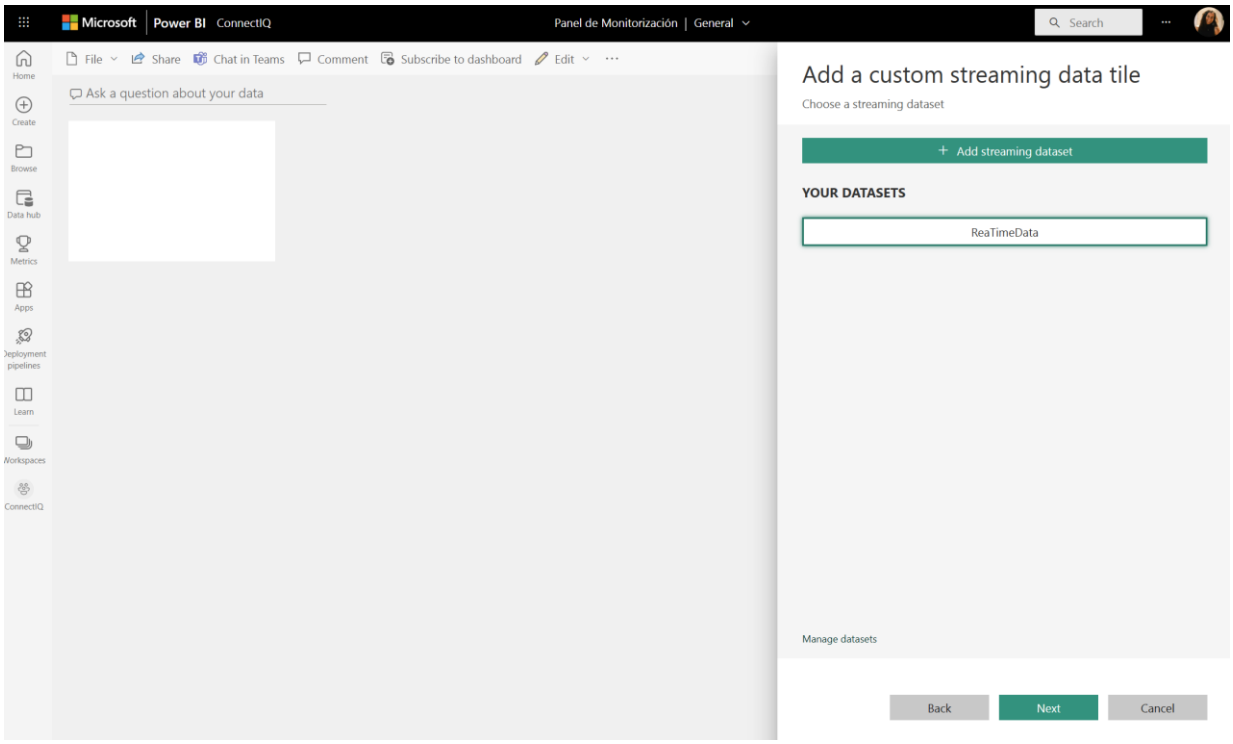

Figura 63: Añadir nuevo Custom Streaming Data en Power BI 2

Ahora ya se está en disposición de configurar el nuevo título.

El primer título que implementar tiene como objetivo representar el ritmo cardiaco de los atletas a lo largo del tiempo. Por esta razón, como se muestra en la figura 64, se han establecido los siguientes valores:

- *Visualization Type.* El tipo de visualización más adecuada es time chart.
- *Axis*. En el eje horizontal se selecciona EventEnqueuedUtcTime, es decir, el momento en el cual se encoló el evento en event hub.
- *Legend.* Establecemos como legenda el UserID, dado que es necesario identificar a que usuario pertenecen los datos de frecuencia cardiaca recibidos.
- *Values.* Como valores a mostrar se selecciona heartRate, es decir, los datos de frecuencia cardiaca registrados por los relojes.
- *Time window to display*. Se establece una ventana de tiempo de 1 minuto.

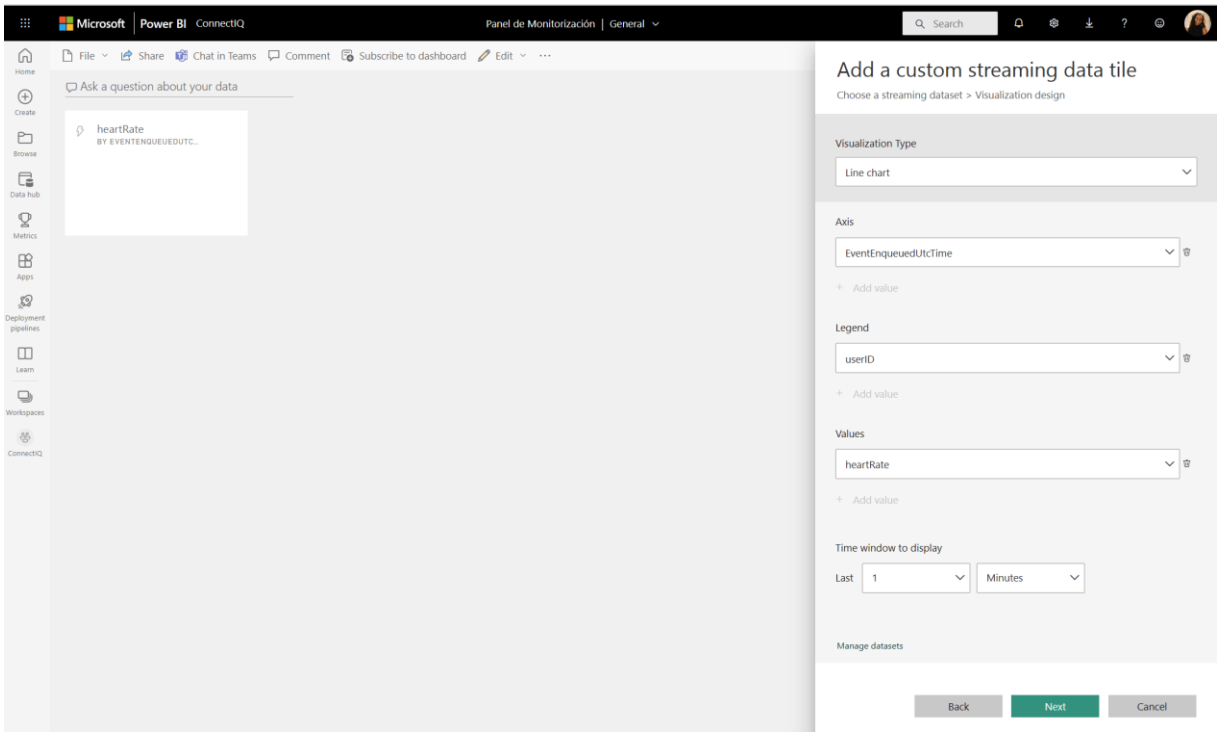

Figura 64: Visualization design para nuestro title

En la pestaña de *Title details* se dejan los datos por defecto y se selecciona Apply, como puede observarse en la figura 65.

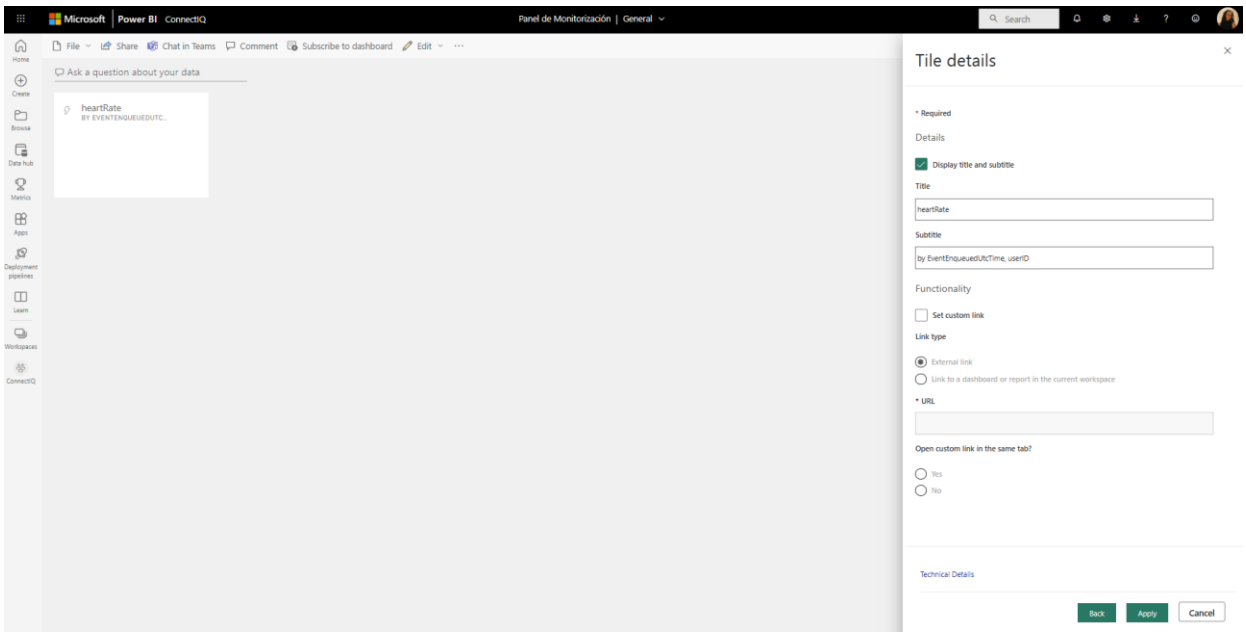

Figura 65: Title details

Como resultado, en la figura 66 se puede comprobar como en negro se representan los datos de frecuencia cardiaca en tiempo real registrados por el reloj (figura 67) mientras que en azul tenemos registrados por el simulador (figura 68).

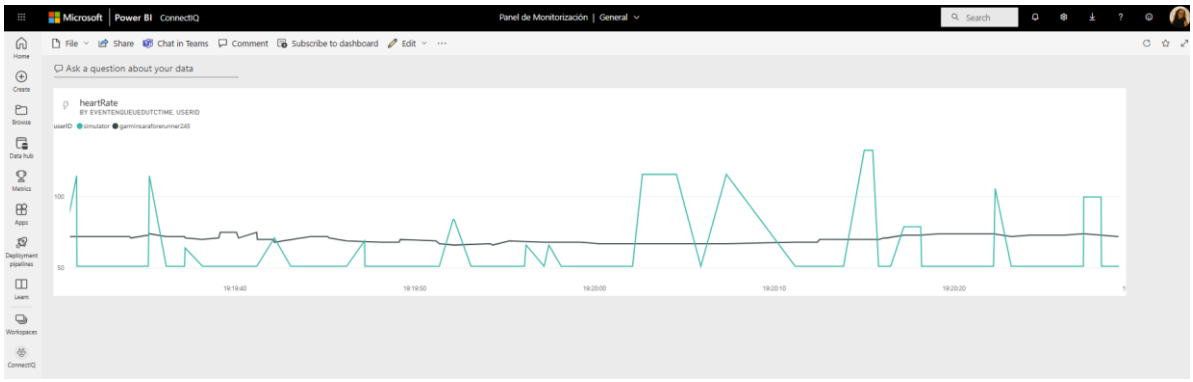

Figura 66: Frecuencia cardiaca registrada por el simulador y el reloj en tiempo real

Figura 67: Frecuencia cardiaca registrada por el reloj en tiempo real

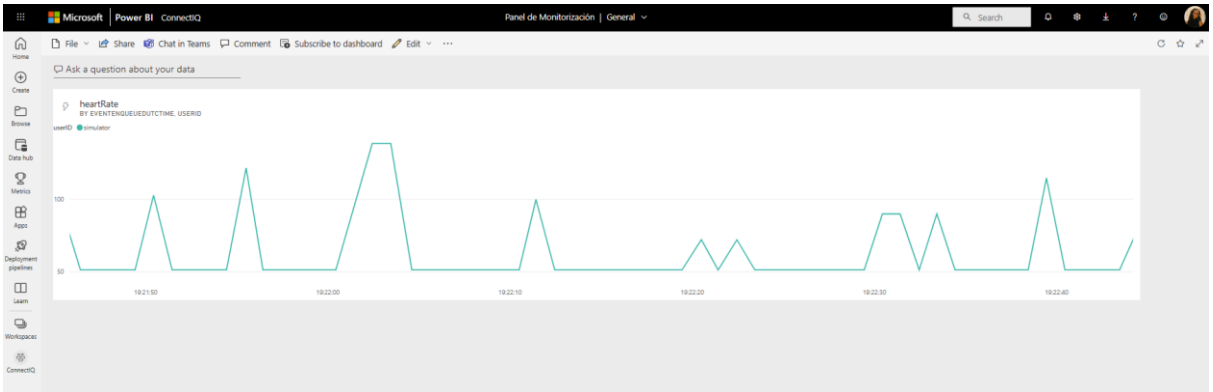

Figura 68: Frecuencia cardiaca registrada por el simulador en tiempo real

Otra forma de añadir nuevas visuales para representar la información de monitorización es la que se dispone en la figura 69 seleccionando *Ask a question about your data.*

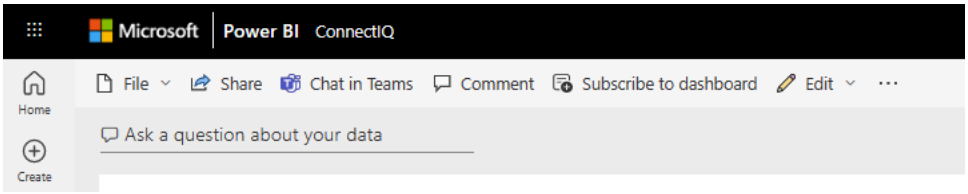

Figura 69: Ask a question about your data 1

Una vez se hace clic en *Ask a question abut your data* se abre la ventana mostrada en la figura 70 en la que se puede escribir cualquier cuestión sobre los datos recibidos o utilizar algunos de los ejemplos ofrecidos por PowerBI.

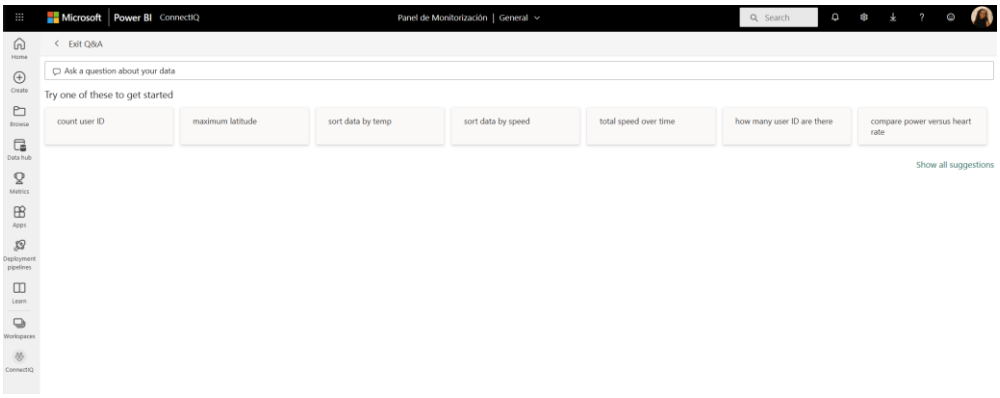

Figura 70: Ask a question about your data 2

Con el objetivo de añadir una tarjeta o *card* que muestre el número de usuarios coenctados a la plataforma de monitorización en tiempo real se selecciona count user ID. El resultado se recoge en la figura 71.

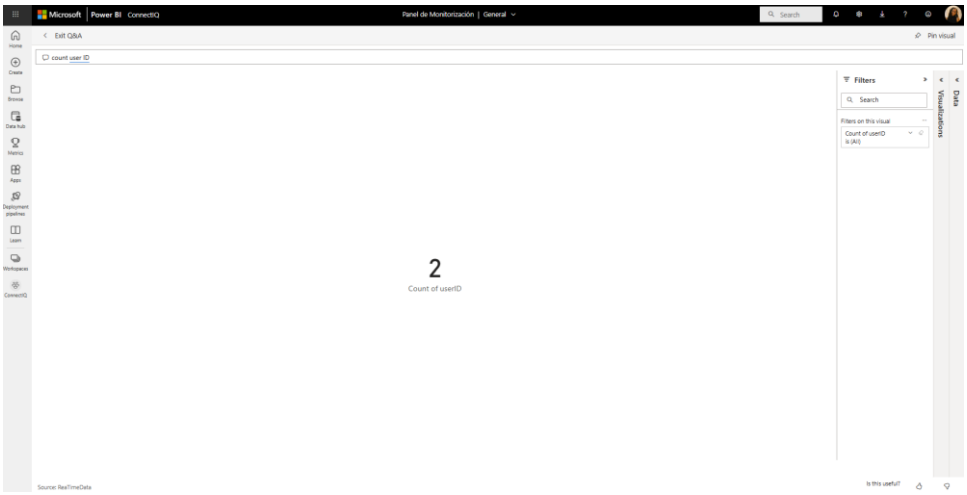

Figura 71: Creación de un nuevo visual a Power BI 1

A continuación, se muestra en la figura 72 como para añadir el nuevo visual al panel de monitorización, se hace clic en *pin visual* en la esquina superior derecha.

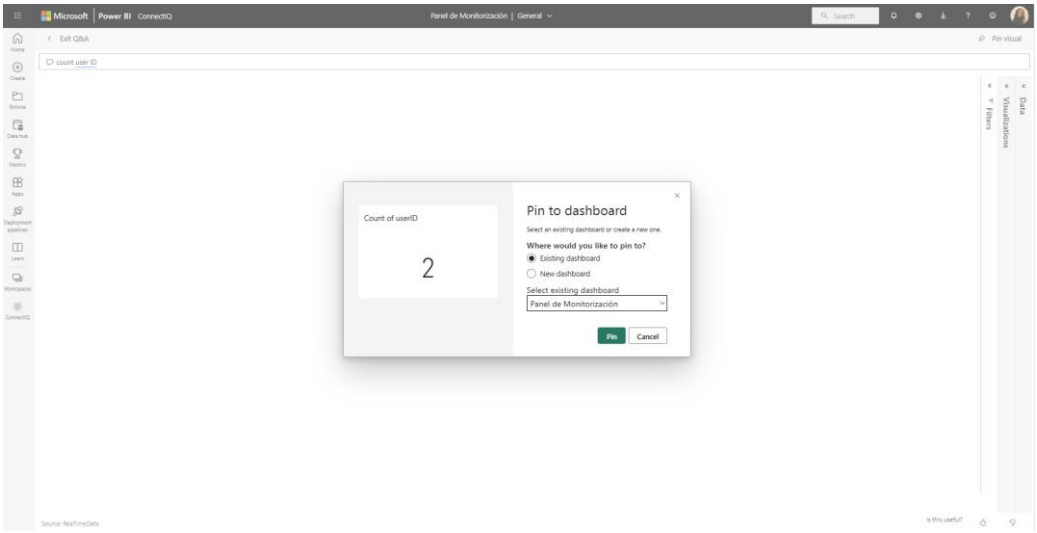

Figura 72: Creacón de un nuevo visual a Power BI 2

Si se vuelve al *dashboard* se observa como efectivamente se ha añadido la nueva tarjeta con el número de usuarios conectados en cada momento.

A continuación, en la figura 73, se muestra como añadir al panel un mapa con la ubicación de los corredores en tiempo real. Para ello, debe hacerse nuevamente clic en *Ask a question about your data* e introducir *latitude and longitude per user ID per event enqueued utc time*. En la pestaña de *visualization* s selecciona *Map*.

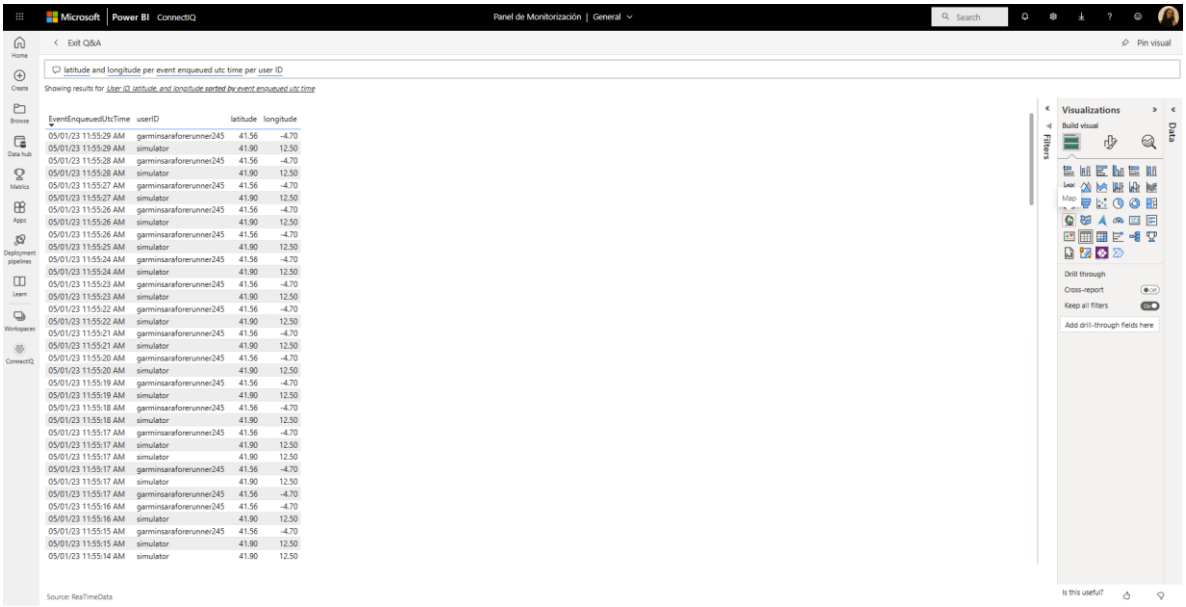

Figura 73: Creación de un mapa para la visualización de la ubicación de corredores

Una vez seleccionado map es posible elegir entre distintas topologías de mapa, tal y como puede observarse en la figura 74.

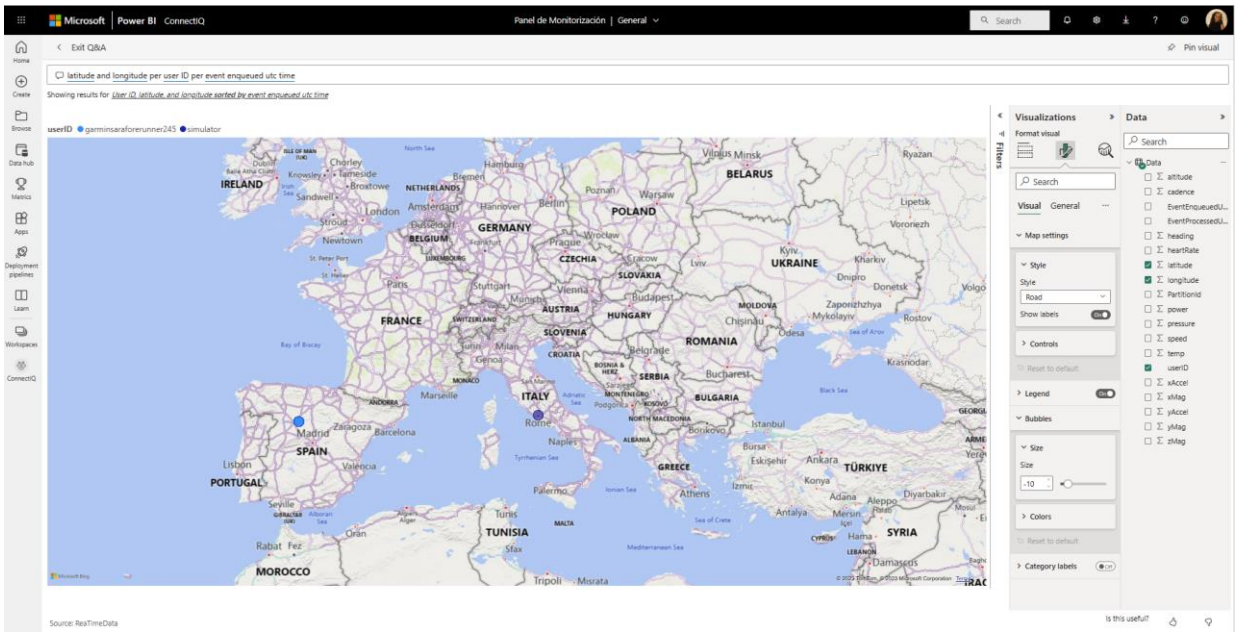

Figura 74: Vista del mapa 1

De forma análoga al caso anterior se debe seleccionar pin visual en la esquina superior derecha para añadir el mapa al *Dashboard*. En la figura 75 se aprecia como el reloj real se encuentra ubicado en Valladolid mientras que el lo está en Roma.

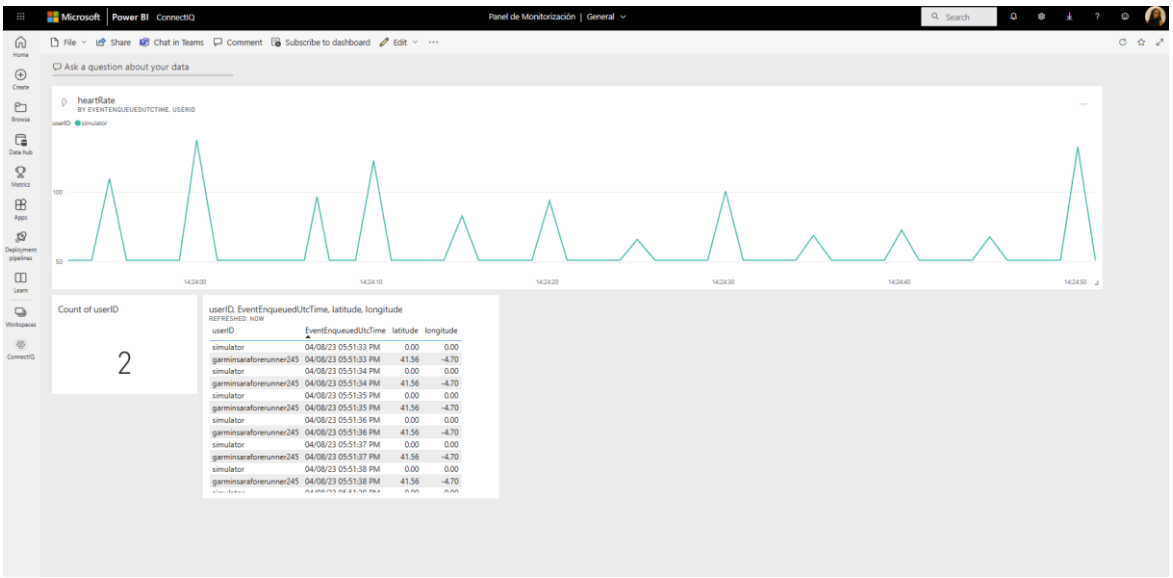

Figura 75: Vista del dashboard de monitorización

Aunque en el *dashboard* esta información se añade como una tabla, si se selecciona *Go to Q&A* como se plasma en la figura 76, es posible visualizar la información nuevamente en el mapa, como se muestra en la figura 77.

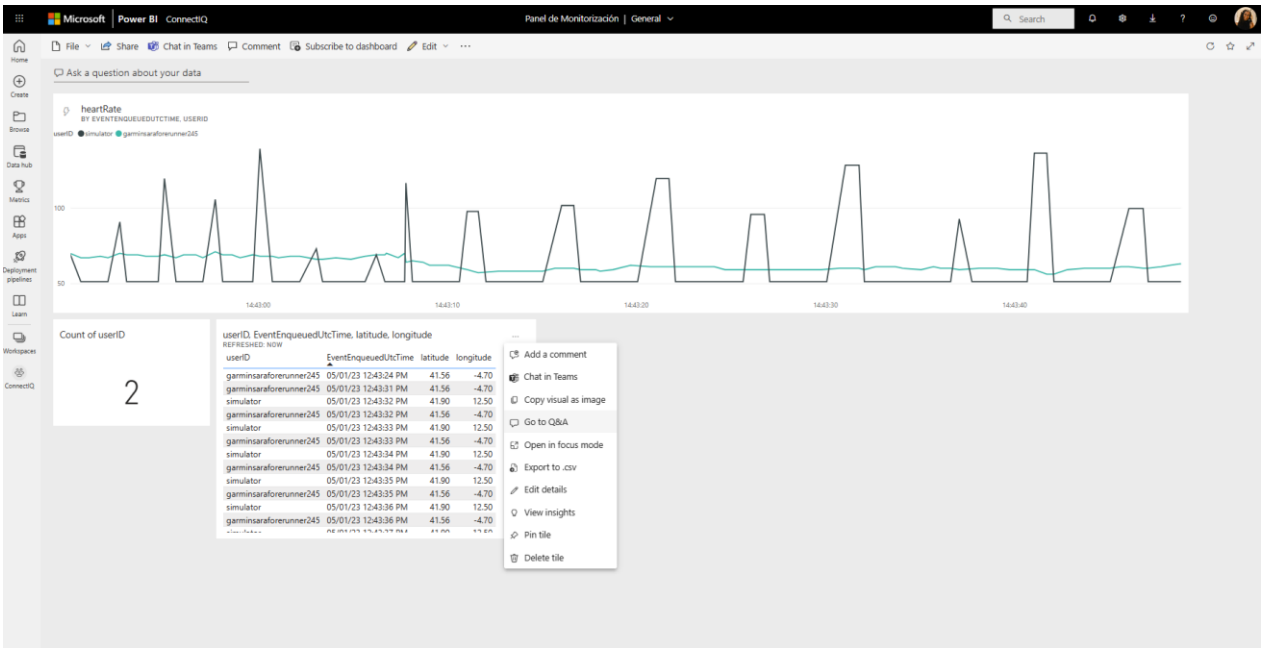

Figura 76: Acceso al mapa 1

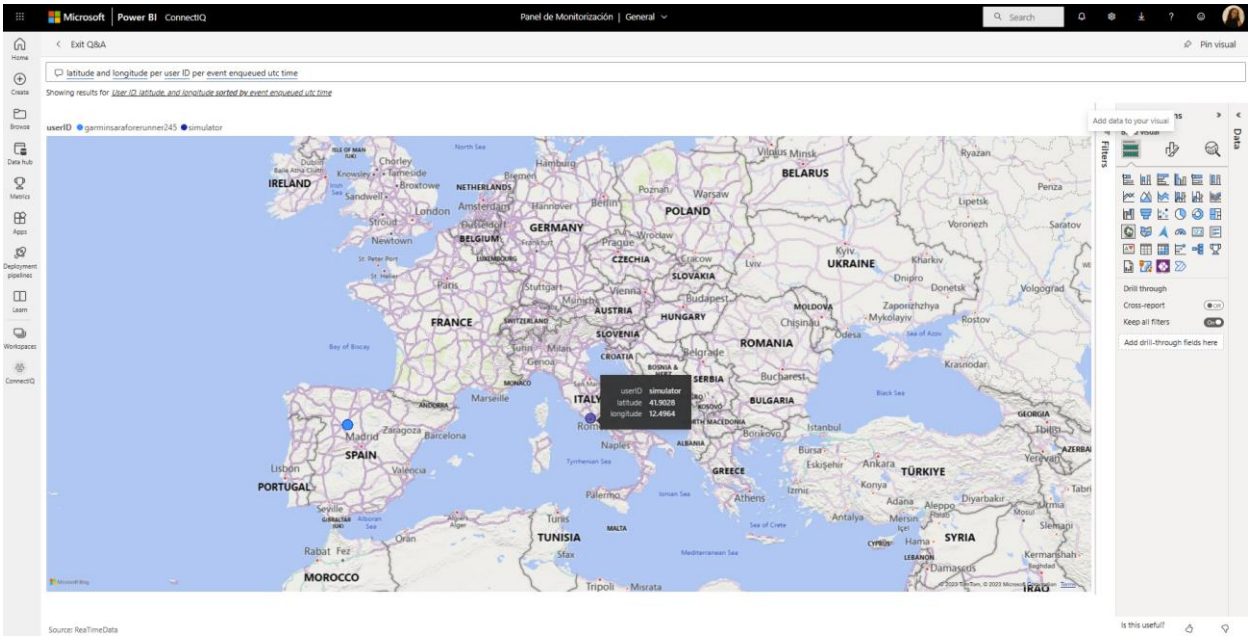

Figura 77: Acceso al mapa 2

A continuación, se procede a añadir un nuevo objeto de visualización con el objetivo de mostrar la frecuencia máxima para cada usuario. De forma análoga a como se ha hecho anteriormente, se selecciona *Ask a question about your data*. Después, se escribe *max heart rate garminsaraforerunner245* y se selecciona pin visual, tal y como se ve en la figura 78.

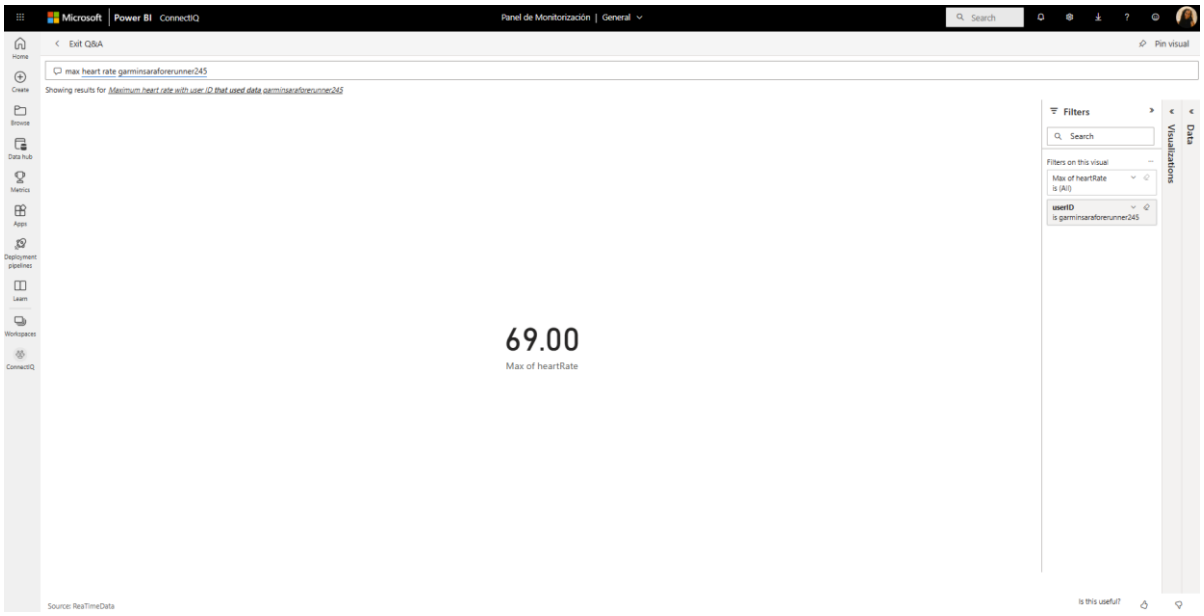

Figura 78: Adición de un nuevo visual a Power BI 3

Se repite en la figura 79 el mismo procedimiento para el simulador.

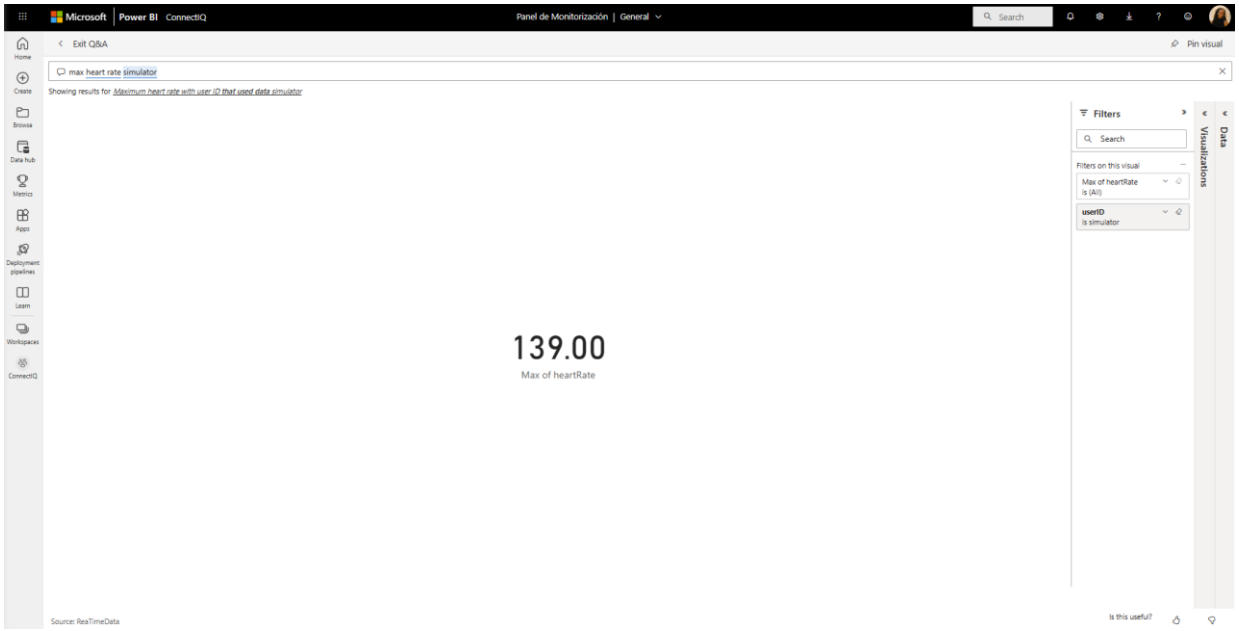

Figura 79: Adición de un nuevo visual a Power BI 4

Para personalizar la vista de nuestro *dashboard* se muestra en la figura 80 como se debe seleccionar Edit *Dashboard* theme.

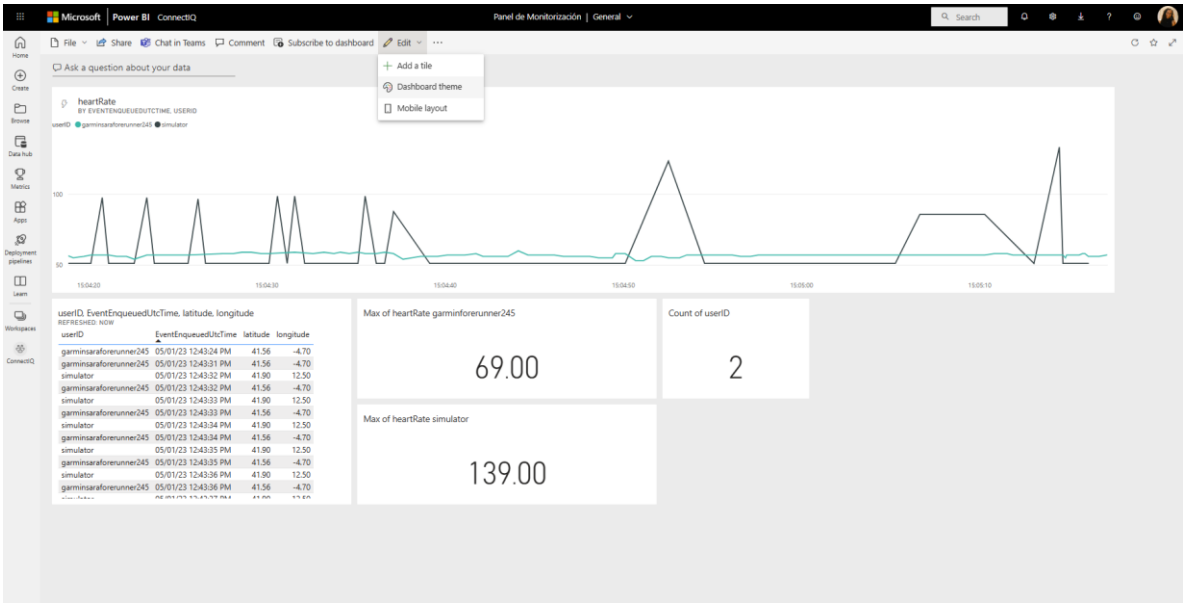

Figura 80: Dashboard de monitorización

En la nueva de la figura 81 puede modificarse la vista del panel de monitorización como se considere oportuno.

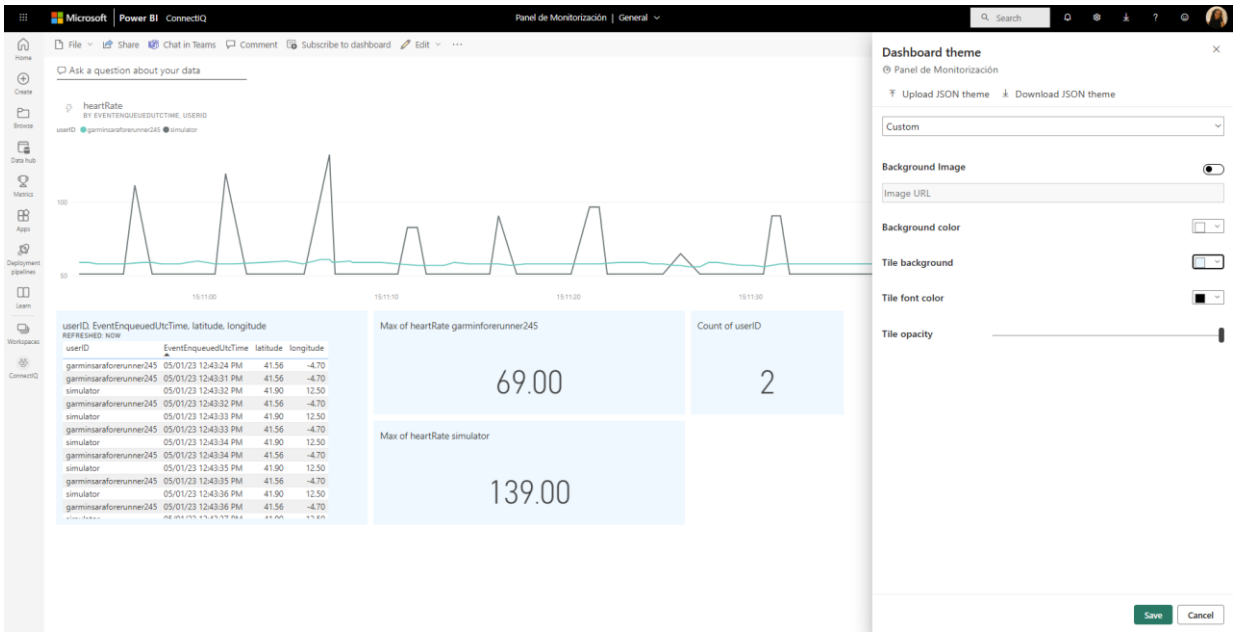

Figura 81: Personalización del Dashboard de monitorización

Una característica muy interesante que ofrecen las tarjetas o cards de cara a una solución de monitorización es la opción de establecer alertas. Si por ejemplo se desea recibir una notificación en Power BI cuando un nuevo dispositivo se coencta a la plataforma de monitorización, se debe seleccionar los 3 puntos en la esquina superior derecha de la tarjeta y hacer clic en *manage alerts*, tal y como se refleja en la figura 82.

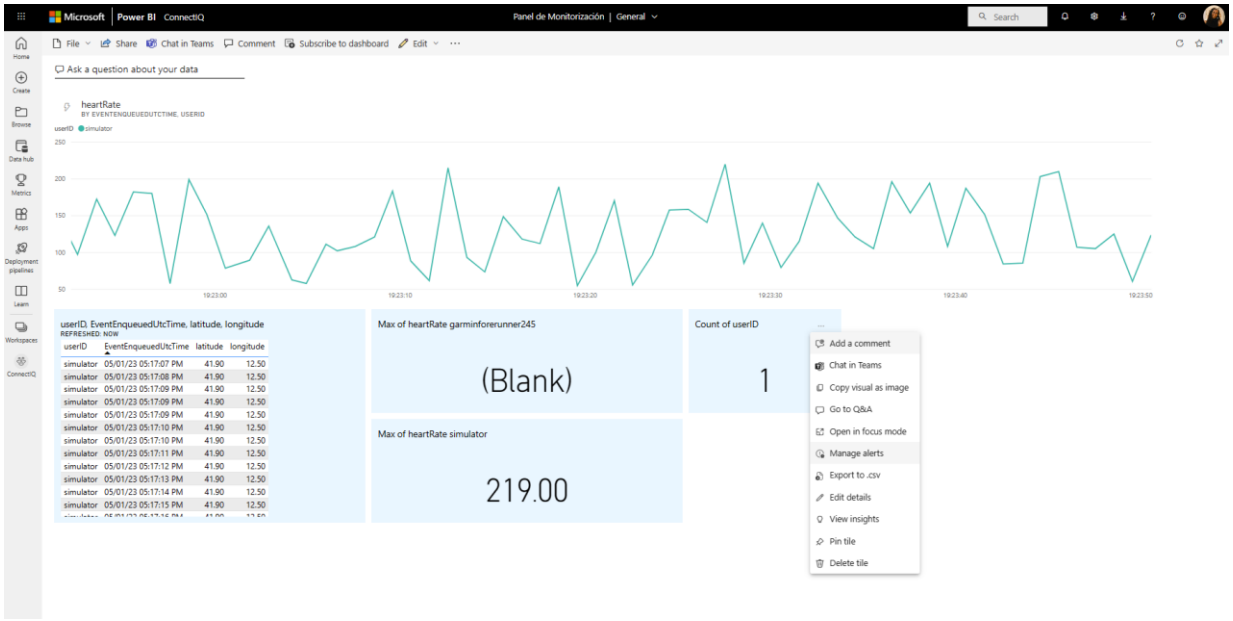

Figura 82: Generación de alertas 1

Esto abrirá el siguiente panel de la figura 83 en el que podemos configurar una nueva alerta.

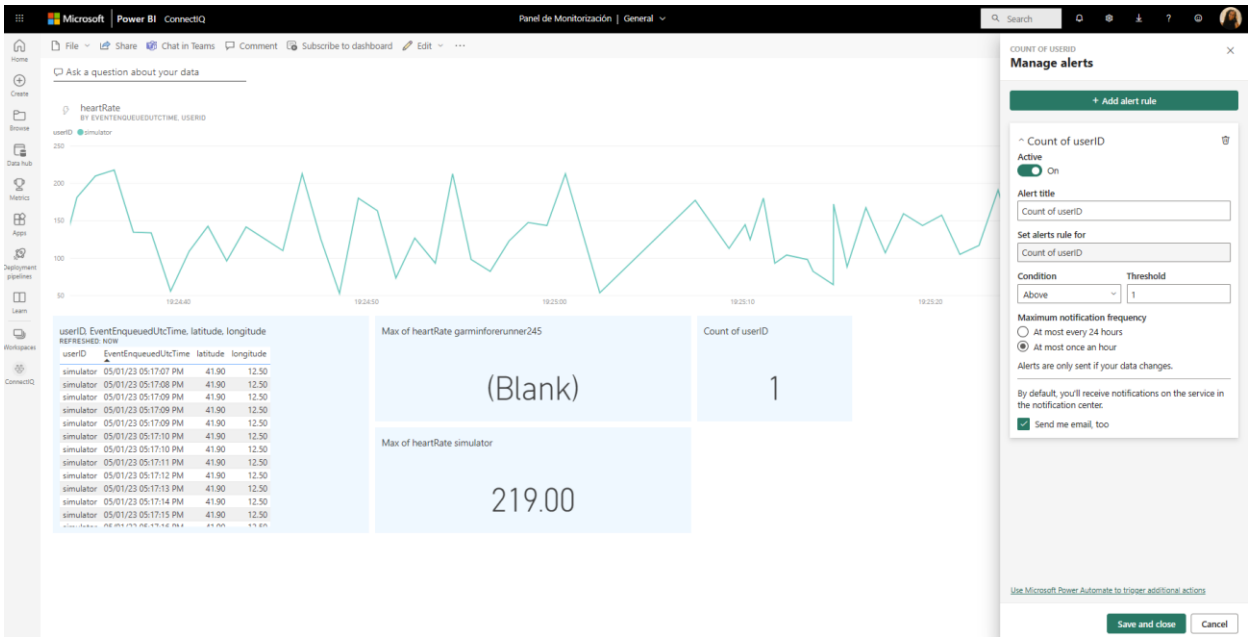

Figura 83: Generación de alertas 2

Al conectar un nuevo dispositivo se recibe la alerta de la figura 84.

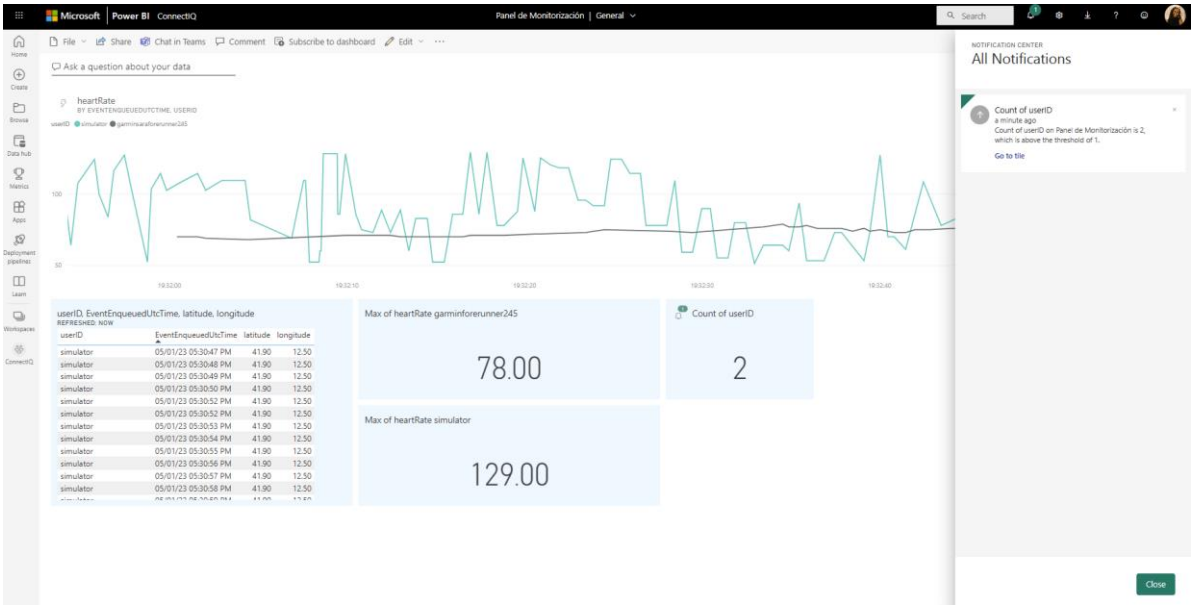

Figura 84: Generación de alertas 3

3.2.7. Logic App

En este apartado se van a presentar los pasos seguidos para la implementación de una Logic App con un desencadenador HTTP. Para ello, el primer paso consiste en ir a Logic apps desde el portal de Azure y seleccionar + Add, como se recoge en la figura 85.

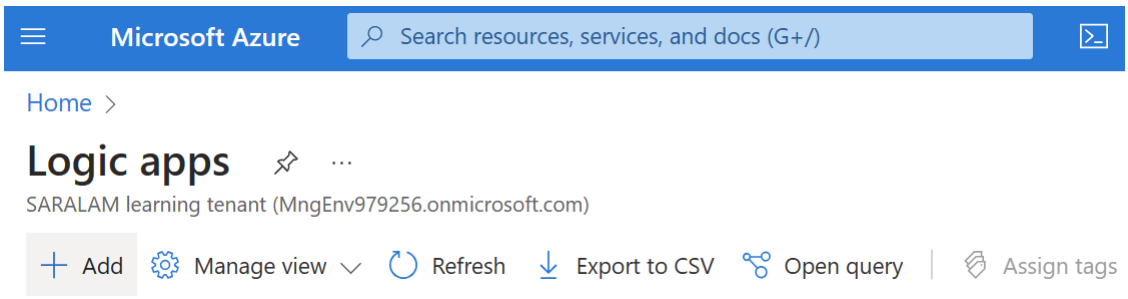

Figura 85: Creación de una nueva Logic App 1

A continuación, en la figura 86 se muestran los datos que se deben rellenar para la creación de la nueva Logic App:

- Suscripción
- Grupo de recursos
- Nombre la Logic App
- Publicador que puede ser un flujo de trabajo o un contenedor doker
- Región
- Tipo de plan que puede ser o *Standard* o *Consumption*. Dado que se esta implementando un prototipo se ha seleccionado la opción *Consumption*
- Redundancia de zona que puede habilitarse o deshabilitarse. En este escenario no es necesario contar con redundancia de zona.

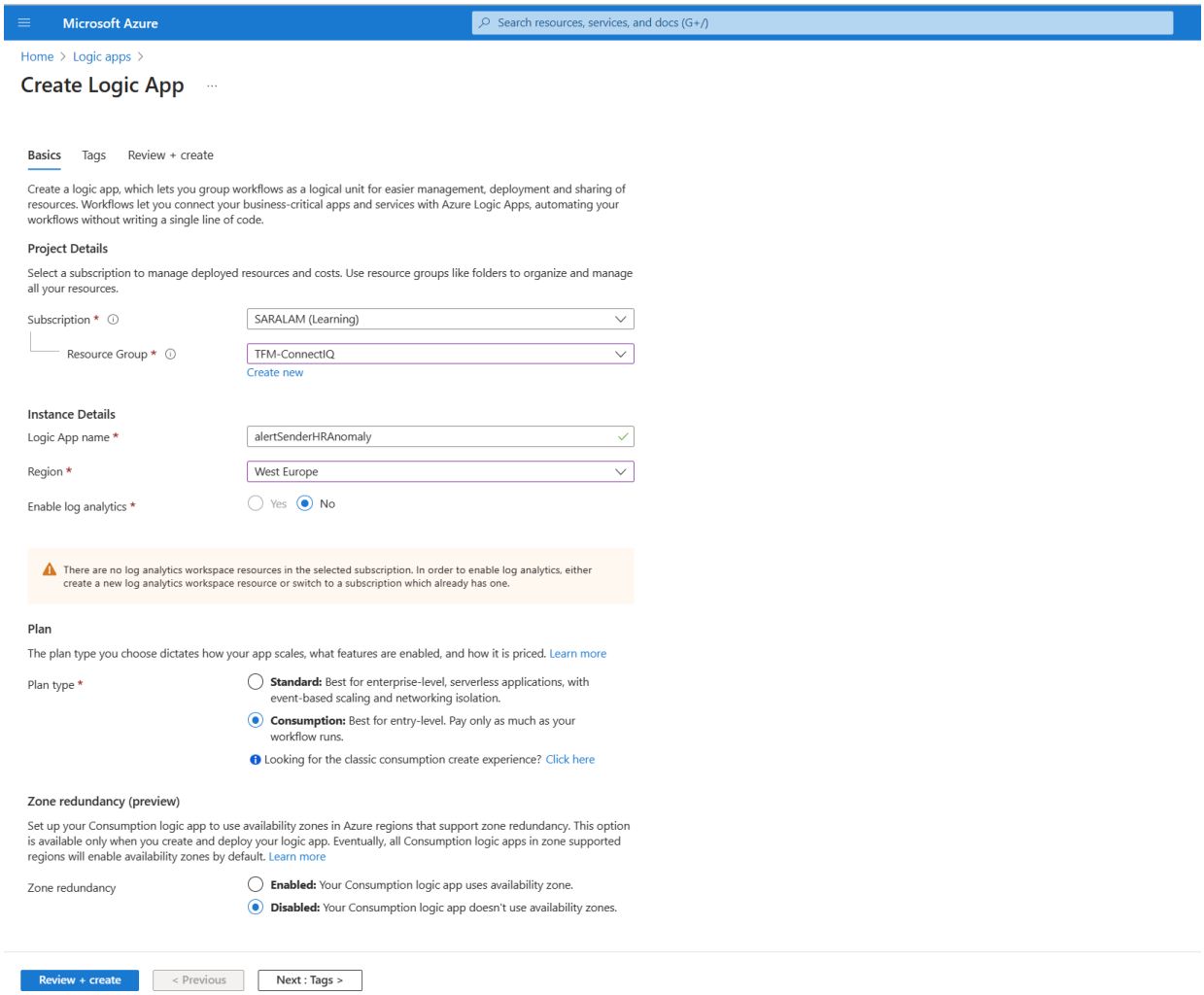

Figura 86: Creación de una nueva Logic App 2

Tras la finalizar el proceso de creación de la Logic App, es posible ir a la pestaña *Logic App Designer,* tal y como se muestra en la figura 87. Una vez en el diseñador, se selecciona *When an HTTP request is received* (Cuando se recibe una solicitud HTTP) con el fin de agregar el desencadenador HTTP a la Logic App.

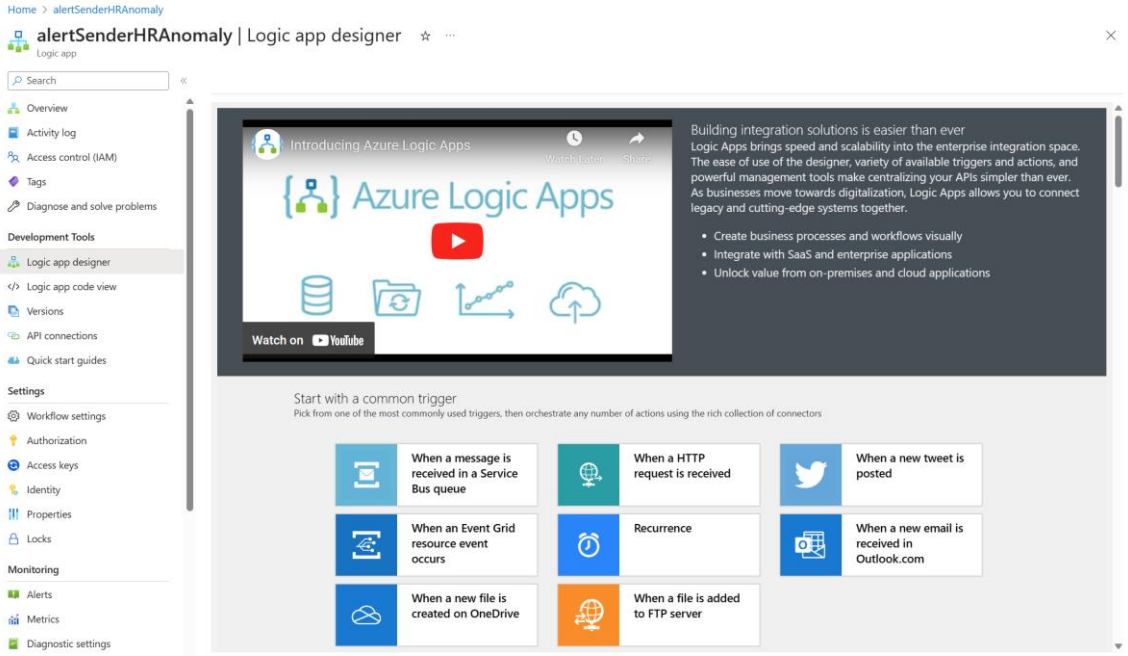

Figura 87: Creación de un nuevo flujo de trabajo 1

Tras seleccionar *When HTTP request is received*, como se muestra en la figura 88, es posible añadir un nuevo paso haciendo clic en *+ New Step*.

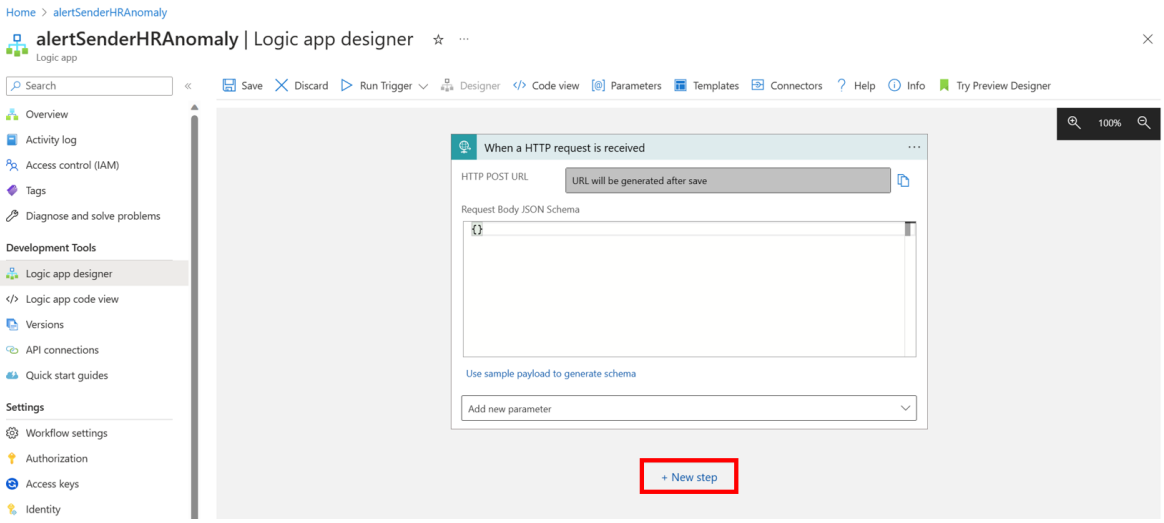

Figura 88: Creación de un nuevo flujo de trabajo 2

A continuación, se abre el desplegable de la figura 89.

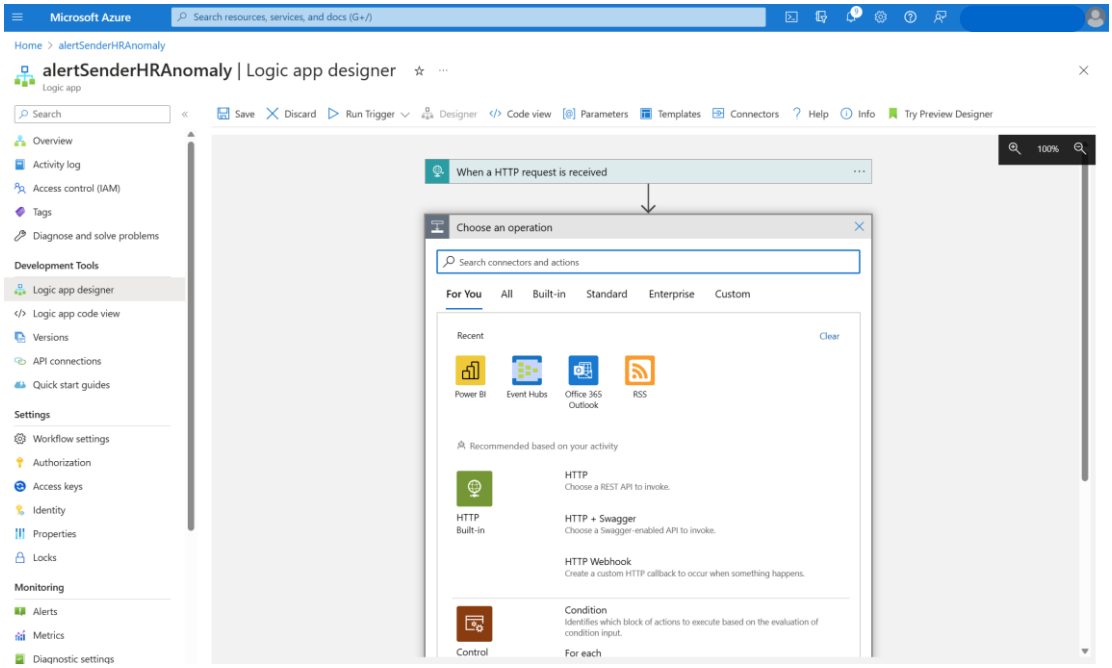

Figura 89: Creación de un nuevo flujo de trabajo 3

Dado que lo que se desea es enviar un correo electrónico cuando se desencadena la Logic App, se muestra en la figura 90 como se procede a seleccionar la acción *send an email* (enviar un correo electrónico).

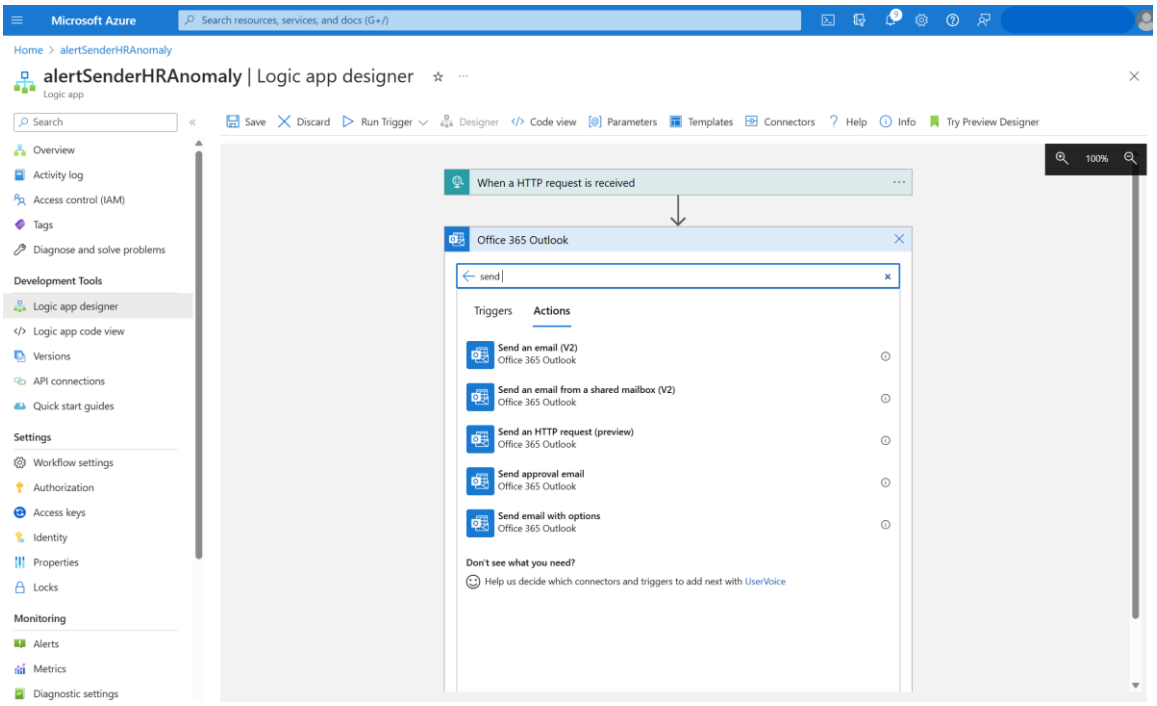

Figura 90: Creación de un nuevo flujo de trabajo 4

Después, se observa en la figura 91 como se precisa iniciar sesión con una cuenta de Outlook.

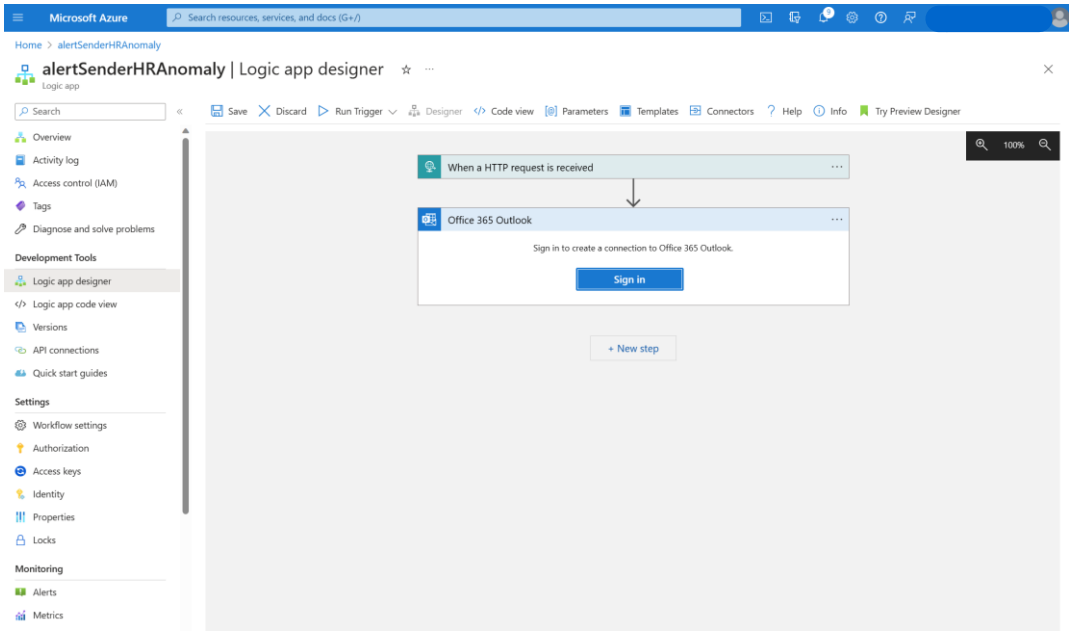

Figura 91: Creación de un nuevo flujo de trabajo 5

Tras señeccionar *Sign in* es posible iniciar sesión como se captura en la figura 92.

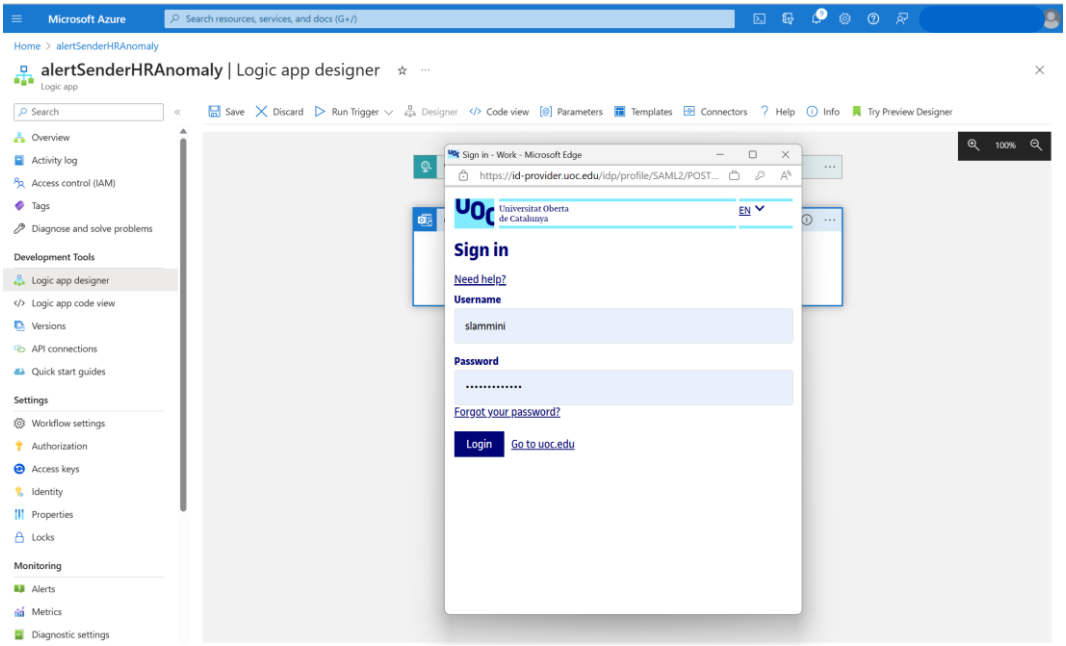

Figura 92: Creación de un nuevo flujo de trabajo 6

Llegados a este punto, es momento de especificar la información necesaria, como la dirección del correo electrónico del destinatario, el asunto y el cuerpo del correo electrónico de la forma que se recoge en la figura 93.

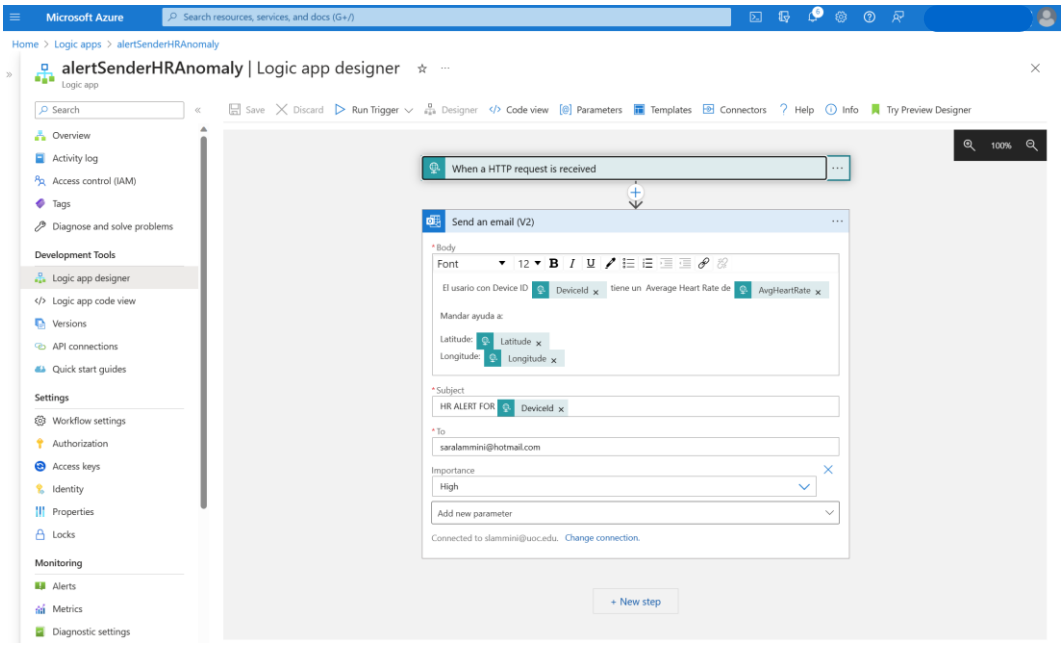

Figura 93: Creación de un nuevo flujo de trabajo 7

Una vez se hace clic en guardar en la parte superior del diseñador, se muestra en la figura 94 como se genera la URL del desencadenador HTTP.

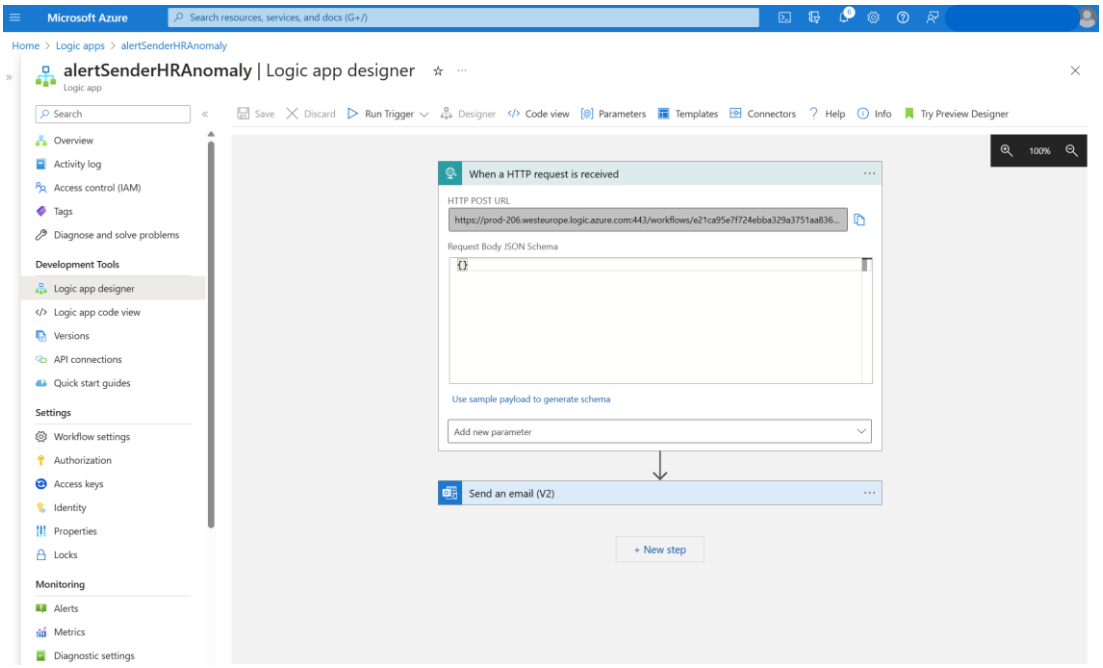

Figura 94: Creación de un nuevo flujo de trabajo 8

Tras la finalización de todos estos pasos se obtiene una Logic App con un desencadenador HTTP y una acción para enviar un correo electrónico. La URL del desencadenador HTTP se utilizará para llamar a la Logic App desde nuestra *function*.

Antes de pasar a la implementación de la *function*, es conveniente testar el funcionamiento de la Logic App que recién creada para verificar que es el esperado. Para ello, tal y como se observa en la figura 95, se debe seleccionar *Run Trigger -> Run trigger with playload* en el menú superior. En *playload* debe ingresarse las propiedades que se desean incluir en formato JSON.

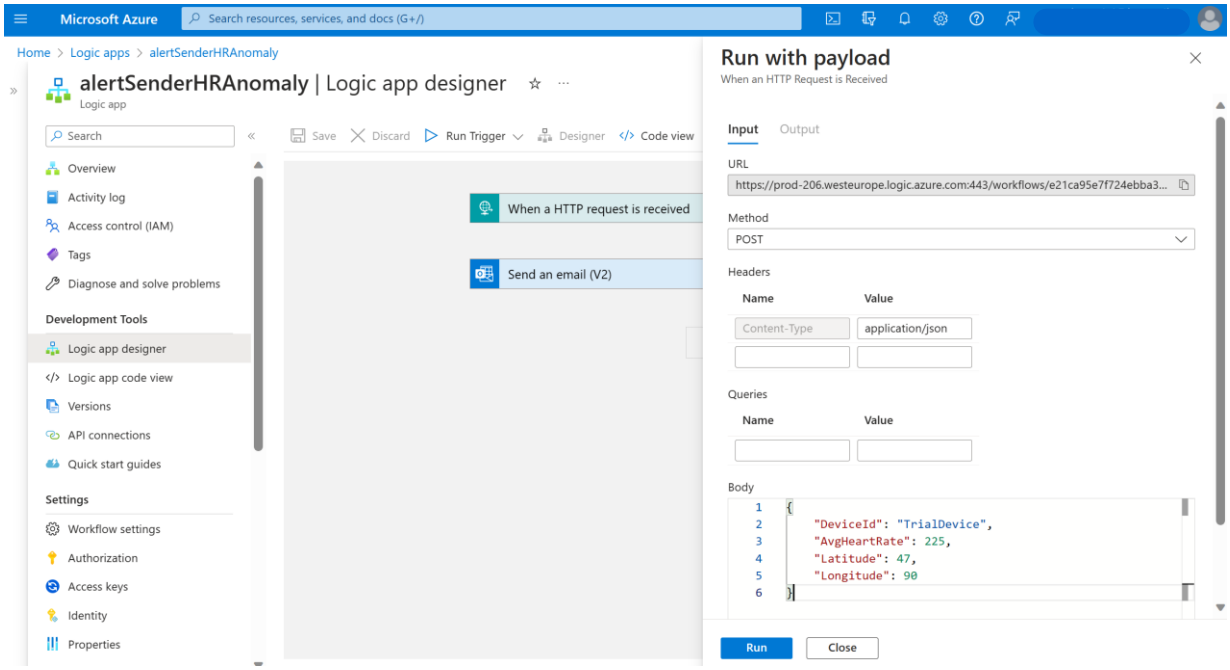

Figura 95: Test Logic App 1

Puede comprobarse en la figura 96 como efectivamente el flujo de trabajo se ejecuta correctamente en menos de un segundo siguiendo el comportamiento esperado.

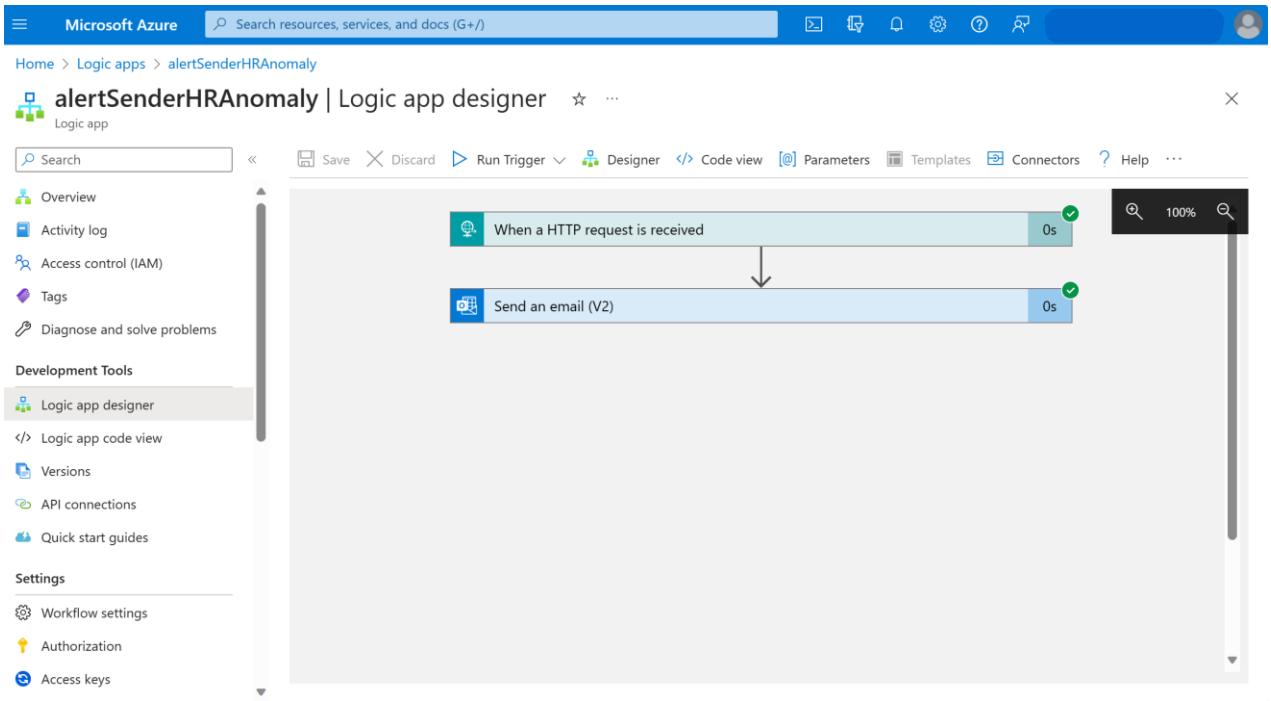

Figura 96: Test Logic App 2

Si se comprueba el correo electrónico puede verificarse que efectivamente se ha recibido el email correctamente, tal y como se muestra en la figura 97.

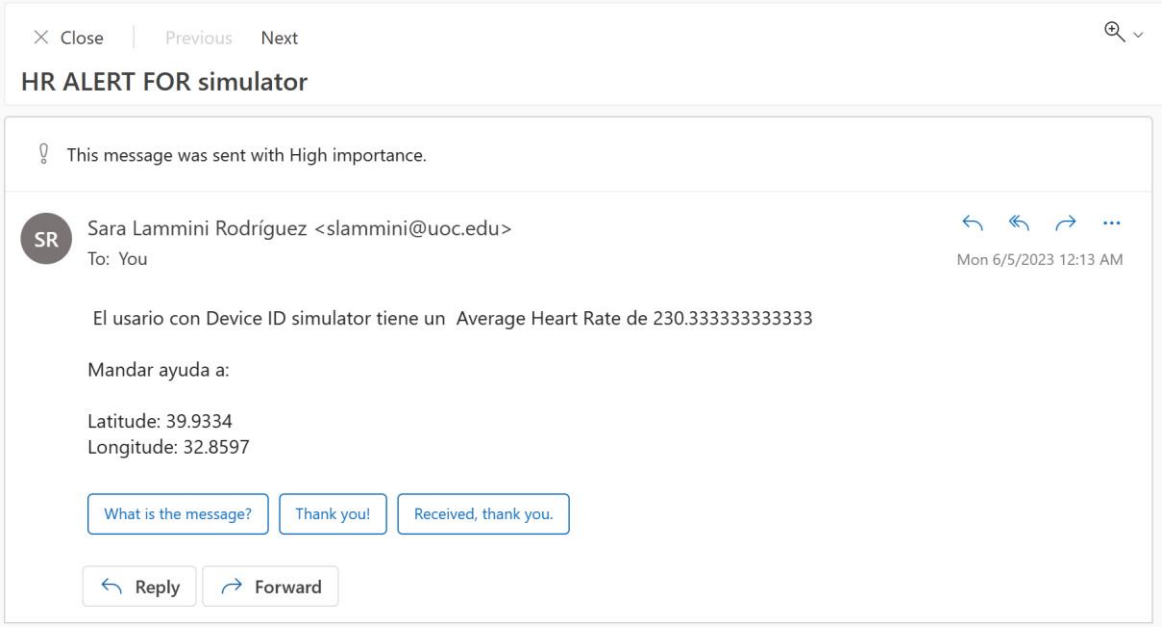

Figura 97: Test Logic App 3

3.2.8. Azure Function

En esta sección se describen los pasos seguidos para crear la *function* responsable de desencadenar la Logic App que va a enviar el correo de solicitud de auxilio. Para ello, el primer paso consiste en buscar Function App en el buscador del portal de Azure y hacer clic en + *Create*, tal y como se recoge en la figura 98.

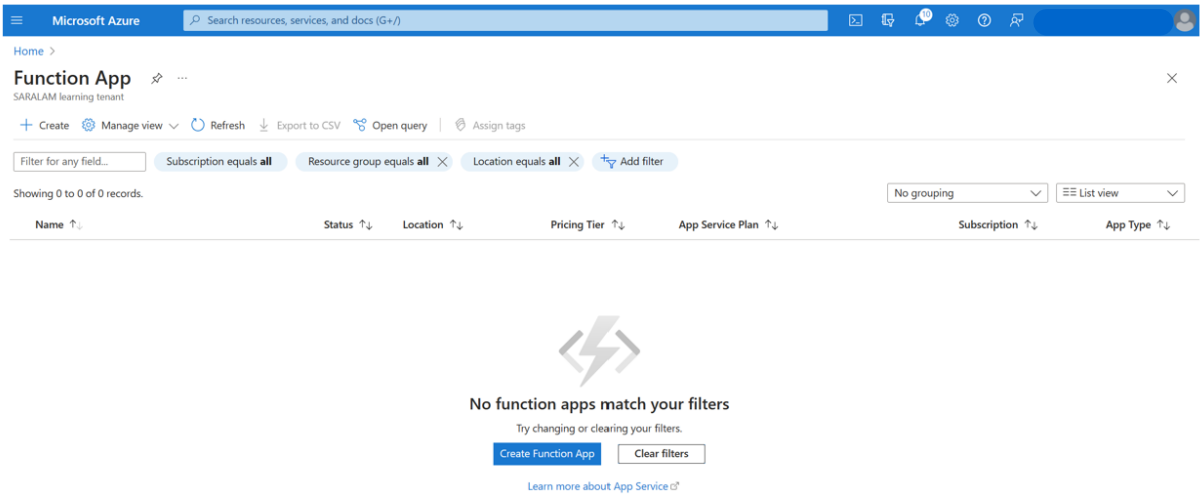

Figura 98: Creacion Function App 1

A continuación, debe especificarse la configuración deseada para la *Function* App tal y como se muestra en la figura 99, rellenando los siguientes campos:

- Suscripción
- Grupo de recursos
- Nombre de la Function App
- Si se desea desplegar a partir de código o de la imagen de un contenedor. Para este escenario se selecciona código.
- *Runtime stack*
- Versión
- Región
- Sistema operativo
- Las opciones de *hosting* a elegir entre severless, function o App service plan.

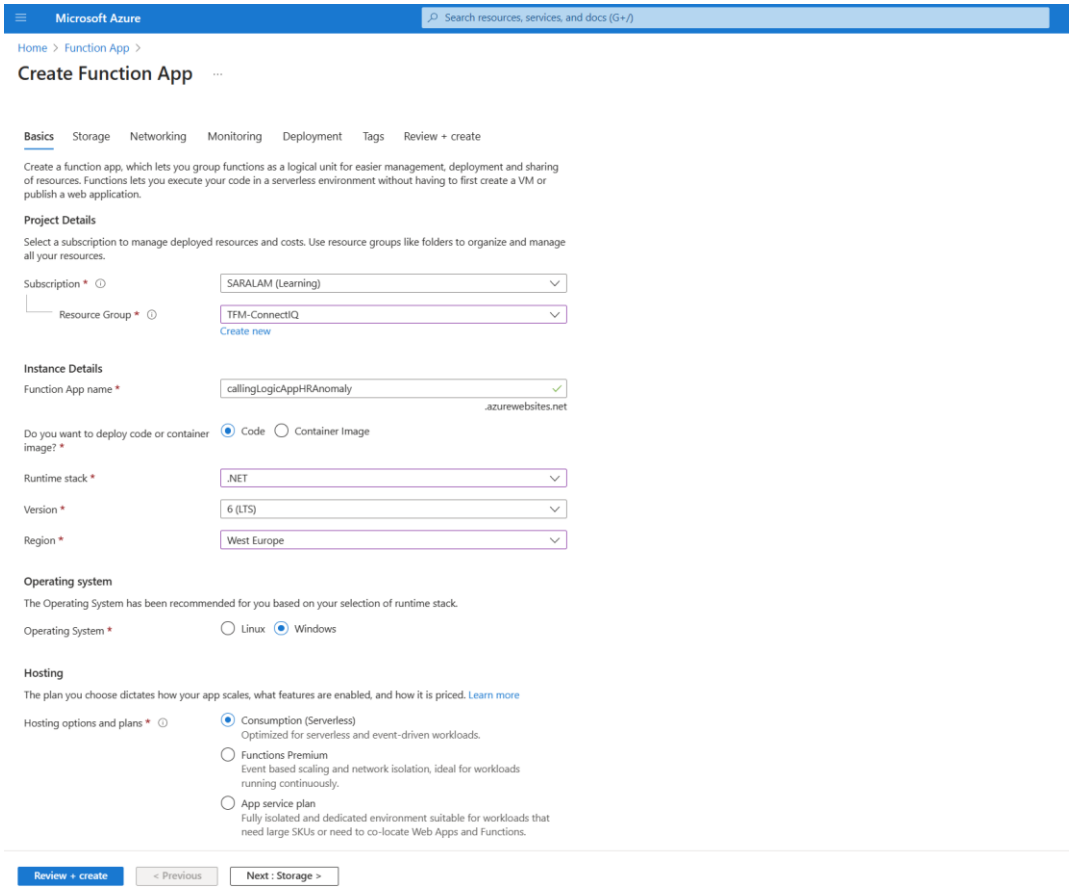

Figura 99: Creación Function App 2

Una vez creada la *function* App, desde la pestaña *Function*s se puede crear una nueva función haciendo clic en *create*, tal y como se muestra en la figura 100.

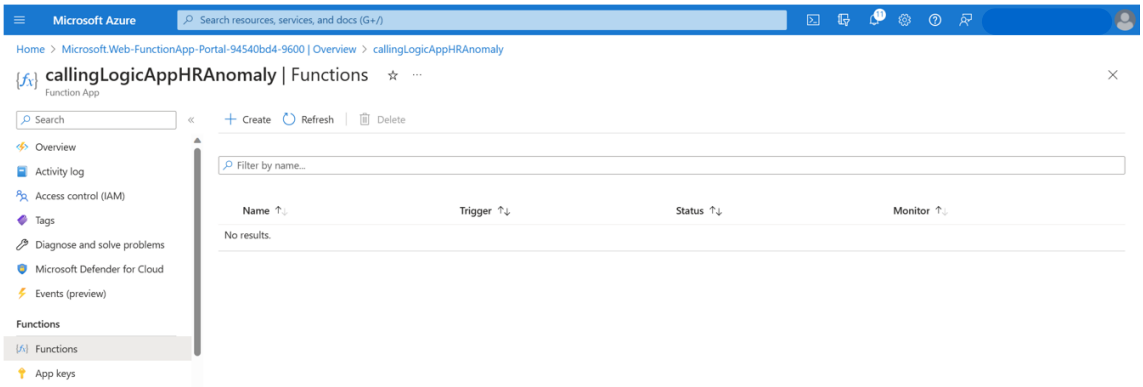

Figura 100: Creación de una Function 1

Para crear la función se ha seleccionado como *trigger HTTP trigger* (desencadenador HTTP), como se registra en la figura 101.

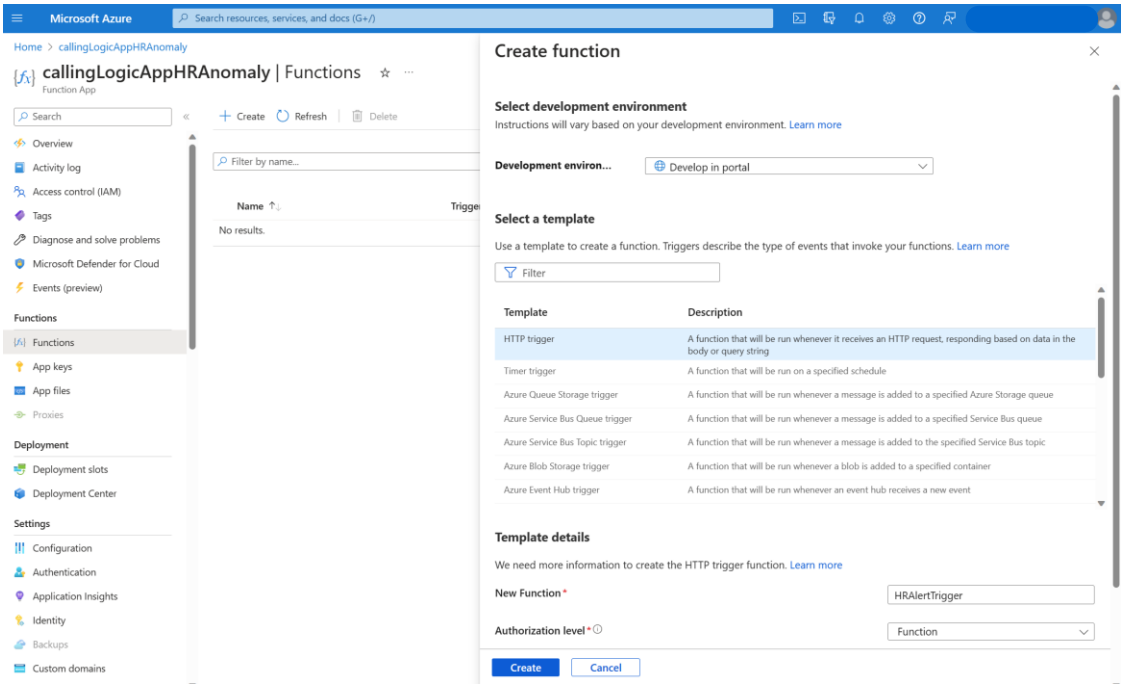

Figura 101: Creación de una Function 2

Con respecto al nivel de autorización existen tres opciones posibles:

- *Anonymous:* No se requiere clave de API para invocar la función. Cualquiera que tenga la URL de la función puede invocarla. Este nivel es menos seguro, pero puede ser útil en casos en los que la seguridad no es una preocupación importante.

- *Function*: Se requiere una clave de API específica de la función para invocarla. Este nivel proporciona un control de acceso básico y es adecuado para la mayoría de los casos de uso.

- *Admin:* Se requiere una clave de API de host para invocar la función. Este nivel proporciona un control de acceso más estricto, ya que la clave de API de host es compartida por todas las funciones en la misma aplicación de funciones.

En el contexto de la solución diseñada en este TFM, dado que la función se utilizará para activar la Logic App, se va a utilizar el nivel de autorización *Function*. Esto proporciona un control de acceso básico y asegura que solo las entidades autorizadas puedan invocar la función.

Una vez creada la *function* desde la pestaña Code + Test es posible especificar el código pertinente para la función, como se ve en la figura 102.

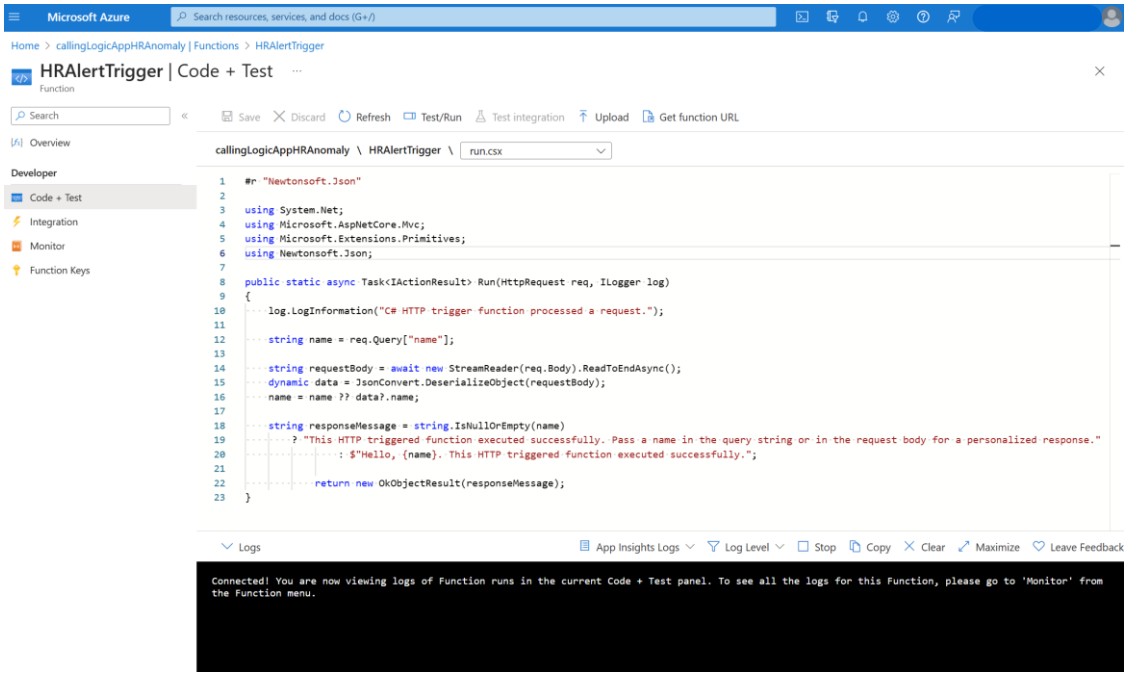

Figura 102: Código Function 1

Partiendo de este ejemplo se va a modificar el código de forma que cada vez que se produce una llamada a la función, esta invoque a la Logic App que creada en el paso inmediatamente anterior. Para ello, en primer lugar se debe obtener la URL de la Logic App. Desplegando la caja de *When a HTTP request is received* dentro de la pestaña *Logic App* Designer dentro de la Logic App, es posible copiar la URL de la Logic App, como se muestra en la figura 103.

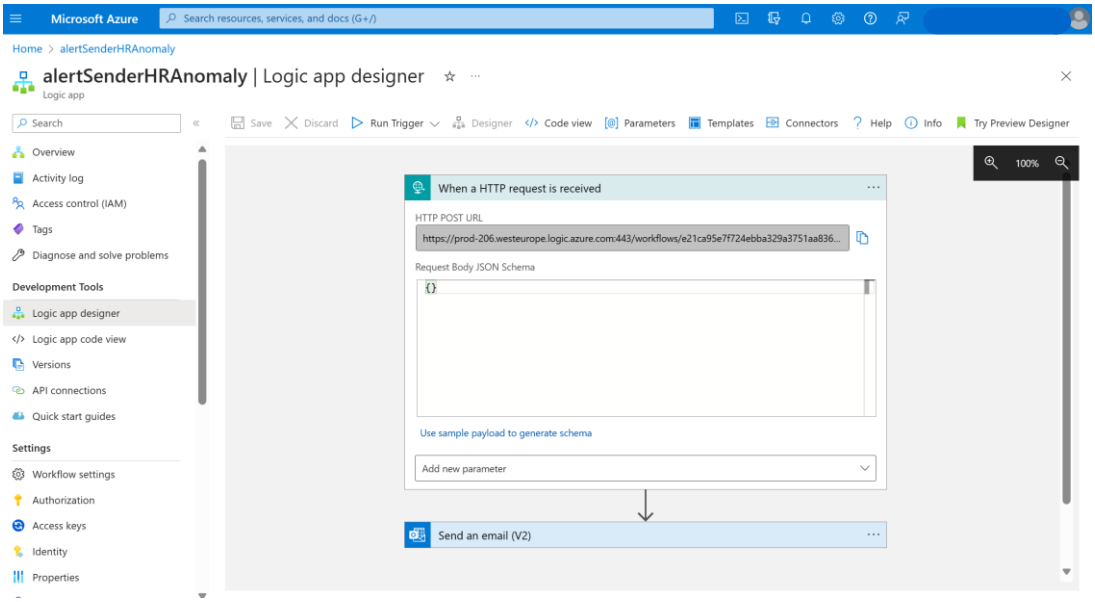

Figura 103: URL Logic App

El código final para invocar a la Logic App puede encontrarse en el Aznexo 2 de la memoria. A continuación, se selecciona en el menú superior Test/Run de la figura 104 con el objetivo de comprobar que la solicitud se lleva a cabo satisfactoriamente.

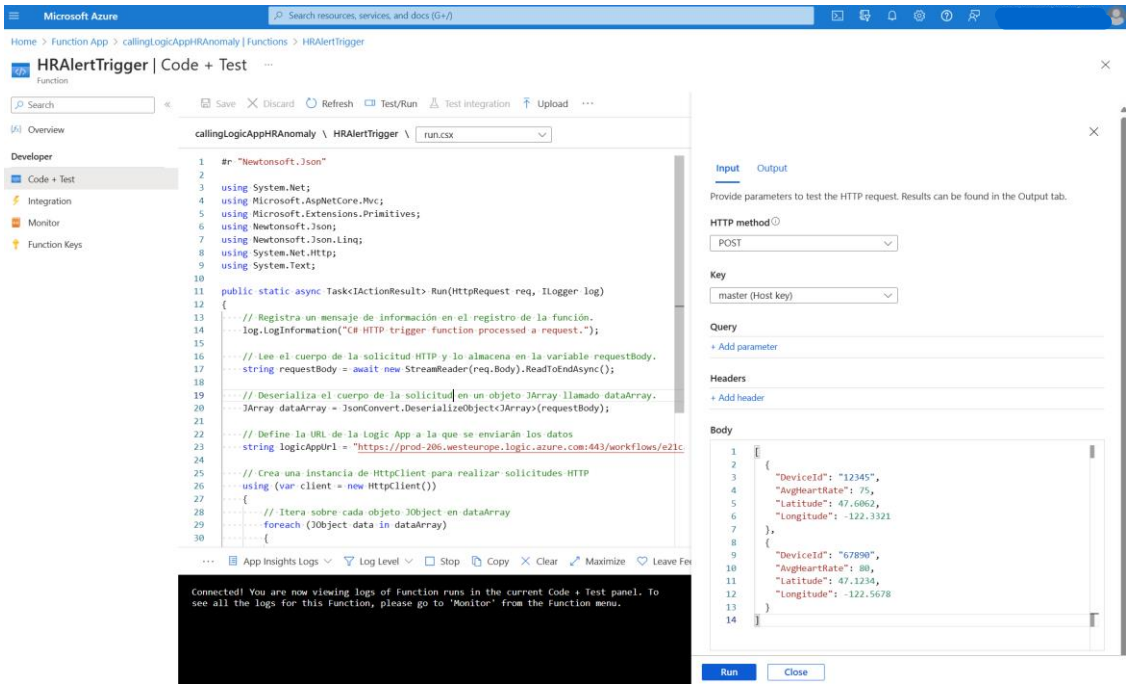

Figura 104: Comprobación del funcionamiento de la Function 1

En la figura 105 se demuestra como la función se ha ejecutado sin ningún error.

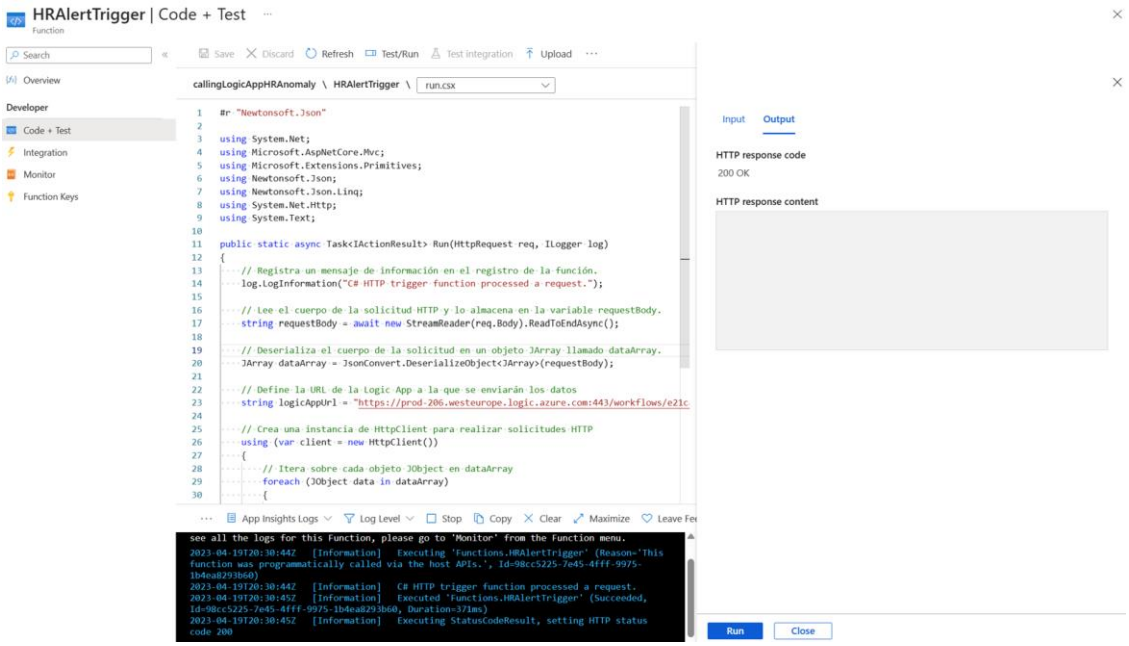

Figura 105: Comrpobación del funcionamiento de la Function 2

Por otro lado, en la figura 106 se observa como si se verifica la bandeja de entrada del correo electrónico es posible comprobar que el email con la alerta ha sido generado correctamente.

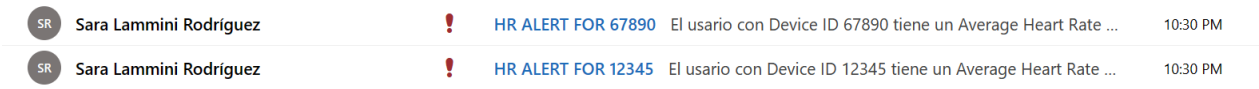

Figura 106: Comprobación del funcionamiento de la Function 3

3.2.9. Azure Stream Analytics - HeartRateAnomalyDetection

En este apartado se describen los pasos seguidos para la implementación del job HeartRateAnomaly Detection, job encargado de identificar en tiempo real cuando se registran pulsaciones anómalas y llamar a la *function* responsable de desencadenar la Logic App. Para ello, en primer lugar, se debe crear un nuevo Azure Stream Analytics job como se muestra en la figura 107. Los campos a rellenar son: la suscripción, el grupo de recursos, el nombre la instancia, la región, si se desea que el *hosting environment* se *cloud* o *edge* y el número de streaming units, es decir, el número de recursos de computación destinados a la ejecución del job.

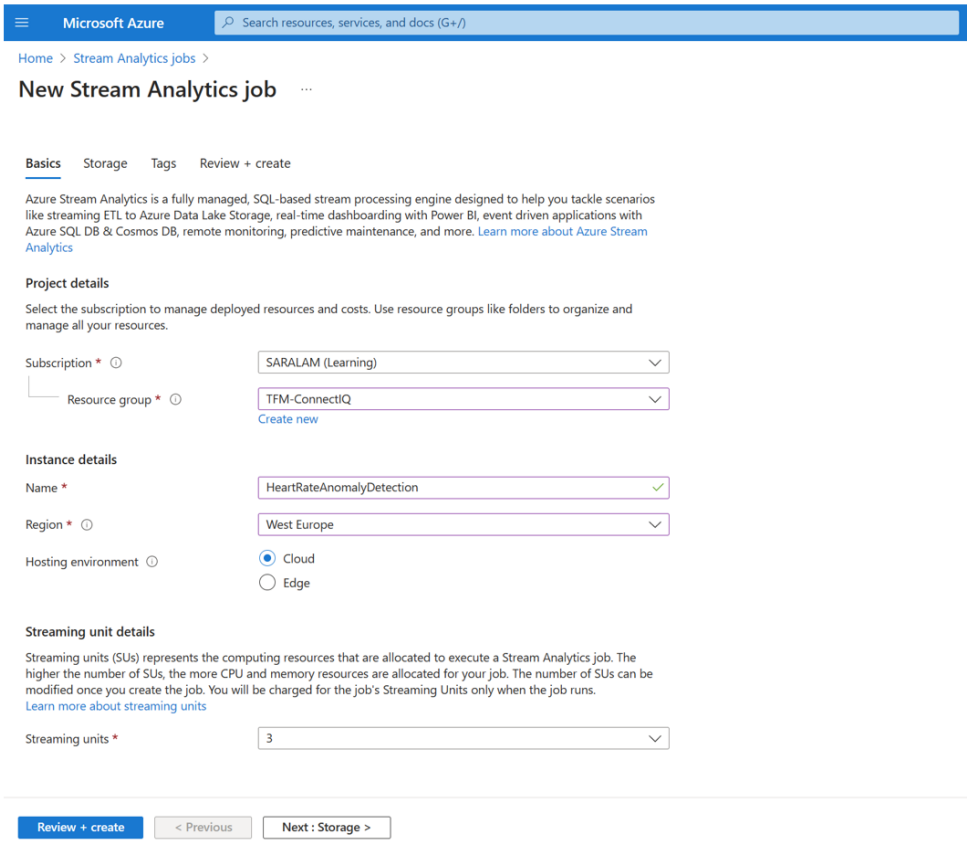

Figura 107: Creación de un nuevo Azure Stream Analytics Job

Una vez ha finalizada la creación del Azure Stream Analytics job, es momento de configurar la entrada o *input*. Para ello, como se refleja en la figura 108, es necesario ir a la pestaña de *input* e indicar el Event Hub.

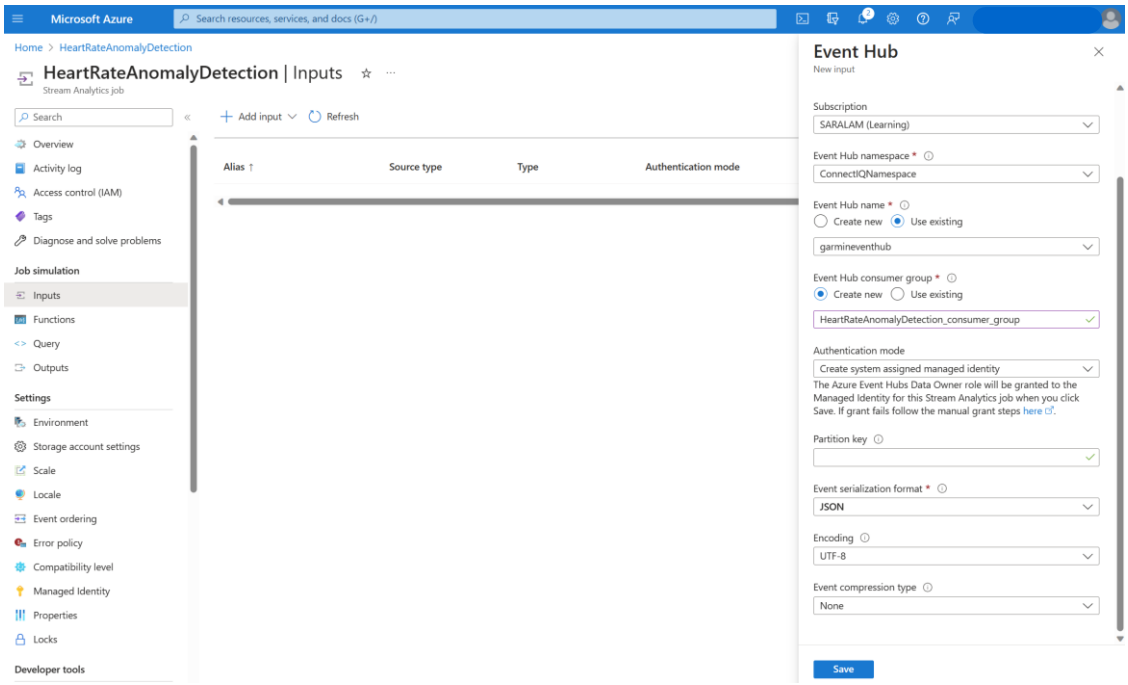

Figura 108: Configuración del Input para el Azure Stream Analytics Job

A continuación, es momento de configurar la salida u *output*. Para ello, desde la pestaña de *output* y se debe especificar la Azure *Function* creada previamente, tal y como puede observarse en la figura 109.

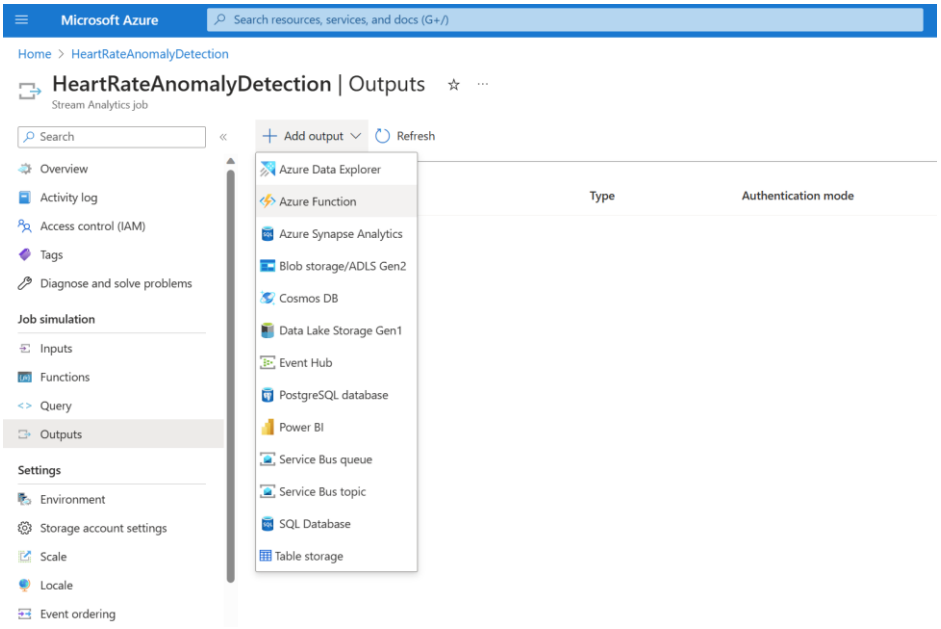

Figura 109: Configuración del output para el Azure Stream Analytics Job

Como a la hora de crear nuestra *function* se ha especificado como *authorization level function* para una mayor seguridad, se requiere una clave de función válida para realizar la autenticación al llamar a la función. Por esta razón estamos en el apartado de *Function Keys* se debe seleccionar *default,* como puede observarse en la figura 110.

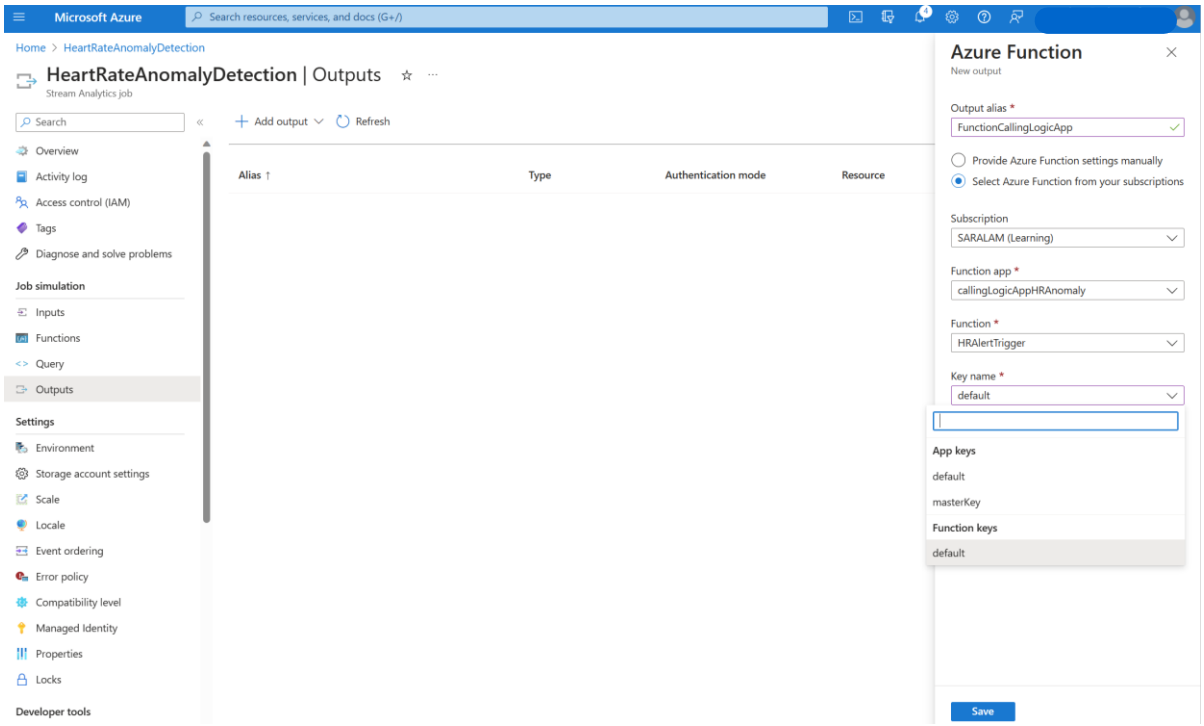

Figura 110: Configuración de la autenticación contra la Function 1

Desde la *function* podemos obtenerse el valor de esta *function* key, tal y como se muestra en la figura 111.

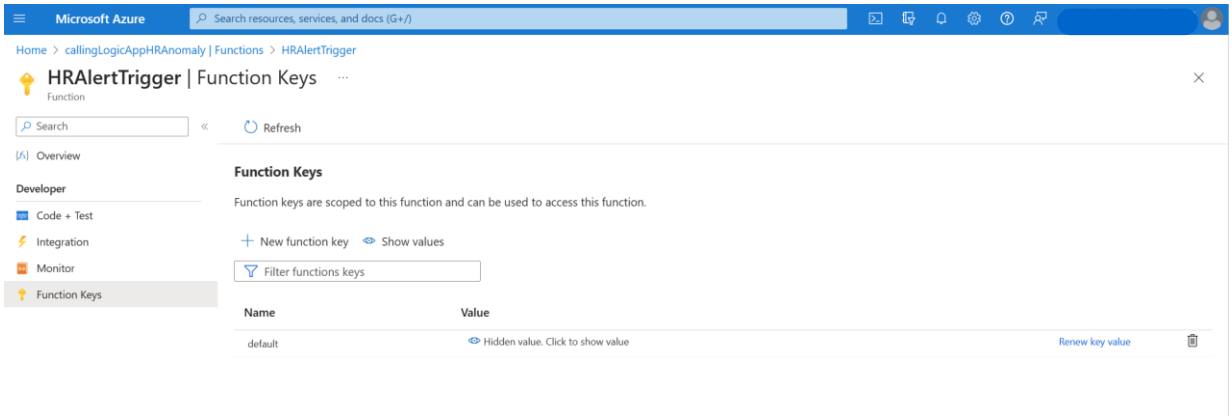

Figura 111: Function Key

Volviendo al output del Azure Stream Analytics Job, si se selecciona Test, es posible comprobar que la conexión se realiza correctamente. El job llama a la *function* que a su vez invoca a la Logic App y, como se verifica en la figura 112, se recibe el email con la alerta en la bandeja de entrada.

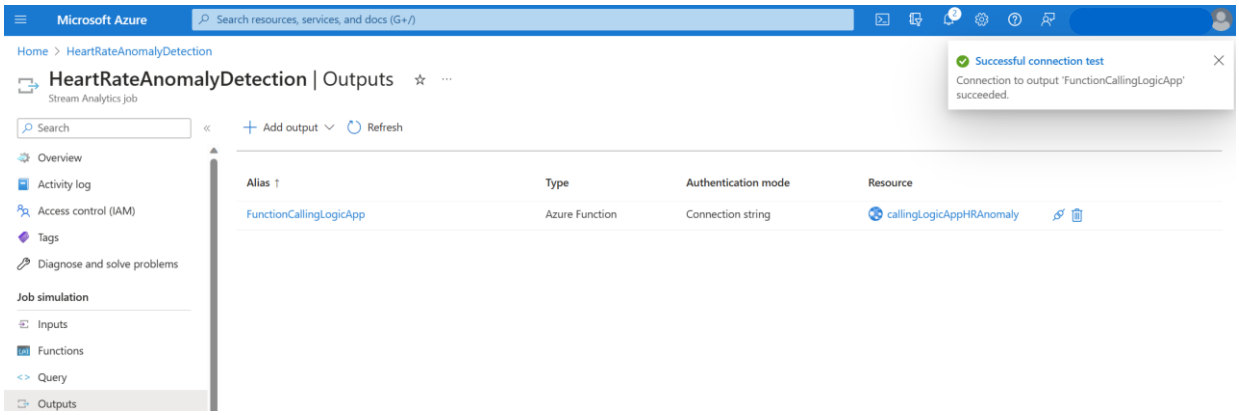

Figura 112: Test de conexión satisfactorio entre el Azure Stream Analytics Job y la Auzre Function

Ahora, en la pestaña de Query es necesario configurar la consulta de manera que el trabajo pueda identificar cuándo un usuario en particular supera una frecuencia cardíaca de 200 pulsaciones por minuto dentro de una ventana de tiempo de 3 segundos. En la figura 113 se presenta la consulta elaborada para este fin. Esta consulta procesa los datos de entrada provenientes de myEventHub. Agrupa los datos en ventanas de tiempo de 3 segundos (TumblingWindow(second, 3)) y calcula el promedio de heartRate (AVG(heartRate)), la latitud máxima (MAX(latitude)) y la longitud máxima (MAX(longitude)) para cada usuario (userId) en cada ventana de tiempo. La consulta filtra los resultados para incluir solo aquellos registros donde el promedio de heartRate en esos 3 segundos supera 200 pulsaciones por minuto (HAVING AVG(heartRate) > 200).

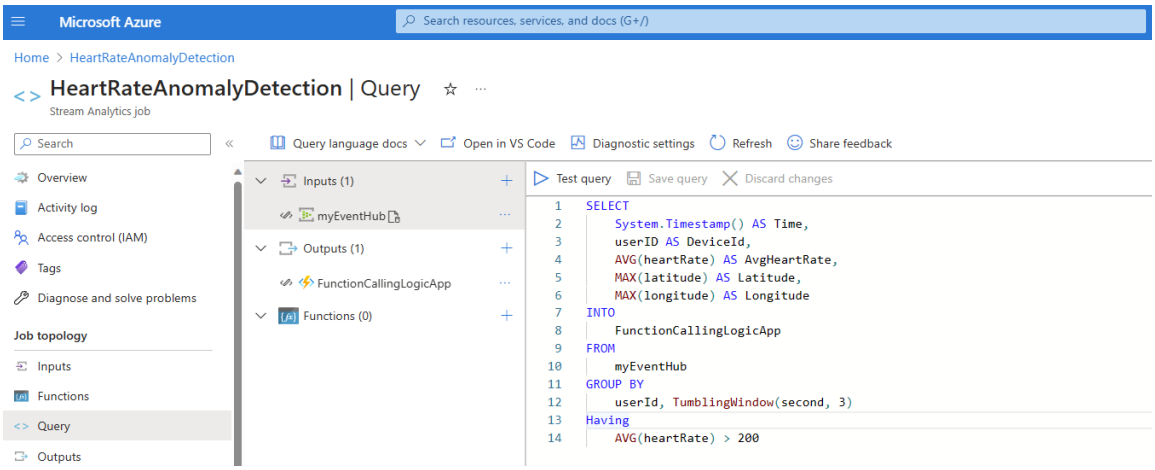

Figura 113: Configuración de la Query para el Azure Stream Analytics

4. Resultados

Este capítulo tiene como objetivo presentar los resultados obtenidos a raíz de la solución diseñada e implementada en el capítulo 3 de la presente memoria para una plataforma de monitoreo de corredores en tiempo real con relojes inteligentes y Azure.

Tal como se discutió en la sección 3.1, dedicada al diseño de la solución, la plataforma desarrollada ofrece tres funcionalidades principales. La primera funcionalidad es la de crear un registro histórico de los datos enviados por los dispositivos conectados para poder realizar análisis y auditorías posteriores. La segunda funcionalidad se centra en la visualización en tiempo real de los datos enviados por los relojes inteligentes conectados, a través de un panel de monitorización construido con Power BI. Por último, la tercera funcionalidad se encarga de generar alertas cuando alguno de los relojes registra pulsaciones anómalas.

Este capítulo se divide en cuatro subcapítulos. En el primero, se presentará la disposición final de los recursos en el portal de Azure. El segundo mostrará el registro histórico generado. El tercer subcapítulo se centrará en discutir los resultados obtenidos en relación al panel de monitorización. Por último, el cuarto subcapítulo se enfocará en los resultados relacionados con la generación de alertas en tiempo real.

4.1. Disposición final de los recursos

En esta sección se presentará la disposición final de los recursos implementados en Azure para el correcto funcionamiento de la plataforma de monitoreo de corredores en tiempo real, desarrollada en el presente TFM.

Conforme se mencionó en el capítulo de implementación, todos los recursos han sido desplegados en un grupo de recursos denominado TFM-ConnectIQ. En la figura 114 se recogen los recursos que finalmente forman parte de TFM-ConnectIQ: el espacio de nombres de Event Hub, la *storage account*, los dos Stram Analytics Jobs, la Function App, la Logic App y una conexión API creada automáticamente cuando se realiza la conexión de la Logic App con Office 365 para el envío de correos electrónicos.

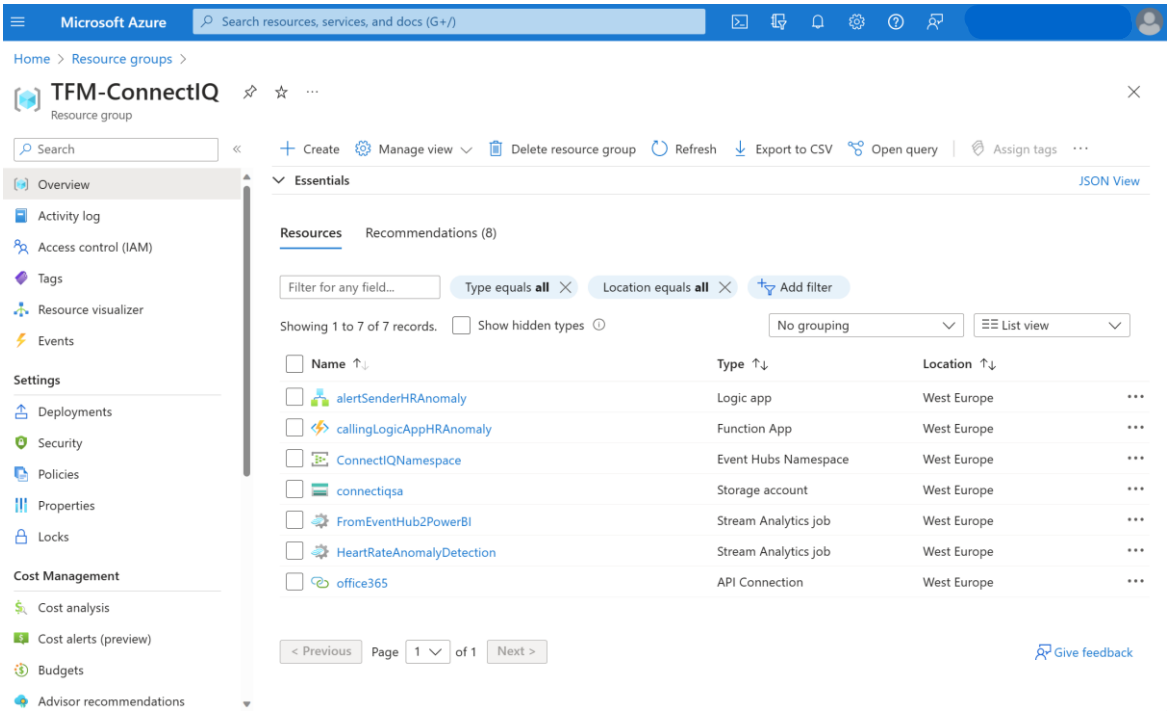

Figura 114: Vista de los recursos que forman parte del grupo de recursos TFM – ConnectIQ

Si en el menú de la izquierda de la figura 114 se hace clic en *Resource Visualizer* se obtiene la vista presentada en la figura 115 donde puede apreciarse todos los recursos que conforman la solución, así como las dependencias entre estos. Por ejemplo, puede que verse como la *API Connection* guarda una dependencia con la Logic App.

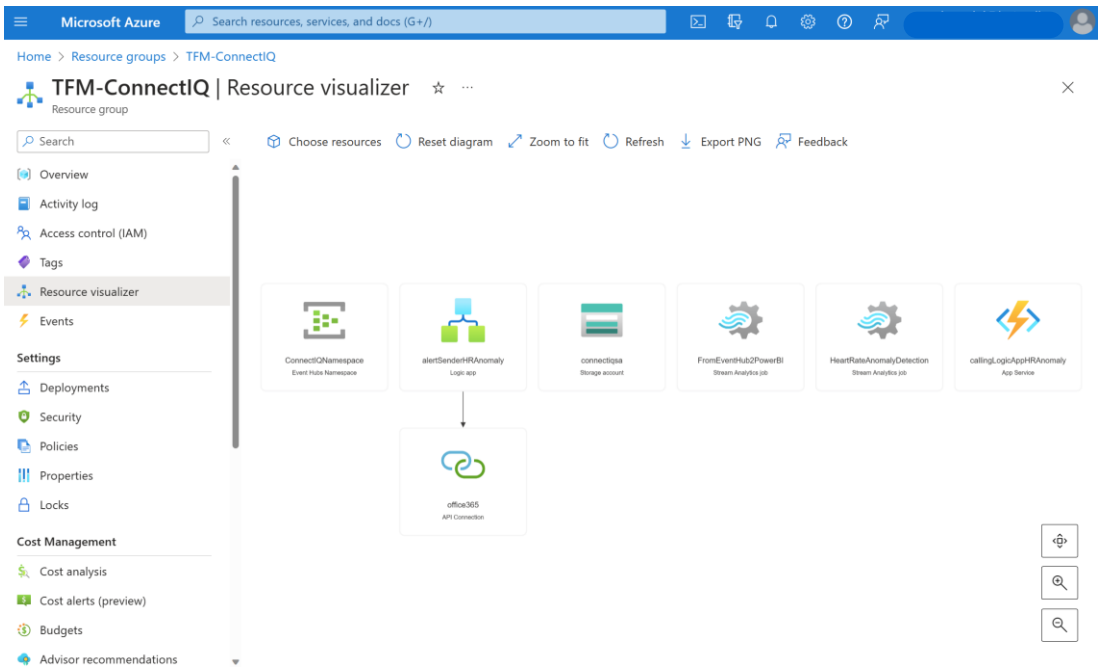

Figura 115: Vista de las dependencias entre los recursos de TFM-Connect IQ

Otra vista interesante de cara a entender cómo se encuentran desplegados los componentes de la solución se obtiene al volver a la pestaña de *Overview,* seleccionando *Summary View* en lugar de *List View*. Como se aprecia en la figura 116, existen 3 opciones a la hora de generar las *summary views*: por ubicación, por tipo de recurso o por *Edge*. En la figura 116 se ha seleccionado por ubicación para mostrar cómo los 7 recursos han sido despegados en la región *West Europe*.

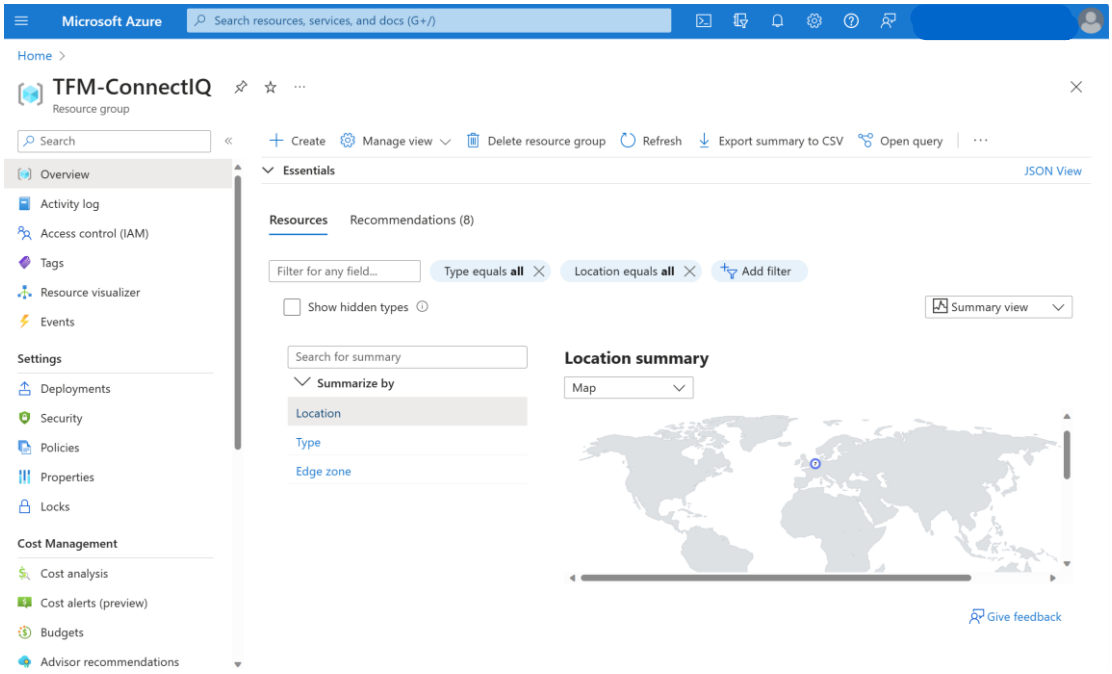

Figura 116: Ubicación de los recursos del grupo de recursos TFM-ConnectIQ

En la figura 117 se muestra este mapa con un nivel de zoom superior para poder apreciar con un mayor grado de detalle como efectivamente los 7 recursos de los que consta la aplicación han sido desplegados en la región *West Europe*.

Figura 117: Ubicación de los recursos del grupo de recursos TFM-ConnectIQ en mayor tamaño

Si se selecciona *summarize by type* se obtiene la vista mostrada en la figura 118 con el número de recursos de cada tipo. En esta figura, se puede observar como la solución se compone de 2 instancias de Azure Stream Analytics job, 1 de *API Connection,* 1 de espacio de nombres de Event Hub, 1 de Function App, 1 de Logic App y 1 de *storage account.*

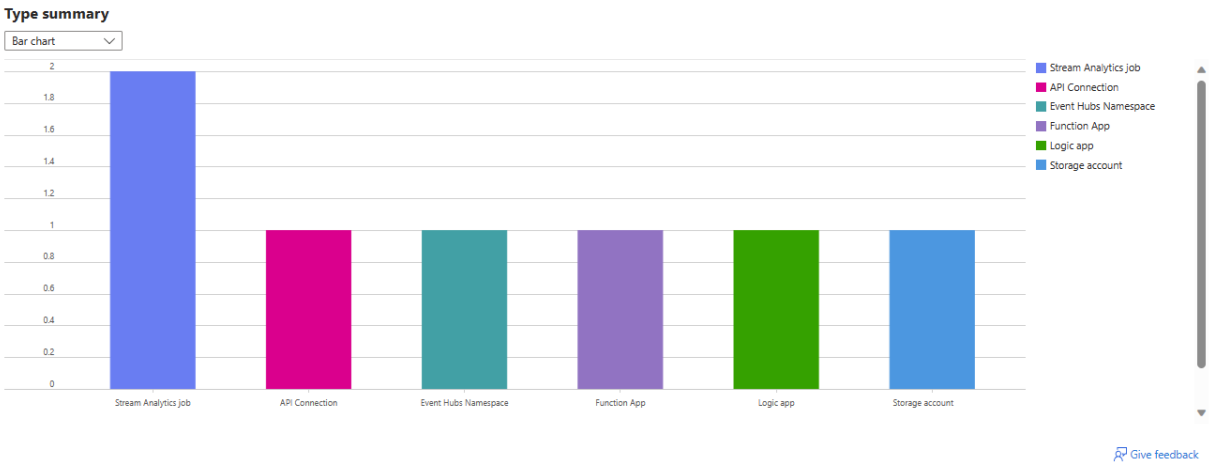

Figura 118: Número de cada tipo de recurso del grupo de recursos TFM-ConnectIQ

La última vista a destacar es la que se obtiene si se selecciona en el menú de la izquierda *Cost análisis*. Como se muestra en la figura 118, esta vista proporciona información de los costes en los que ha incurrido la solución en el mes de mayo. Estos costes se recogen por servicio, por ubicación y por recurso. Puede observarse como el coste total de la solución en el mes de mayo fue de $105,45, siendo los recursos más costosos los dos Stream Analytics con un coste de $36,85 para fromeventhub2powerbi y $35,77 para heartrateanomalydetection, seguidos de el espacio de nombre del event hub que incurrió en unos costes de $32.62.

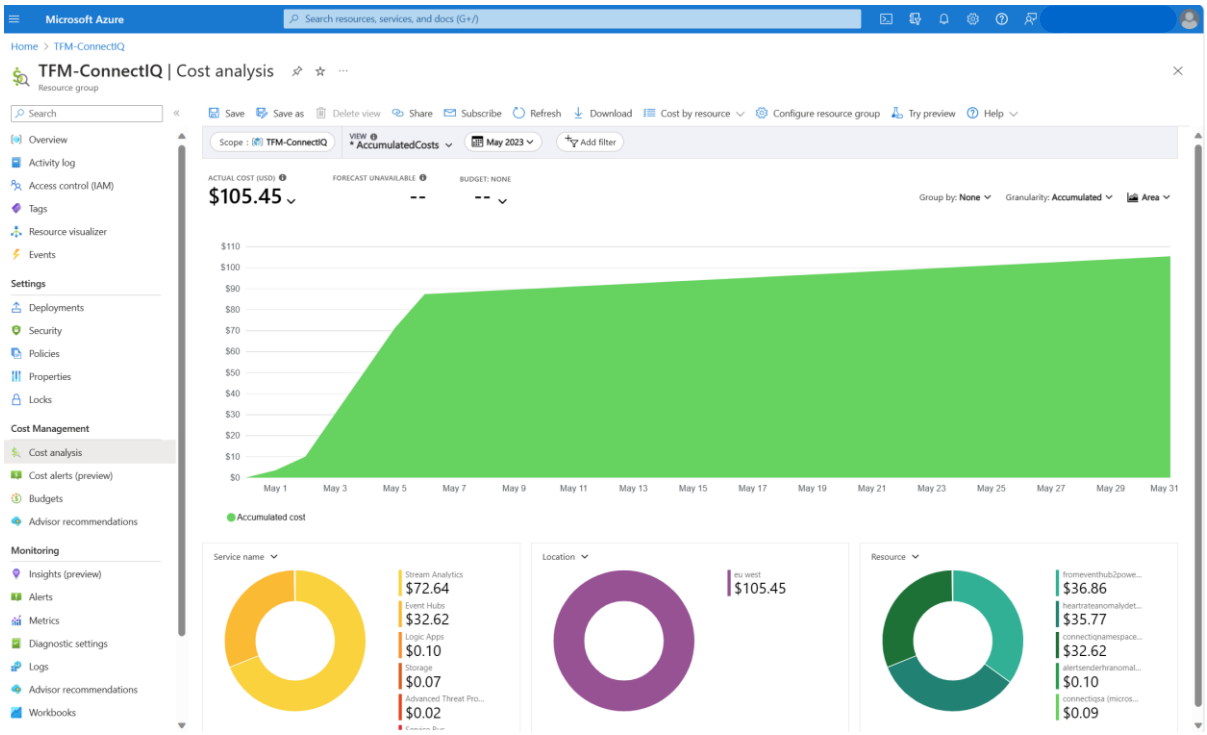

Figura 119: Análisis de costes para el grupo de recursos.

4.2. Registro histórico

Esta sección aborda el resultado del registro histórico creado a partir de los datos enviados por los relojes inteligentes conectados a la plataforma de monitorización de corredores en tiempo real.

Como se muestra en la figura 120, los paquetes recibidos en el Azure Event Hub son almacenados en el contenedor garmindatacontainer dentro de la *storage account* connectiqsa siguiendo el formato: {Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/ {Day}/{Hour}/{Minute}/{Second}.

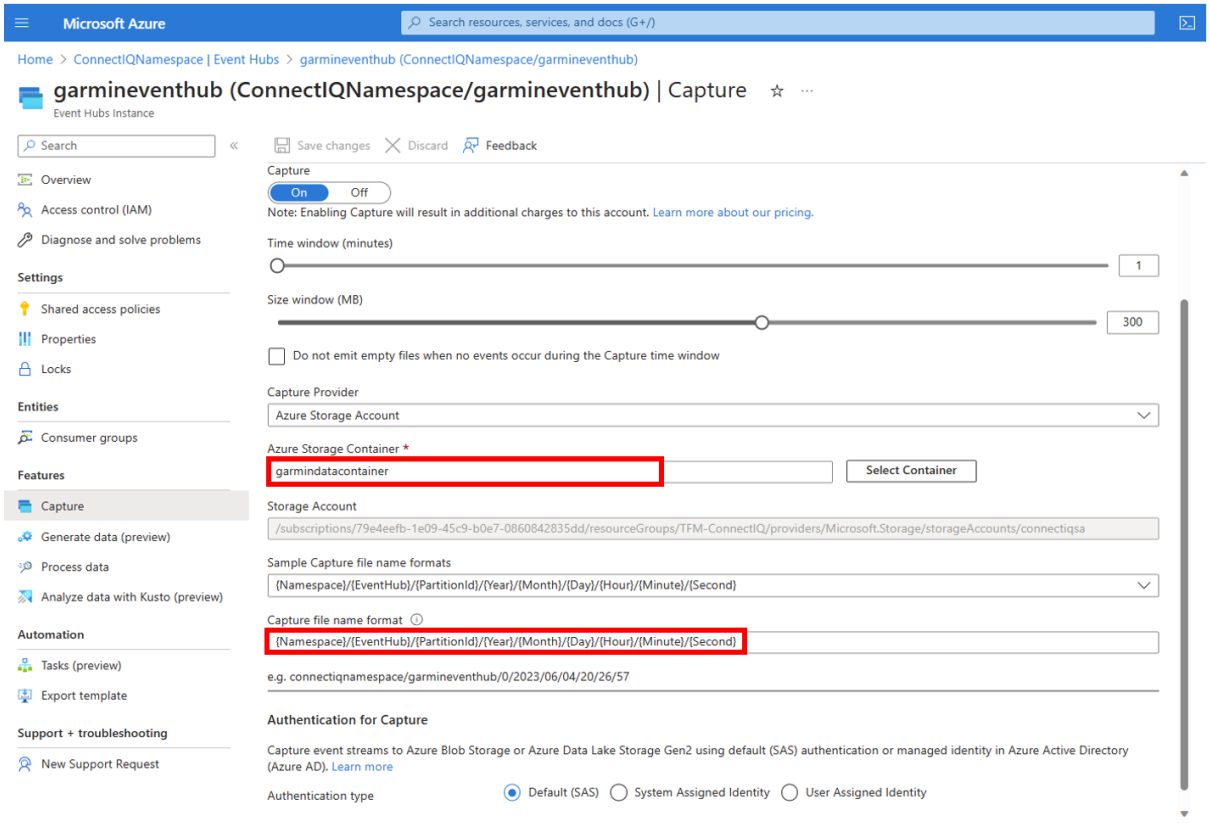

Figura 120: Proceso de guardado de los paquetes recividos en Azure Event Hub

Si se va al contenedor garmindatacontainer como se muestra en la figura 121, puede observarse como efectivamente los datos enviados se están guardando con el formato especificado:

{connectiqnamespace}/{garmineventhub}/{0}/{2023}/{06}/ {04}/{20}/{03}/{36}.arvo

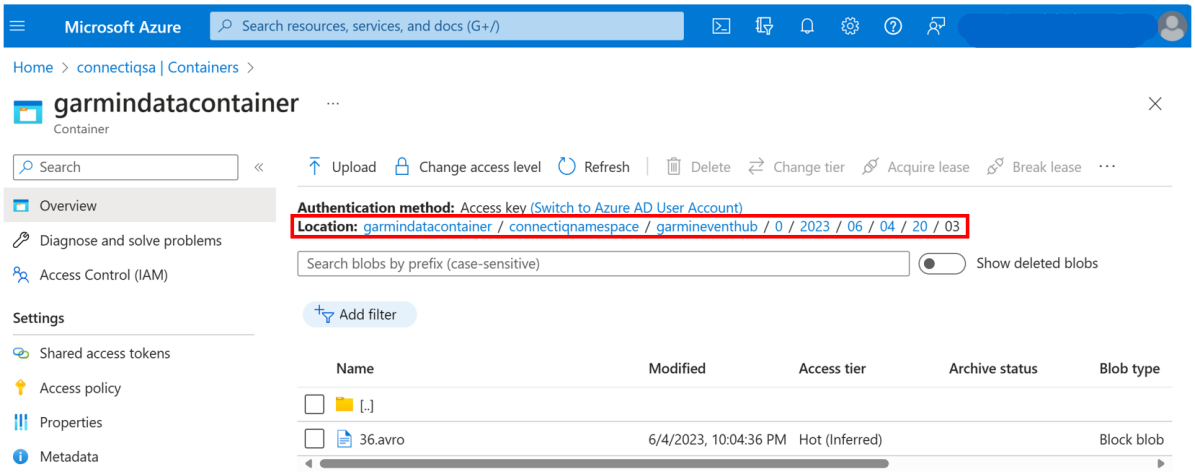

Figura 121: Vista de los paquetes almacenados automáticamente en el contenedor garmindatacontainer desde el portal de Azure

Otra forma de comprobar que los paquetes se están guardando correctamente es desde la aplicación Microsoft Azure Storage Explorer, tal y como se presenta en la figura 122.

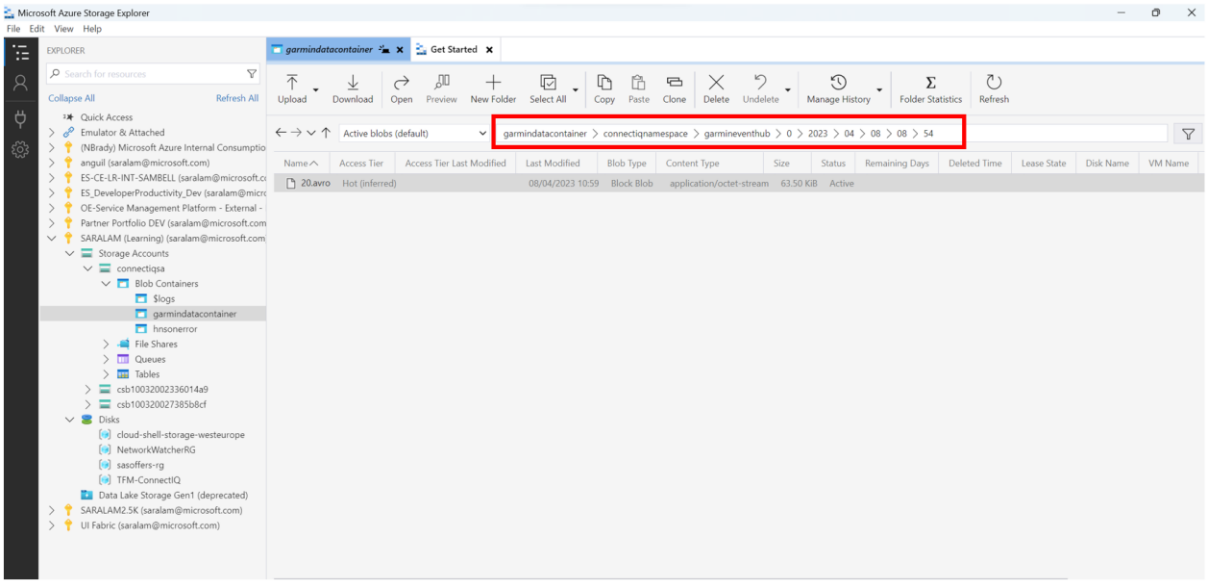

Figura 122: Vista de los paquetes almacenados automáticamente en el contenedor garmindatacontainer desde el Microsoft Azure Storage Explorer

Desde el portal de Azure es posible descargar uno de estos ficheros para su posterior examinación haciendo seleccionando el fichero y haciendo clic en *Download* tal y como se muestra en la figura 123.

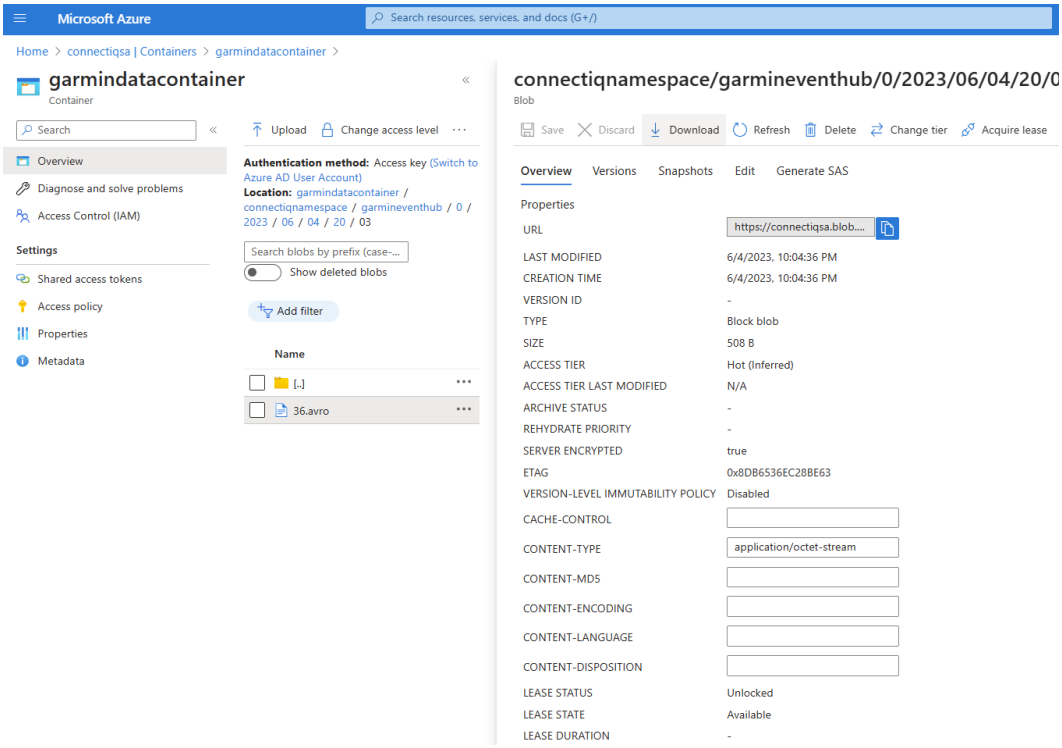

Figura 123: Descarga de los paquetes almacenados en el contenedor

Si se abre el fichero desde un bloc de notas se comprueba como los datos se están almacenando correctamente siguiendo el formato siguiente:

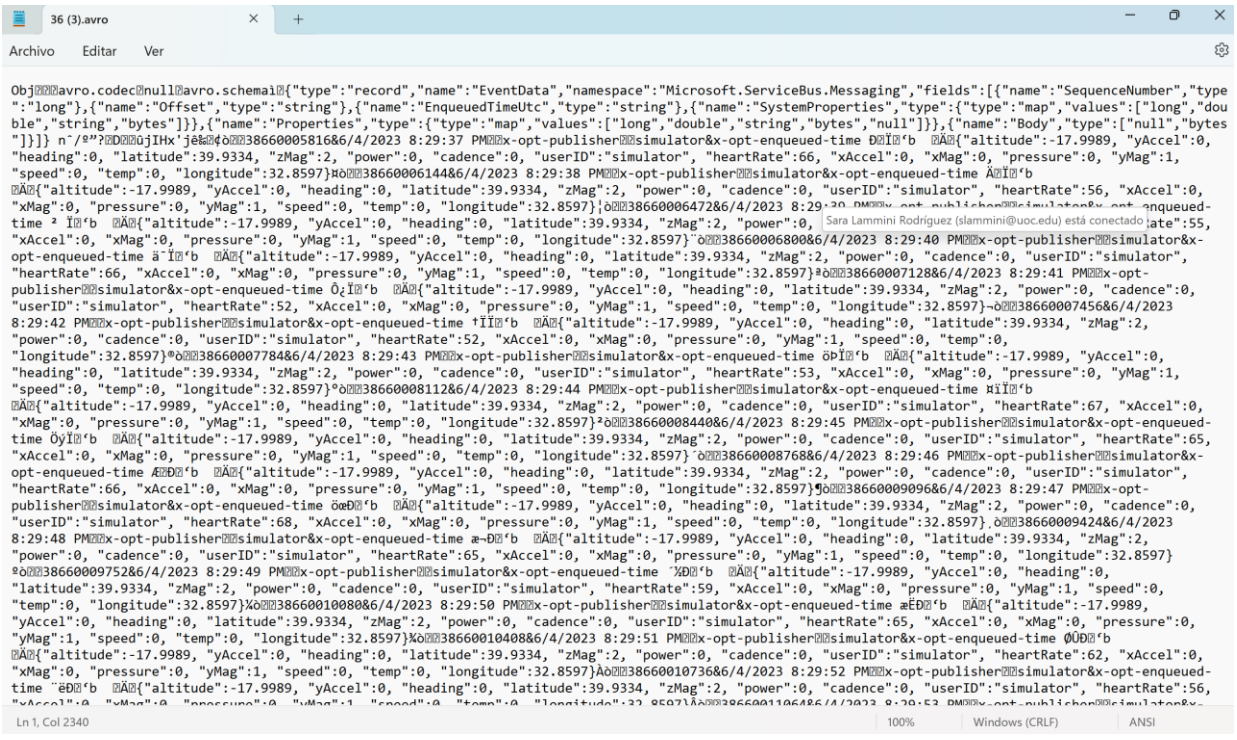

Figura 124: Datos almacenados en el contenedor

4.3. Panel de monitorización el tiempo real

En esta tercera parte del capítulo, se examinan los resultados obtenidos al monitorear a los atletas en tiempo real. La Figura 125 muestra el panel de monitorización final desarrollado en Power BI. En dicha figura, se puede observar cómo el panel recopila información de diversa índole. A continuación, se detalla la información presentada en el panel. Para facilitar la relación entre las explicaciones y las secciones del panel, se ha asignado un número en rojo a cada sección.

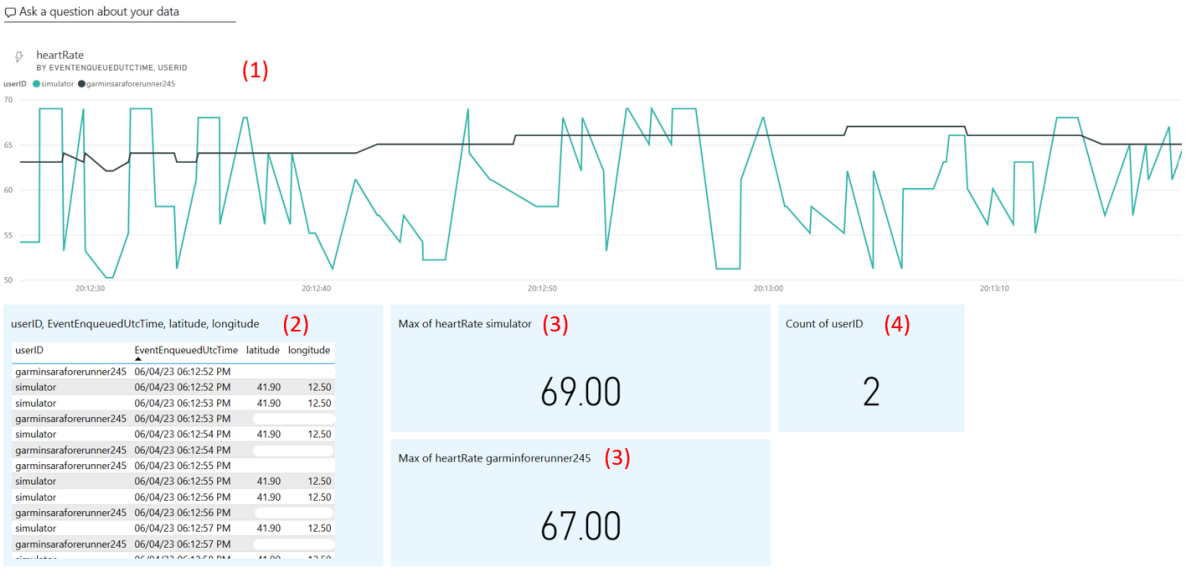

Figura 125: panel de monitorización de corredores en tiempo real final desarrollado en Power BI

(1) Este gráfico resume la evolución de la frecuencia cardiaca a lo largo del tiempo registrada por cada uno de los dispositivos conectados a la plataforma de monitorización a lo largo del tiempo en una ventana de 1 minuto.

En la Figura 126, se muestra la vista obtenida cuando dos dispositivos están conectados simultáneamente a la plataforma de monitoreo. Se puede apreciar cómo la evolución de las pulsaciones a lo largo del tiempo se representa en verde para el simulador, mientras que en negro se muestra la correspondiente al reloj real.

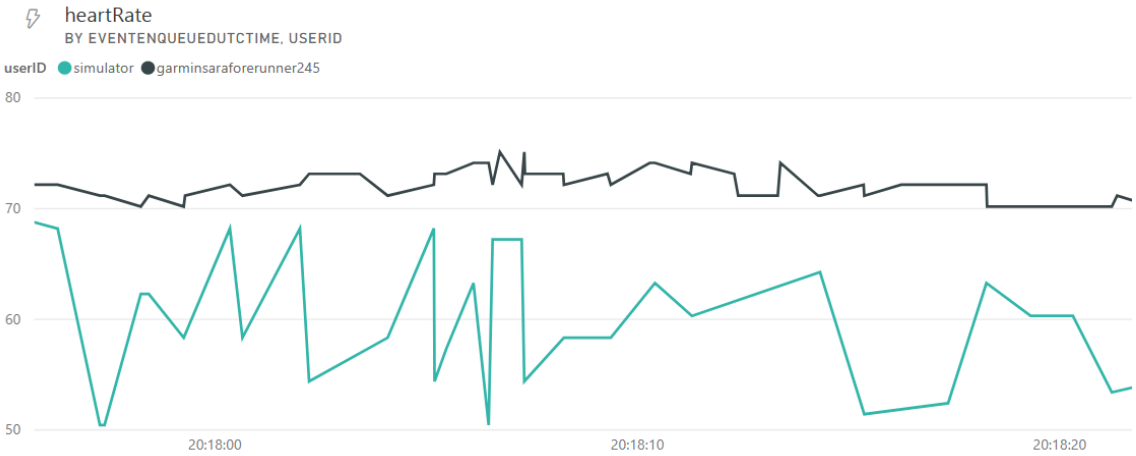

Figura 126: Evolución de la frecuencia cardiaca a lo largo del tiempo cuando hay dos dispositivos conectados a la plataforma

Si únicamente se encuentra el reloj conectado a la plataforma de monitorización la gráfica presenta el aspecto de la figura 127.

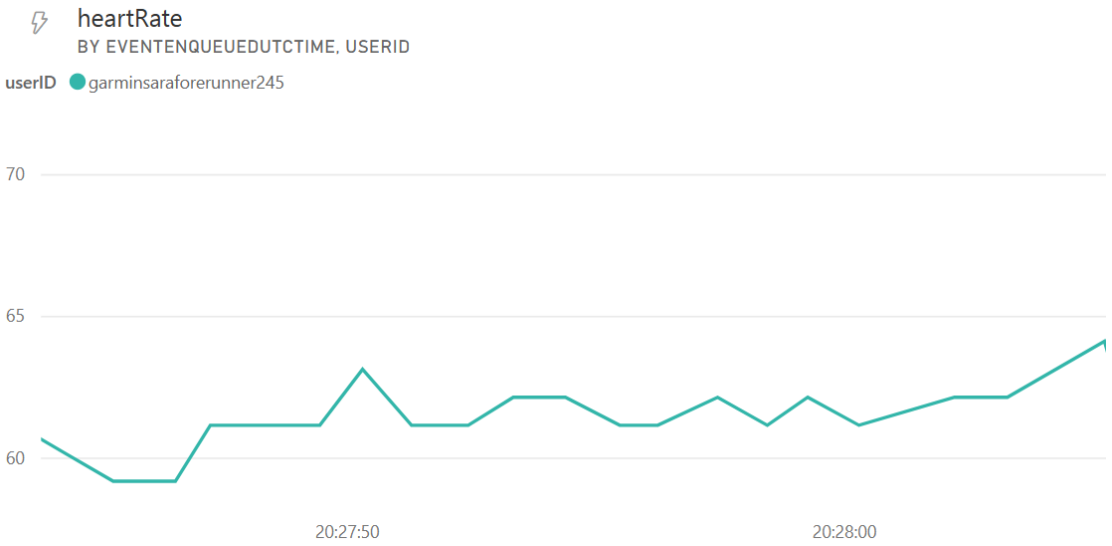

Figura 127: Evolución de la frecuencia cardiaca a lo largo del tiempo cuando únicamente el reloj real se encuentra conectado a la plataforma

Si por el contrario solo se encuentra conectado el simulador a la plataforma de monitorización, la gráfica es la que se recoge en las figuras 128 y 129. En la figura 128 se recoge la gráfica cuando las pulsaciones simuladas se encuentran comprendidas en el intervalo 50-70 pulsaciones por minutos mientras que en la figura 129 las pulsaciones son de entre 180 y 250 pulsaciones por minuto.

Figura 128: Evolución de la frecuencia cardiaca a lo largo del tiempo cuando únicamente el reloj real se encuentra conectado el simulador a la plataforma. Valores de frecuencia cardiaca comprendidos entre 50 y 70 pulsaciones por minuto.

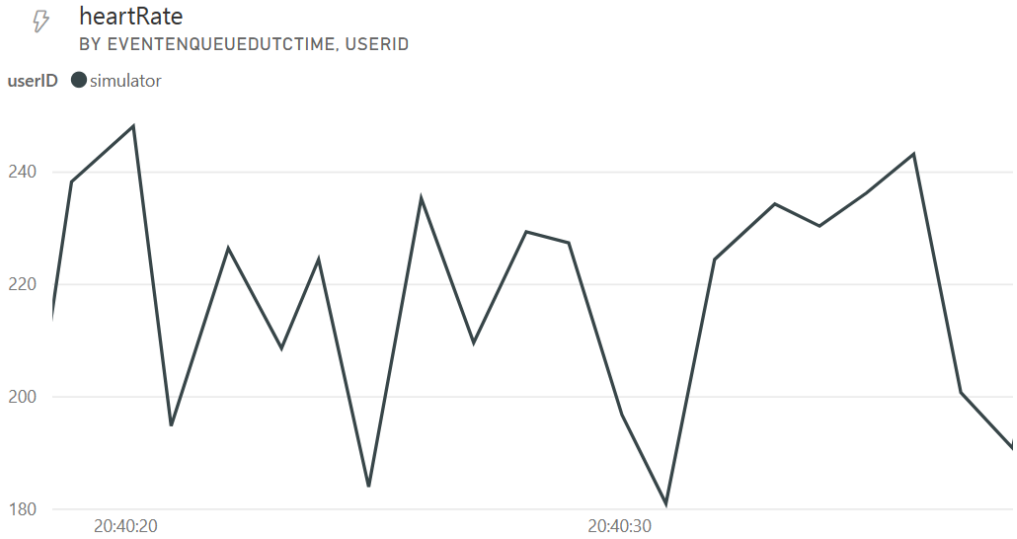

Figura 129: Evolución de la frecuencia cardiaca a lo largo del tiempo cuando únicamente el reloj real se encuentra conectado el simulador a la plataforma. Valores de frecuencia cardiaca comprendidos entre 180 y 250 pulsaciones por minuto.

Es importante mencionar que la frecuencia cardiaca enviada al Event Hub es el resultado del fragmento de código mostrado en la Figura 130. En dicho código, se puede observar que, en el caso de contar con un sensor real (es decir, para el dispositivo Garmin Forerunner 245), la información enviada corresponde a la registrada por el sensor del dispositivo. Por otro lado, en el caso de la simulación, los datos se generan siguiendo la expresión hR=Math.rand()%20+50 si se desea generar valores entre 50 y 70 pulsaciones por minuto (figura 128), o hR=Math.rand()%70+180 (figura 129) si se desea que las pulsaciones generadas estén comprendidas entre 180 y 250. El código completo se encuentra en el Anexo 1 de esta memoria.

```
141         //Heartrate
142         if (sensorInfo has :heartRate && sensorInfo.heartRate != null) {
143 
144             hR = sensorInfo.heartRate;
145         }
146         else {
147             // Generamos pulsaciones de forma aleartorias entre 180 y 250
148             //hR=Math.rand()%70+180;
149 
150             // Generamos pulsaciones de forma aleartorias entre 50 y 70
151             hR=Math.rand()%20+50;
152         }
```

Figura 130: Código para el registro o simulación de datos de frecuencia cardíaca

(2) Esta tabla recoge la información en tiempo real de las coordenadas de los usuarios que portan los dispositivos conectados a la plataforma de monitorización continua. En la figura 131 se encuentra una visión ampliada de esta tabla.

userID, EventEnqueuedUtcTime, latitude, longitude

userID	EventEnqueuedUtcTime	latitude	longitude
garminsaraforerunner245	06/04/23 06:46:40 PM		
simulator	06/04/23 06:46:40 PM	41.90	12.50
garminsaraforerunner245	06/04/23 06:46:41 PM		
simulator	06/04/23 06:46:41 PM	41.90	12.50
garminsaraforerunner245	06/04/23 06:46:42 PM		
simulator	06/04/23 06:46:42 PM	41.90	12.50
garminsaraforerunner245	06/04/23 06:46:43 PM		
simulator	06/04/23 06:46:43 PM	41.90	12.50
garminsaraforerunner245	06/04/23 06:46:44 PM		
simulator	06/04/23 06:46:44 PM	41.90	12.50
garminsaraforerunner245	06/04/23 06:46:45 PM		
simulator	06/04/23 06:46:45 PM	41.90	12.50

Figura 131: tabla de coordenadas registradas por los dispositivos en tiempo real

Si se hace clic en la tabla y se selecciona mapa, como se muestra en la figura 132, es posible abrir una vista en la que visualizar la ubicación de los usuarios sobre un mapa como el de la figura 133.

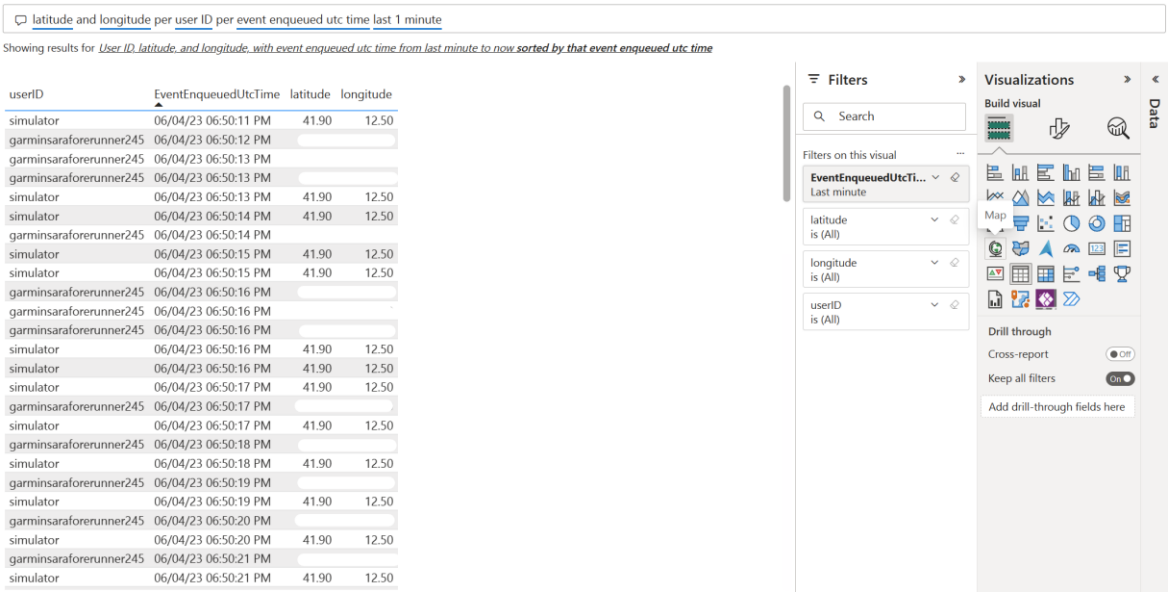

Figura 132: visualización de la ubicación de los corredores en tiempo real sobre un mapa

Tras hacer clic en mapa se abre la vista de la figura 133 en la que se muestra la ubicación de los dispositivos conectados. Puede verse como el reloj real se encuentra en Valladolid mientras que el simulador se encuentra en Roma.

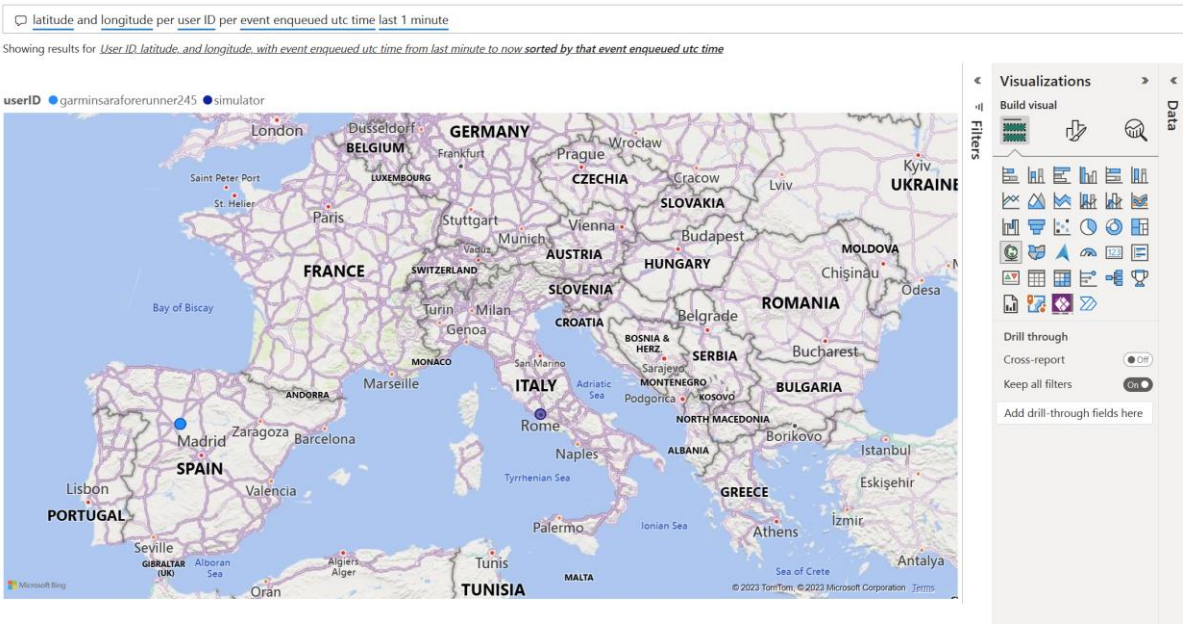

Figura 133: Ubicación de los dispositivos en tiempo real sobre el mapa

Es interesante señalar que las coordenadas enviadas al Event Hub son el resultado del fragmento de código recogido en la figura 134. El código completo puede encontrarse en el anexo 1 de la presente memoria.

```
214         //Position
215         if( positionInfo != null ) {
216             if (positionInfo has :position && positionInfo.position != null) {
217                 var location = positionInfo.position.toDegrees();
218                 latitude = location[0];
219                 longitude = location[1];
220             }
221         }
222         else {
223             latitude = 41.9028;
224             longitude = 12.4964;
225         }
```

Figura 134: Código para el registro o simulación de coordenadas

En este código puede observarse como en el caso de un reloj real, la ubicación se determina en base a las coordenadas registradas por el sensor del dispositivo. Por el contrario, para el caso del simulador, las coordenadas están siendo forzadas a una latitud de 41.9028 y longitud de 12.4964. En la figura 135 se muestra como efectivamente si se coloca el ratón sobre el círculo correspondiente al simulador las coordenadas son exactamente las especificadas en el código.

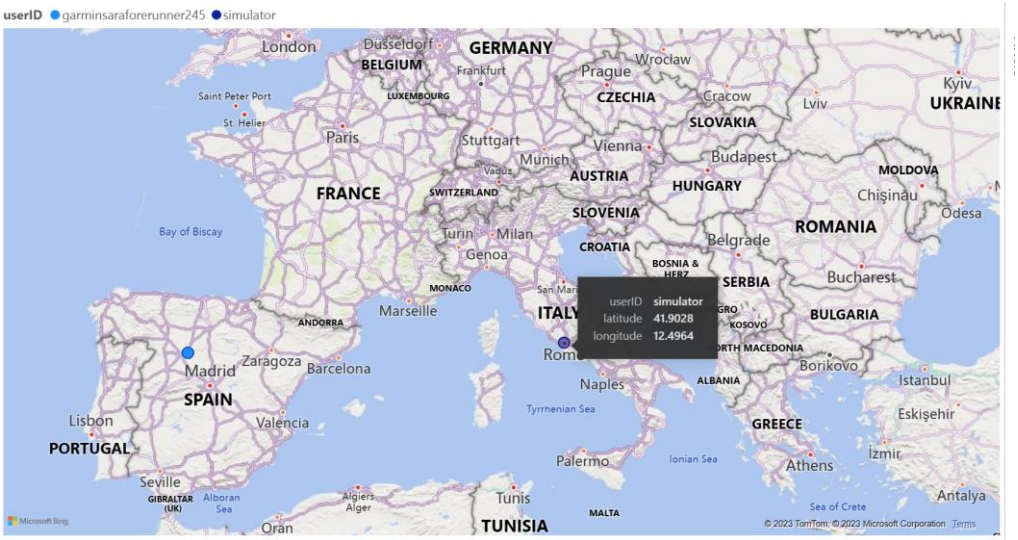

Figura 135: Ubicación del simulador sobre el mapa 1 cuando las coordenadas se sitúan en Italia

Si el código se modifica tal y como se muestra en la figura 136 con las coordenadas de Turquía puede observarse en la figura 137 como la ubicación sobre el mapa de modifica correctamente.

```
214         //Position
215         if( positionInfo != null ) {
216             if (positionInfo has :position && positionInfo.position != null) {
217                 var location = positionInfo.position.toDegrees();
218                 latitude = location[0];
219                 longitude = location[1];
220             }
221         }
222         else {
223             //latitude = 41.9028;
224             //longitude = 12.4964;
225
226             latitude = 39.9334;
227             longitude = 32.8597;
228         }
```

Figura 136: Modificación del código para situar el simulador en Turquía

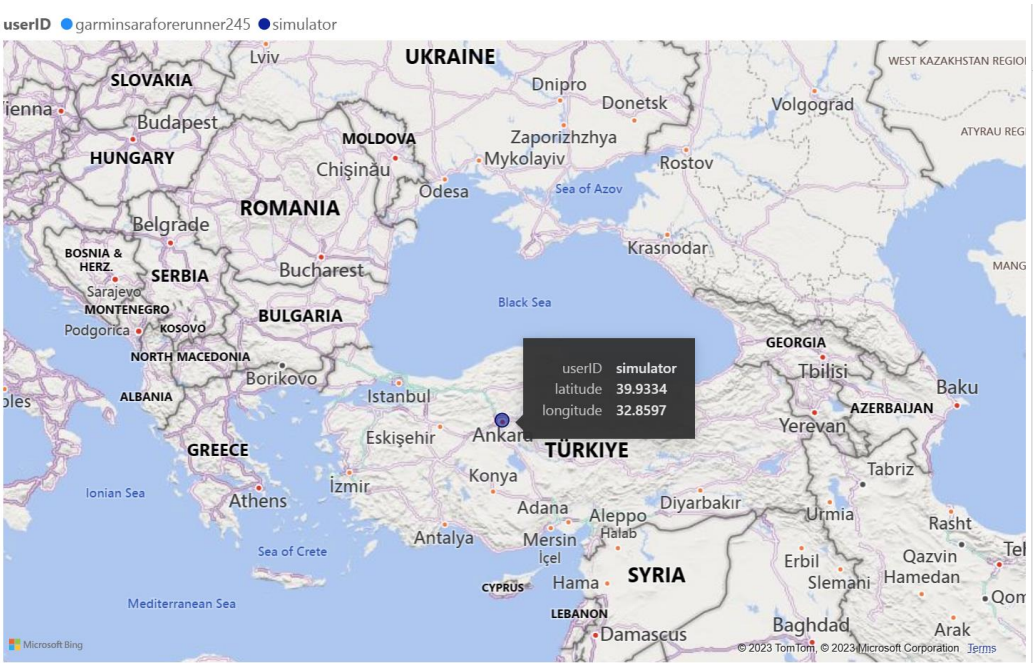

Figura 137: ubicación del simulador sobre el mapa 1 cuando las coordenadas se sitúan en Turquía

Un aspecto interesante a destacar es las posibilidades de interacción con el mapa. En la figura 138 se muestra como yendo a la pestaña de *Fromat visual* es posible modificar la vista del mapa.

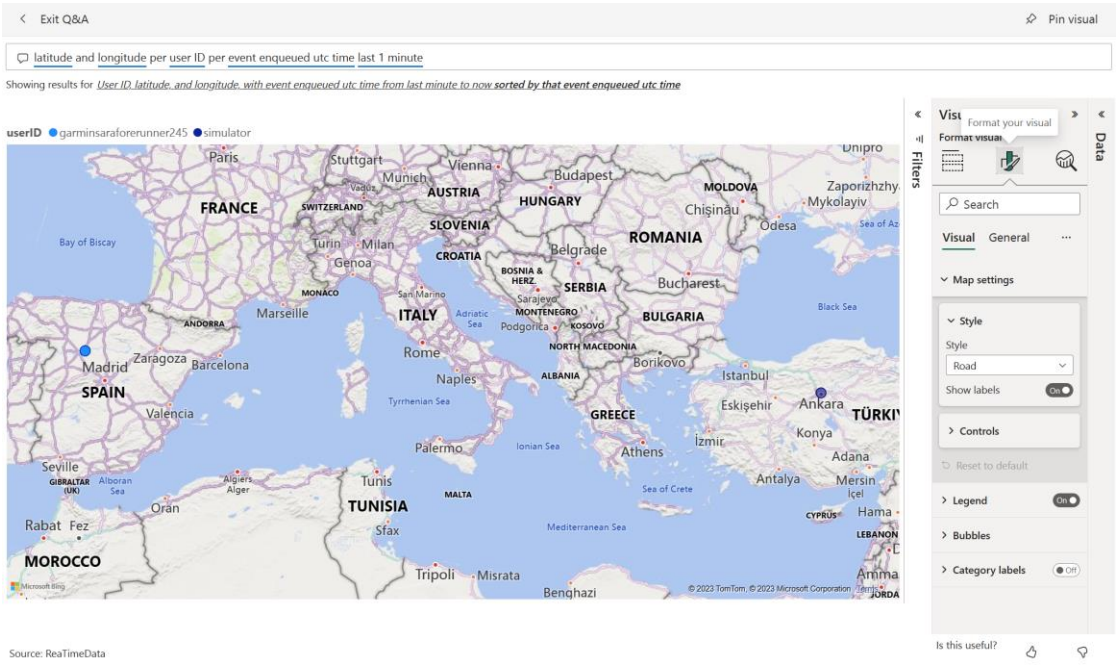

Figura 138: Opciones de interacción con el mapa

Si se selecciona el estilo Aerial la vista del mapa sería la mostrada en la figura 139.

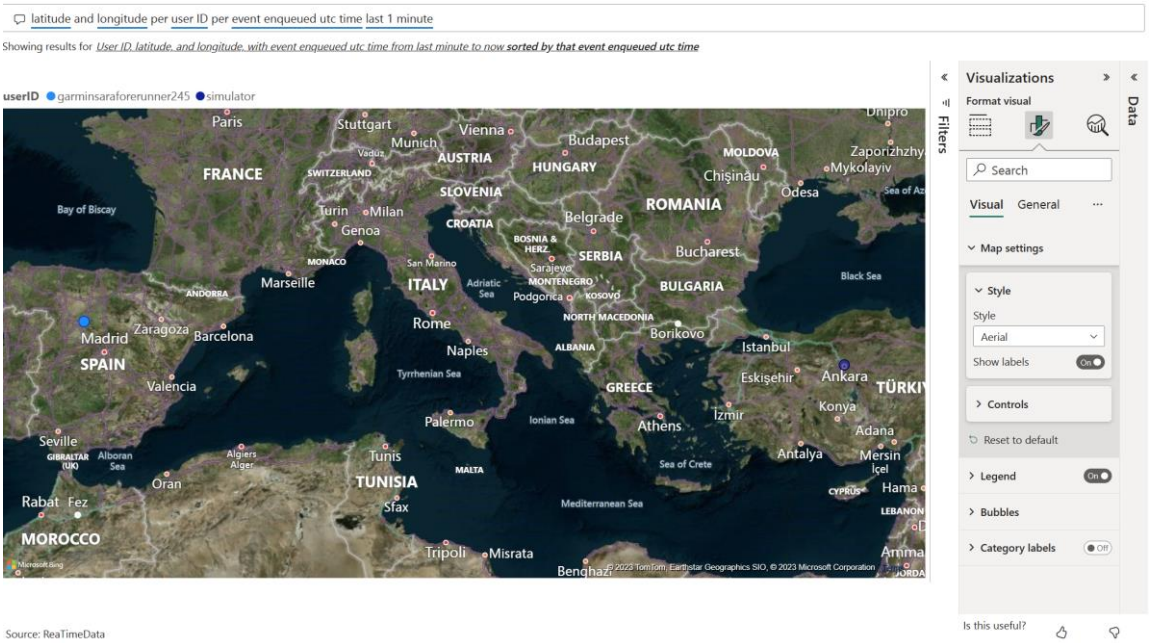

Figura 139: Vista del mapa con estilo Aerial

Si en lugar de Aerial se selecciona *grayscale* la vista sería de la figura 140.

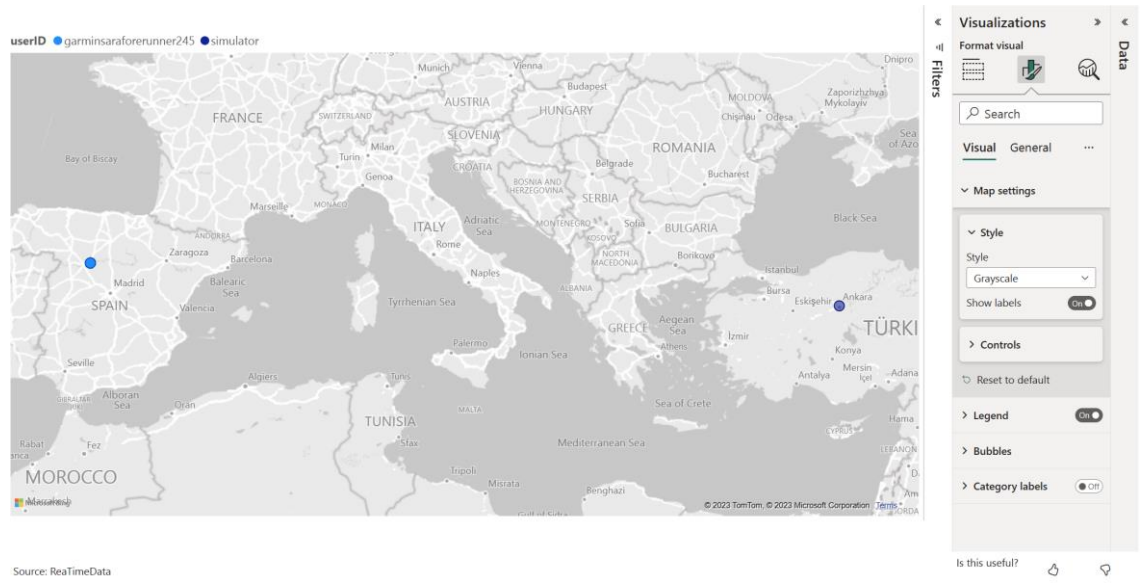

Figura 140: Vista mapa estilo grayscale

Además, como se muestra en la figura 141, es posible aumentar el zoom para ver exactamente la ubicación de un determinado usuario.

Figura 141: Zoom sobre el mapa

(3) Estas tarjetas muestran las pulsaciones máximas registradas por cada dispositivo en la ventana de un minuto. En la figura 136 puede observarse como en el panel superior se recogen las pulsaciones máximas registradas por el simulador mientras que en el panel inferior se muestran las registradas por el reloj.

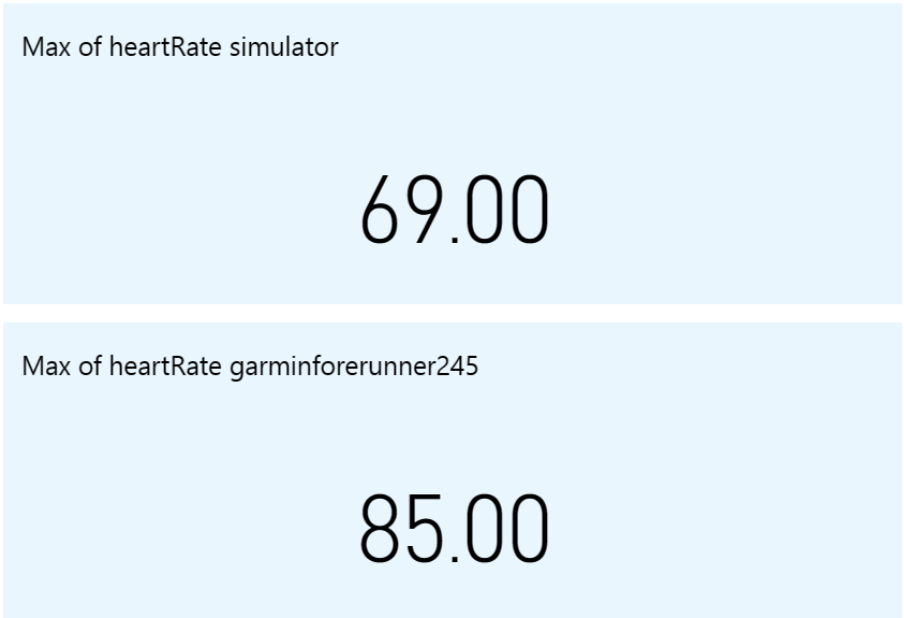

Figura 142: Frecuencia cardiaca máxima registrada por el simulador y por el reloj

Cuando un dispositivo no está conectado a la plataforma, su tarjeta correspondiente muestra en valor de (Blank), tal y como se muestra en la figura 137.

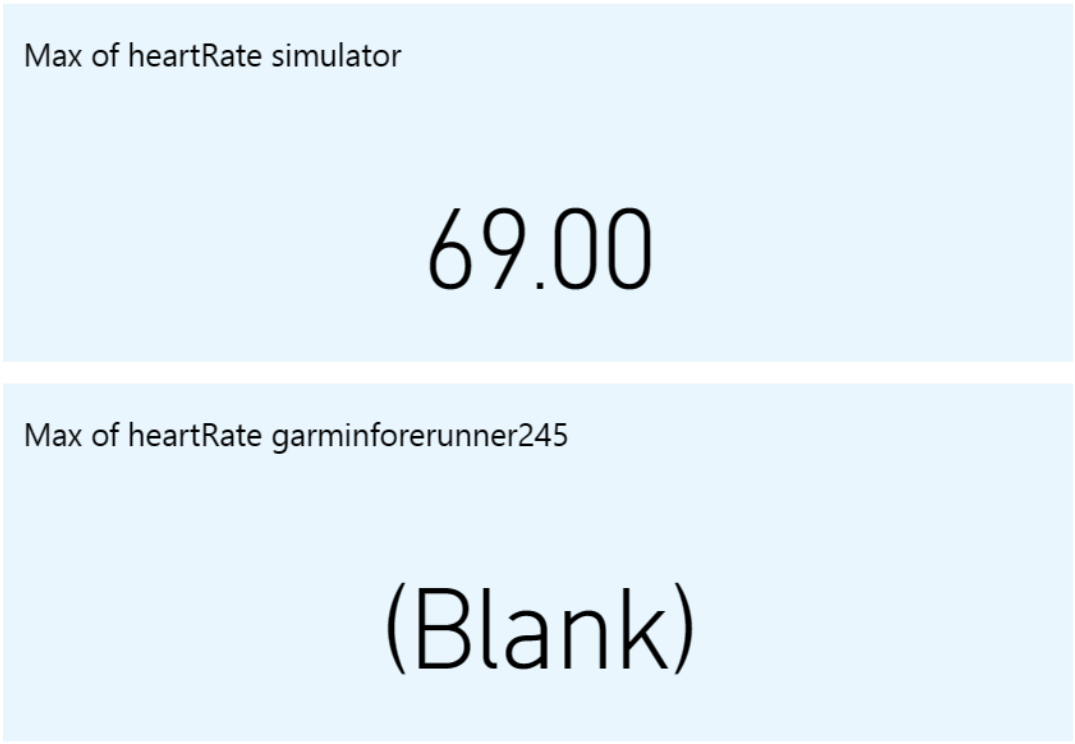

Figura 143: Frecuencia cardiaca máxima registrada cuando un dispositivo no se encuentra conectado

(4) Este último panel muestra l número de dispositivos conectados en un momento dado. En la figura 144 se muestra el aspecto de este panel cuando se tienen 2 dispositivos conectados.

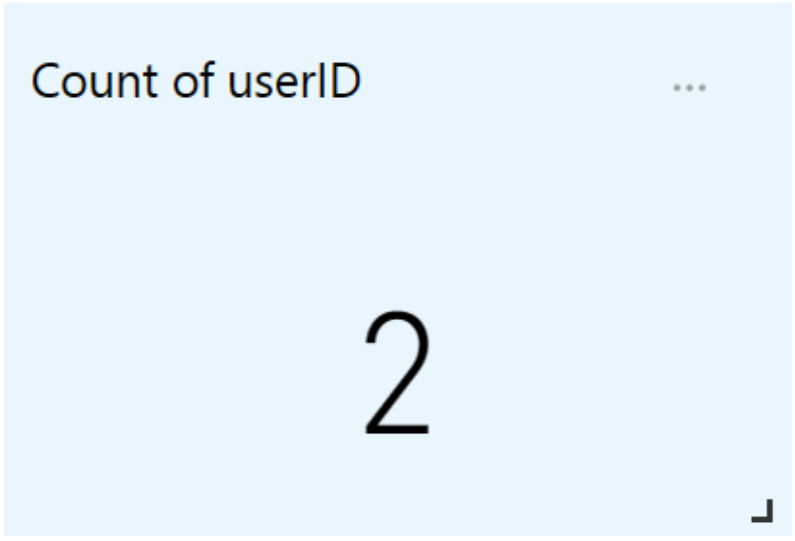

Figura 144: Número de dispositivos conectados

4.4. Generación de alertas

En esta última sección se analizan los resultados obtenidos cuando se genera una alerta ante la detección de pulsaciones anómalas. Para ello, se analiza el funcionamiento de todas las fases del proceso de generación de alertas, desde que comienzan a llegar datos al Azure Event Hub hasta que se envía el correo electrónico solicitando ayuda.

Desde la pestaña de *Overview* dentro del Event Hub garmineventhub, tal y como se muestra en la figura 145, es posible analizar las peticiones, los mensajes y el *throughput* del Azure Event Hub.

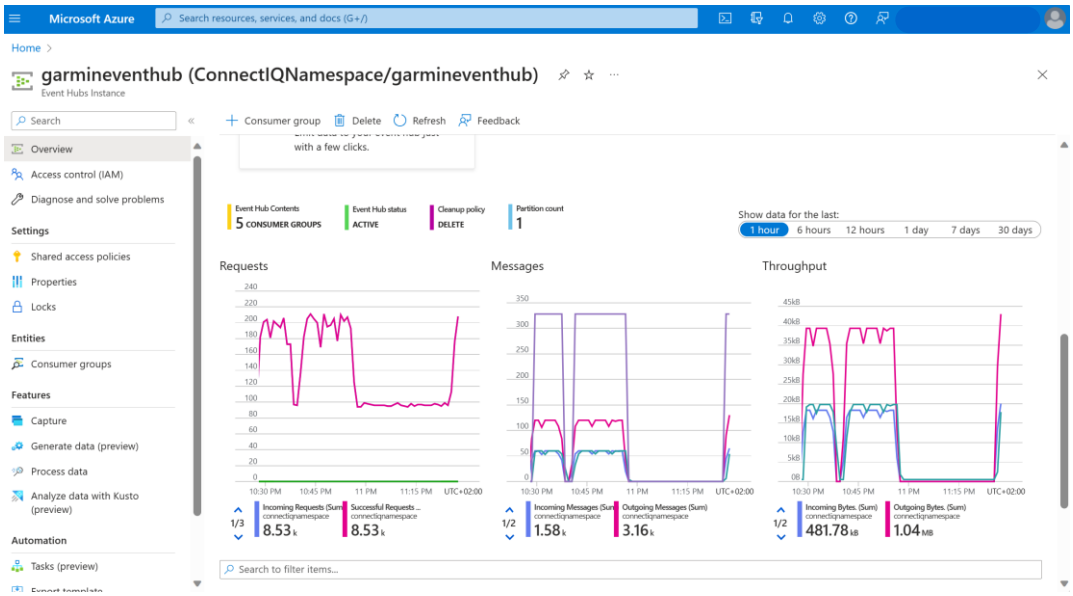

Figura 145: Peticiones, mensajes y thorughput del Azure Event Hub garmineventhub

Puede observarse que ya hay dispositivos enviando datos al Event Hub y que los dos Azure Stream Analytics Jobs se encuentran en funcionamiento dado que, como se aprecia en la figura 146 con mayor detalle que en la figura 147, los mensajes enviados desde el Event Hub (*outgoing messages*) son el doble de los que llegan al Event Hub (*incoming messages*).

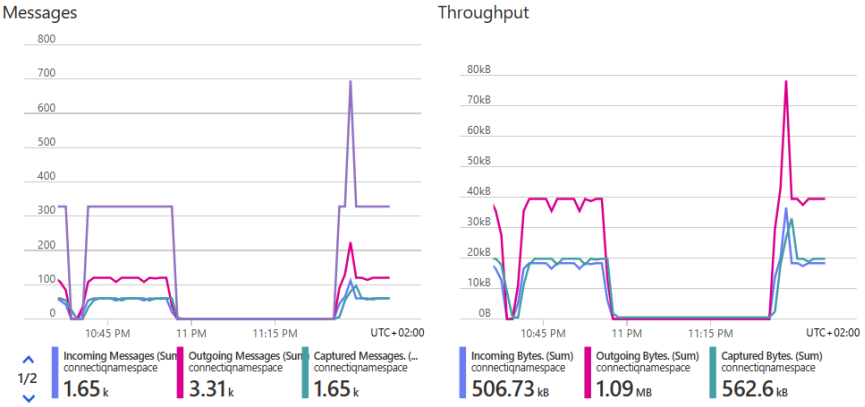

Figura 146: Mensajes y Throughput en el Azure Event Hub garmineventhub

Si se va al HearRateAnomalyDetection Azure Stream Analytics Job, como se ilustra en la figura 147, puede comprobarse que efectivamente el job está corriendo adecuadamente. Si se analiza la gráfica *Events count* puede observarse como se han producido 1.76k eventos de entrada en la última hora. Dado que no se ha detectado ninguna pulsación anómala, el número de eventos de salida es cero.

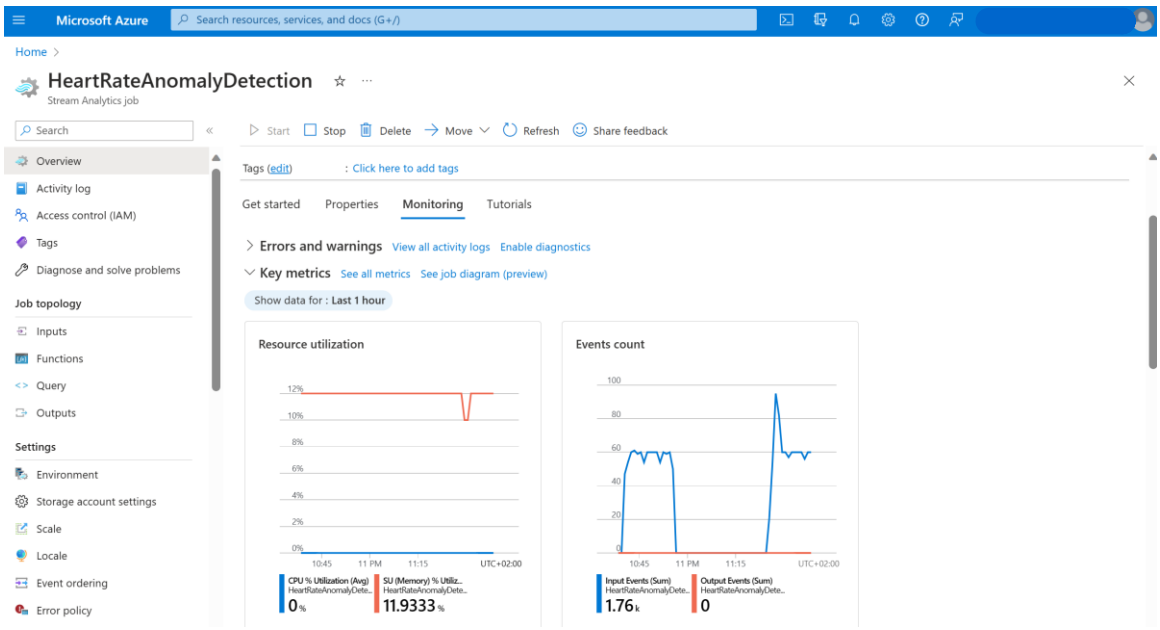

Figura 147: hearRateAnomalyDetection Overview

Si se selecciona la pestaña *Query* tal y como se ilustra en la figura 148, es posible probar que la *query* funciona correctamente. En la pestaña *input* se puede verificar que en estos momentos no se están registrando pulsaciones anómalas.

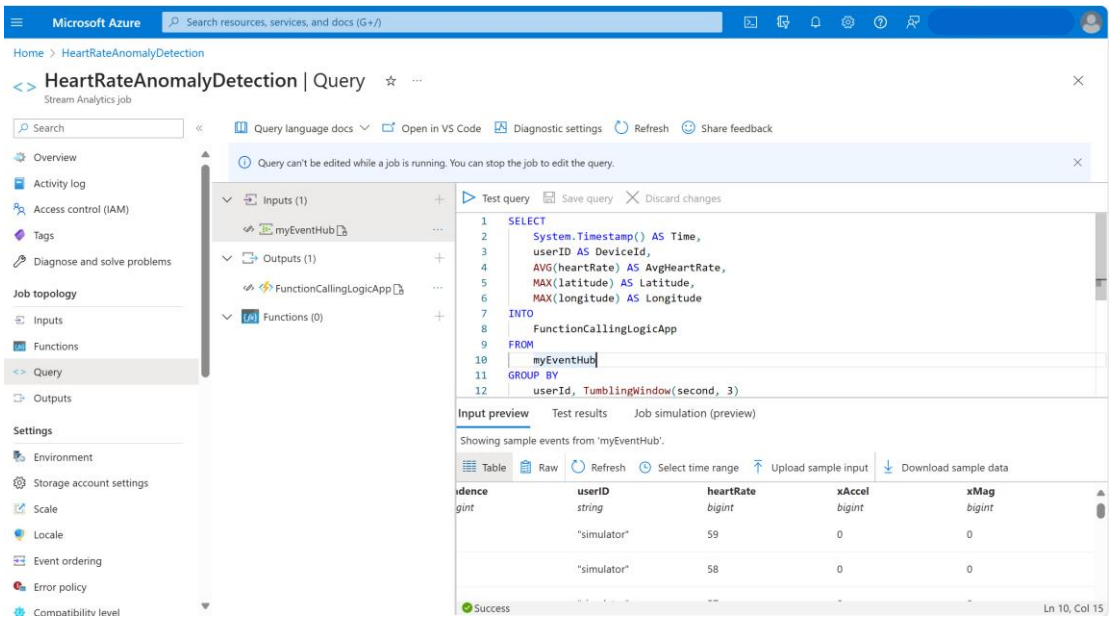

Figura 148: Test del correcto funcioamiento de la query del HeartRateAnomalyDetection Stream Analytics Job

Dado que no se están produciendo pulsaciones anómalas la ventana de *Test Results* se encuentra vacía, tal y como se ilustra en la figura 149.

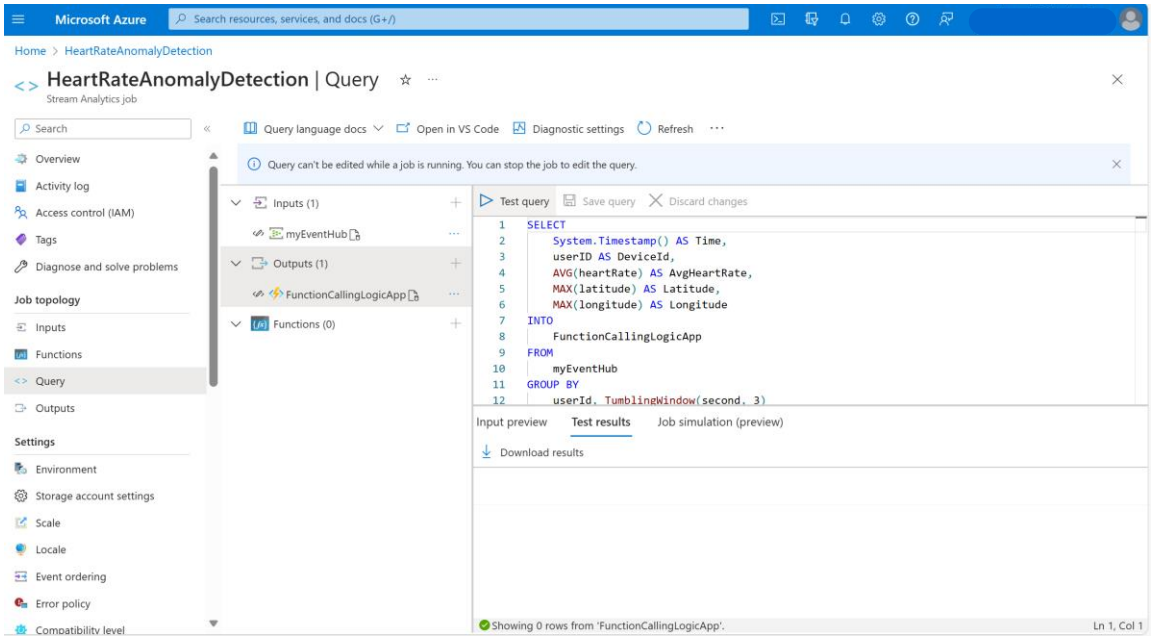

Figura 149: Resultado del test de la query cuando no se producen pulsaciones anómalas

Con el fin de generar pulsaciones anómalas que desencadenen la generación de una alerta, va a modificarse para la simulación de valores de frecuencia cardiaca tal y como se muestra en la figura 150. En concreto se va a generar pulsaciones aleatorias comprendidas en el intervalo 180 y 250. El código completo puede encontrarse en el anexo 1.

```
145         //Heartrate
146         if (sensorInfo has :heartRate && sensorInfo.heartRate != null) {
147
148             hR = sensorInfo.heartRate;
149
150         }
151         else
152             //Generación de pulsaciones aleatorias entre 180 y 250
153             hR = Math.rand()%70+180;
154
155             //Generación de pulsaciones aleatorias entre 50 y 70
156             //hR = Math.rand()%20+50;
157
158         }
```

Figura 150: Modificación del código de nuestra aplicación para la simulación de las pulsaciones deseadas

Dado que el job ha sido programado para detectar cuando en una ventana de tiempo de 3 segundos un usuario determinado supera la frecuencia cardiaca media de 200 pulsaciones por segundo, si se vuelve a testar la query como se ilustra en la figura 151, se puede comprobar como en esta ocasión ha detectado unas pulsaciones medias de 215,74 y la ventana de *Test results* ya no se encuentra vacía.

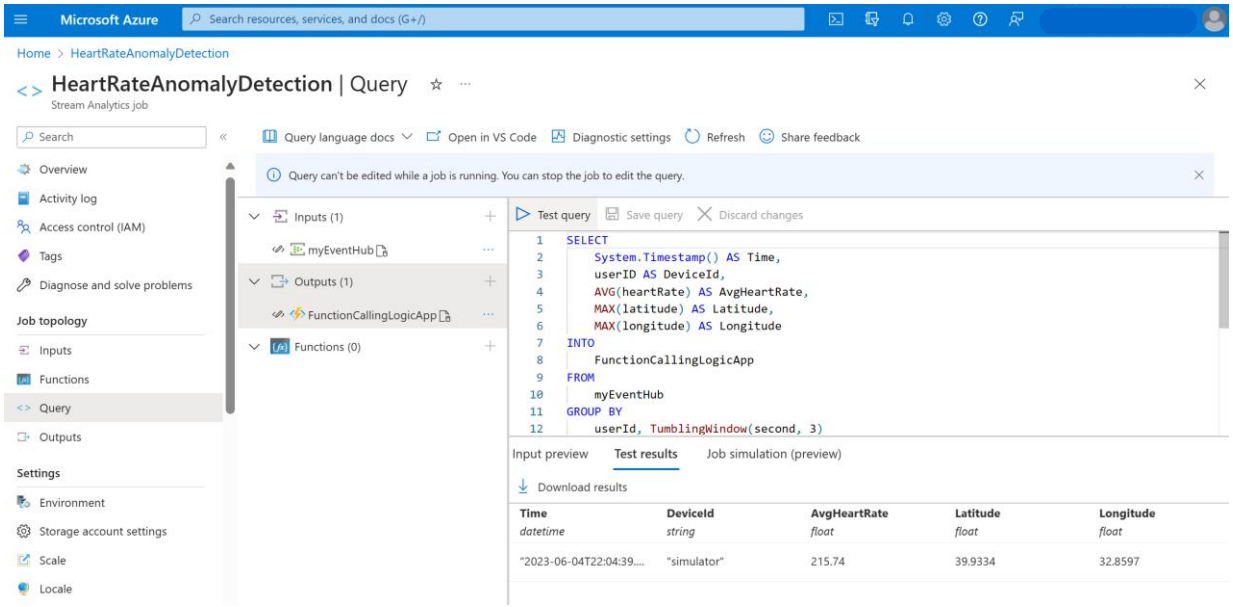

Figura 151: Resultado del test del Azure Stream Analytics Job cuando se generan pulsaciones anómalas

Si se vuelve a la pestaña de *Overview* como se muestra en la figura 152, es posible observar como en la gráfica de *Events count* ahora sí se producen eventos de salida y se está invocando a la función.

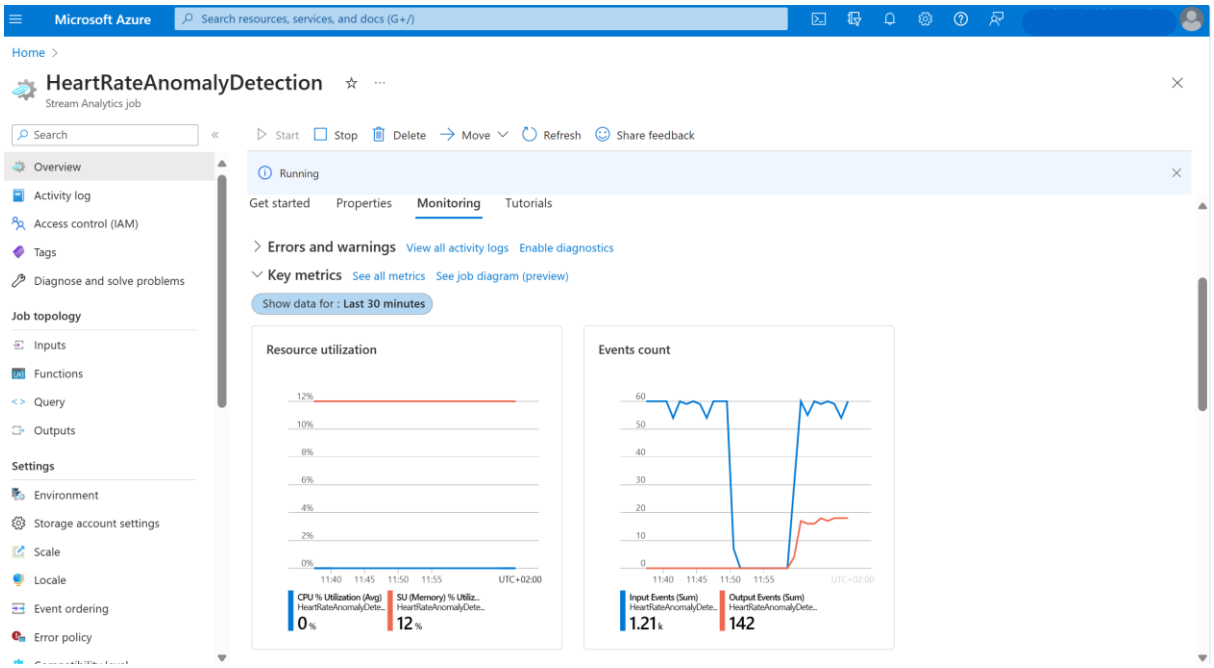

Figura 152: Overview del job HeartRateAnomalyDetection cuando se generan pulsaciones anómalas

Si se va a la pestaña de overview de la function tal y como se muestra en la figura 153, es posible comprobar que efectivamente la función está siendo invocada y ejecutada satisfactoriamente.

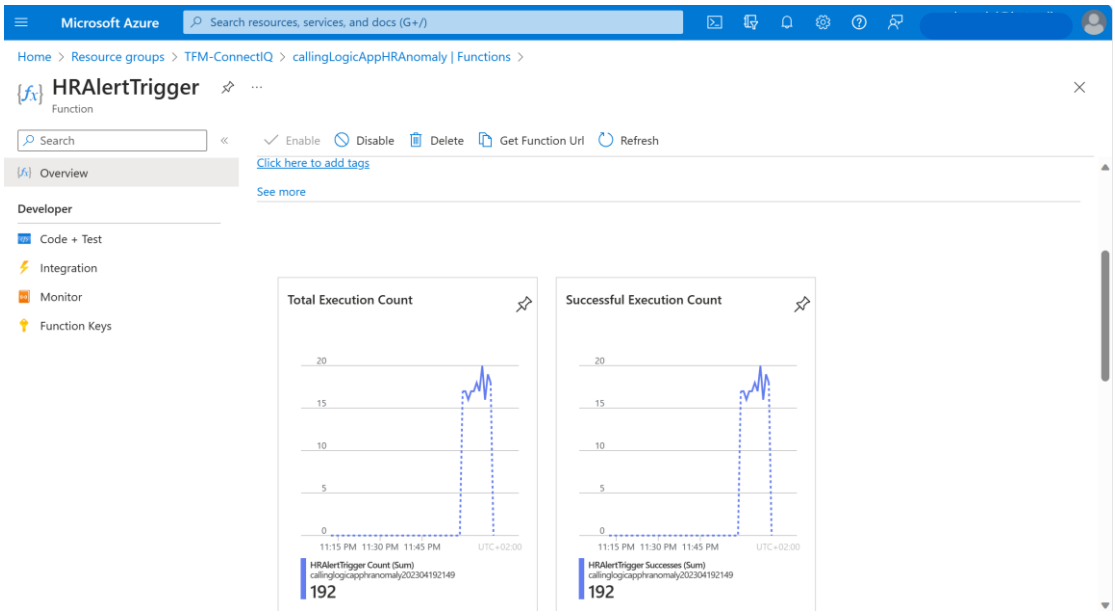

Figura 153: Gráfica de ejecuciones de la function como resultado de pulsaciones anómalas

Si en lugar de ir a la pestaña de *Overview* se va a la pestaña de *Monitor* tal y como se recoje en la figura 154, se puede observar la duración en milisegundos de la ejecución de la función. En esta figura puede observarse como la duración mínima ha sido de 119 ms y la máxima de 235 milisegundos.

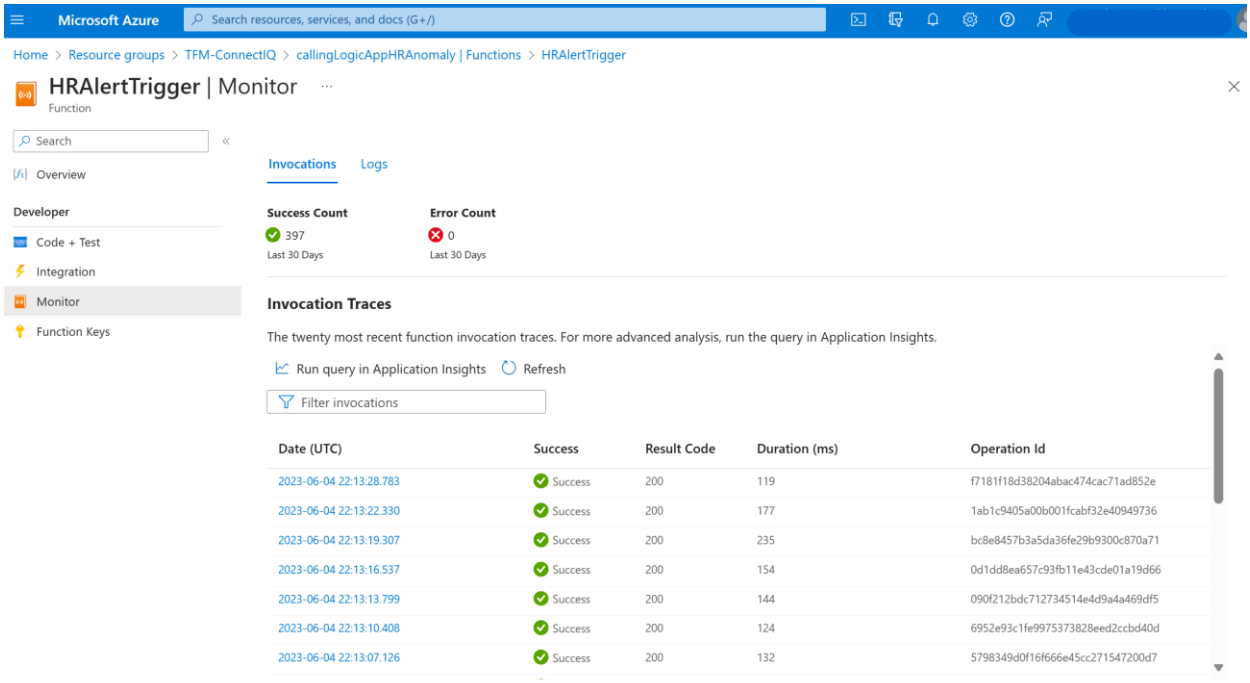

Figura 154: Tiempo de ejecución de la function

Si se va a la pestaña de *Overview* de la Logic App puede verse como también está siendo desencadenada correctamente. Desde *Overview* se puede obtener información acerca de las ejecuciones, los triggers y las métricas. En la figura 155, en la sección *Trigger history,* puede comprobarse como la Logic App está siendo desencadenada correctamente por la function.

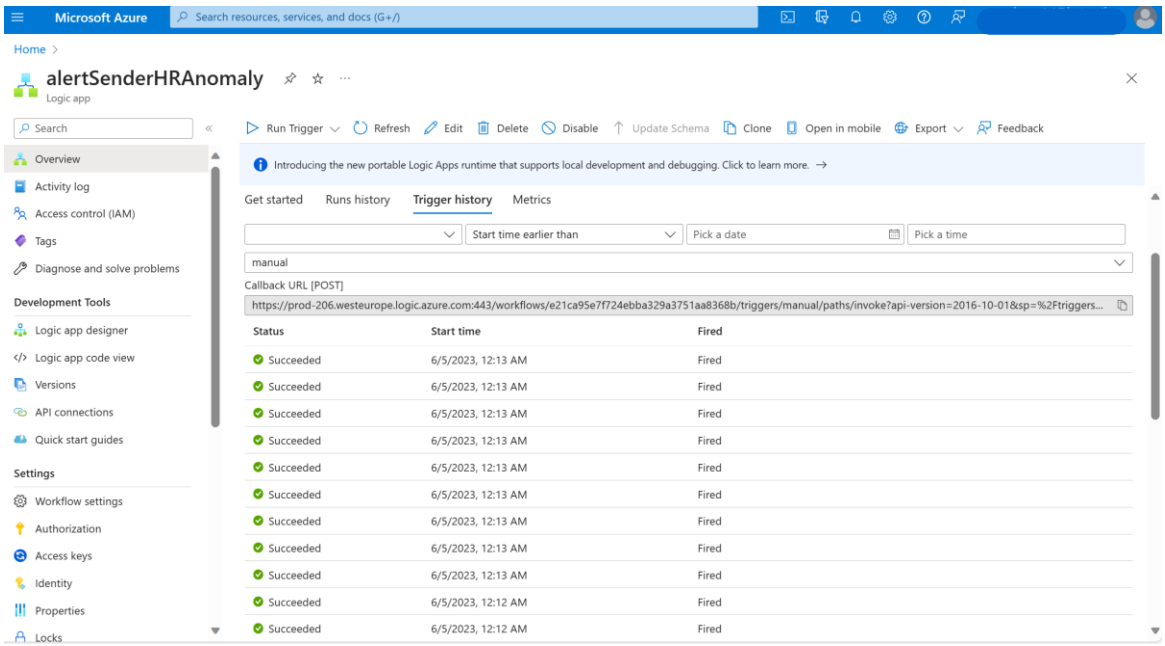

Figura 155: Registro de triggers de la Logic App

Desde la sección Runs history puede comprobarse como la Logic App se ejecuta correctamente, así como la duración de esta ejecución.

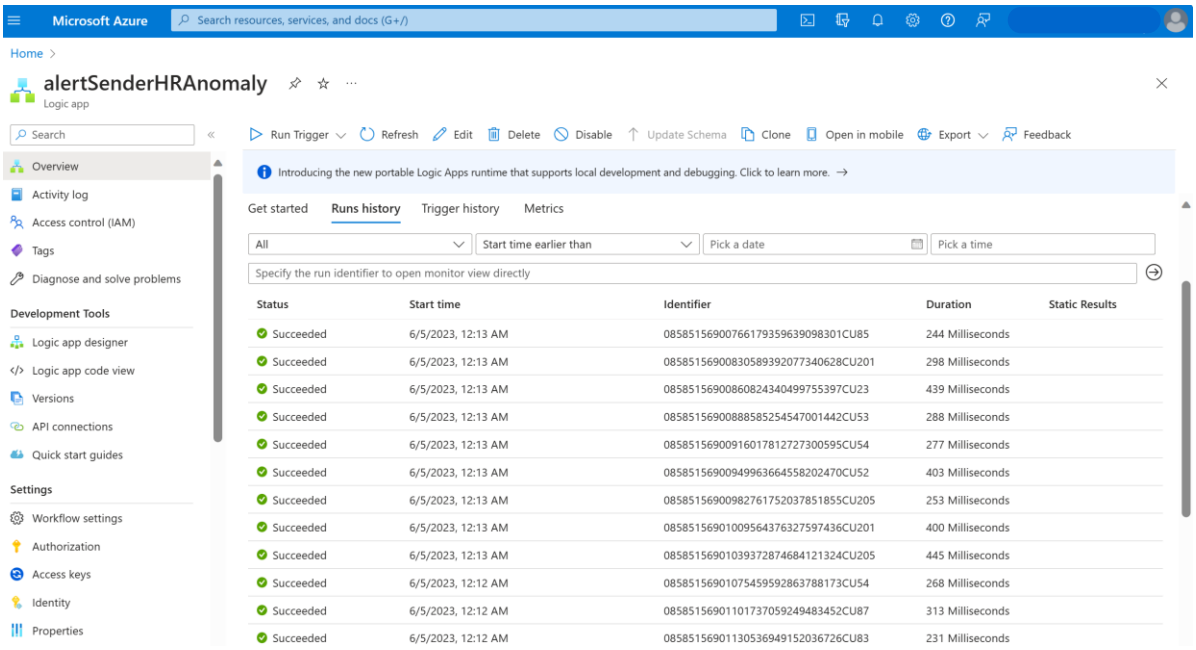

Figura 156: Registro de ejecución de la Logic App

Por último, en la sección de métricas se obtiene una gráfica con el número de veces que ha sido desencadenada la Logic App.

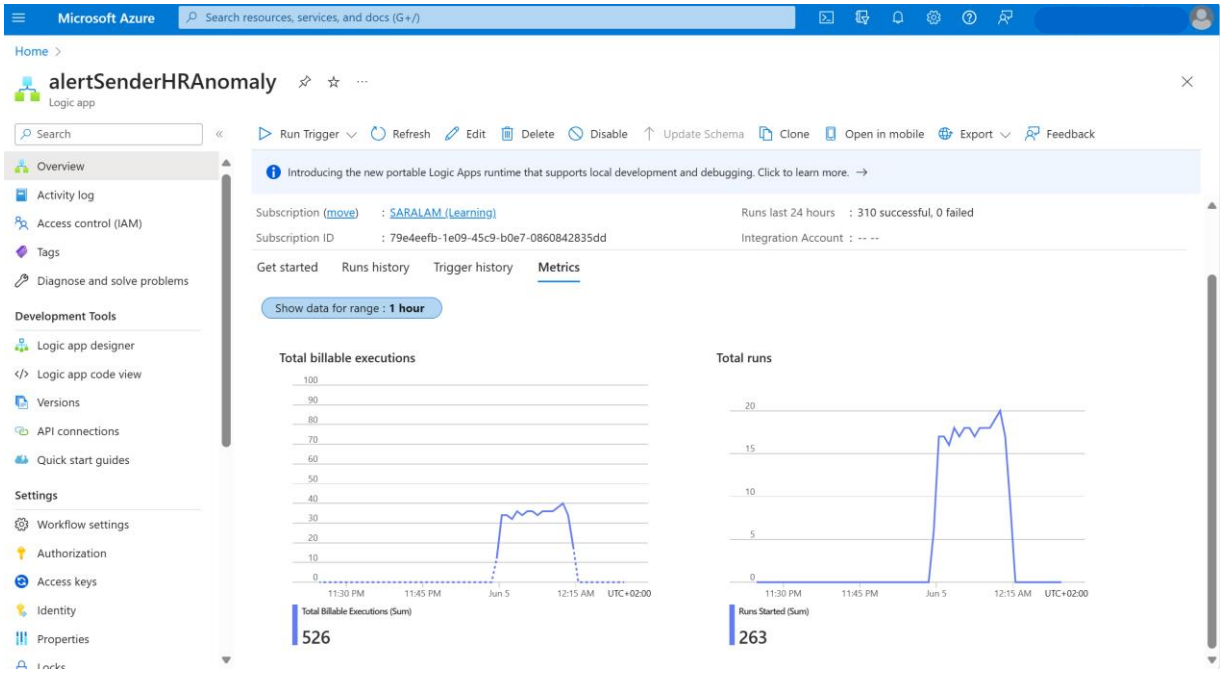

Figura 157: Métricas de ejecución de la Logic App

Por último, si se va al correo electrónico, es posible comprobar como se están recibiendo correctamente los correos electrónicos enviados por la Logic App, tal y como se ilustra en la figura 158.

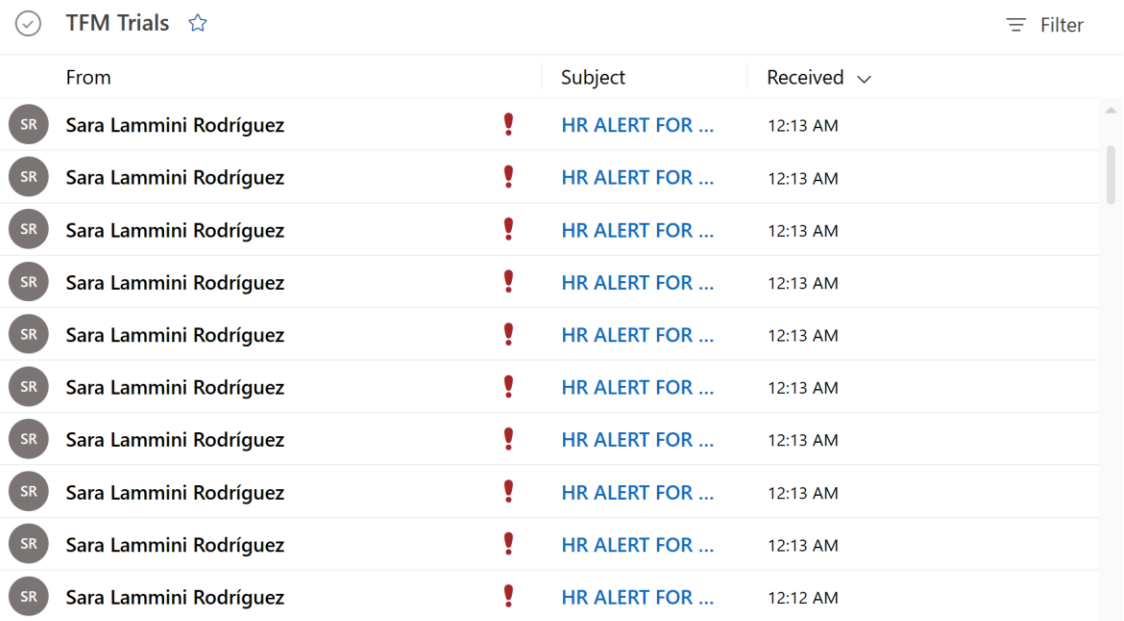

Figura 158: Correos electrónicos generados por la Logic App

Si se abre uno de estos correos, como se refleja en la figura 159, es posible comprobar como el sistema envía correctamente el email solicitando ayuda indicando el usuario que ha desencadenado la alerta, la frecuencia cardiaca media registrada y las coordenadas en las que se encuentra el usuario.

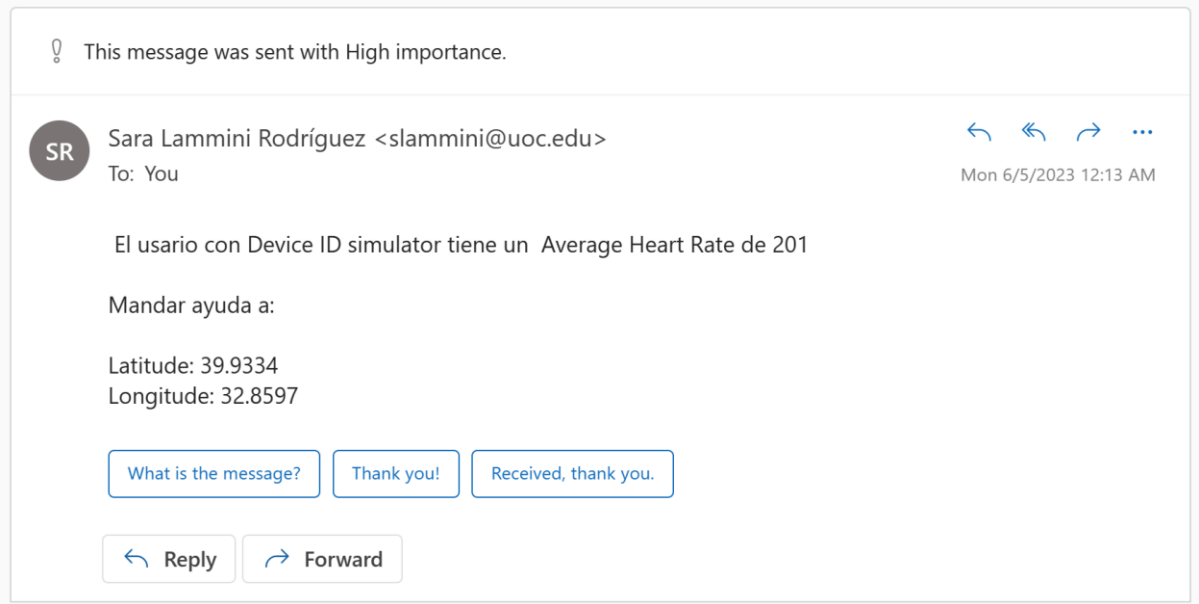

Figura 159: Correo electrónico de alerta

5. Conclusiones y trabajos futuros

Este capítulo tiene como propósito realizar una revisión retrospectiva de todo el trabajo realizado durante el cuatrimestre y reflejado en la presente memoria, con el fin de describir las conclusiones alcanzadas y explorar posibles líneas de investigación futuras que no han podido ser abordadas. Para lograr este objetivo, se ha dividido el capítulo en dos secciones. La primera sección se dedica al análisis de las conclusiones alcanzadas, mientras que la segunda sección se enfoca en la exploración de las posibles líneas para futuras investigaciones.

5.1. Conclusiones

En la primera sección de este capítulo, se describirán las conclusiones del trabajo, realizando una reflexión crítica sobre el logro de los objetivos planteados inicialmente, así como la evaluación de si se ha seguido la planificación establecida al inicio del cuatrimestre. También se evaluará si el impacto ético-social, de sostenibilidad y de diversidad estudiado en el capítulo 1 se ha alcanzado conforme a lo esperado.

La conclusión principal alcanzada es que es factible desarrollar una plataforma de monitorización continua de corredores en tiempo real a partir de relojes inteligentes y Azure. En el capítulo 4, dedicado a presentar los resultados de este TFM y describir el comportamiento de la plataforma desarrollada, se demuestra cómo es posible construir un panel sobre Power BI que permita la monitorización de las constantes vitales y la ubicación de los corredores en tiempo real. También se observa la viabilidad de desarrollar una lógica que permita generar alertas cuando un usuario registra pulsaciones anómalas, enviando un mensaje a los servicios de emergencia con la ubicación del corredor que requiere asistencia urgente.

Considerando todos estos aspectos, se puede afirmar que se ha logrado el objetivo principal establecido para este trabajo: desarrollar una plataforma innovadora para el monitoreo en tiempo real de corredores a partir de sus relojes inteligentes y los servicios ofertados por Azure. En relación a los objetivos secundarios, se han cumplido los objetivos planteados inicialmente. A lo largo de todo el trabajo, se ha realizado un análisis exhaustivo de los riesgos asociados con el deporte de carrera con el fin de identificar los riesgos a mitigar y las necesidades a cubrir mediante la herramienta a desarrollar. También se han evaluado diferentes tecnologías y servicios proporcionados por Azure y otras opciones de nube pública para satisfacer las necesidades de la solución. Finalmente, se ha llevado a cabo el diseño e implementación de la solución que permite el monitoreo de los atletas y la generación de alertas en caso necesario.

La herramienta desarrollada tiene un impacto positivo en las dimensiones ético-sociales, de sostenibilidad y de diversidad. En primer lugar, la herramienta permite monitorizar en todo momento las constantes vitales de los corredores y generar alertas cuando se registran valores anómalos. Esto reduce algunos de los riesgos asociados con la carrera, como

ataques cardíacos o muerte súbita, al permitir una respuesta rápida, lo que tiene un impacto socialmente beneficioso pues, como se veía en el capítulo de 1, la muerte de deportistas aparentemente sanos es fuente de gran conmoción social. Además, al facilitar la monitorización continua de la ubicación de los deportistas contribuye a la diversidad. Como se describía en el capítulo 1, algunos de los riesgos asociados a la práctica de carrera son el hecho de sufrir tropiezos y caídas o enfrentarse a situaciones de riesgo, miedo especialmente común entre mujeres corredoras. Por lo tanto, gracias a esta herramienta los corredores pueden sentirse más seguros al saber que su ubicación está siendo monitorizada en todo momento. Además, al basarse la plataforma desarrollada en la tecnología en la nube, es posible realizar un escalado de los recursos acorde con la demanda de cada momento. De esta forma, se logra un impacto positivo en términos de sostenibilidad al reducir el desperdicio energético y de recursos.

Con respecto a la planificación, no se ha requerido la utilización de otros recursos además de los listados inicialmente. Con respecto al cronograma, se puede concluir que, en líneas generales, cada una de las fases en las que se ha dividido la realización del presente trabajo se ha cumplido en el orden y tiempo estipulado.

5.2. Trabajos futuros

En esta segunda sección se plantearán posibles líneas de trabajo futuro que no han podido explorarse en el presente TFM y han quedado pendientes. Estas líneas son las siguientes:

- Integración de la herramienta desarrollada con tecnologías de inteligencia artificial y aprendizaje automático para detectar patrones y anomalías en los datos recogidos. En el presente trabajo se ha implementado una plataforma que permite detectar anomalías cardiacas en tiempo real. También se ha desarrollado la lógica necesaria para crear un registro histórico de los datos enviados por los dispositivos conectados. No obstante, sería interesante implementar soluciones de inteligencia artificial y aprendizaje automático que permitan estudiar el histórico de datos de cada usuario para detectar patologías subyacentes de forma económica, escalable y sostenible.

- Extender la solución a otras marcas de relojes inteligentes. Aunque Garmin es una de las marcas de relojes deportivos más popular, sería conveniente desarrollar las aplicaciones pertinentes a otras marcas de relojes populares como Apple o Suunto con el fin de aumentar la accesibilidad de la solución implementada.

- Estudiar la posibilidad de incorporar comunicación bidireccional entre los relojes y la solución para que los usuarios puedan solicitar ayuda en caso de encontrarse en una situación de riesgo sin esperar a que se detecte alguna anomalía.

6. Bibliografía

1. Runner's World. 12 beneficios de correr para que te animes a ser runner [Internet]. 2022 [cited 2023 Mar 2]. Available from: https://www.runnersworld.com/es/salud-lesiones-runner/a40141551/beneficios-correr/
2. Sanitas. Beneficios de correr [Internet]. [cited 2023 Mar 2]. Available from: https://www.sanitas.es/sanitas/seguros/es/particulares/biblioteca-de-salud/ejercicio-deporte/Consejos-para-correr/san041669wr.html
3. Women's Running. Is running good for you? [Internet]. 2022 [cited 2023 Mar 2]. Available from: https://www.womensrunning.co.uk/health/is-running-good-for-you/
4. Morentin B, Suárez-Mier MP, Monzó A, Ballesteros J, Molina P, Lucena J. Muerte súbita relacionada con la actividad deportiva en España. Estudio poblacional multicéntrico forense de 288 casos. Rev Española Cardiol [Internet]. 2021 Mar 1 [cited 2023 Mar 4];74(3):225–32. Available from: http://www.revespcardiol.org/es-muerte-subita-relacionada-con-actividad-articulo-S0300893220303304
5. Rich MW, Maron BJ. Risk for sudden cardiac death associated with marathon running [3]. J Am Coll Cardiol. 1997;29(1):224.
6. Noticias de la Universidad del País Vasco. La muerte súbita asociada al deporte es muy infrecuente y difícil de predecir [Internet]. 2022 [cited 2023 Mar 4]. Available from: https://www.ehu.eus/es/-/la-muerte-subita-asociada-al-deporte-es-muy-infrecuente-y-dificil-de-predecir
7. What Is Cardiac Arrest? [Internet]. National Heart, Lung and Blood Institute. 2022 [cited 2023 Mar 4]. Available from: https://www.nhlbi.nih.gov/health/cardiac-arrest
8. Fredette A, Roy JS, Perreault K, Dupuis F, Napier C, Esculier JF. The Association Between Running Injuries and Training Parameters: A Systematic Review. J Athl Train [Internet]. 2022 Jul 1 [cited 2023 May 14];57(7):650–71. Available from: https://meridian.allenpress.com/jat/article/57/7/650/470023/The-Association-Between-Running-Injuries-and
9. CinfaSalud. Lesiones más habituales en el running y cómo evitarlas [Internet]. [cited 2023 May 14]. Available from: https://cinfasalud.cinfa.com/p/lesiones-en-el-running/
10. Logo C. How to Run Alone Safely: What You Need to Know [Internet]. CNET. 2022 [cited 2023 Mar 4]. Available from: https://www.cnet.com/health/fitness/how-to-run-alone-safely-what-you-need-to-know/
11. Runner's World. 60% of women we surveyed have been harassed while running [Internet]. Runner's World. 2021 [cited 2023 Mar 4]. Available from: https://www.runnersworld.com/uk/training/a36278390/reclaim-the-run/
12. Brooks A. Confidently Running Alone: Tips for Running Safety [Internet]. RunToTheFinish. [cited 2023 Mar 4]. Available from: https://www.runtothefinish.com/running-safely/
13. Spector N. Scared to run alone? Women runners share their best safety tips [Internet]. 2018 [cited 2023 Mar 4]. Available from: https://www.nbcnews.com/better/health/scared-run-alone-female-runners-share-how-they-stay-safe-ncna935186
14. Trota M. Guía del corredor novato: la montaña [Internet]. [cited 2023 Mar 4]. Available from: https://carreraspopulares.com/noticia/guia-del-corredor-novato-la-montana
15. Microsoft. Tutorial: Deploy and review the continuous patient monitoring application template [Internet]. Microsoft Learn . 2023 [cited 2023 Mar 4]. Available from: https://learn.microsoft.com/en-us/azure/iot-central/healthcare/tutorial-continuous-patient-monitoring
16. Jat AS, Grønli TM. Smart Watch for Smart Health Monitoring: A Literature Review. Lect Notes Comput Sci (including Subser Lect Notes Artif Intell Lect Notes Bioinformatics) [Internet]. 2022 [cited 2023 Mar 4];13346 LNBI:256–68. Available from:

https://link.springer.com/chapter/10.1007/978-3-031-07704-3_21
17. Gómez R. El reloj inteligente, el wearable preferido [Internet]. Deporte Tecnológico. 2020 [cited 2023 Mar 4]. Available from: https://deportetecnologico.com/el-reloj-inteligente-lo-preferido-por-los-corredores/
18. Suárez-Miera MP, Aguilera B. Causas de muerte súbita asociada al deporte en España. Revista Española de Cardiología [Internet]. 2002 [cited 2023 Mar 4]; Available from: https://www.revespcardiol.org/es-pdf-13029695
19. Boraita A. Muerte súbita y deporte. ¿Hay alguna manera de prevenirla en los deportistas? [Internet]. Revista Española de Cardiología. 2002 [cited 2023 Mar 4]. Available from: https://www.revespcardiol.org/es-muerte-subita-deporte-hay-alguna-articulo-13029693
20. Cardiac Arrest Treatment [Internet]. National Heart, Lung and Blood Institute. 2022 [cited 2023 Mar 4]. Available from: https://www.nhlbi.nih.gov/health/cardiac-arrest/treatment
21. Clínica Cemtro. Reconocimientos Medicos Deportivos [Internet]. [cited 2023 Mar 8]. Available from: https://www.clinicacemtro.com/medicina-deportiva/reconocimientos-medicos-deportivos/
22. Sede electrónica del ayuntamiento de Madrid. Programas Médico Deportivos. Reconocimientos Médico-Deportivos . Gestiones y Trámites [Internet]. [cited 2023 Mar 8]. Available from: https://sede.madrid.es/portal/site/tramites/menuitem.62876cb64654a55e2dbd7003a8a409a0/?vgnextoid=4b1ddd9d6baed010VgnVCM2000000c205a0aRCRD&vgnextchannel=ae29a38813180210VgnVCM100000c90da8c0RCRD&vgnextfmt=pd
23. Arango Astorga P, García García Y. Internet de las cosas en el ámbito de la atención médica: tendencias y desafíos [Internet]. Revista Cubana de Informática Médica. 2022 [cited 2023 May 27]. Available from: http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1684-18592022000100014
24. Newton-Smith C. Reimagining healthcare with Azure IoT [Internet]. 2020 [cited 2023 Mar 4]. Available from: https://azure.microsoft.com/es-es/blog/reimagining-healthcare-with-azure-iot/
25. Pradhan B, Bhattacharyya S, Pal K. IoT-Based Applications in Healthcare Devices. J Healthc Eng. 2021;2021.
26. Syed L, Jabeen S, S. M, Alsaeedi A. Smart healthcare framework for ambient assisted living using IoMT and big data analytics techniques. Futur Gener Comput Syst. 2019 Dec 1;101:136–51.
27. Colombage L, Amarasiri T, Sanjeewani T, Senevirathne C, Panchendrarajan R. SmartCare: Detecting Heart Failure and Diabetes Using Smartwatch. Proc - 2022 IEEE Int Conf Smart Internet Things, SmartIoT 2022. 2022;20–7.
28. Avram MR, Pop F. Real-time running workouts monitoring using Cloud–Edge computing. Neural Comput Appl [Internet]. 2022 Jan 10 [cited 2023 Mar 23];1–20. Available from: https://link.springer.com/article/10.1007/s00521-021-06675-3
29. Şengül G, Karakaya M, Misra S, Abayomi-Alli OO, Damaševičius R. Deep learning based fall detection using smartwatches for healthcare applications. Biomed Signal Process Control. 2022 Jan 1;71:103242.
30. Apple. Apple Watch, una ayuda para que tus pacientes lleven una vida más sana. [Internet]. [cited 2023 Mar 23]. Available from: https://www.apple.com/es/healthcare/apple-watch/
31. Garmin. Garmin Health [Internet]. [cited 2023 Mar 23]. Available from: https://www.garmin.com/es-ES/health/business-solutions/research/
32. Labfront. On a Mission to Democratize Research [Internet]. [cited 2023 May 28]. Available from: https://www.labfront.com/about
33. Labfront Team. Labfront and Garmin Collaboration: Accelerating Wearable Tech in Academic Research. 2022 Jun 20 [cited 2023 Mar 23]; Available from: https://www.labfront.com/blog/labfront-and-garmin-collaboration

34. Labfront. Guide for Participant Onboarding Using Garmin Connect and Labfront Companion [Internet]. [cited 2023 May 28]. Available from: https://help.labfront.com/participant-onboarding-garmin-and-labfront
35. HumanITCare. HumanITcare [Internet]. [cited 2023 May 28]. Available from: https://humanitcare.com/
36. HumanITcare. HumanITcare collaborates with Garmin to provide expanded access to connected health . 2023 Jan 23 [cited 2023 Mar 23]; Available from: https://humanitcare.com/en/humanitcare-collaborates-with-garmin-to-provide-expanded-access-to-connected-health/
37. Chronolife. Chronolife [Internet]. [cited 2023 May 28]. Available from: https://www.chronolife.net/
38. Chronolife. Chronolife announces collaboration with Garmin to further enrich continuous remote patient monitoring in real-life conditions [Internet]. [cited 2023 Mar 23]. Available from: https://www.chronolife.net/chronolife-garmin-remote-patient-monitoring/
39. Chronolife. Chronolife, a Multisource Medical Data Hub [Internet]. [cited 2023 May 28]. Available from: https://www.chronolife.net/about/
40. Centro de Asistencia Garmin. Configuración de la Detección de Incidentes en un Dispositivo Garmin [Internet]. [cited 2023 Mar 23]. Available from: https://support.garmin.com/es-ES/?faq=RfaXahBWkH8Q7pVFLsuUmA
41. Microsoft Learn. What is Azure IoT Central? [Internet]. 2022 [cited 2023 Mar 24]. Available from: https://learn.microsoft.com/en-us/azure/iot-central/core/overview-iot-central
42. Grey Matter. The health data journey, from device to doctor [Internet]. 2020 [cited 2023 May 28]. Available from: https://greymatter.com/content-hub/the-health-data-journey-from-device-to-doctor/
43. Microsoft Learn. IoT concepts and Azure IoT Hub . 2023 Mar 23 [cited 2023 Mar 24]; Available from: https://learn.microsoft.com/en-us/azure/iot-hub/iot-concepts-and-iot-hub
44. Benmekki A. What is Azure IoT Hub? [Internet]. 2022 [cited 2023 May 28]. Available from: https://blog.cellenza.com/en/data/what-is-azure-iot-hub/
45. Amazon Web Services. AWS IoT Core [Internet]. [cited 2023 Mar 24]. Available from: https://aws.amazon.com/es/iot-core/
46. Microsoft Azure. Event Hubs: ingesta de datos en tiempo real [Internet]. [cited 2023 Mar 24]. Available from: https://azure.microsoft.com/es-es/products/event-hubs/
47. de Groot M. Azure Event Hub vs IoT Hub [Internet]. 2016 [cited 2023 May 28]. Available from: https://microsoft-bitools.blogspot.com/2016/12/azure-event-hub-vs-iot-hub.html
48. AWS. Amazon Kinesis. Procesamiento de datos en tiempo real [Internet]. [cited 2023 May 28]. Available from: https://aws.amazon.com/es/kinesis/
49. Google Cloud. ¿Qué es Pub/Sub? [Internet]. [cited 2023 May 28]. Available from: https://cloud.google.com/pubsub/docs/overview?hl=es-419
50. Ezquerro Saenz M. Como usar Cloud IoT Core gateways con una Raspberry Pi [Internet]. 2019 [cited 2023 May 28]. Available from: https://www.cloudespañol.com/2019/01/como-usar-cloud-iot-core-gateways-con.html
51. Garmin. Hello Monkey C! [Internet]. 2023. Available from: https://developer.garmin.com/connect-iq/monkey-c/
52. Microsoft Learn. Create a storage account [Internet]. 2023. Available from: https://learn.microsoft.com/en-us/azure/storage/common/storage-account-create?tabs=azure-portal
53. Microsoft Learn. Enable capturing of events streaming through Azure Event Hubs [Internet]. 2023 [cited 2023 May 19]. Available from: https://learn.microsoft.com/en-us/azure/event-hubs/event-hubs-capture-enable-through-portal
54. Microsoft Learn. Introduction to Azure Data Lake Storage Gen2 [Internet]. 2023 [cited 2023

May 19]. Available from: https://learn.microsoft.com/en-us/azure/storage/blobs/data-lake-storage-introduction#what-is-a-data-lake
55. Microsoft Learn. Azure Event Hubs. A big data streaming platform and event ingestion service [Internet]. 2023. Available from: https://learn.microsoft.com/en-us/azure/event-hubs/event-hubs-about
56. Microsoft Learn. Monitoring Azure Event Hubs data reference [Internet]. 2023. Available from: https://github.com/MicrosoftDocs/azure-docs/blob/main/articles/event-hubs/monitor-event-hubs-reference.md
57. Microsoft Learn. Connecting IoT Devices to Azure: IoT Hub and Event Hubs [Internet]. 2023 [cited 2023 May 19]. Available from: https://learn.microsoft.com/en-us/azure/iot-hub/iot-hub-compare-event-hubs
58. Microsoft Learn. Choose an Internet of Things (IoT) solution in Azure [Internet]. [cited 2023 May 19]. Available from: https://learn.microsoft.com/en-us/azure/architecture/example-scenario/iot/iot-central-iot-hub-cheat-sheet
59. Microsoft Learn. Outputs from Azure Stream Analytics [Internet]. 2023. Available from: https://learn.microsoft.com/en-us/azure/stream-analytics/stream-analytics-define-outputs
60. Microsoft Learn. Process data from your event hub using Azure Stream Analytics [Internet]. 2022 [cited 2023 May 19]. Available from: https://learn.microsoft.com/en-us/azure/event-hubs/process-data-azure-stream-analytics
61. Microsoft Learn. What is Power BI? [Internet]. 2023. Available from: https://learn.microsoft.com/en-us/power-bi/fundamentals/power-bi-overview
62. Microsoft Learn. Real-time streaming in Power BI [Internet]. 2023. Available from: https://learn.microsoft.com/en-us/power-bi/connect-data/service-real-time-streaming
63. Microsoft Learn. Introduction to Azure Functions. 2023; Available from: https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview
64. Microsoft Learn. Azure Functions hosting options [Internet]. Available from: https://learn.microsoft.com/en-us/azure/azure-functions/functions-scale
65. Microsoft Learn. Azure Functions triggers and bindings concepts [Internet]. 2023. Available from: https://learn.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings?tabs=csharp
66. Microsoft Learn. Service Bus queues, topics, and subscriptions [Internet]. 2022. Available from: https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-queues-topics-subscriptions
67. Microsoft Learn. What is Azure Logic Apps? [Internet]. 2023. Available from: https://learn.microsoft.com/en-us/azure/logic-apps/logic-apps-overview
68. Microsoft Learn. List of all Logic Apps connectors [Internet]. 2023. Available from: https://learn.microsoft.com/en-us/connectors/connector-reference/connector-reference-logicapps-connectors
69. SendGrid. SendGrid [Internet]. Available from: https://sendgrid.com/solutions/email-api/
70. Soporte técnico de Microsoft 365. ¿Qué es Outlook? [Internet]. Available from: https://support.microsoft.com/es-es/office/-qué-es-outlook-10f1fa35-f33a-4cb7-838c-a7f3e6228b20
71. Visual Studio: IDE y Editor de código para desarrolladores de software y Teams [Internet]. [cited 2023 May 13]. Available from: https://visualstudio.microsoft.com/es/
72. Microsoft Learn. Administración de grupos de recursos de Azure con Azure Portal [Internet]. 2023 [cited 2023 May 2]. Available from: https://learn.microsoft.com/es-es/azure/azure-resource-manager/management/manage-resource-groups-portal
73. Microsoft Learn. Manage resource groups - Azure CLI - Azure Resource Manager | Microsoft Learn [Internet]. 2023. [cited 2023 May 2]. Available from: https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/manage-resource-groups-cli

74. Microsoft Learn. Manage resource groups - Azure PowerShell - Azure Resource Manager | Microsoft Learn [Internet]. [cited 2023 May 2]. Available from: https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/manage-resource-groups-powershell
75. Microsoft Learn. Manage resource groups - Python - Azure Resource Manager | Microsoft Learn [Internet]. 2023 [cited 2023 May 2]. Available from: https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/manage-resource-groups-python?tabs=windows
76. Microsoft Learn. Guía de inicio rápido de Azure: Creación de un centro de eventos mediante Azure Portal [Internet]. [cited 2023 May 2]. Available from: https://learn.microsoft.com/es-es/azure/event-hubs/event-hubs-create
77. Microsoft Learn. Quickstart - Create an event hub using Azure CLI [Internet]. 2023 [cited 2023 May 2]. Available from: https://learn.microsoft.com/en-us/azure/event-hubs/event-hubs-quickstart-cli
78. Quickstart: Create an event hub using PowerShell [Internet]. [cited 2023 May 2]. Available from: https://learn.microsoft.com/en-us/azure/event-hubs/event-hubs-quickstart-powershell
79. Microsoft Learn. Quickstart: Create an event hub with consumer group using Bicep [Internet]. [cited 2023 May 2]. Available from: https://learn.microsoft.com/en-us/azure/event-hubs/event-hubs-bicep-namespace-event-hub?tabs=CLI
80. Microsoft Learn. Quickstart: Create an event hub by using an ARM template [Internet]. [cited 2023 May 2]. Available from: https://learn.microsoft.com/en-us/azure/event-hubs/event-hubs-resource-manager-namespace-event-hub
81. MicrosoftDocs. Features and terminology in Azure Event Hubs [Internet]. [cited 2023 May 2]. Available from: https://github.com/MicrosoftDocs/azure-docs/blob/main/articles/event-hubs/event-hubs-features.md
82. Microsoft. Authenticate access to Azure Event Hubs with shared access signatures [Internet]. 2023 [cited 2023 Feb 11]. Available from: https://learn.microsoft.com/en-us/azure/event-hubs/authenticate-shared-access-signature
83. Microsoft Learn. Authenticate an application to access Azure Event Hubs resources [Internet]. [cited 2023 May 13]. Available from: https://learn.microsoft.com/en-us/azure/event-hubs/authenticate-application

7. Anexos

7.1. Anexo 1 – Código de la aplicación

En este Anexo se recoge el código desarrollado para a la aplicación GarminToAzure. El objetivo de esta aplicación es el de recoger los datos generados por los relojes Garmin y enviarlos al Azure Event Hub. Esta aplicación también nos permite simular los datos necesarios para nuestras pruebas.

Este es el esquema de los ficheros de los que se compone nuestra solución:

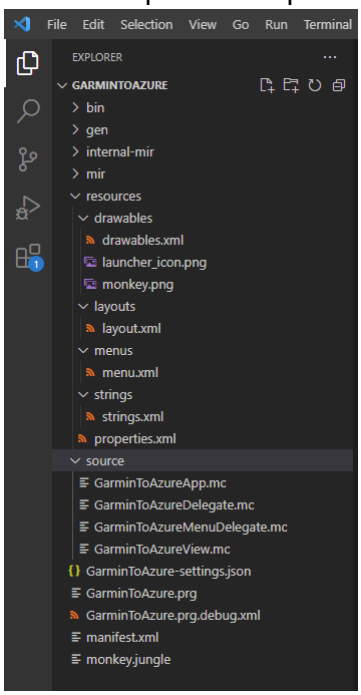

Figura 160: Estructura de los ficheros de los que se compone nuestra solución

Como podemos observar dentro de la carpeta resources, tenemos las 3 carpetas siguientes con los siguientes contenidos:

- Drawables:
  - Drawables.xml. Este fichero contiene la configuración del launcher icon que se muestra cuando se inicia la aplicación.

```
<drawables>
    <bitmap id="LauncherIcon" filename="launcher_icon.png" />
</drawables>
```

  - launcher_icon.png. Icono de la figura siguiente:

Figura 161: launcher_icon.png

- monkey.png. Icono de la figura siguiente:

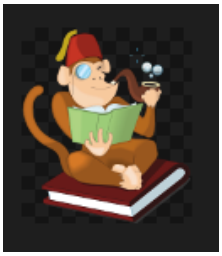

Figura 162: monkey.png

- Layouts
  - layout.xml: configuración del layout de nuestra aplicación. Esta configuración se traduce en el layout mostrado en la figura 115.

```
<layout id="MainLayout">
    <label x="center" y="170" text="@Strings.prompt" color="Graphics.COLOR_WHITE"
justification="Graphics.TEXT_JUSTIFY_CENTER" />
    <bitmap id="id_monkey" x="center" y="58" filename="../drawables/monkey.png" />
</layout>
```

Figura 163: Layout de la aplicación

- Menus
  - Menú.xml

```
<menu id="MainMenu">
    <menu-item id="item_1" label="@Strings.menu_label_1" />
    <menu-item id="item_2" label="@Strings.menu_label_2" />
</menu>
```

- Strings
  - String.xml. Se definen los strings que vamos a utilizar

```
<strings>
    <string id="AppName">GarminToAzure</string>

    <string id="prompt">Hello Azure</string>

    <string id="menu_label_1">Item 1</string>
    <string id="menu_label_2">Item 2</string>

    <string id="field1">Status</string>
    <string id="field2">xAccel</string>
    <string id="field3">yAccel</string>
    <string id="field4">HR</string>
</strings>
```

- Properties.xml. Se definen las propiedades y variables de la aplicación.

```
<resources>

    <properties>


        <property id="eHNamespace" type="string">connectiqnamespace</property>
        <property id="eHEventHub" type="string">garmineventhub</property>
        <property id="eHSASKeyName" type="string">garmin</property>
        <property id="eHSASKey" type="string">key </property>
        <property id="userID" type="string">simulator</property>
        <property id="timer" type="number">1000</property>
    </properties>

    <strings>
       <string id="eHNamespace_title">Your Namespace</string>
       <string id="eHEventHub_title">Your Event Hub</string>
       <string id="eHSASKeyName_title">SAS Key Name</string>
       <string id="eHSASKey_title">SAS Key</string>
       <string id="userID_title">User ID</string>
```

```
        <string id="timer_title">Timer Delay</string>
    </strings>

    <settings>
        <setting propertyKey="@Properties.eHNamespace"
title="@Strings.eHNamespace_title">
            <settingConfig type="alphaNumeric" />
        </setting>
        <setting propertyKey="@Properties.eHEventHub"
title="@Strings.eHEventHub_title">
            <settingConfig type="alphaNumeric" />
        </setting>
        <setting propertyKey="@Properties.eHSASKeyName"
title="@Strings.eHSASKeyName_title">
            <settingConfig type="alphaNumeric" />
        </setting>
        <setting propertyKey="@Properties.eHSASKey" title="@Strings.eHSASKey_title">
            <settingConfig type="alphaNumeric" />
        </setting>
        <setting propertyKey="@Properties.userID" title="@Strings.userID_title">
            <settingConfig type="alphaNumeric" />
        </setting>
        <setting propertyKey="@Properties.timer" title="@Strings.timer_title">
            <settingConfig type="numeric" />
        </setting>

    </settings>
</resources>
```

Dentro de la carpeta source, vamos a tener los siguientes ficheros:

- GarminToAzureApp.mc Se trata del archivo principal del código fuente de la aplicación y es el fichero utilizado para compilar la aplicación en un archivo binario ejecutable que se puede instalar y ejecutar en el dispositivo. El código de este fichero es el siguiente:

```
import Toybox.Application;
import Toybox.Lang;
import Toybox.WatchUi;
import Toybox.Position;

class GarminToAzureApp extends Application.AppBase {

    var GarminView;
```

```
    function initialize() {
        AppBase.initialize();
    }

    // onStart() is called on application start up
    function onStart(state as Dictionary?) as Void {
        //Position.enableLocationEvents(options, listener)
        //LOCATION_CONTINUOUS=>Enables continuous Location Tracking
        //onPosition es una funcion que se llama cuando cambia la posicion
        Position.enableLocationEvents(Position.LOCATION_CONTINUOUS, method(
:onPosition ));
    }

    // onStop() is called when your application is exiting
    function onStop(state as Dictionary?) as Void {
        //LOCATION_DISABLE=>Disables Location Tracking
        Position.enableLocationEvents(Position.LOCATION_DISABLE, method( :onPosition
));
    }


    function onPosition(info as  Position.Info) as Void{
        GarminView.setPosition(info);
    }

    // Return the initial view of your application here
    function getInitialView() as Array<Views or InputDelegates>? {
        //return [ new GarminToAzureView(), new GarminToAzureDelegate() ] as
Array<Views or InputDelegates>;
        GarminView= new GarminToAzureView();
        return [ GarminView, new GarminToAzureDelegate() ] as Array<Views or
InputDelegates>;
    }

}

function getApp() as GarminToAzureApp {
    return Application.getApp() as GarminToAzureApp;
}
```

- GarminToAzureDelegate.mc. Es el fichero empleado para definirlos controladores de eventos de la aplicación. Los controladores de eventos son funciones que se ejecutan automáticamente cuando se da un evento específico en la aplicación, como tocar un

botón o recibir una notificación. Este fichero permite la separara la lógica de la aplicación de la interfaz de usuario. El código de este fichero es el siguiente:

```
import Toybox.Lang;
import Toybox.WatchUi;

class GarminToAzureDelegate extends WatchUi.BehaviorDelegate {

    function initialize() {
        BehaviorDelegate.initialize();
    }

    function onMenu() as Boolean {
        WatchUi.pushView(new Rez.Menus.MainMenu(), new GarminToAzureMenuDelegate(),
WatchUi.SLIDE_UP);
        return true;
    }

}
```

- GarminToAzureMenuDelegate.mc.Este fichero se emplea para definir el controlador de eventos del menú principal de la aplicación. El código es el siguiente:

```
import Toybox.Lang;
import Toybox.System;
import Toybox.WatchUi;

class GarminToAzureMenuDelegate extends WatchUi.MenuInputDelegate {

    function initialize() {
        MenuInputDelegate.initialize();
    }

    function onMenuItem(item as Symbol) as Void {
        if (item == :item_1) {
            System.println("item 1");
        } else if (item == :item_2) {
            System.println("item 2");
        }
    }

}
```

- GarminToAzureView.mc. En este fichero definimos la interfaz de usuario y funciones auxiliares que nos permiten modificar el comportamiento del dispositivo. El código de la aplicación es el siguiente:

```
import Toybox.Graphics; // Para dibujar en la pantalla
import Toybox.WatchUi; // Controlar y crear vistas en la pantalla
import Toybox.Communications; //With the Communications module, widgets and apps will be able to communicate with a mobile phone via Bluetooth Low Energy (BLE). The mobile phone may be sharing data with the device, or it may act as a bridge between the app and the Internet. This allows the device to become part of the Internet of Things.
import Toybox.Lang;
import Toybox.Math;

class GarminToAzureView extends WatchUi.View {

    //var positionInfo;
    //set up display variables
    var status = "Waiting";
    var field2 = "0";
    var field3 = "0";
    var field4 = "0";
    var positionInfo = null;
    //set up the text labels
    hidden var dispField1;
    hidden var dispField2;
    hidden var dispField3;
    hidden var dispField4;

    //Set up the timer. La clase Timer se utiliza para ejecutar una tarea (método) de manera repetida en intervalos de tiempo específicos
    var dataTimer = new Timer.Timer();
    //Fill in this variable with the time interval in ms that you want to use for submitting data. Lower values will cause more calls so will cost more!
    var timer = 1000;

    var mySettings = System.getDeviceSettings();
    var publisher = mySettings.uniqueIdentifier;

    var contador=0;

    //IoT Hub
    //var sas="SharedAccessSignature sr=https%3A%2F%2FIotHubGarmin.azure-devices.net%2Fdevices%2F95d07e48105e043b14739885a4c18eb27a6114f2%2Fmessages%2Fevents%3Fapi-version%3D2021-09-01&sig=TkTZxnQ5Paz%2Bd85p5%2BzS48hVW9N42jfcQ96y67Wb6m0%3D&se=1681897985&skn=GarminSara";
    //var url = "https://" + Application.Properties.getValue("IoTHubName") + ".azure-devices.net/devices/" + publisher + "/messages/events?api-version=2021-09-01";
```

```
    //Event Hub
    var url =
"https://connectiqnamespace.servicebus.windows.net/garmineventhub/publishers/simulato
r/messages";
    var sas = "SharedAccessSignature
sr=https%3a%2f%2fconnectiqnamespace.servicebus.windows.net%2fgarmineventhub%2fpublish
ers%2fsimulator%2fmessages&sig=O4imNc4mvaMzGeDMdQ9%2fgIR%2boCIe0x2B9JQRx1vPokQ%3d&se=
1712529746&skn=garmin";


    function initialize() {
        View.initialize();

        var mySettings = System.getDeviceSettings();
        var publisher = mySettings.uniqueIdentifier;

        //url Azure IoT Hub
        //var url = "https://" + Application.Properties.getValue("IoTHubName") +
".azure-devices.net/devices/" + publisher + "/messages/events?api-version=2021-09-
01";
        //System.println(url);

        //url Azure Event Hub
        var url =
"https://connectiqnamespace.servicebus.windows.net/garmineventhub/publishers/garminsa
raforerunner245/messages";

        //Key Azure Event Hub
        var mySASKey = Application.Properties.getValue("eHSASKey");

        //key Azure IoT Hub
        //var mySASKey = Application.Properties.getValue("IoTHubSASKey");

        //Set up the SAS key for HMAC (Hash-Based Message Authentication Code)
encryption
        var keyConvertOptions = {
        :fromRepresentation => StringUtil.REPRESENTATION_STRING_PLAIN_TEXT,
        :toRepresentation => StringUtil.REPRESENTATION_BYTE_ARRAY,
        :encoding => StringUtil.CHAR_ENCODING_UTF8
        };
        //Convert SAS Key to Byte Array
        var mySASKeyByteArray = StringUtil.convertEncodedString(mySASKey,
keyConvertOptions);

        //Set up string to convert
```

```
        //current time
        var UTCNow = Time.now().value();
        //duration for key to last
        var duration = 2678400; //month
        var keyExpiry = (UTCNow + duration).toString();
        var stringToConvert = Toybox.Communications.encodeURL(url) + "\n" +
keyExpiry;
        var bytesToConvert = StringUtil.convertEncodedString(stringToConvert,
keyConvertOptions);

        //Set up HMAC (HashBasedMessageAuthenticationCode)
        var HMACOptions = {
            :algorithm => Cryptography.HASH_SHA256,
            :key => mySASKeyByteArray
        };

        var HMAC = new Cryptography.HashBasedMessageAuthenticationCode(HMACOptions);
        //convert the string
        HMAC.update(bytesToConvert);
        var encryptedBytes = HMAC.digest();

        HMAC.initialize(HMACOptions);
        //convert the string
        HMAC.update(bytesToConvert);
        encryptedBytes = HMAC.digest();

        var keyConvertOptions2 = {
            :fromRepresentation => StringUtil.REPRESENTATION_BYTE_ARRAY,
            :toRepresentation => StringUtil.REPRESENTATION_STRING_BASE64,
            :encoding => StringUtil.CHAR_ENCODING_UTF8
            };

        var myoutput = StringUtil.convertEncodedString(encryptedBytes,
keyConvertOptions2);

        //sas = "SharedAccessSignature sr=" + Toybox.Communications.encodeURL(url) +
"&sig=" + Toybox.Communications.encodeURL(myoutput) + "&se=" + keyExpiry + "&skn=" +
Application.Properties.getValue("eHSASKeyName");
        var sas = "SharedAccessSignature sr=" + Toybox.Communications.encodeURL(url)
+ "&sig=" + Toybox.Communications.encodeURL(myoutput) + "&se=" + keyExpiry + "&skn="
+ Application.Properties.getValue("IoTHubSASKeyName");
        //System.println(sas);


        // Set up a timer
```

```
        timer = Application.Properties.getValue("timer");
        dataTimer.start(method(:timerCallback), timer, true);


    }

    function timerCallback() {
        var sensorInfo = Sensor.getInfo();
        var xAccel = 0;
        var yAccel = 0;
        var hR = 0;
        var altitude = 0;
        var cadence = 0;
        var heading = 0;
        var xMag = 0;
        var yMag = 0;
        var zMag = 0;
        var power = 0;
        var pressure = 0;
        var speed = 0;
        var temp = 0;
        var latitude = 0;
        var longitude = 0;
        var userID = Application.Properties.getValue("userID");


        //Collect Data
        //Accelerometer
        if (sensorInfo has :accel && sensorInfo.accel != null) {
            var accel = sensorInfo.accel;
            xAccel = accel[0];
            yAccel = accel[1];
        }
        else {
            xAccel = 0;
            yAccel = 0;
        }
        //Heartrate
        if (sensorInfo has :heartRate && sensorInfo.heartRate != null) {

            hR = sensorInfo.heartRate;

        }
        else {
```

```
        //Generamos pulsaciones de forma aleatoria entre 180 y 250
        //hR=Math.rand()%70+180;

        //Generamos pulsaciones de forma aleatoria entre 50 y 7'
        hR=Math.rand()%20+50;


    }
    //Altitude
    if (sensorInfo has :altitude && sensorInfo.altitude != null) {
        altitude = sensorInfo.altitude;
    }
    else {
        altitude = 0;
    }
    //Cadence
    if (sensorInfo has :cadence && sensorInfo.cadence != null) {
        cadence = sensorInfo.cadence;
    }
    else {
        cadence = 0;
    }
    //heading
    if (sensorInfo has :heading && sensorInfo.heading != null) {
        heading = sensorInfo.heading;
    }
    else {
        heading = 0;
    }
    //magnetometer
    if (sensorInfo has :mag && sensorInfo.mag != null) {
        var mag = sensorInfo.mag;
        xMag = mag[0];
        yMag = mag[1];
        zMag = mag[2];
    }
    else {
        xMag = 0;
        yMag = 1;
        zMag = 2;
    }
    //Power
    if (sensorInfo has :power && sensorInfo.power != null) {
        power = sensorInfo.power;
    }
```

```
        else {
            power = 0;
        }
        //Pressure
        if (sensorInfo has :pressure && sensorInfo.pressure != null) {
            pressure = sensorInfo.pressure;
        }
        else {
            pressure = 0;
        }
        //Speed
        if (sensorInfo has :speed && sensorInfo.speed != null) {
            speed = sensorInfo.speed;
        }
        else {
            speed = 0;
        }
        //Temperature
        if (sensorInfo has :temp && sensorInfo.temp != null) {
            temp = sensorInfo.temp;
        }
        else {
            temp = 0;
        }
        //Position
        if( positionInfo != null ) {
            if (positionInfo has :position && positionInfo.position != null) {
                var location = positionInfo.position.toDegrees();
                latitude = location[0];
                longitude = location[1];
            }
        }
        else {
            latitude = 41.557849884033203;
            longitude = -8.397369384765625;
        }

        //Send the data to the REST API

        var params = {
            "userID" => userID,
            "heartRate" => hR.toNumber(),
            "xAccel" => xAccel.toNumber(),
            "yAccel" => yAccel.toNumber(),
            "altitude" => altitude.toFloat(),
```

```
            "cadence" => cadence.toNumber(),
            "heading" => heading.toFloat(),
            "xMag" => xMag.toNumber(),
            "yMag" => yMag.toNumber(),
            "zMag" => zMag.toNumber(),
            "power" => power.toNumber(),
            "pressure" => pressure.toFloat(),
            "speed" => speed.toFloat(),
            "temp" => temp.toNumber(),
            "latitude" => latitude.toFloat(),
            "longitude" => longitude.toFloat()
        };

        //set the display field variables
        field2 = xAccel.toString();
        field3 = yAccel.toString();
        field4 = hR.toString();

        var headers = {
            "Content-Type" => Communications.REQUEST_CONTENT_TYPE_JSON,
            "Authorization" => sas
        };
        var options = {
            :headers => headers,
            :method => Communications.HTTP_REQUEST_METHOD_POST,
            :responseType => Communications.HTTP_RESPONSE_CONTENT_TYPE_JSON
        };
        // Make the Communications.makeWebRequest() call

        Communications.makeWebRequest(url, params, options, method(:onReceive));


        //Troubleshooting
        //System.println("URL: " + url);
        //System.println("SAS Token: " + sas);
    }



    function onReceive(responseCode as Toybox.Lang.Number, data as Null or
Toybox.Lang.Dictionary or Toybox.Lang.String) as Void {
        //Uncomment for debug
        //System.println("Response code: " + responseCode);
        //System.println("Data: " + data);
```

```
        switch (responseCode.toString()) {
            case "-400":
                status = "OK";
                break;
            case "201":
                status = "OK";
                break;
            case "-101":
                status = "BLE Queue full";
                break;
            case "-104":
                status = "No Connection";
                break;
            case "--2":
                status = "Host Timeout";
                break;
            default:
                status = responseCode.toString();
                break;
        }

        WatchUi.requestUpdate();

    }



    // Load your resources here
    function onLayout(dc as Dc) as Void {
        setLayout(Rez.Layouts.MainLayout(dc));
    }

    // Called when this View is brought to the foreground. Restore
    // the state of this View and prepare it to be shown. This includes
    // loading resources into memory.
    function onShow() as Void {
    }

    // Update the view
    function onUpdate(dc) {
        View.onUpdate(dc);
        dc.clear();
        // Call the parent onUpdate function to redraw the layout
        dispField1 = new WatchUi.Text({
            :text=>status,
```

```
            :color=>Graphics.COLOR_BLACK,
            :font=>Graphics.FONT_LARGE,
            :locX =>WatchUi.LAYOUT_HALIGN_CENTER,
            :locY=>30,
            :justification=>Graphics.TEXT_JUSTIFY_CENTER
        });
        dispField2 = new WatchUi.Text({
            :text=>field2,
            :color=>Graphics.COLOR_BLACK,
            :font=>Graphics.FONT_LARGE,
            :locX =>60,
            :locY=>110,
            :justification=>Graphics.TEXT_JUSTIFY_CENTER
        });
        dispField3 = new WatchUi.Text({
            :text=>field3,
            :color=>Graphics.COLOR_BLACK,
            :font=>Graphics.FONT_LARGE,
            :locX =>180,
            :locY=>110,
            :justification=>Graphics.TEXT_JUSTIFY_CENTER
        });
        dispField4 = new WatchUi.Text({
            :text=>field4,
            :color=>Graphics.COLOR_BLACK,
            :font=>Graphics.FONT_LARGE,
            :locX =>WatchUi.LAYOUT_HALIGN_CENTER,
            :locY=>190,
            :justification=>Graphics.TEXT_JUSTIFY_CENTER
        });
       dispField1.draw(dc);
       dispField2.draw(dc);
       dispField3.draw(dc);
       dispField4.draw(dc);

    }

    //yo
    function setPosition(info) {
        positionInfo = info;
        WatchUi.requestUpdate();
    }

    // Called when this View is removed from the screen. Save the
    // state of this View here. This includes freeing resources from
```

```
    // memory.
    function onHide() as Void {
    }

}
```

El fichero manifest.xml es el fichero de configuración utilizado en aplicaciones Monkey C para definir la información básica de la aplicación, como el nombre, versión, permisos requeridos, producto, … El código de este fichero es el siguiente:

```
<?xml version="1.0"?>
<!-- This is a generated file. It is highly recommended that you DO NOT edit this
file. -->
<iq:manifest version="3" xmlns:iq="http://www.garmin.com/xml/connectiq">
    <!--
        Use "Monkey C: Edit Application" from the Visual Studio Code command palette
        to update the application attributes.
    -->
    <iq:application id="91b6f63f-4047-4a16-a811-688cbfab16fd" type="watch-app"
name="@Strings.AppName" entry="GarminToAzureApp"
launcherIcon="@Drawables.LauncherIcon" minApiLevel="3.3.0">
        <!--
            Use the following from the Visual Studio Code comand palette to edit
            the build targets:
            "Monkey C: Set Products by Product Category" - Lets you add all products
                                        that belong to the same product category
            "Monkey C: Edit Products" - Lets you add or remove any product
        -->
        <iq:products>
            <iq:product id="fr245"/>
        </iq:products>
        <!--
            Use "Monkey C: Edit Permissions" from the Visual Studio Code command
            palette to update permissions.
        -->
        <iq:permissions>
            <iq:uses-permission id="Sensor"/>
            <iq:uses-permission id="Positioning"/>
            <iq:uses-permission id="Communications"/>
        </iq:permissions>
        <!--
            Use "Monkey C: Edit Languages" from the Visual Studio Code command
```

```
            palette to edit your compatible language list.
        -->
        <iq:languages/>
        <!--
            Use "Monkey C: Configure Monkey Barrel" from the Visual Studio Code
            command palette to edit the included barrels.
        -->
        <iq:barrels/>
    </iq:application>
</iq:manifest>
```

7.2. Anexo 2 – Código de la function

```
#r "Newtonsoft.Json"

using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System.Net.Http;
using System.Text;

public static async Task<IActionResult> Run(HttpRequest req, ILogger log)
{
    // Registra un mensaje de información en el registro de la función.
    log.LogInformation("C# HTTP trigger function processed a request.");

    // Lee el cuerpo de la solicitud HTTP y lo almacena en la variable requestBody.
    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();

    // Deserializa el cuerpo de la solicitud en un objeto JArray llamado dataArray.
    JArray dataArray = JsonConvert.DeserializeObject<JArray>(requestBody);

    // Define la URL de la Logic App a la que se enviarán los datos
    string logicAppUrl = "https://prod-
206.westeurope.logic.azure.com:443/workflows/e21ca95e7f724ebba329a3751aa8368b/triggers/
manual/paths/invoke?api-version=2016-10-
01&sp=%2Ftriggers%2Fmanual%2Frun&sv=1.0&sig=tUqK3GyK3gYCNfdOzSXft4II5u-Z_4YMAxh5QkF-N4M";

    // Crea una instancia de HttpClient para realizar solicitudes HTTP
    using (var client = new HttpClient())
    {
        // Itera sobre cada objeto JObject en dataArray
        foreach (JObject data in dataArray)
        {
            // Convierte el objeto JObject en una cadena JSON y crea un objeto StringContent
con la codificación UTF-8 y el tipo de contenido "application/json"
            var content = new StringContent(JsonConvert.SerializeObject(data), Encoding.UTF8,
"application/json");

            // Envía una solicitud POST a la URL de la Logic App con el contenido JSON
            var response = await client.PostAsync(logicAppUrl, content);

            // Verifica si la respuesta no indica un estado exitoso
            if (!response.IsSuccessStatusCode)
```

```
            {
                // Registra un mensaje de error en el registro de la función si la respuesta
 no fue exitosa
                log.LogError($"Error sending data to Logic App: {response.StatusCode}");
            }
        }
    }

    return new OkResult();
}
```