E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Desarrollo de una plataforma FIWARE para la implementación de un dispositivo IoT

Grado en Ingeniería
en Tecnologías Industriales

Trabajo Fin de Grado

Asier Martínez San Esteban

Ángel María Andueza Unanua

Pamplona, 08/06/2023

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Trabajo Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Desarrollo de una plataforma FIWARE para la implementación real de un dispositivo IoT

Autor:
Asier Martínez San Esteban

Tutor:
Ángel María Andueza Unanua

Dpto. Ingeniería eléctrica, electrónica y de comunicación

E.T.S. de Ingenierías Informática y de Telecomunicación

Universidad Pública de Navarra

Pamplona, 8 de junio de 2023

Resumen

En este documento se pretende mostrar el proceso completo de creación de una plataforma FIWARE que permita la implementación real y funcional de un dispositivo IoT. Primero se lleva a cabo una revisión del estado del arte, mostrando las bases teóricas necesarias para poder seguir el proyecto y seguidamente se procede con el asunto principal, la creación paso a paso de la plataforma. Existen, como se puede ver, diferentes etapas en este proceso; comenzando por el diseño y la definición de funcionalidades, los componentes empleados y la interconexión entre los mismos. Se documentan las diferentes pruebas realizadas y los resultados obtenidos, mostrando el correcto funcionamiento del proyecto; con el objetivo principal de demostrar la utilidad de este tipo de plataforma y su potencial escalabilidad.

Finalmente, en los anexos se encuentran las guías de instalación de cada componente software, los *scripts* completos utilizados y el contenido de los archivos de configuración.

Palabras clave: plataforma FIWARE, Node-RED, FIWARE, Orion, Draco, MySQL, dispositivo IoT, Docker, Postman, ESP32, entidad, IoT Agent, suscripción, notificaciones, sistema de alerta, máquina virtual, Arduino, JSON, HTTP, *Docker-compose*.

Abstract

This document aims to show the complete process of creating a FIWARE platform that allows for the real and functional implementation of an IoT device. First, a review of the state of the art is carried out, showing the necessary theoretical foundations to be able to follow the project, and then the main subject, the step-by-step creation of the platform, is addressed. As can be seen, there are different stages in this process; starting with the design and definition of functionalities, the components used, and the interconnection between them. The different tests carried out and the results obtained are documented, showing the correct functioning of the project, with the main objective of demonstrating the usefulness of this type of platform and its potential scalability.

Finally, in the annexes, the installation guides for each software component, the complete scripts used, and the contents of the configuration files are included.

Keywords: FIWARE platform, Node-RED, FIWARE, Orion, Draco, MySQL, IoT device, Docker, Postman, ESP32, entity, IoT Agent, subscription, notifications, alert system, virtual machine, Arduino, JSON, HTTP, Docker-compose.

Índice

Resumen ***V***
Abstract ***V***
Índice ***VII***
Índice de Tablas ***IX***
Índice de Figuras ***X***
Glosario ***XIII***
1. Motivación ***1***
2. Objetivos ***1***
3. Introducción ***2***
3.1. Descripción del proyecto *2*
3.2. Organización del proyecto y plan de trabajo *4*
3.2.1. Fase I. Búsqueda de información. Análisis de funcionalidades *4*
3.2.2. Fase II. Primeros pasos. Envío de datos *4*
3.2.3. Fase III. Recepción de datos y pruebas *5*
3.2.4. Fase IV. Otras funcionalidades *6*
3.2.5. Fase V. Fase final *7*
4. Estado del arte ***8***
4.1. Internet of Things *8*
4.2. FIWARE *9*
4.2.1. Orion Context Broker *9*
4.2.2. Arquitectura y comunicación *10*
4.2.3. MongoDB *11*
4.2.4. IoT Agent *12*
4.2.5. Draco *13*
4.2.6. ¿Por qué FIWARE? *13*
4.3. Recursos software *14*
4.3.1. Máquina virtual *14*
4.3.2. Docker *15*
4.3.3. Postman *16*
4.3.4. Node-RED *17*
4.3.5. Otros programas *17*
5. Diseño de la plataforma ***19***
5.1. Hardware empleado *19*
5.1.1. Sensor DHT11 *20*
5.1.2. Placa ESP32 *20*
5.2. Software necesario *21*
5.2.1. Recogida de datos *22*
5.2.2. Formato JSON. *23*
5.2.3. Envío por WiFi de datos mediante protocolo HTTP *24*
5.3. Configuración de red *26*
5.3.1. Tipos de configuraciones *26*
5.3.2. Configuración empleada *27*
5.4. Prueba de comunicación *30*
6. FIWARE ***34***
6.1. Despliegue de MongoDB y OCB *34*
6.2. Entidades *38*
6.2.1. Estructura *38*
6.2.2. Cabeceras *39*
6.3. Despliegue de IoT Agent *40*
6.4. Aprovisionamiento *42*

6.4.1. Grupo de servicio 42
6.4.2. Dispositivo 43
6.5. Envío de datos a la plataforma FIWARE 44
6.6. Datos persistentes 45
6.6.1. Arquitectura 45
6.6.2. Configuración 46
6.6.3. Ver la base de datos MySQL 47
6.7. Suscripciones 49
6.7.1. Draco 50
6.7.2. Node-RED 51
7. Visualización Power BI 52
7.1. MySQL Workbench 52
7.2. Despliegue y configuración 52
7.3. Configuración de las fuentes de datos 53
7.4. Dashboard 55
8. Sistema de alertas y notificaciones 57
8.1.1. Recepción de los datos en Node-RED 57
9. Resultados y discusión 62
9.1. Pruebas realizadas 62
9.2. Resultados obtenidos 63
9.2.1. Envío y recepción de los datos provenientes del sensor. 63
9.2.2. Correcto almacenamiento de la información 64
9.2.3. Visualización de datos actualizados 65
9.2.4. Avisos cuando ocurre un evento 65
9.3. Tiempos 66
9.4. Revisión de seguridad 67
9.5. Requisitos para fase de producción 67
10. Conclusiones y líneas futuras 68
10.1. Conclusiones 68
10.2. Líneas futuras 69
11. Referencias 70
Anexo A. Instalación y despliegue 72
A.1 Instalación de una máquina virtual 72
A.2 Instalación de Docker 74
A.3 Instalación de Draco 76
A.4 Instalación de Arduino IDE 80
A.4.1 Gestión de Placas 80
A.4.2 Librerías necesarias 82
A.5 Instalación de Node-RED 84
A.6 Instalación de Postman 87
A.7 Instalación de MySQL Workbench 87
A.8 Instalación de Power BI 88
Anexo B. Scripts de Arduino 89
B.1 Recogida de datos con sensor DHT11 89
B.2 Implementación lenguaje JSON 90
B.3 Conexión WiFi y comunicación HTTP 91
B.4 Script completo 92
Anexo C. Scripts Node-RED 94
C.1 Script para prueba de comunicación 94
C.1 Script final Node-RED 94
Anexo D. Configuración de archivo docker-compose.yml 96

Índice de Tablas

Tabla 3-A Recuento y comparación Fase I *4*
Tabla 3-B Recuento y comparación Fase II. *5*
Tabla 3-C Recuento y comparación Fase III *5*
Tabla 3-D Recuento y comparación Fase IV. *6*
Tabla 3-E Recuento y comparación Fase V. *7*
Tabla 3-F Recuento de horas y comparación final. *7*

Índice de Figuras

Figura 3.1 Estructura del proyecto por partes. 2
Figura 3.2 Arquitectura del proyecto al completo. 3
Figura 3.3 Diagrama resumen de fases e hitos. 4
Figura 4.1 Esquema básico OCB. 9
Figura 4.2 Arquitectura FIWARE completa. 11
Figura 4.3 Arquitectura y comunicación de IoT Agent. 12
Figura 4.4 Logo FIWARE 13
Figura 4.5 Representación de una máquina virtual. 14
Figura 4.6 Comparativa del nivel de funcionamiento entre Docker y una máquina virtual 15
Figura 4.7 Logo de Postman. 16
Figura 4.8 Interfaz para la creación de peticiones. 16
Figura 4.9 Logo de Node-RED. 17
Figura 4.10 Flujo de Node-RED. 17
Figura 4.11 Logo Arduino IDE. 18
Figura 4.12 Logo MySQL Workbench 18
Figura 4.13 Logo Power BI Desktop. 18
Figura 5.1 Conexión del dispositivo IoT. 19
Figura 5.2 Distribución de pines del sensor DHT11 en su versión PCB 20
Figura 5.3 Distribución de pines ESP32_C3_DevKitM_1 . 21
Figura 5.4 Acceso a la Configuración de Red. 26
Figura 5.5 Conexiones de red cuando el punto de acceso WiFi está desactivado. 27
Figura 5.6 Conexiones de red cuando el punto de acceso WiFi está activado. 28
Figura 5.7 Detalles de la conexión de red seleccionada. 28
Figura 5.8 Configuración del Adaptador 1 de la máquina virtual. 28
Figura 5.9 Configuración del Adaptador 2 de la máquina virtual. 29
Figura 5.10 Configurar IP fija en la máquina virtual. 29
Figura 5.11 Diagrama de red. 30
Figura 5.12 Flujo Node-RED para prueba de comunicación. 31
Figura 5.13 Función que implementa el bloque JSON or URL Encoded. 32
Figura 5.14 Importar nodos en Node-RED. 32
Figura 5.15 Dashboard Node-RED. 32
Figura 5.16 Configuración de los tres indicadores y del dashboard general. 33
Figura 6.1 Estructura de plataforma FIWARE 34
Figura 6.2 Comprobación del funcionamiento de Orion y MongoDB. 37
Figura 6.3 Cabeceras empleadas. 39
Figura 6.4 GET para ver la entidad creada. 39
Figura 6.5 Arquitectura y puertos del entorno FIWARE. 40
Figura 6.6 Comprobación del funcionamiento del IoT Agent. 41
Figura 6.7 Petición GET para observar cambios en los valores de la entidad. 44
Figura 6.8 Diagramas y puertos de la arquitectura al completo. 45
Figura 6.9 Ejecutar un cliente MySQL 47

Figura 6.10 Bases de datos por defecto. Draco inactivo. 48
Figura 6.11 Nueva base de datos "openiot" al activar el servicio. 48
Figura 6.12 Mostrar las tablas de la base de datos "openiot". 48
Figura 6.13 Tabla que contiene los datos enviados por el sensor. 48
Figura 6.14 Suscribir Draco a los cambios de contexto. 50
Figura 7.1 Menú de inicio Power BI. 52
Figura 7.2 Instalación de conector adicional. 53
Figura 7.3 Menú de obtención de datos. 53
Figura 7.4 Conectar base de datos MySQL. 53
Figura 7.5 Base de datos en bruto. 54
Figura 7.6 Base de datos formateada. 54
Figura 7.7 Panel para visualización de la información. 55
Figura 7.8 Creación de panel con filtros. 55
Figura 7.9 Creación de panel de histórico. 56
Figura 8.1 Flujo de Node-RED completo. 57
Figura 8.2 Recepción del mensaje HTTP de Orion. 57
Figura 8.3 Mensaje JSON recibido por Node-RED. 58
Figura 8.4 Bloques para la extracción de los valores. 58
Figura 8.5 Valores de temperatura y humedad extraídos del mensaje JSON y mostrados en el debug. 59
Figura 8.6 Contenido de ambos bloques. 59
Figura 8.7 Últimos bloques del flujo. 59
Figura 8.8 Contenido de ambos bloques. 60
Figura 8.9 Mensaje obtenido en el debug. 60
Figura 8.10 Contenido del bloque gmail notif. 60
Figura 8.11 Prueba de envío de mensajes Whatsapp a través de Node-RED. 61
Figura 8.12 Alerta recibida al correo electrónico. 61
Figura 9.1 Información en el monitor serie confirma que el envío y la recepción son correctos. 63
Figura 9.2 Entidad con valores actualizados. 64
Figura 9.3 MySQL Workbench para verificar creación y existencia de la base de datos. 64
Figura 9.4 Datos actualizados. 65
Figura 9.5 Recepción de correo electrónico. 65
Figura 9.6 Diagrama de tiempos. 66
Figura A. 1. Descargar VirtualBox. 72
Figura A. 2. Descargar imagen de Ubuntu. 72
Figura A. 3. Instalación Visual C++ 2022. 73
Figura A. 4. Crear una nueva máquina virtual en VirtualBox. 73
Figura A. 5 Portal Apache NiFi. Opción Template 76
Figura A. 6 Template MYSQL_TUTORIAL. 76
Figura A. 7 Configuración general. 77
Figura A. 8 Configuración - Controller services. 77
Figura A. 9 Credenciales como los de la configuración MySQL. 77
Figura A. 10 Lanzar el controlador. 77

Figura A. 11 Se configura el procesador NGSIToMySQL. 78
Figura A. 12 Flujo lanzado. 78
Figura A. 13 Ver flujo de datos. 79
Figura A. 14 Ver datos. 79
Figura A. 15 Configuración de las preferencias. 80
Figura A. 16 Insertar URL`s adicionales 81
Figura A. 17 Gestor de placas. 81
Figura A. 18 Selección de placa 81
Figura A. 19 Librerías empleadas. 82
Figura A. 20 Descargar librería para HTTP. 83
Figura A. 21. Ubicación librería HTTPClient.h 83
Figura A. 22. Ubicación archivos de configuración. 84
Figura A. 23. Carpeta personal en Ubuntu. 84
Figura A. 24. Apartado Server Settings de archivo settings.js. 85
Figura A. 25. Apartado Security de archivo settings.js. 85
Figura A. 26. Log de inicio de servidor. 85
Figura A. 27. Seleccionar opción Manage palette. 86
Figura A. 28. Librerías Node-RED 86
Figura A. 29. Instalación de Postman. 87
Figura A. 30. Menú de inicio de MySQL Workbench. 87
Figura A. 31. Configurar una nueva conexión. 88
Figura A. 32. Conexión con base de datos configurada con éxito. 88
Figura B. 1 Output al ejecutar el script anterior, obteniendo valores del sensor DHT11. 89
Figura B. 2 Output al ejecutar el script anterior, implementando JSON. 90
Figura B. 3 Output al ejecutar el script anterior, IP asignada por conexión a WiFi. 91
Figura B. 4 Output al ejecutar el script completo, comunicación efectiva. 93

Glosario

IoT: Internet of Things (internet de las cosas)

OCB: Orion Context Broker

SQL: Structured Query Language (lenguaje de consulta estructurada)

PC: Personal Computer

SO: Sistema operativo

MV: Máquina virtual

SoC: System on a Chip

API: Application programming interface (interfaz de programación de aplicaciones)

HTTP: Hypertext Transfer Protocol (protocolo de transferencia de hipertexto)

REST: Representational State Transfer (transferencia de estado representacional)

JSON: JavaScript Object Notation (notación de objetos de JavaScript)

OMA: Open Mobile Alliance

NGSI: Next Generation Service Interface

XML: Extensible Markup Language (lenguaje de marcado extensible)

1. Motivación

Este Trabajo de Fin de Grado se ha llevado a cabo durante mi estancia en **NAITEC**, empresa en la cual a lo largo de este octavo cuatrimestre he realizado mis **prácticas:** "Desarrollo de plataforma de datos FIWARE para aplicación de Smart City". Esta plataforma almacena información de sensores colocados en diferentes ubicaciones para su posterior análisis y utilización, por ejemplo, en un modelo de tráfico predictivo. El servidor empleado se aloja en la red privada de NAITEC y al estar la plataforma ya implementada, no es buena idea comenzar a hacer pruebas y aprender en este entorno. Un pequeño error podría tirar por tierra fácilmente el trabajo de mis compañeros.

Así pues, con la idea de conocer este tipo de plataformas a fondo y llevar a la realidad los conceptos e ideas que se plantean desde el comienzo, decidí junto con mi tutor que lo mejor sería utilizar un **ejemplo** a parte e ir desarrollando la plataforma propia paso a paso; desde las primeras ideas hasta las últimas funcionalidades.

Es entonces cuando surge este proyecto, con la idea de implementar un sensor de temperatura y humedad junto con una ESP32 en una plataforma FIWARE creada desde cero; con el objetivo de conseguir funcionalidades interesantes como un sistema de alertas con notificaciones útiles y poder mostrar la información gráficamente.

2. Objetivos

El objetivo de este trabajo no es solamente conseguir que el proyecto funcione de la forma deseada, sino también demostrar con él la simplicidad y el potencial de escalabilidad que presentan las plataformas FIWARE. Para ello se emplea un ejemplo sencillo en términos de hardware, ya que una vez se desarrolla e implementa de manera funcional la estructura de comunicación base, escalar el proyecto se convierte en una cuestión de tiempo, estandarización y añadir gradualmente elementos al modelo básico. Por lo tanto, uno de los objetivos es adquirir los conocimientos necesarios para poder crear una plataforma FIWARE básica, identificar los problemas y dificultades, y proponer soluciones.

Además, se busca continuar y consolidar los estándares utilizados en este tipo de plataformas en términos de nomenclatura y arquitectura, y también crear nuevos estándares dentro del proyecto con normas que faciliten la escalabilidad de este, garantizando un crecimiento ordenado y fácil de comprender para un nuevo usuario que no esté familiarizado con FIWARE.

No obstante, el requisito fundamental e indispensable es llevar a cabo el proyecto y conseguir un resultado óptimo con las funcionalidades deseadas y de esta forma probar la simplicidad y utilidad de las plataformas FIWARE. Esto implica implementar y documentar adecuadamente la estructura de comunicación requerida, realizar y documentar las pruebas necesarias, e incluir funcionalidades innovadoras que puedan ser implementadas en otros proyectos de la empresa, como sistemas de alarma y notificaciones mediante correo electrónico o WhatsApp.

Por último, se pretende revisar el apartado de seguridad de la plataforma y comentar los requisitos necesarios para escalar una plataforma de estas características a una fase de producción.

3. Introducción

La siguiente memoria contiene toda la información necesaria para implementar un dispositivo IoT en una plataforma FIWARE. Se detalla cada concepto y proceso llevado a cabo, desde la captura de datos hasta su uso de manera inteligente. Cada etapa y subproceso se analizan en profundidad para conocer en detalle el proceso de implementación, los problemas que han surgido y cómo se han resuelto.

3.1. Descripción del proyecto

De manera sencilla, el proyecto consiste en la creación de una plataforma capaz de recoger y almacenar los datos recogidos por un sensor ambiental. Estos datos se podrán mostrar de manera gráfica, servirán para generar un histórico, y con ellos se podrá construir un sistema de alarmas y avisos en caso de que se cumplan ciertos patrones.

Para ello, el proceso de desarrollo comienza con un simple sensor y termina con un aviso al correo electrónico cuando los datos muestran ciertos patrones. Así pues, se distinguen cuatro partes principales en el desarrollo técnico del proyecto, tal como se muestra en la Figura 3.1.

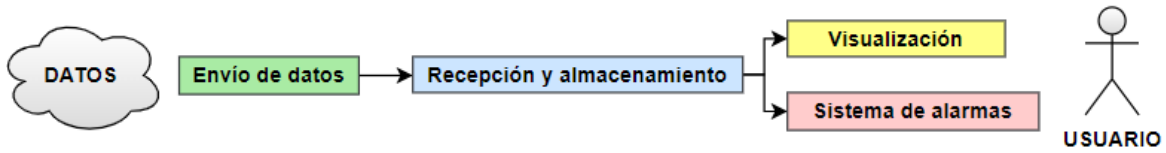


Figura 3.1 Estructura del proyecto por partes.

- *Envío de datos:* se emplea un sensor de temperatura y humedad (DHT11) para recoger los valores de estas dos variables. Para hacerlo funcionar se conecta a una placa de desarrollo basada en un ESP32 (tiene conectividad WiFi) formando un dispositivo IoT, con el que se logra enviar la información vía WiFi mediante protocolo HTTP.

- *Recepción y almacenamiento:* la información llega a la plataforma FIWARE, y concretamente la recibe un componente llamado IoT Agent, que es capaz de extraer los datos relevantes del mensaje y actualizar una entidad creada en el gestor de contexto Orion. Los datos se almacenan mediante otro componente llamado Draco en una base de datos MySQL.

- *Visualización:* se pretende mostrar la información del sensor mediante gráficos empleando el conocido programa Power BI.

- *Sistema de alarmas:* cuando los datos cumplan una serie de normas establecidas, se lanzan notificaciones por correo electrónico y *Whatsapp* para avisar a quien sea necesario.

Hasta ahora, todo esto puede resultar confuso y complejo. A lo largo del documento se irá explicando cada parte y componente de forma que se pueda comprender de manera sencilla cada fase.

A modo de resumen, el proyecto consiste en implementar una **plataforma** que permita disponer de la **información actualizada** de uno o varios dispositivos de manera controlada. Con un mayor nivel de profundidad, la Figura 3.2 muestra la arquitectura completa de la plataforma y cómo los diferentes componentes interaccionan entre sí para lograr el objetivo deseado. Como se puede observar, se respetan las diferentes partes mencionadas anteriormente, profundizando en cada una de ellas.

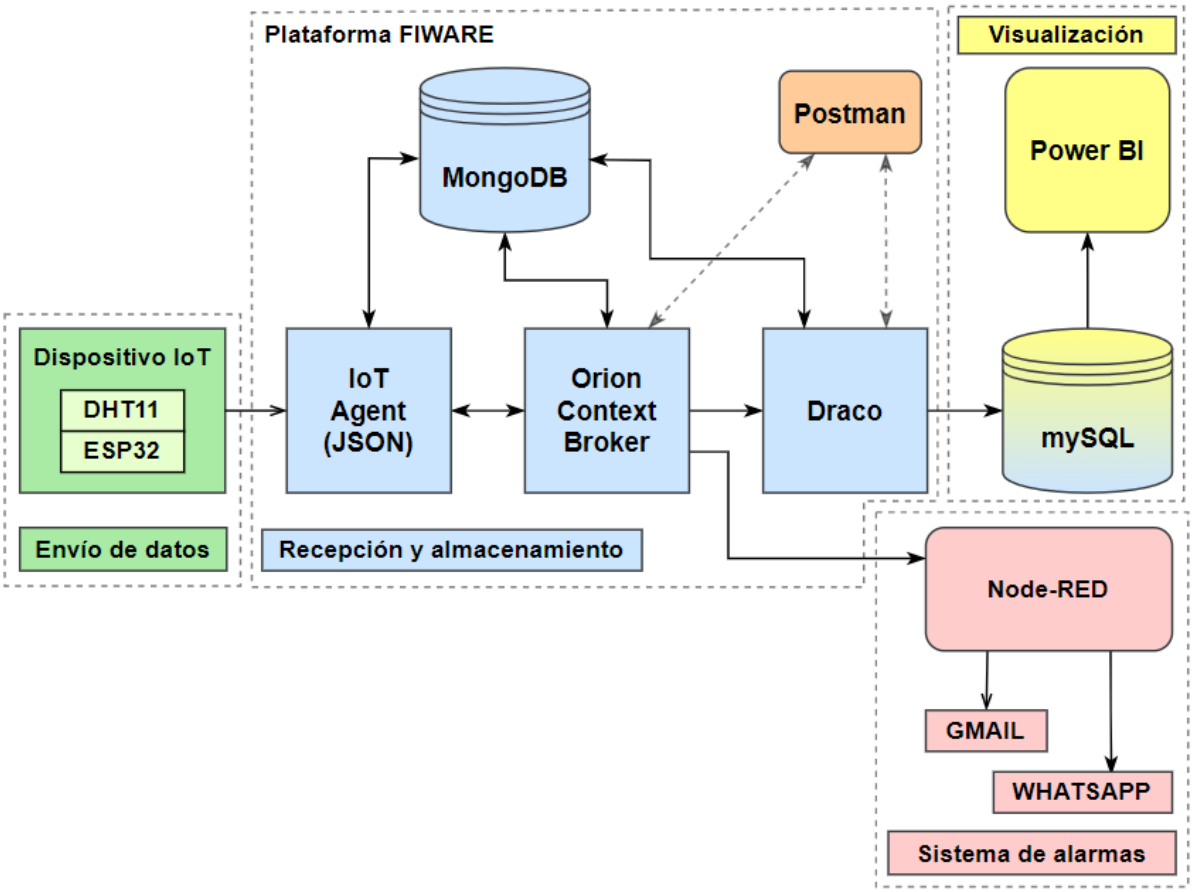


Figura 3.2 Arquitectura del proyecto al completo.

- *Dispositivo IoT:* formado por un sensor (DHT11) y una placa con conexión WiFi (ESP32) que envía datos de temperatura y humedad gracias a un *script* específico de Arduino mediante protocolo HTTP.

- *Plataforma FIWARE:* recibe y gestiona la información proveniente del sensor.
  - *IoT Agent*: recibe la información del sensor, la "transcribe" al protocolo NGSI-v2 y la suministra al Orion Context Broker.
  - *Orion Context Broker:* gestiona la información de contexto y el acceso a ésta.
  - *MongoDB:* almacena la información de configuración de los componentes de la plataforma, pero no de los datos como tal.
  - *Draco:* componente que se suscribe al gestor de contexto para recibir avisos cuando se produzcan actualizaciones o cambios en la información y actualizar la base de datos.
  - *Postman:* no es un componente FIWARE como tal. Envía y recibe peticiones HTTP.

Para la persistencia de los datos recogidos por el sensor es necesaria una base de datos externa, en este caso MySQL. Se conecta con la aplicación de visualización.

- *Visualización:* se emplea **Power BI** para visualizar la información.

- *Alarma:* se gestionan diferentes alarmas con distintos tipos de avisos mediante Node-RED.

Los únicos componentes hardware son el PC, el sensor y la placa, que conforman el dispositivo IoT. El resto son componentes software que interactúan entre sí al estar correctamente configurados, proporcionando una **plataforma capaz de gestionar la información** recogida por el dispositivo IoT.

3.2. Organización del proyecto y plan de trabajo

En este apartado se van a comentar las actividades llevadas a cabo para la realización del proyecto, incluyendo todo tipo de tareas realizadas en los diferentes ámbitos y el tiempo dedicado a cada una de ellas.

El proyecto se divide en diferentes **fases** con el objetivo de establecer una línea temporal y funcional para el desarrollo del proyecto, como se muestra en la Figura 3.3. Se establecen una serie de **hitos** necesarios para el desarrollo del proyecto, que sirven de indicadores de progreso.

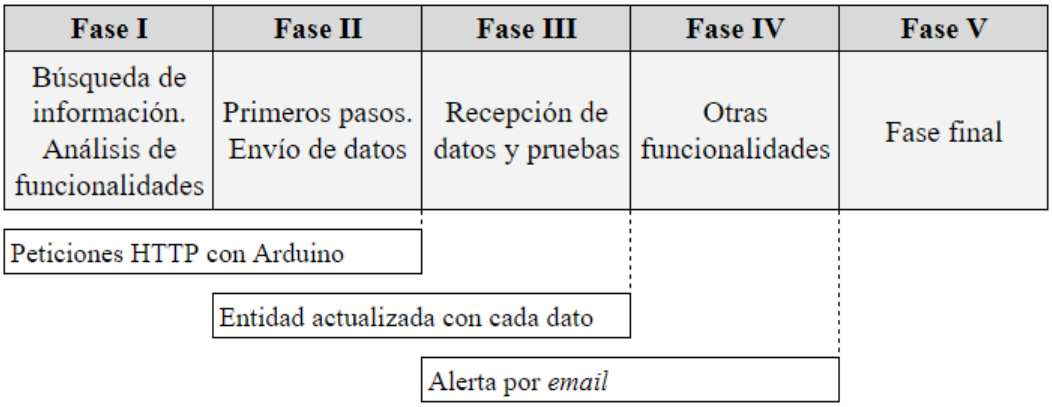


Figura 3.3 Diagrama resumen de fases e hitos.

3.2.1. Fase I. Búsqueda de información. Análisis de funcionalidades

La primera fase de este proyecto consiste en buscar información para adquirir los conocimientos previos necesarios para poder entender el concepto de *plataforma FIWARE* que será empleada. Se realiza una previsión de las funcionalidades que se van a implementar en el proyecto para focalizar la búsqueda de información y continuar buscando documentación específica de FIWARE y sus componentes.

Búsqueda de información. Análisis de funcionalidades	Tiempo estimado	Tiempo real
Información básica previa	8	8
Definición de funcionalidades	8	6
Documentación específica FIWARE	8	6
TOTAL	**24**	**20**

Tabla 3-A Recuento y comparación Fase I

3.2.2. Fase II. Primeros pasos. Envío de datos

Esta fase consiste en dar los primeros pasos en la implementación del proyecto, comenzando por la configuración del equipo y la máquina virtual. Se instala el IDE de Arduino y se procede a conectar el dispositivo IoT con el PC. Al no ser una placa oficial Arduino esto puede acarrear problemas. Esta fase finaliza al conseguir implementar las funciones principales en el IDE de Arduino, desde la recogida de datos, formato necesario y comunicación HTTP viable.

Primeros pasos. Envío de datos	**Tiempo estimado**	**Tiempo real**
Configuración de entorno		
Instalación de máquina virtual	6	4
Instalación de Arduino	2	1
TOTAL	**8**	**5**
Conexión Hardware		
Configuración de la placa	4	7
Instalación de librerías	4	6
TOTAL	**8**	**13**
***Scripts* Arduino**		
Script para mostrar datos DHT11 en formato ultralight	4	3
Implementar JSON	4	3
Implementar comunicación HTTP	8	20
TOTAL	**16**	**26**

Tabla 3-B Recuento y comparación Fase II.

3.2.3. Fase III. Recepción de datos y pruebas

La fase dos finaliza al conseguir enviar peticiones HTTP desde la tarjeta ESP32 programada con el IDE de Arduino. Lo próximo es recibir estas peticiones y ver los datos. Para ello, primero se realiza una prueba en Node-RED ya que es un entorno más sencillo que no requiere configurar Linux. Una vez validado el *script* de Arduino, al verificar la recepción de datos en el formato correcto en Node-RED, la siguiente tarea es implementar la plataforma FIWARE en la máquina virtual.

Recepción de datos y pruebas	**Tiempo estimado**	**Tiempo real**
Prueba Node-RED		
Configuración de Red	12	24
Configuración del panel de Node-RED	5	12
Librerías Node-RED	5	6
TOTAL	**22**	**42**
Implementación de componentes FIWARE		
Docker en Linux	5	8
Instalación y aprendizaje Postman	6	10
Configuración *docker-compose.yml*	8	20
Configuración entidades	8	18
Provisionar IoTA	8	12
TOTAL	**35**	**68**

Tabla 3-C Recuento y comparación Fase III

Una vez finalizada la Fase III se da por finalizada la parte de comunicación, superando todos los impedimentos que pueda presentar esto. La parte más importante del proyecto estará en principio configurada y funcionando, por lo que éste es un *checkpoint* importante que coincide con uno de los tres hitos principales definidos al principio.

3.2.4. Fase IV. Otras funcionalidades

A continuación, se procede con la Fase IV, que consiste en implementar funcionalidades interesantes que permitan aprovechar la información obtenida por el sensor. La idea es desarrollar e implementar nuevas funcionalidades que en un futuro se puedan añadir a la plataforma FIWARE de Naitec.

Las dos principales funcionalidades son la visualización de los datos con una herramienta nueva, Power BI y el sistema de alertas. Se busca implementar Power BI como plataforma para mostrar la información ya que esta tiene multitud de herramientas visuales en este ámbito. Además, posibilita el desarrollo de aplicaciones móviles para gestión de alertas, lo cual puede ser una opción interesante.

Las principales funcionalidades que se quieren implementar para la gestión de alertas son el envío de *email* o *SMS,* y el envío de un mensaje por *Whatsapp* si se producen una serie de condiciones determinantes.

Otras funcionalidades	Tiempo estimado	Tiempo real
Datos persistentes		
Suscribir Draco y configurar Apache NiFi	8	15
Configuración MySQL	8	17
TOTAL	**16**	**32**
Visualización de datos con Power BI		
Instalación MySQL Workbench	1	1
Instalación Power BI	1	1
Conexión de Power BI con MySQL	8	10
Creación de panel	6	8
TOTAL	**16**	**20**
Alertas		
Envío de datos de Orion a Node-RED	8	14
Implementación de alertas	6	12
TOTAL	**14**	**26**

Tabla 3-D Recuento y comparación Fase IV.

3.2.5. Fase V. Fase final

La fase final del proyecto consiste principalmente en la revisión de seguridad general, comentando las principales debilidades y todas las herramientas de autenticación no implementadas (por comodidad en el desarrollo y facilitar el manejo). Se va a llevar a cabo también una revisión de los requisitos indispensables para poder escalar una plataforma de este estilo a un nivel de producción.

Finalmente, las tareas de redacción y preparación de los entregables.

Fase final	Tiempo estimado	Tiempo real
Revisión de la solución		
Revisión de seguridad en cada elemento	2	4
Requisitos para fase de producción	3	3
TOTAL	**5**	**7**
Redacción y maquetado		
Redacción de la memoria	130	150
Preparación de la presentación	10	15
TOTAL	**140**	**165**

Tabla 3-E Recuento y comparación Fase V.

A modo de resumen y para observar la diferencia entre las horas planteadas en un inicio y las horas reales invertidas en este proyecto, se recogen a continuación los totales.

Resumen	Tiempo estimado	Tiempo real
Suma y comparación	**304**	**424**

Tabla 3-F Recuento de horas y comparación final.

4. Estado del arte

En este apartado se va a justificar el uso y describir las diferentes herramientas empleadas a lo largo de todo el proyecto con el objetivo de conocerlas previamente para poder entender el documento. Se exponen las características principales y comentan otras alternativas.

4.1. Internet of Things

Cada vez es más común escuchar este término en lo que a comunicaciones y nuevas tecnologías se refiere. En castellano, "Internet de las Cosas"; hace referencia a la interconexión de numerosos dispositivos, objetos de uso diario que incorporan sensores capaces de recabar y transmitir información relevante. De esta forma se generan **redes de transmisión de datos** para el intercambio de esta información con poca o nula intervención humana, redes autónomas e inteligentes cuyo objetivo es la recopilación y análisis de grandes cantidades de información con multitud de **aplicaciones útiles**, como la mejora en la eficiencia y funcionalidad de objetos y sistemas. [1]

El impacto de esta tecnología en la sociedad actual ha sido relevante en los últimos años. Gracias a esta tecnología, se ha reinventado la forma en la que las personas actúan con el entorno, abriendo la puerta a nuevas posibilidades en diferentes espacios. Un ejemplo de ello son los **hogares inteligentes**, donde objetos cotidianos como las bombillas y persianas pueden ser controlados y automatizados a través de aplicaciones móviles, mensajes, y la propia voz. También ha mejorado la **eficiencia energética** en hogares y oficinas gracias a la monitorización y control de los diferentes sistemas de acondicionamiento.

En industria y agricultura, esta tecnología permite recopilar y analizar una gran cantidad de información específica, lo que ayuda a reducir costos y optimizar procesos. Ejemplos de ello son el mantenimiento predictivo, monitoreo de los cultivos y prevención en general de eventos perjudiciales.

La perspectiva de futuro del IoT es muy prometedora. A medida que esta tecnología evoluciona los dispositivos se vuelven más asequibles, favoreciendo su implantación en diferentes sectores como la movilidad y la salud. Con esto se va a mejorar la eficiencia de los envíos y sistemas de distribución, el tráfico en las ciudades y el control de información médica en tiempo real. [2]

Para ello, la integración del IoT en cualquier entorno presenta varios **requisitos**.

- Redes de comunicación capaces de manejar grandes cantidades de datos en tiempo real de forma estable y fiable.
- Plataforma IoT que pueda recopilar y gestionar los datos recopilados. Debe ser escalable, segura y capaz de integrarse con otros sistemas.
- Análisis de los datos para crear información valiosa.
- Seguridad frente a ataques
- Estandarización e interoperabilidad entre productos de diferentes fabricantes mediante un protocolo común.

El primer requisito depende en gran medida del avance tecnológico, sin embargo, los cuatro siguientes se pueden conseguir en mayor o menor medida con una planificación cuidadosa, infraestructura sólida y una implementación a conciencia. Para ello, la Comisión Europea ha impulsado desde 2011 una alternativa de código abierto llamada FIWARE que persigue estos mismos objetivos.

4.2. FIWARE

FIWARE es una plataforma de código abierto compuesta por diferentes componentes se pueden emplear para el desarrollo de plataformas dirigidas al desarrollo de soluciones inteligentes y servicios de datos. Contribuye a la estandarización e interoperabilidad de los diferentes dispositivos, ofreciendo a cada usuario la posibilidad de tomar el control de su plataforma de gestión de datos.

Se trata por tanto de software libre y gratuito, impulsado por la colaboración entre la Comisión Europea y la industria europea con el objetivo de desarrollar un ecosistema abierto y estandarizado para facilitar la creación de nuevas soluciones inteligentes en multitud de ámbitos. Es una gran oportunidad para el desarrollo de tecnologías IoT por las facilidades que presenta a la hora de recopilar información de diferentes sensores que operan con diferentes lenguajes y traducirlos a un protocolo/lenguaje común.

Como se ha visto, esta plataforma se alinea claramente con los requisitos necesarios para la implantación del IoT, se trata por tanto de una herramienta para la adquisición, gestión y análisis de los datos recopilados por multitud de dispositivos, con la idea de obtener información útil para diferentes aplicaciones.

El único requisito obligatorio de cualquier plataforma o solución para que pueda ser denominada *Powered by FIWARE* es el uso de Orion Context Broker (OCB), componente principal que permite administrar la información de contexto, actualizándola y gestionando el acceso a la misma.[3]

4.2.1. Orion Context Broker

Es el gestor de contexto por excelencia de la plataforma FIWARE. Su nombre deja entrever que se trata de un componente capaz de gestionar la información, actualizarla y consultarla cuando sea necesario. Par ello, se sirve de productores (sensores, cámaras, fuentes de datos) y provee a consumidores (aplicaciones, visualización), tal y como se muestra en la Figura 4.1.

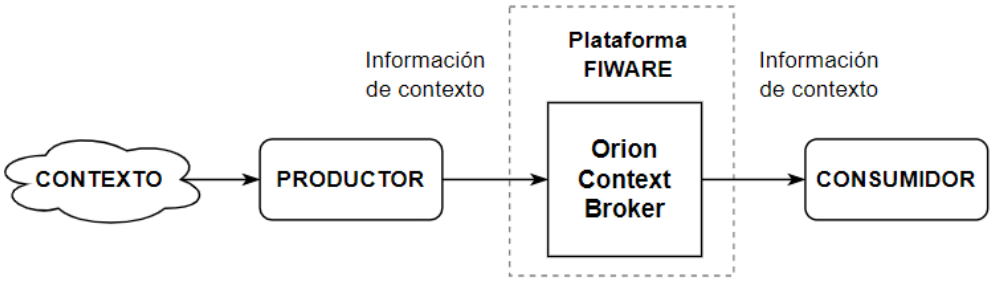


Figura 4.1 Esquema básico OCB.

Para entender mejor esta idea, es necesario profundizar en algunos conceptos:

- **Contexto:** es la información que rodea a un acontecimiento, a partir de la cual se puede interpretar o entender mejor el suceso. Información relevante del entorno y sus usuarios.
- **Información de contexto:** proviene de cámaras de vídeo, radares, sensores (productores) y tiene multitud de aplicaciones (consumidores), por ejemplo, un servicio de aparcamiento inteligente, sistemas de riego, gestión del tráfico, alumbrado público etc.

En resumen, el OCB gestiona información de contexto, registra los proveedores de contexto, actualiza la información, notifica cuando haya cambios y consulta la información.

Se trata del elemento central. Logra una disociación total entre productores y consumidores de contexto de forma que los productores publican datos sin saber qué, dónde ni cuándo los consumidores los consultarán y los consumidores disponen de la información de su interés sin conocer al productor que la publica ya que su interés está únicamente en el evento.

Para implementar funcionalidades inteligentes se debe reunir, gestionar y mantener actualizada una gran cantidad de información de contexto. Para ello, Orion organiza la información en diferentes entidades, caracterizadas por diferentes atributos que tienen asociado un valor, y referencias entre las diferentes entidades.

- **Entidad:** una entidad representa un objeto del mundo real en el entorno digital. Lo hace conteniendo diferentes atributos que caracterizan a dicho objeto, por ejemplo, la entidad que representa a un sensor de temperatura tendrá un atributo dinámico (variable en el tiempo) que será el valor de temperatura medido, pero también se caracterizará por otros atributos estáticos (no varían, o lo hacen con poca frecuencia) como un número de serie, una fecha de instalación, su fabricante o su ubicación.

  En resumen, la información relevante que permite caracterizar de manera inequívoca un objeto del mundo real se contiene en la entidad correspondiente por medio de atributos.

- **Atributo:** como se ha visto, puede ser estático o dinámico. Contiene una información en concreto del objeto al que se refiere.

Esto se conoce como formato NGSI, empleado por el OCB para sus comunicaciones y operaciones. Es un formato de la especificación OMA NGSI [4] que permite la representación de la información de contexto mediante estructuras de datos conocidas como entidades. Existen diferentes versiones, en este proyecto se emplea la *v2* con alguna característica de la versión *LD*. [5]

Más adelante se caracterizará el dispositivo IoT empleado en el proyecto como una entidad con sus atributos de manera sencilla y esto quedará ejemplificado de forma clara.

4.2.2. Arquitectura y comunicación

Como se ha mencionado, una plataforma se considera *Powered by FIWARE* si cumple un único requisito, que el Context Broker que implementa sea Orion. Existen multitud de componentes para distintas aplicaciones que pueden formar parte de la plataforma o no, según los objetivos y funcionalidades de esta. También se puede incluir software propio o de terceros, ya que se trata de código abierto y pretende cumplir con el requisito de interoperabilidad.

Una de las principales características de FIWARE que determina su arquitectura es precisamente su modo de comunicación. Es capaz de recibir información de dispositivos que emplean lenguajes diversos y unificar todos estos lenguajes y protocolos en uno solo, el ya mencionado NGSI en sus diferentes versiones. Logra por tanto abstraer la capa de adquisición de información de la capa de conocimiento y, sobre todo, cumple con el requisito de estandarización, permitiendo combinar dispositivos diferentes de fabricantes diferentes. En niveles altos de la arquitectura FIWARE, los componentes se comunican entre sí con este formato.

El cuadro más general que muestra la mayoría de los componentes FIWARE es el presentado en la Figura 4.2. [6]

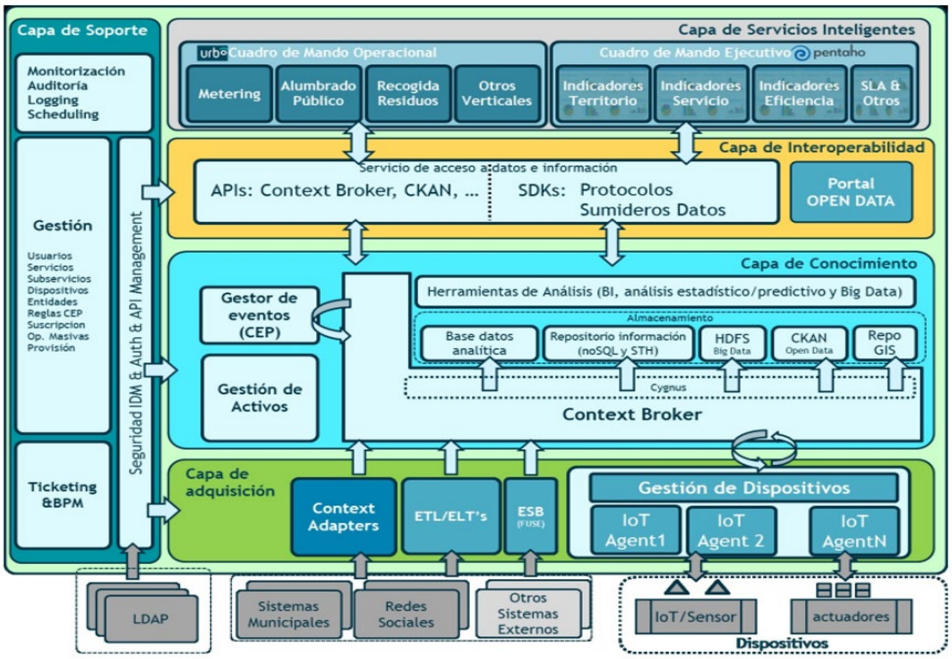


Figura 4.2 Arquitectura FIWARE completa.

En este proyecto en concreto, se han empleado cuatro componentes FIWARE, que son Orion, un IoT Agent de tipo **JSON**, Draco y MongoDB. Serán estos los que se expliquen en profundidad, así como sus homólogos, para explicar por qué se optó por uno u otro en cada caso.

JSON

JSON (JavaScript Object Notation) [7] es un formato ligero de intercambio de datos de alto nivel. Es un formato de texto completamente independiente del lenguaje empleado. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos. JSON está constituido por una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un objeto.

Ejemplo: `{"departamento":"compras", "numero":8}`

4.2.3. MongoDB

Es una base de datos NoSQL de código abierto comúnmente empleada en las plataformas FIWARE. Se trata por tanto de una base de datos que emplea un modelo de datos no estructurado o semiestructurado, como documentos, grafos, o pares *key-value*. Los datos pueden ser por tanto almacenados y consultados en un formato más flexible. [8]

En ella se almacenan las entidades creadas, las configuraciones del IoT Agent y las subscripciones al OCB en el mismo formato JSON en el que se crean. De esta forma, la configuración de cada elemento de la plataforma se guarda en MongoDB, haciendo posible desactivar la plataforma y volver a activarla de nuevo más tarde, sin perder la configuración de esta.

La diferencia con bases de datos tradicionales SQL es que estas emplean modelos de datos estructurados basados en tablas y relaciones, y las consultas se realizan mediante *querys.*[1]

[1] Las consultas en bases de datos de tipo SQL (Structured Query Language) son una forma de solicitar información específica de una base de datos relacional.

4.2.4. IoT Agent

Es un componente FIWARE que sirve para integrar de dispositivos IoT, sensores y actuadores en la plataforma. Se encarga de recopilar datos de los dispositivos IoT y enviarlos a la plataforma de forma estructurada y estandarizada, permitiendo que sean procesados y analizados de forma eficiente.

El IoT Agent funciona como un intermediario entre el dispositivo IoT y la plataforma FIWARE. Existen diferentes IoT Agents según el protocolo mediante el cual se recibe la información. Además, el IoT Agent se encarga de gestionar la seguridad y de la autenticación de los dispositivos IoT, garantizando que sólo se recibe información de los dispositivos autorizados. La característica principal del IoT Agent es su capacidad para transformar los datos recibidos desde los dispositivos IoT en formato NGSI que puede ser interpretado por la plataforma FIWARE, así como traducir de NGSI al formato que sea necesario una instrucción para un actuador. [9]

A continuación, se muestra un esquema que ejemplifica el entramado de comunicación necesario para el funcionamiento de un IoT Agent, en este caso, uno que opera con JSON.

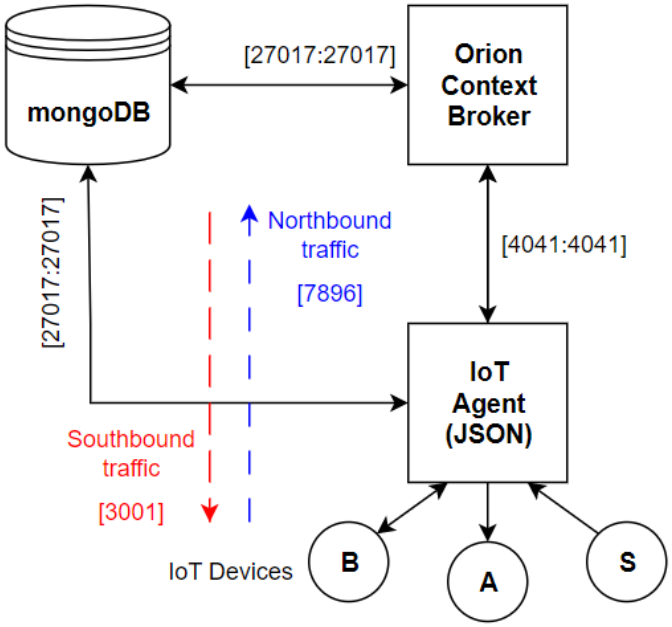


Figura 4.3 Arquitectura y comunicación de IoT Agent.

Como se puede ver en la Figura 4.3, el IoT Agent hace uso de Mongo DB para guardar su información de configuración. Para la comunicación con los diferentes elementos, se emplean los siguientes **puertos**.

- Comunicación con MongoDB: 27017
- Comunicación *Northbound* con OCB: 7896 (información proveniente del dispositivo)
- Comunicación *Southbound* con OCB: 3001 (información hacia un actuador o dispositivo)
- Comunicación para configuración con OCB: 4041

En resumen, un IoT Agent es un componente clave de la plataforma FIWARE para la integración de dispositivos IoT en la plataforma. El IoT Agent se encarga de recopilar y enviar datos de los dispositivos a la plataforma, gestionar la seguridad y la autenticación, y transformar los datos en un formato estandarizado y semántico.

4.2.5. Draco

Es un componente del entorno FIWARE que se emplea para la gestión de grandes volúmenes de datos en tiempo real [10]. Es un sistema poderoso y confiable, basado en Apache NiFi [11], razón por la que presenta una cómoda interfaz que permite visualizar el flujo de datos y añade un nivel más de control.

Con este componente se pretende lograr la persistencia de la información de contexto, generando una base de datos MySQL que más adelante pueda ser consultada para la visualización de los datos obtenidos.

Una alternativa al uso de Draco es el componente Cygnus [12]. Se trata de un conector que cumple la función de almacenar la información en una base de datos de terceros, como MySQL. Está basado en Apache Flume [13]. Se opta por implementar Draco para incluir el panel de control ya mencionado.

Ambos componentes funcionan mediante una suscripción al OCB a través de la cual reciben los cambios en la información de contexto seleccionada, que puede ser filtrada con condicionantes en la propia suscripción. Este tema se abordará más adelante.

4.2.6. ¿Por qué FIWARE?

A modo de resumen, FIWARE presenta una alternativa de código abierto para que sea el propio usuario quien gestione su propia plataforma, sin depender de servicios de terceros como puede ser AWS o Azure.

La principal ventaja consiste en la posibilidad de integrar diferentes tipos de sensores en una misma plataforma gracias a su capacidad de estandarización y el empleo del formato NGSI para ello. Basta con una configuración adecuada del IoT Agent para integrar cualquier dispositivo.

Presenta un gran potencial escalable, puede manejar grandes volúmenes de datos sin afectar el rendimiento. Esto es especialmente importante para implementar soluciones en tiempo real. La posibilidad de integrar diferentes componentes hace que sea un entorno muy flexible, capaz de adaptarse a las necesidades de cualquier desarrollador.

En el aspecto de ciberseguridad, FIWARE emplea una estructura de capas que garantiza la protección de los datos y la privacidad de los usuarios. Cuenta con diversos mecanismos de autenticación que garantizan que sólo los usuarios autorizados puedan acceder a según qué datos y servicios.

Al ser un software de código abierto desarrollado por la comunidad, no tiene un único creador, lo que implica que el conocimiento se distribuye en muchas personas. Esto se traduce a un gran número de expertos trabajando para mejorar y evolucionar la plataforma.

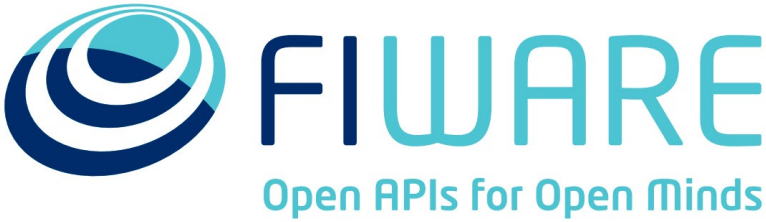


Figura 4.4 Logo FIWARE

4.3. Recursos software

Para el desarrollo de este proyecto se han empleado multitud de recursos software los cuales se van a comentar y explicar en este apartado. Se comentan en profundidad los programas de mayor interés o más novedosos,

4.3.1. Máquina virtual

Una máquina virtual es, como su propio nombre indica, **un ordenador** (máquina) que **no existe físicamente** (virtual). ¿Cómo es esto posible? Se trata de un software capaz de cargar en su interior otro sistema operativo distinto al del ordenador anfitrión en el cual se ejecuta dicho software. Simula por tanto un **PC independiente** diferente que emplea los recursos físicos del PC en el que se aloja, el cual recibe el nombre de anfitrión, o en inglés, *host;* representado en el centro de la Figura 4.5.

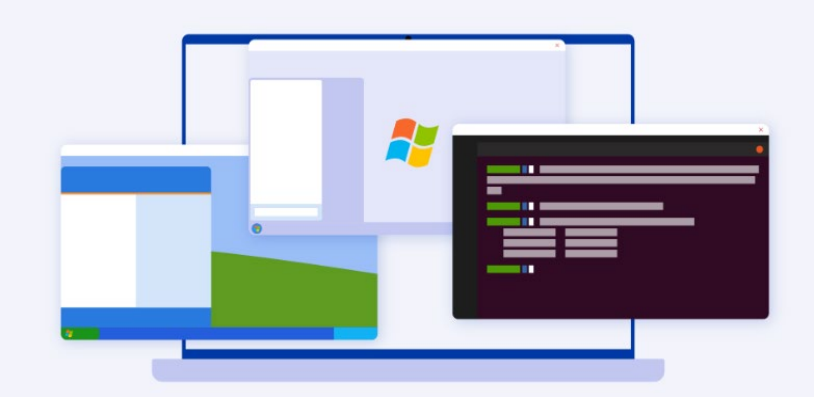


Figura 4.5 Representación de una máquina virtual.

Esta imagen ejemplifica de forma visual el concepto y utilidad de una (o varias) máquinas virtuales. Con **un sólo equipo físico** que corre un único SO, se pueden lanzar **diferentes máquinas** virtuales independientes con cualquier SO que exista.

Esto tiene varias implicaciones ventajosas para los desarrolladores. En primer lugar, la máquina virtual es **independiente del anfitrión**, por lo que, si se destruye, deja de funcionar, o es infectada por un virus informático; basta con eliminar la máquina y crear otra nueva. El PC anfitrión permanecerá intacto. Resulta muy útil para realizar **pruebas de comunicación** entre aplicaciones cliente-servidor, sin necesidad de tener dos equipos físicos independientes, basta con crear una máquina virtual y configurar la red correctamente.

Además, en la máquina virtual **el administrador es el propio usuario**, por lo que el usuario cuenta con todos los **privilegios** para hacer y deshacer a su antojo. En el caso de organizaciones en las que los equipos cuentan con limitaciones a la hora de configurar los cortafuegos, descargar aplicaciones y en general realizar cambios en el equipo, esto resulta una gran ventaja.

En este caso, la máquina virtual se va a emplear para correr la plataforma FIWARE, ya que ésta se despliega mediante **Docker**, que únicamente funciona de manera correcta en **Linux**. Es por ello por lo que se va a habilitar una máquina virtual con sistema operativo *Ubuntu 22.04.1* en su versión *desktop*. A su vez, se imita el funcionamiento de la plataforma de la empresa, ubicada en una máquina virtual que corre en un servidor privado.

4.3.2. Docker

Docker es un software libre que crea contenedores ligeros y portables que pueden ser abiertos en otro equipo que únicamente tenga instalado Docker, sin necesidad de que este equipo tenga que cumplir con ningún requisito extra. Para hacer esto posible, cuando se construye un contenedor para una aplicación determinada, este incluye toda la información y requisitos extra: código, configuraciones, versiones, dependencias y librerías.

Es similar en cierto modo al concepto de máquina virtual, salvando las distancias, como se puede ver en la comparación de la Figura 4.6.

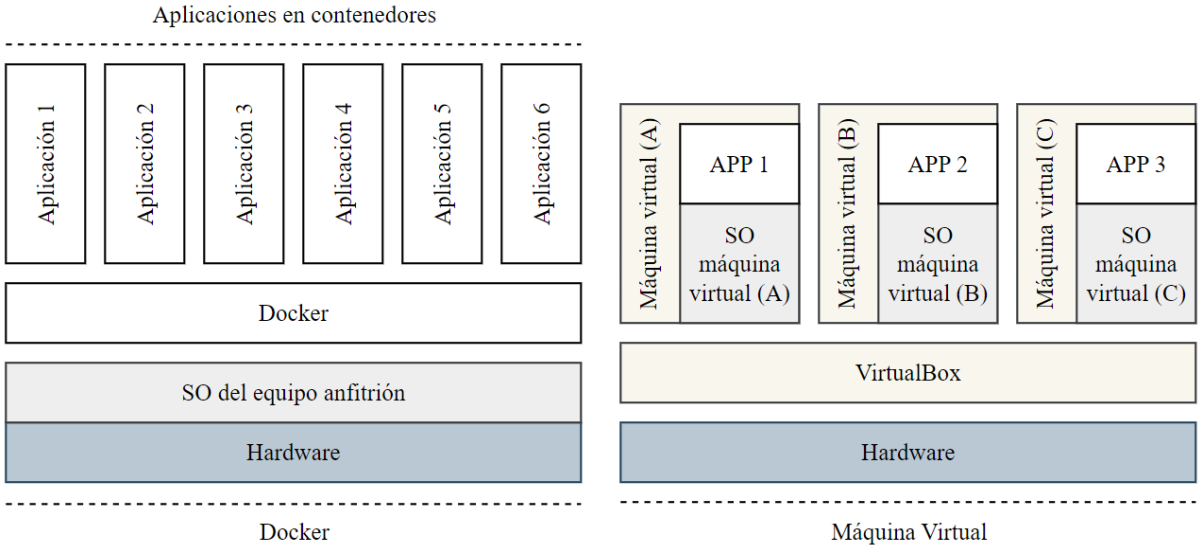


Figura 4.6 Comparativa del nivel de funcionamiento entre Docker y una máquina virtual

IMÁGENES Y CONTENEDORES

En el entorno de Docker es habitual hablar de imágenes y contenedores. Un desarrollador puede crear una aplicación y empaquetarla junto con sus dependencias en lo que se denomina la "imagen" de la aplicación. Por lo tanto, una imagen es una representación estática de la aplicación/servicio y de su configuración y dependencias. Si se desea desplegar una aplicación, se crea una instancia de la imagen de dicha aplicación para crear un contenedor, que se ejecutará en el host de Docker [14].

DOCKER COMPOSE

Como se irá viendo, a lo largo del proyecto va a ser necesario desplegar varias imágenes al mismo tiempo. Por ello, para evitar problemas por dependencias entre estas y garantizar el correcto funcionamiento, así como una forma sencilla de configuración, se emplea Docker Compose. Así, mediante la creación y configuración de un archivo YAML llamado *docker-compose.yml* se logra desplegar multitud de imágenes con un solo comando, respetando las dependencias entre ellas.

A modo de recopilación, las principales características que presenta Docker y son de interés para este proyecto son las siguientes.

- Requiere un SO Linux para funcionar correctamente
- Permite descargar y ejecutar software de manera sencilla y cómoda.
- Los contenedores e imágenes son sencillos de manejar.

4.3.3. Postman

Para la comunicación con la plataforma FIWARE mediante peticiones HTTP se emplea Postman, un programa empleado para desarrollar y probar APIs que facilita la labor de construcción, prueba y monitorización de estas [15]. La principal característica de esta herramienta es su capacidad para crear, gestionar y mandar peticiones HTTP a servicios REST. Presenta una interfaz gráfica que facilita esta tarea por ser muy intuitiva y sencilla de manejar. Ofrece la posibilidad de guardar las peticiones y ordenarlas por carpetas, para poder lanzarlas de forma sencilla y ordenada.

Figura 4.7 Logo de Postman.

SERVICIO REST

Los servicios REST [16] (Representational State Transfer) permiten la comunicación entre sistemas software a través de internet. REST se basa en el protocolo HTTP y utiliza los métodos HTTP para definir las operaciones que pueden realizarse en un recurso. Estas operaciones son, entre otras; GET, POST, PUT y DELETE.

Utilizan el formato de intercambio de datos JSON o XML (Extensible Markup Language) para transmitir datos entre el servidor y el cliente.

La principal ventaja de los servicios REST es su independencia del lenguaje de programación utilizado. Cualquier sistema que pueda realizar una solicitud HTTP y comprender el formato de los datos de respuesta puede interactuar con un servicio REST. Además, son altamente escalables y se pueden utilizar en diferentes entornos, desde aplicaciones móviles hasta aplicaciones empresariales complejas.

En resumen, Postman implementa de manera ágil, accesible y gratuita el protocolo HTTP. La interfaz para la creación de peticiones que va a aparecer de manera recurrente a lo largo del documento se muestra en la Figura 4.8.

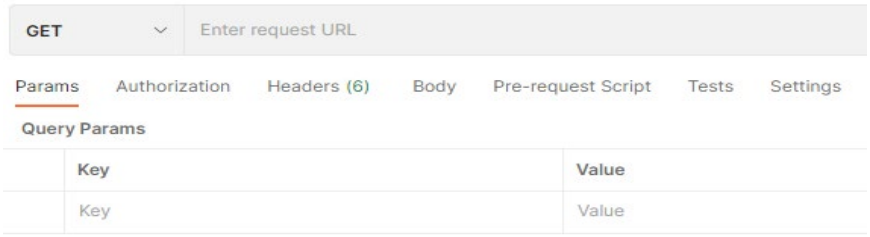


Figura 4.8 Interfaz para la creación de peticiones.

Los elementos principales son, comenzando por la esquina superior izquierda, un menú de selección del tipo de petición a realizar. A su derecha, se introduce la dirección URL del destino al cual se dirige la petición.
Justo debajo, se encuentra una fila con diferentes campos, de los cuales se van a usar principalmente:

- *Params:* configuración de la petición que se va a realizar por medio de pares *key-value.*
- *Headers:* cabeceras del mensaje, con especial relevancia: *fiware-service* y *fiware-servicepath.*
- *Body:* cuerpo del mensaje, contiene la información relevante. En este proyecto se emplea el formato JSON en este campo.

4.3.4. Node-RED

Node-RED se emplea para realizar las pruebas de comunicación y para integrar el sistema de alarmas. Se trata de una herramienta que sirve para establecer comunicación entre diferentes piezas de hardware y múltiples servicios de manera rápida, sencilla y visual. Con su interfaz de programación basada en bloques, se facilita a nuevos usuarios la tarea de programar diferentes funcionalidades [17].

En 2016, IBM contribuyó con Node-RED como un proyecto de OpenJS Foundation de código abierto [18].

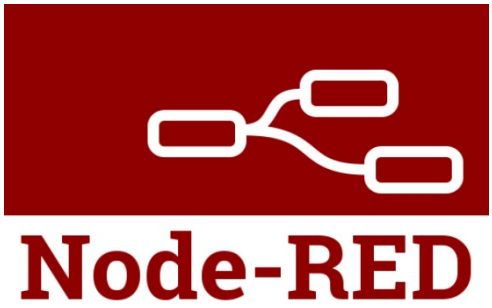


Figura 4.9 Logo de Node-RED.

Node-RED se caracteriza por su interfaz para la edición de flujos. Se ejecuta en un navegador web una vez iniciado en el equipo en el que está instalado. Se basa por tanto en la creación de diferentes flujos de información por medio de bloques que se conectan entre sí permitiendo implementar de manera sencilla multitud de funciones, como se muestra en la Figura 4.10.

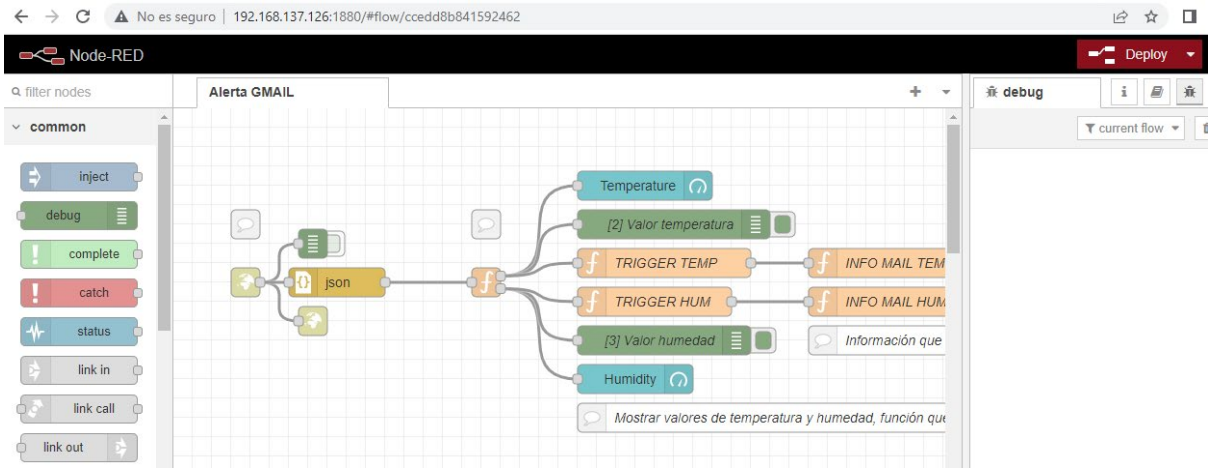


Figura 4.10 Flujo de Node-RED.

Cuenta con una amplia variedad de librerías de código abierto para añadir al Node-RED básico nuevos bloques con nuevas funciones. Está basado en Node.js [19] y los flujos creados se pueden guardar en un simple archivo de texto ya que se exporta toda la información necesaria para crear el flujo en formato JSON.

4.3.5. Otros programas

Arduino IDE: Por sus siglas *Integrated Development Environment,* es una aplicación multiplataforma (Windows, macOS, Linux) que ofrece un conjunto de herramientas software para desarrolladores. Es un entorno que facilita la tarea de desarrollar y grabar el código necesario para hacer funcionar la placa con la que se va a trabajar (puede ser Arduino o similar, pero también puede funcionar

en placas de desarrollo de otros proveedores con pequeñas adaptaciones). Está escrita en Java y admite los lenguajes C y C++.

Figura 4.11 Logo Arduino IDE.

MySQL Workbench: Es una herramienta empleada para la gestión de bases de datos relacionales (tipo SQL). Presenta un panel visual que simplifica la tarea de desplazarse por los menús de consola de MySQL. Facilita la tarea de creación, edición y gestión de bases de datos. Se puede conectar a una base de datos de forma remota, siempre que se tenga acceso al anfitrión de dicha base de datos y se introduzcan las credenciales correctamente.

Figura 4.12 Logo MySQL Workbench

Power BI: Es una herramienta de análisis y visualización de datos desarrollada por Microsoft. Permite la conexión y tratamiento de múltiples fuentes de datos, así como la creación de paneles con diferentes tipos de gráficos que permitan al desarrollador mostrar la información de la manera más adecuada. Se utiliza la versión gratuita, Power BI Desktop.

Figura 4.13 Logo Power BI Desktop.

5. Diseño de la plataforma

En este apartado se va a describir en profundidad la implementación que se va a realizar a lo largo del proyecto. Como se ha comentado, el proyecto surge de la necesidad de aprender cómo funcionan las plataformas FIWARE, cómo se llevan a la práctica todos los conceptos y funcionalidades de los que se ha hablado anteriormente y obtener los conocimientos necesarios para poder aportar al desarrollo de una plataforma más grande y compleja de este tipo.

Es por ello por lo que se ha optado por implementar en una plataforma FIWARE un dispositivo IoT sencillo y fácil de manejar, pero que va a presentar muchas de las dificultades presentes en una plataforma de mayores dimensiones.

Con todo, se va a desarrollar una plataforma FIWARE para gestionar la información proveniente de un sensor DHT11. Existen varias formas de llevar a la práctica esta idea tan general, a lo largo de este apartado se va a ir describiendo de manera más precisa el camino elegido.

5.1. Hardware empleado

Como se ha comentado, se va a implementar un dispositivo IoT en una plataforma FIWARE, pues bien, en este apartado se va a describir detalladamente este dispositivo. Está formado por un sensor y una placa con conectividad WiFi, concretamente por el sensor DHT11 y la placa ESP32-C3-DevKitM-1.

Los únicos componentes físicos empleados en el proyecto son los siguientes.

- Sensor DHT11 1x
- Placa ESP32-C3-DevKitM-1 1x
- Cable *hembra - hembra* 3x
- Cable USB2.0 (Standard-A to Micro-B) 1x

El montaje del dispositivo consiste en la conexión de ambos componentes, se realiza de manera sencilla de la forma en la que se muestra en la Figura 5.1.

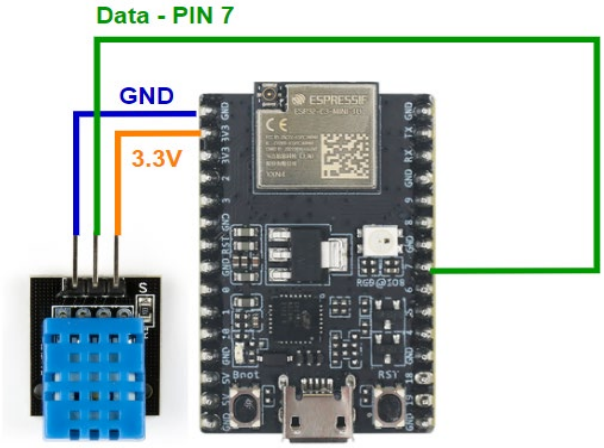


Figura 5.1 Conexión del dispositivo IoT.

5.1.1. Sensor DHT11

El sensor empleado para la realización de este proyecto es el conocido DHT11, un **sensor de temperatura y humedad** que utiliza un sensor capacitivo de humedad y un termistor para medir el aire circundante. Los datos obtenidos son enviados en forma de señal digital a través de su pin de datos [20].

Se emplea la versión PCB del DHT11 por ser esta muy sencilla de manejar y de pequeñas dimensiones [23 x 12 x 05 (mm)]. Sus características principales son las siguientes.

- Rango de temperatura: 0ºC - 50ºC con una resolución de ± 2.0ºC
- Rango de humedad:20% a 90% con una resolución de ± 5.0% (humedad relativa)
- Alimentación de 3.3V a 5V DC.
- Frecuencia de muestreo < 1Hz

El sensor contiene tres pines, dos de alimentación y otro de datos, como se muestra en la Figura 5.2.

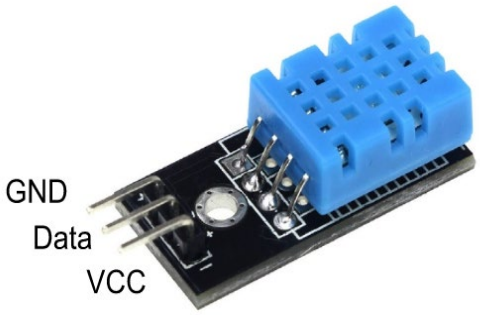


Figura 5.2 Distribución de pines del sensor DHT11 en su versión PCB

Existen multitud de sensores ambientales, pero se ha elegido el DHT por facilidad de disposición. La diferencia con otro sensor que mida más parámetros es simplemente la cantidad de atributos que va a contener la entidad, por lo que la dificultad no varía con respecto a utilizar, por ejemplo, un sensor BME680 de más altas prestaciones.

5.1.2. Placa ESP32

La placa empleada es un *Developer Kit* fabricado por *Espressif* que implementa un chip *ESP32-C3-MINI-1U* y cuenta con funcionalidad WiFi y Bluetooth.

ESP32 [21]es la denominación de una familia de chips SoC de bajo coste y consumo de energía con tecnología WiFi y Bluetooth integrada. Emplea un microprocesador *Tensilica Xtensa LX6* de simple o doble núcleo. El ESP32 fue creado y desarrollado por Espressif *Systems* y es fabricado por TSMC utilizando su proceso de 40 nm.1 Es un sucesor de otro SoC, el ESP8266.

Espressif es uno de los líderes mundiales en chips y módulos de comunicación WiFi y Bluetooth, desarrollados específicamente para dar la mejor solución a la actual demanda del Internet of Things [22].

La carga de software desde el PC se realiza a través de un cable USB - MiniUSB, y en este proyecto, la placa se alimenta a través de este mismo cable, ya que es la opción recomendada por el fabricante [23]. Como se puede ver en su *datasheet,* existen varias formas de alimentar la placa.

En la Figura 5.3 se puede ver la distribución de pines de la placa.

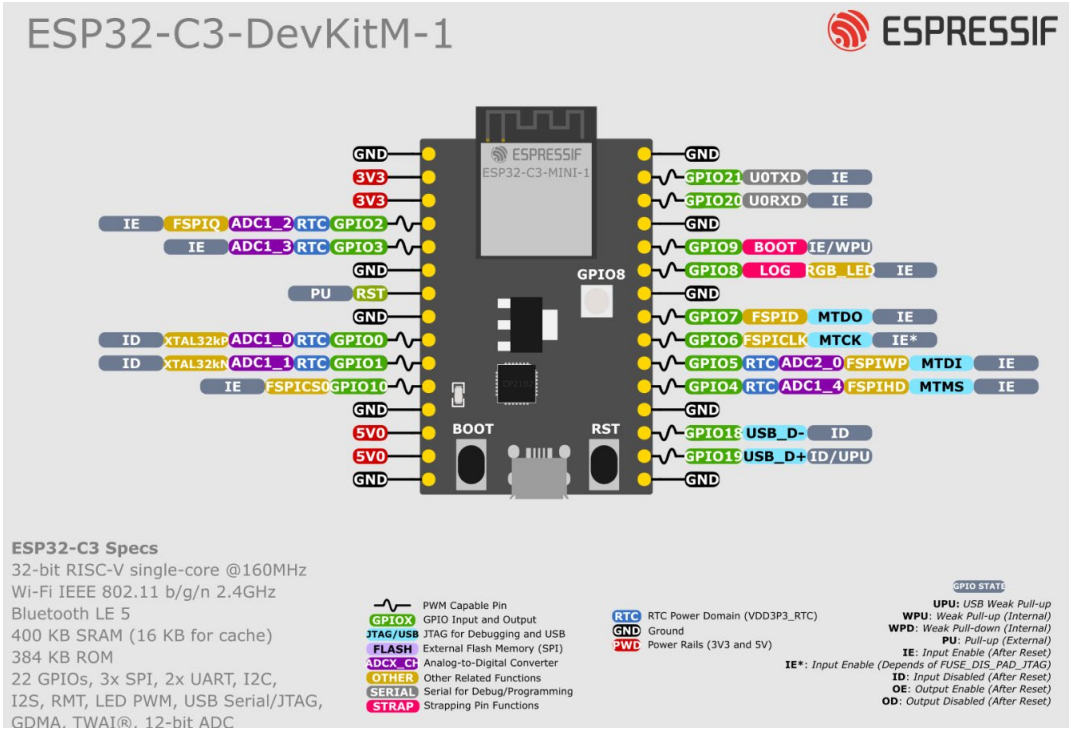


Figura 5.3 Distribución de pines ESP32_C3_DevKitM_1 .

5.2. Software necesario

El sensor se conecta a la placa y es esta quien contiene y ejecuta el software que el usuario desea. Como se ha comentado, el volcado a la placa de este software se realiza de manera sencilla mediante un cable USB 2.0.

Para programar este dispositivo existen varias alternativas, entre las cuales se elige el software Arduino IDE. Se lleva a cabo la instalación de Arduino IDE desde su página oficial de manera gratuita para comenzar a programar los scripts. La tarea que debe realizar la placa ESP32 se puede dividir en tres partes fundamentales que son las siguientes.

- Recogida de datos: *script* que habilita al sensor DHT11 y que almacena en dos variables *t* y *h* los valores de temperatura y humedad relativa respectivamente.
- Traducción a JSON: código que permite crear una cadena JSON para poder enviar en el formato correcto la petición HTTP, ya que se usa un IoT Agent de tipo JSON
- Envío por WiFi de datos mediante protocolo HTTP: este *script* debe contener lo necesario para conectar la ESP32 a la red WiFi correspondiente y capacidad para realizar peticiones HTTP.

Los *scripts* completos se encuentran en el Anexo A. La instalación y descripción de las librerías empleadas en los diferentes scripts se recoge en el Anexo A.4.2. A continuación, se va a profundizar en cada parte para explicar con detalle el código empleado en cada función, para conocer en profundidad la función de cada línea, ayudando esto a la optimización y posible reutilización del código.

5.2.1. Recogida de datos

La primera tarea consiste en la recogida de los datos de interés mediante el sensor DHT11. Para ello, realizada la conexión del dispositivo IoT como se explicó anteriormente, se procede a la creación de un nuevo *script*.

En primer lugar, se debe incluir la librería para el manejo del DHT11

```
#include <DHT.h>
```

Se definen tanto el tipo de sensor como el pin de la placa, en este caso al pin digital 7.

```
#define DHTTYPE DHT11
#define DHTPIN 7
```

Se crea el objeto DHT que va a usar las funciones ligadas a la librería DHT.h. Con el objeto DHT se puede llamar más tarde a las funciones readHumidity y readTemperature.

```
DHT dht(DHTPIN,DHTTYPE);
```

En el bucle *void setup()* se inicializa el puerto serie a 115200 baudios y se inicializa el sensor.

```
void setup() {
  Serial.begin(115200);
  dht.begin();
}
```

A continuación, se quiere almacenar en las variables t y h los valores de temperatura y humedad recogidos. Para tener un suministro periódico de datos, se incluye en un bucle.

```
void loop() {
  delay(5000);
```

Para leer la humedad se designa una variable de tipo *int* y se emplea el comando a continuación.

```
int h = dht.readHumidity();
```

Lo mismo para la temperatura

```
int t = dht.readTemperature();
Se comprueba si hay error en la lectura con:
 if (isnan(h) || isnan(t)) {
    Serial.println("Error obteniendo los datos del sensor DHT11");
    return;
  }
```

Es importante acordarse de cerrar el bucle.

```
}
```

5.2.2. Formato JSON.

Para que el mensaje sea correctamente recibido debe estar en formato JSON, ya que el IoT Agent empleado "entiende" este lenguaje. Por ello, se busca transmitir un dato con la siguiente estructura: {"t": "*valor",* "h": "*valor"*}.

En primer lugar, se deben incluir las librerías necesarias.

```
#include <ArduinoJson.h>
#include <ArduinoJson.hpp>
```

Se crea una *String* con un nombre arbitrario, en este caso *json.*

```
String json;
```

A continuación, se debe crear un documento JSON, *JsonDocument,* que puede ser de tipo dinámico o estático. La diferencia entre ambos es que *StaticJsonDocument* se genera durante la compilación y se almacena en la pila, mientras que *DynamicJsonDocument* emplea memoria dinámica y se almacena en la memoria de variables, es un poco más lento, pero permite generar ficheros más grandes. El propio desarrollador de la librería Arduino Json aconseja emplear documentos estáticos para ficheros menores de 1KB y dinámicos si sobrepasan este tamaño. [24]

Se debe indicar la memoria que se destina al documento, *doc(<memoria>)*. Debe ser mayor que el tamaño del objeto a codificar y es difícil de calcular. En fases de desarrollo basta con que sea suficientemente grande, y posteriormente se puede ajustar para ahorrar recursos si se necesita.

```
DynamicJsonDocument doc(1024);
```

El lenguaje JSON se compone de pares *key-value*. Para asignar un *value* a su *key*, se emplea la siguiente función.

```
doc["t"]= t;
doc["h"]= h;
```

Se serializa el *JsonDocument*

```
serializeJson(doc, json);
```

Ahora, al imprimir la *String* llamada *json* por pantalla, se obtiene el dato JSON deseado.

```
Serial.println(json);
```

5.2.3. Envío por WiFi de datos mediante protocolo HTTP

La ESP32 se conecta a la WiFi del punto de acceso del PC anfitrión, adquiriendo una IP en su misma red cada vez que esta conexión se produce. El envío de datos se realiza mediante peticiones HTTP de tipo POST que apuntan al destino web en el que se ubica el IoT Agent.

Se incluyen las librerías necesarias para manejar tanto la conexión WiFi como el protocolo HTTP.

```
#include <WiFi.h>
#include <HTTPClient.h>
```

Hay que especificar las credenciales de acceso a la red WiFi

```
const char* ssid = "XXXX";
const char* password = "XXXX";
```

Se especifica también el dominio al cual se van a enviar las peticiones HTTP

```
String serverPath = "http://192.168.137.126:7896/iot/json?i=DHT11&k=apikey";
```

Esta línea en particular puede ocasionar algunos problemas. Lo que se hace aquí es crear una *String* de nombre *serverPath* y que contiene la dirección http del IoT Agent (JSON over HTTP) empleado en el proyecto y otros elementos.

Desgranando el destino empleado:

- *http://192.168.137.126:7896* Contiene la IP del IoT Agent y el puerto por el cual escucha las comunicaciones *Northbound* (provenientes del dispositivo, hacia OCB).
- */iot/json:* Destino especificado en *service* del IoT Agent.
- *?i=<device_id>:* Campo especificado en *devices* del IoT Agent, que es DHT11 .
- *&k=<apikey>:* Campo especificado en *service* del IoT Agent, en este caso “apikey”: “apikey” para facilitar las pruebas, se debería colocar una *key* compleja para que ningún dispositivo desconocido pueda enviar datos.

En el bucle *void setup():*

- Se inicializa el puerto serie a 115200 baudios y se inicializa la WiFi.

```
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
```

- Se muestra por pantalla el mensaje *Conectando…* y se añade un “.” cada 500 ms que pasen sin que se logre establecer la conexión.

```
  Serial.print("Conectando...");
```

```
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
```

- Una vez establecida la conexión con la red, se muestra en la siguiente línea un mensaje que muestra la IP asignada a la ESP32.

```
  Serial.println("");
  Serial.print("Conectado a WiFi con IP: "); Serial.println(WiFi.localIP());
}
```

En el bucle principal *void loop():*

- Se inicializa el protocolo HTTP

```
void loop() {
  http.begin(serverPath.c_str());
```

- Se introducen las credenciales de acceso al servidor si es necesario

```
  http.setAuthorization("admin", "1234");
```

- Para enviar una petición tipo POST (peticiones empleadas)

```
  http.addHeader("Content-Type", "application/json");
  int httpResponseCode = http.POST("<mensaje>");
```

- Como información extra, si se quiere enviar una petición tipo GET se deberá escribir:

```
  int httpResponseCode = http.GET();
```

- Si se recibe un código de respuesta (200, 404…) imprime dicho código y guarda el mensaje en una string llamada *payload*, luego imprime el mensaje.

```
  if (httpResponseCode>0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
    String payload = http.getString();
    Serial.println(payload);
  }
```

- Si no recibe ningún código, muestra por pantalla un mensaje de error

```
   else {
    Serial.print("Error code");
   }
  http.end();
}
```

5.3. Configuración de red

Un apartado que ha resultado bastante problemático para el desarrollo de las primeras fases del proyecto ha sido la configuración de red de la máquina virtual.

Para acceder a la configuración de red, se inicia *VirtualBox* y se selecciona *Configuración* > *Red.* En la Figura 5.4 se pueden ver dos máquinas virtuales creadas, *ubuntuserver22* y *ubuntudesktop22.copia.* La primera, como su nombre indica, corre una imagen de Ubuntu en versión *Server*, descargada por curiosidad para realizar alguna prueba. La segunda es la que se emplea a lo largo del proyecto, corre la imagen *Ubuntu 22.04.1-desktop-amd64.iso.*

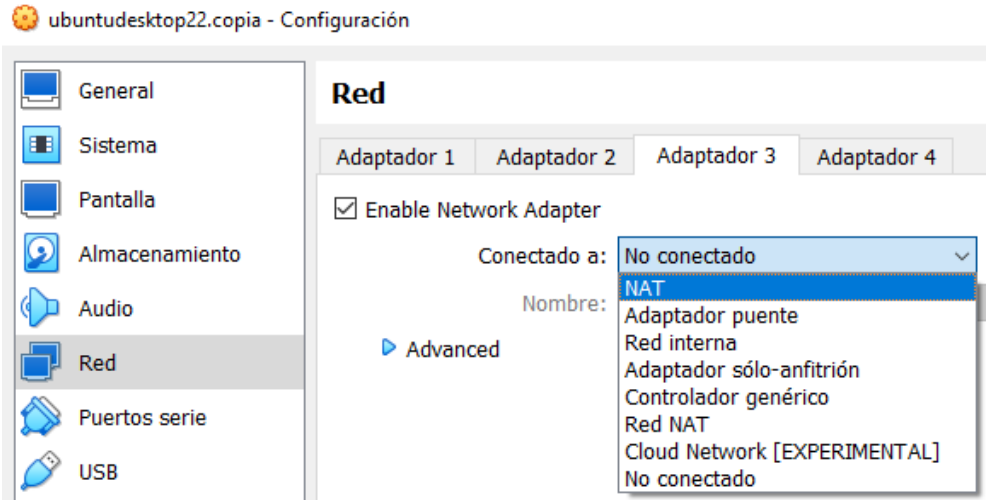


Figura 5.4 Acceso a la Configuración de Red.

5.3.1. Tipos de configuraciones

En la Figura 5.4 se pueden ver las diferentes configuraciones de red que existen para la máquina virtual. Para poder realizar la configuración de manera correcta en cada caso, es conveniente conocer cada una de ellas. [25]

- *NAT:* con esta configuración, la máquina virtual tiene acceso a internet a través del anfitrión.
- *Adaptador puente*: esta configuración simula que la tarjeta de red de la máquina virtual (virtual) se conecta al mismo switch que la tarjeta física del anfitrión. De esta forma, la máquina virtual funciona como si fuera otro equipo perteneciente a la misma red física que el anfitrión. Exige definir la tarjeta de red a emplear si el anfitrión tiene más de una.
- *Red interna:* configuración empleada para crear redes aisladas en las que sólo interactúan las máquinas virtuales que pertenezcan a una misma red. El anfitrión no puede participar.
- *Adaptador sólo-anfitrión:* es una configuración muy similar a la anterior, con la diferencia de que el anfitrión pertenece a esta red privada. De esta forma, desde el anfitrión se tiene acceso a internet y es posible la comunicación con la máquina virtual, la cual no tiene acceso a internet y sólo se puede comunicar con el anfitrión.
- *Red NAT:* esta configuración funciona como un *router*. Los equipos que están dentro de la misma red NAT pueden comunicarse entre sí y existe conexión a internet.

5.3.2. Configuración empleada

Para la aplicación planteada, es necesario que los tres elementos principales (anfitrión Windows, máquina virtual Linux y ESP32) puedan comunicarse entre sí.

A. Primer intento

Con esta idea, se realizaron diferentes pruebas enfocadas en conseguir comunicación bidireccional entre la máquina y el anfitrión, manteniendo en ambos equipos la conexión a internet. Con este fin, una solución aparentemente buena podría ser habilitar dos adaptadores en la máquina virtual, cada uno con una función y que trabajen de manera combinada.

- *Adaptador 1:* NAT
- *Adaptador 2:* Adaptador puente + Conexión Ethernet

Esta es la configuración más evidente atendiendo a la descripción de los diferentes tipos de configuración de red, sin embargo, no se consigue el objetivo planteado y al intentar enviar datos desde la ESP32 a la máquina virtual, se obtiene también un error de accesibilidad (no se puede apuntar a un destino en la MV). Atendiendo a las explicaciones de cada tipo de configuración de red, el error no es evidente.

Lanzando en la consola de la máquina virtual el comando *ifconfig* para mostrar su IP, se puede ver que no aparece ninguna. Esto se debe a que en la Ethernet no hay direcciones IP disponibles, por lo que el adaptador configurado no es capaz de asignar una IP a la MV para que pueda realizar todas las funciones de red.

B. Configuración definitiva

Una vez detectado el problema existen dos posibles soluciones. Se puede asignar una IP de la red a la máquina virtual, o configurar la máquina para que se conecte a un punto de acceso WiFi que se puede habilitar el PC Host. Para no ocupar una dirección IP se opta por la segunda solución.

En primer lugar, se accede al menú *Conexiones de Red* ubicado en el destino que se muestra en la imagen. Para explicar correctamente este proceso se parte del estado inicial del equipo, con el punto de acceso WiFi desactivado como se muestra en la Figura 5.5.

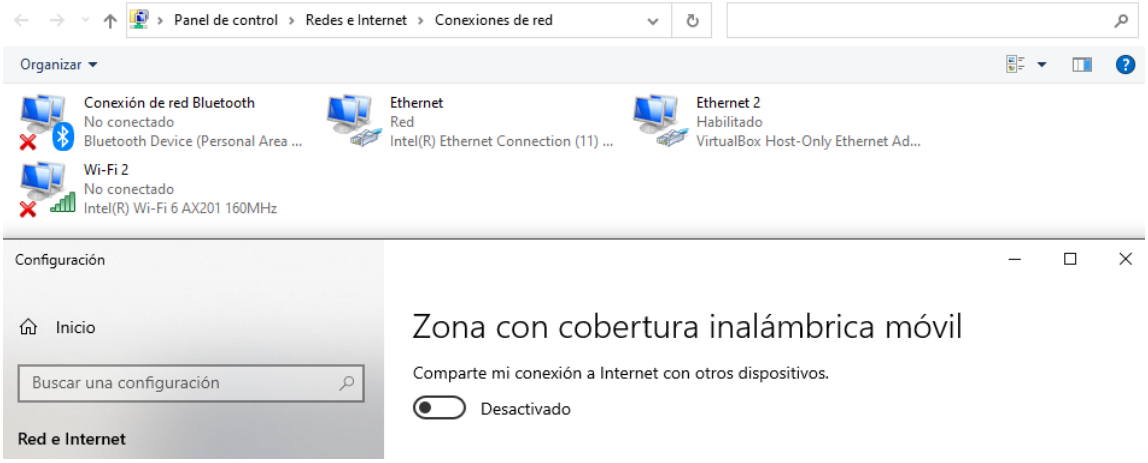


Figura 5.5 Conexiones de red cuando el punto de acceso WiFi está desactivado.

Al activar el Punto de Acceso, aparece una nueva conexión de red con el nombre: *Conexión de área local* 12* en estado habilitado, como se observa en la Figura 5.6.

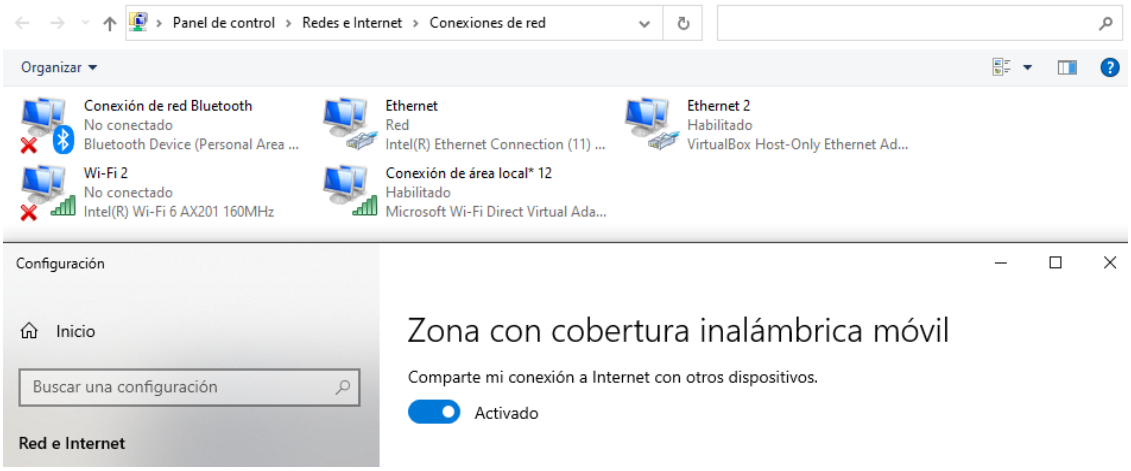
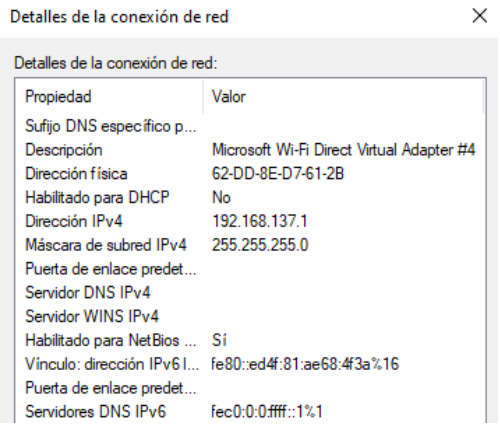


Figura 5.6 Conexiones de red cuando el punto de acceso WiFi está activado.

Hacer doble clic en *Conexión de área local* 12* para ver los detalles:



Se puede ver que la dirección IPv4 es 192.168.137.1 con una máscara 255.255.255.0, lo que indica que se encuentra en la red 192.168.137. También es importante destacar el adaptador de red, *Microsoft WiFi Direct Virtual Adapter #4,* que será relevante más adelante.

Figura 5.7 Detalles de la conexión de red seleccionada.

Con esta información se accede al menú de configuración de red de la máquina virtual, y para el *Adaptador 1* se selecciona la configuración que se muestra en la Figura 5.8.

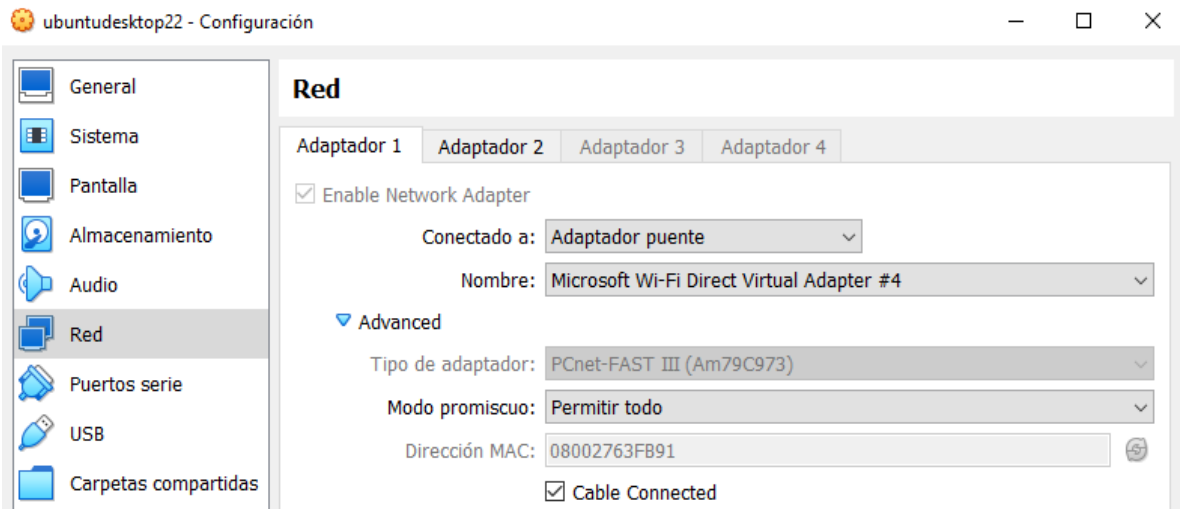


Figura 5.8 Configuración del Adaptador 1 de la máquina virtual.

En el *Adaptador 2* se introduce configuración que se muestra en la Figura 5.9.

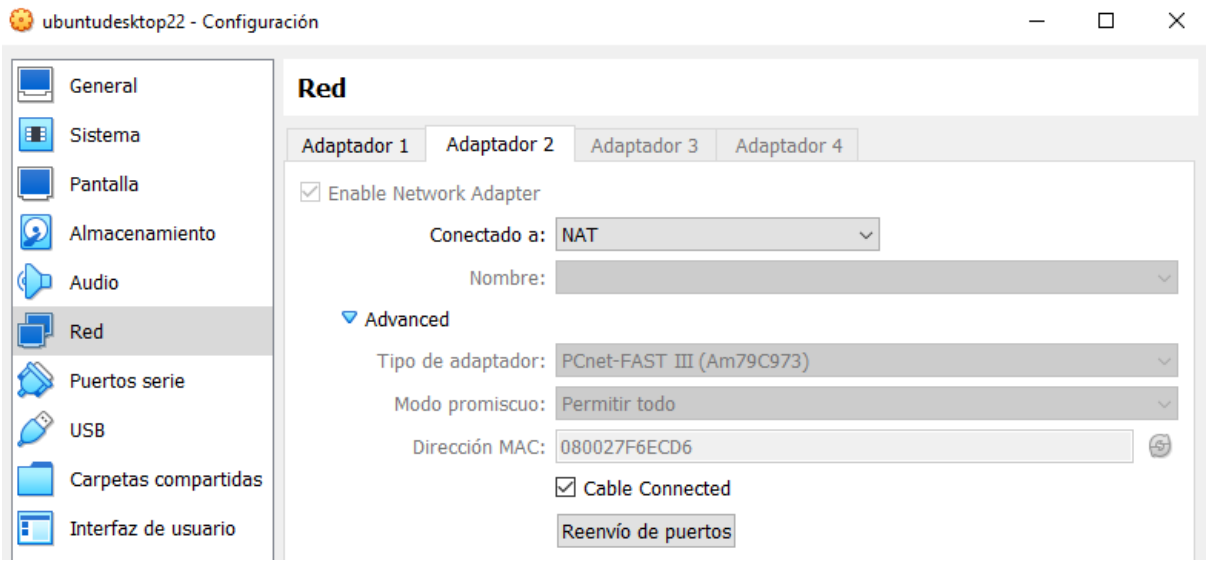


Figura 5.9 Configuración del Adaptador 2 de la máquina virtual.

Con el *Adaptador 1* (conectado a Adaptador Puente) se logra conectar la máquina Ubuntu con el PC Host mediante el punto de acceso WiFi, se le asigna por tanto una IP de la misma red (192.168.137) por lo que se garantiza la conectividad entre ambas y con la ESP32, que, al estar conectada a la misma WiFi, pertenece también a la misma red.

Con el *Adaptador 2* (conectado a NAT) se consigue que la máquina Ubuntu tenga salida a internet. Ambos adaptadores están habilitados al mismo tiempo.

A continuación, para evitar que cada vez que la máquina Ubuntu se conecte a la red WiFi la IP que se le asigne sea diferente; se le asigna una IP fija. En este caso, la primera IP que obtuvo al conectarse a la WiFi. Para hacer esto, se accede a la configuración de red de la máquina Ubuntu mostrada en la Figura 5.10 y se selecciona la opción *Manual.* Se introduce la IP deseada, la máscara y la puerta de enlace, además del DNS (se puede dejar automático).

Figura 5.10 Configurar IP fija en la máquina virtual.

De esta forma, al habilitar el Punto de Acceso WiFi en el Host, la máquina Ubuntu se conecta automáticamente y lo hace con la IP 192.168.137.126. El diagrama de conexiones queda como en la Figura 5.11.

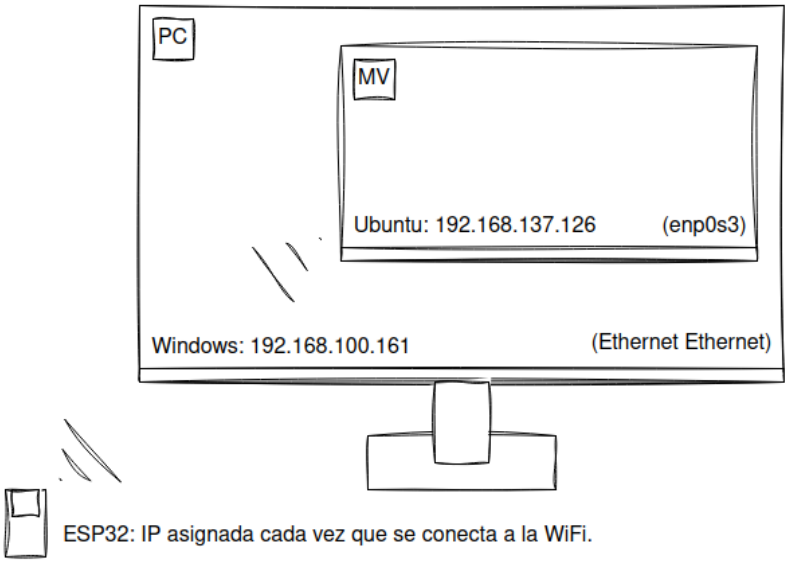


Figura 5.11 Diagrama de red.

5.4. Prueba de comunicación

Una vez realizada la configuración de red como se ha mostrado y empleando los *scripts* de Arduino explicados anteriormente, se procede a realizar una prueba de comunicación para verificar el correcto funcionamiento de la parte comunicativa de la plataforma antes de configurar los componentes FIWARE.

El *script* está diseñado con la finalidad de conectar la ESP32 a un punto de acceso WiFi para poder enviar mediante peticiones tipo POST HTTP los datos de temperatura y humedad recogidos por el sensor DHT11 en formato JSON. El objetivo en este momento es que el portal Node-RED ejecutado en una máquina Ubuntu reciba estos datos y los muestre en un *dashboard* básico, además de contestar a la petición. Para la configuración de la prueba se atiende a la documentación que se encuentra en la siguiente referencia [26].

1. Se emplea el código completo de Arduino, para verificar la recogida de datos, la conversión a formato JSON y la comunicación HTTP vía WiFi.

La información enviada en formato JSON para esta prueba por el ESP32 es: {"sensor":"DHT11", "temp":<valor>,"hum":<valor>}, por lo que se debe añadir el campo "sensor" en la creación de la cadena JSON, lo que se hace modificando el código de la siguiente manera.

```
String json;
DynamicJsonDocument doc(1024);
doc["sensor"] = "DHT11";
doc["temp"]= t;
doc["hum"]= h;
serializeJson(doc, json);
```

2. Para recibir las peticiones en Node-RED, se ejecuta primero el programa, siguiendo los pasos especificados en el Anexo A.5.

3. Se configura un flujo en Node-RED que permita la recepción de peticiones HTTP. El flujo es el siguiente.

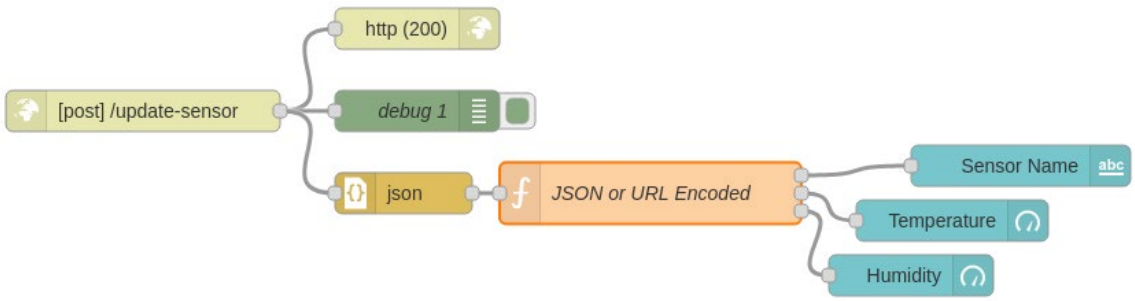


Figura 5.12 Flujo Node-RED para prueba de comunicación.

A continuación, se va a explicar el flujo bloque por bloque.

- *[post] /update-sensor:* recibe las peticiones HTTP de tipo POST que van dirigidas al destino *…/update-sensor.*

- *http (200):* este bloque envía un código 200 cuando se recibe una petición en el bloque anterior. Este mensaje será recibido en Arduino como *httpResponseCode.*

- *debug 1:* muestra en la consola de *debugs* de Node-RED los mensajes recibidos por el bloque anterior para poder ver su contenido y poder gestionarlos.

- *json:* garantiza que el mensaje está en formato JSON, o lo convierte si no lo está.

- *JSON or URL Encoded:* es capaz de extraer tres valores para tres variables diferentes, a los que da salida hacia un *dash* de texto (va a mostrar el nombre del sensor) y dos indicadores que muestran los valores de temperatura y humedad respectivamente.

Figura 5.13 Función que implementa el bloque JSON or URL Encoded.

Para facilitar la creación de este flujo, se puede copiar el código que se recoge en el Anexo C.1 y pegarlo en la ventana que aparece al seleccionar en el menú la opción *Import.*

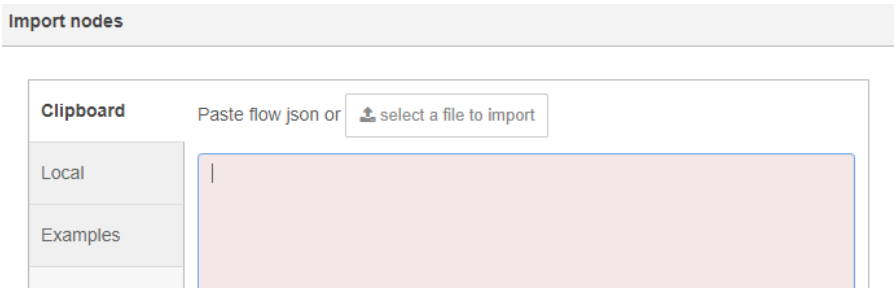


Figura 5.14 Importar nodos en Node-RED.

4. Se modifica la dirección de destino de las peticiones HTTP para apuntar a Node-RED, concretamente al primer bloque, *[post] /update-sensor.*

```
String serverName = "http://192.168.137.126:1880/update-sensor";
```

5. Se carga el código y se comienza a recibir información en Node-RED. En el *dashboard,* se obtiene la información siguiente:

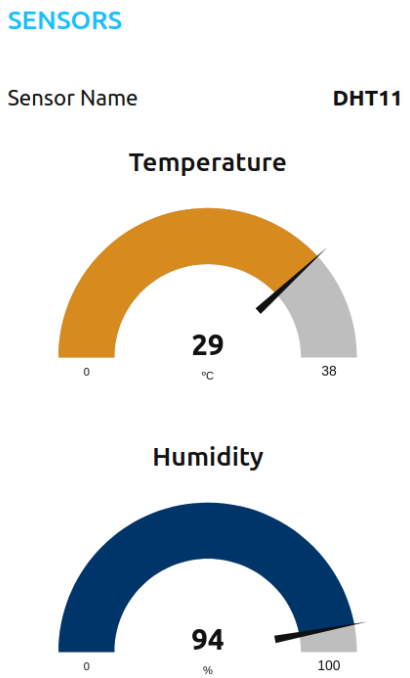


Figura 5.15 Dashboard Node-RED.

Para obtener el diseño de pantalla de la Figura 5.15, la configuración de los distintos elementos del dashboard es la mostrada en la Figura 5.16. Existen muchas variantes, colores, formas y estilos para personalizar la presentación de los datos.

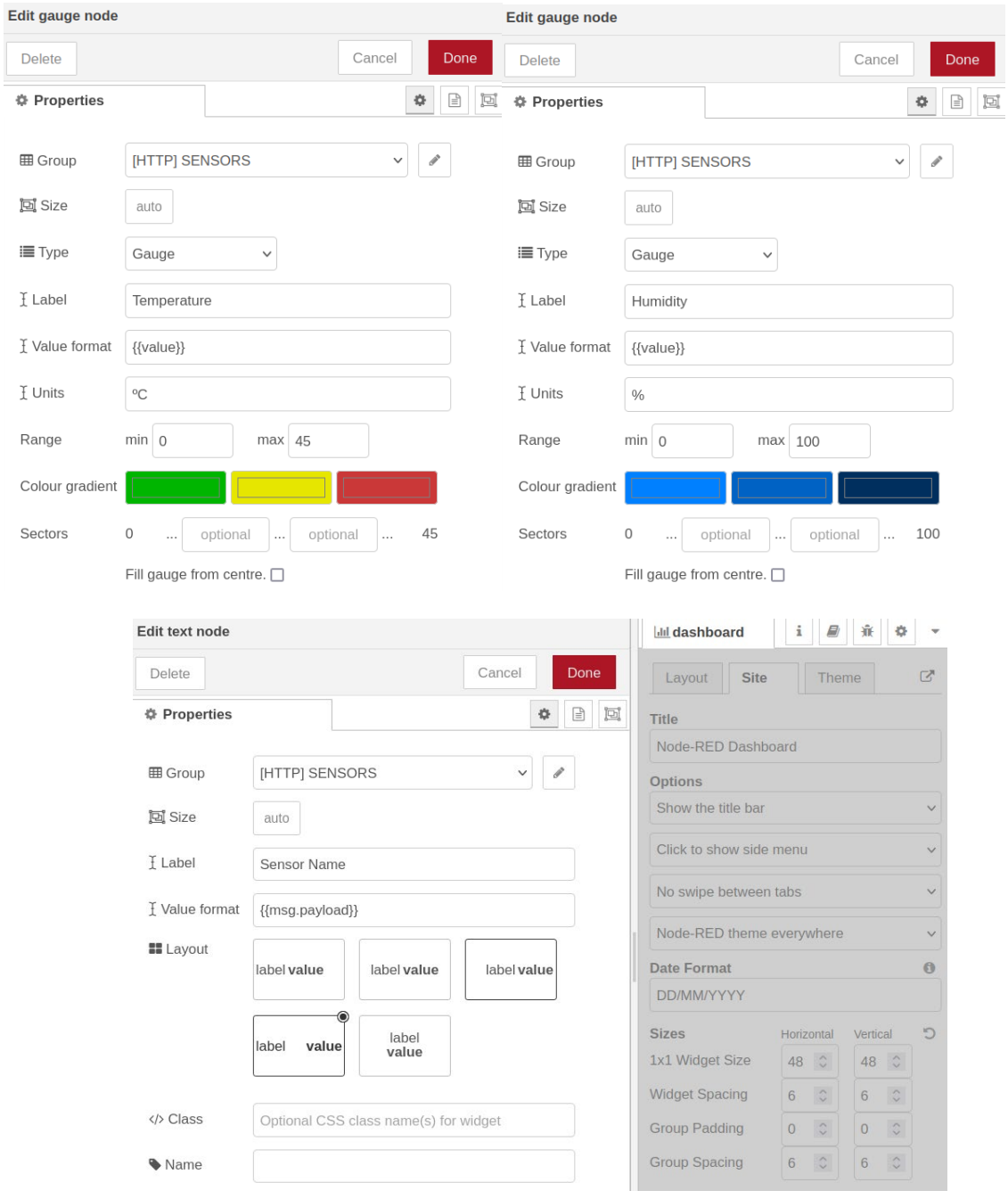


Figura 5.16 Configuración de los tres indicadores y del dashboard general.

De esta forma, se verifica que la comunicación funciona. **El *script* de Arduino queda validado**. Con esta certeza, se puede continuar con el **siguiente paso**, que consiste en la implementación de la **plataforma FIWARE**, desplegando cada componente y configurándose de la manera adecuada para ser capaces de recibir y almacenar la información proveniente del sensor.

6. FIWARE

A continuación, una vez validada la parte de comunicación, se busca desplegar la plataforma FIWARE componente a componente. Como se ha comentado anteriormente, para esta tarea se va a emplear Docker y Docker-Compose, corriendo sobre la máquina virtual Ubuntu.

Así pues, el primer paso es la instalación de Docker en este entorno, proceso recogido en el Anexo A.2.

Una vez hecho esto, se crea un nuevo directorio designado como "fiware" escribiendo en la consola el siguiente comando.

```
sudo mkdir fiware
```

En este directorio, se crea un archivo de texto de tipo YAML denominado *docker-compose.yml* escribiendo el comando siguiente.

```
sudo nano docker-compose.yml
```

Este es el archivo en el cual se van a incluir las configuraciones de los diferentes componentes FIWARE, que se lanzarán como un único contenedor de manera sencilla.

6.1. Despliegue de MongoDB y OCB

Como se ha visto, MongoDB es la base de datos en la que se almacena la información relacionada con Orion. Por ello, para desplegar Orion es necesario desplegar MongoDB primero.

No es necesario descargar las imágenes de ambos componentes para poder lanzar el contenedor, ya que Docker, si la imagen no está instalada en el equipo, la descarga de su repositorio al lanzar el contenedor por primera vez. La arquitectura desarrollada en este apartado se muestra en la Figura 6.1.

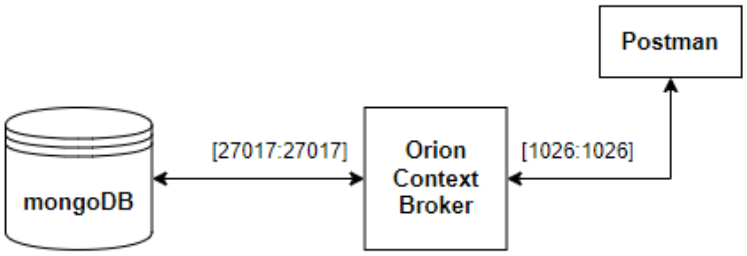


Figura 6.1 Estructura de plataforma FIWARE

El archivo *docker-compose.yml* es un archivo de tipo YAML [27] en el que se definen los diferentes servicios, redes y volúmenes para una aplicación Docker. Es importante respetar la anidación de los diferentes apartados, por ello se muestra el código al completo en el Anexo D.

En el ya mencionado archivo *docker-compose.yml,* la configuración necesaria para poder lanzar tanto MongoDB como Orion es la siguiente. En primer lugar, se define la versión del archivo. Existen diferentes versiones, la 1, 2, 3 y la versión *Compose Specification.* Comparten algunas características entre sí, y las versiones superiores añaden características nuevas que mejoran funcionalidades respecto a las anteriores. En este caso se usa la versión 3, por ser bastante reciente y sobre la que más bibliografía se ha podido encontrar. [28]

En concreto, se usa la 3.4, una versión mejorada con respecto a la versión 3 que introduce nuevos parámetros. Está disponible para versiones de *Docker Engine* superiores a la 17.09.0.

```
version: "3.4"
```

A continuación, se definen los servicios. El orden en el que los diferentes servicios aparecen en el archivo de configuración no es relevante. Se va a comenzar con *orion.* En primer lugar, se escribe la imagen a utilizar, se le asigna al servicio un nombre de anfitrión arbitrario, y se nombra también el contenedor.

```
services:
  orion:
    image: fiware/orion-ld:1.2.0-PRE-1295
    hostname: orion
    container_name: fiware-orion
```

Se especifican los puertos en los que va a funcionar la aplicación.

```
    ports:
      - "1026:1026"
```

Se define la red/es interna en las que va a trabajar la aplicación, en este caso la que se crea por defecto al lanzar el contenedor.

```
    networks:
      - default
```

Se establecen prioridades en el arranque. Como Orion guarda la información en MongoDB, esta se debe iniciar antes.

```
    depends_on:
      - mongo-db
```

Se especifica el nombre de la base de datos anfitrión (dbhost) para que la aplicación sepa dónde encontrar la base de datos y poder realizar una conexión. Se establece el nivel de registro de la aplicación en DEBUG, lo que implica que la aplicación registrará mensajes de depuración detallados para ayudar con la depuración y resolución de problemas.

```
    command: -dbhost mongo-db -logLevel DEBUG
```

A continuación, se incluye el servicio de la base de datos, en concreto mongo-db. Al igual que en el servicio anterior, se asignan valores a los siguientes campos.

```
  mongo-db:
    image: mongo:4.2
    hostname: mongo-db
    container_name: db-mongo
```

Se configuran los puertos de igual manera, en este caso, por defecto

```
    ports:
      - "27017:27017"
```

Se establece la red.

```
    networks:
      - default
```

El siguiente apartado de la configuración de MongoDB, *volumes,* es de gran importancia para el correcto funcionamiento de la aplicación. Los volúmenes son el mecanismo preferido para almacenar la información generada y consumida por los contenedores. [29]

```
    volumes:
      - ./mongodata:/data/db
```

Se especifican condiciones para el despliegue.

```
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure
```

Para finalizar, se declaran las redes, *networks* y los volúmenes, *volumes*.

```
networks:
    default:
volumes:
  mongo-db: ~
```

En resumen, las imágenes han sido seleccionadas con una versión concreta, para evitar posibles fallos o incompatibilidades al ejecutar una versión nueva. Concretamente, se emplean:

- *fiware/orion-ld:1.2.0-PRE-1295*
- *mongo:4.2*

* Para cargar la versión más actual, en vez de especificar la versión, se escribe sólo el nombre de la imagen (*fiware/orion-ld, mongo*).

Ambos tienen asignada la *network* por defecto al crear el contenedor. En este campo se puede asignar una *ipv4* fija a cada contenedor, pero para este proyecto no es algo necesario.

Una vez finalizada la configuración del archivo *docker-compose.yml* se guarda siguiendo las indicaciones de la consola. Una vez hecho esto, se ejecuta el archivo con el siguiente comando.

```
sudo docker-compose -p fiware up -d
```

De esta forma se levantan[2] los contenedores anteriormente configurados y se crea la red de trabajo. Para comprobar que los contenedores se han ejecutado de la manera correcta y que están funcionando se escribe el siguiente comando.

```
sudo docker ps
```

Para verificar si funciona de manera correcta, se emplea Postman para realizar una petición HTTP a Orion de tipo GET con destino *http:///192.168.137.126:1026/v2/,* muy sencilla, cuyo resultado debe ser el mostrado en la Figura 6.2.

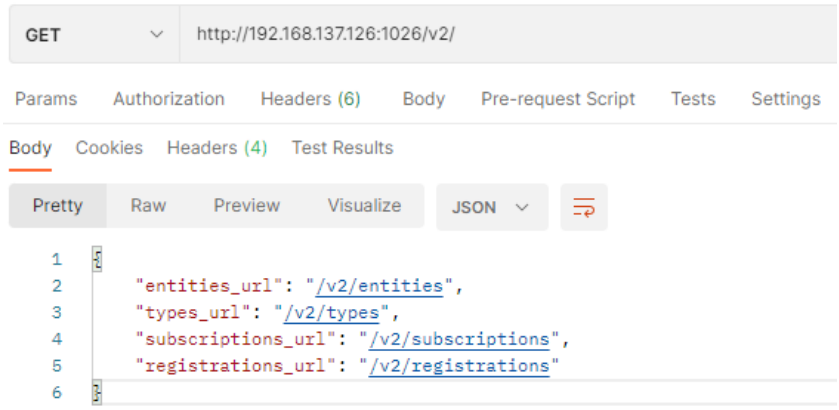


Figura 6.2 Comprobación del funcionamiento de Orion y MongoDB.

[2] Se emplea el verbo levantar para referirse a la acción de "activar/desplegar" un contenedor de Docker ya que el comando ejecutado es "up".

6.2. Entidades

Como se ha explicado, las entidades son la manera de representar en el entorno FIWARE objetos físicos que existen en el mundo real, como el sensor DHT11 que se usa en este proyecto. Por ello, se debe crear una entidad en Orion que contenga la información relevante, tanto datos estáticos como parámetros dinámicos.

6.2.1. Estructura

En este caso, al tratarse de un sensor de temperatura y humedad, la entidad tiene la siguiente forma, en **JSON**:

```
{
    "id": "urn:ngsi-ld:Sensor:001",
    "type": "Sensor",
    "name": {
        "type": "Text",
        "value": "DHT11"
    },
    "temperature": {
        "type": "Integer",
        "value": 24
    },
    "humidity": {
        "type": "Integer",
        "value": 17
    }
}
```

Con esta estructura, los **atributos** empleados para caracterizar la entidad son los siguientes.

- ***id:*** es un atributo indispensable y por tanto **obligatorio** para la creación de la entidad. Designa a la entidad como tal. Su contenido se estructura siguiendo el **formato LD** (*urn:ngsi-ld:<entity-type>:<entity-id>*), útil para la estandarización y escalabilidad, de esta forma no habrá dos *id* iguales en la plataforma.

- ***type:*** es un atributo indispensable y por tanto **obligatorio** para la creación de la entidad. Se refiere al tipo de entidad. Siguiendo el **formato LD**, el valor de este campo debe corresponder con el <*entity-type*> colocado en el campo *id.*

- ***name****:* no es obligatorio. Contiene dos campos, *type* y *value*, que sirven para designar el tipo de dato que se incluye en el campo *value,* y el propio valor.

- ***temperature****:* atributo **dinámico** ya que su valor se actualiza con cada medición del sensor. Al igual que cada atributo, contiene un campo *type* que especifica el tipo de dato, en este caso *Integer* y su *value*, un número. Se puede cargar un número al azar o dejar el *String* vacío.

- ***humidity****:* atributo dinámico de estructura idéntica al anterior.

Como se puede observar, cada atributo se compone de un campo *type* y otro campo *value*. Se puede incluir metadatos con información adicional. La estructura de un metadato se compone de tres campos, *name, type* y *value*.

Se puede añadir de manera sencilla otros atributos como la fecha de instalación, fecha de última revisión, nombre del fabricante, lugar donde se encuentran, número de serie; con el fin de obtener un perfil más completo del mismo y poder localizarlo con mayor facilidad.

6.2.2. Cabeceras

Es necesario añadir otras dos cabeceras, *headers,* a las existentes por defecto. En FIWARE, las cabeceras se emplean para enviar información adicional junto con las peticiones HTTP que no se puede transmitir en el cuerpo de estas.

- *fiware-service:* se especifica el servicio que se está empleando.
- *fiware-servicepath:* sirve para especificar la ruta del servicio que se está empleando.

Esta información se utiliza para identificar el contexto de la solicitud y garantizar que se accede a los datos correctos.

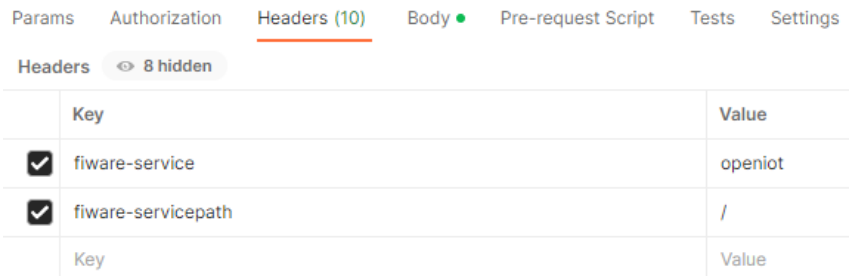


Figura 6.3 Cabeceras empleadas.

Para crear la entidad en Orion, es necesario realizar una petición HTTP tipo POST al destino *http://192.168.137.126:1026/v2/entities* con las cabeceras indicadas anteriormente.

Para comprobar que se ha creado una entidad correctamente, se puede realizar en Postman una petición tipo GET a la dirección *http://192.168.137.126:1026/v2/entities,* incluyendo las cabeceras. Seleccionando el parámetro *options=keyValues* la petición y su respuesta se puede ver en la Figura 6.4.

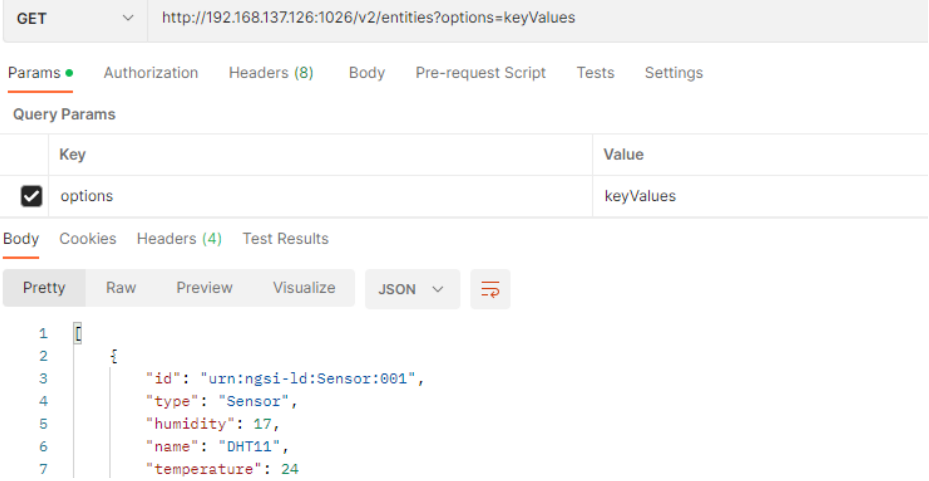


Figura 6.4 GET para ver la entidad creada.

6.3. Despliegue de IoT Agent

El siguiente paso consiste en la configuración del IoT Agent, que será quien reciba la comunicación del dispositivo IoT. Así, la arquitectura implementada se muestra en la Figura 6.5.

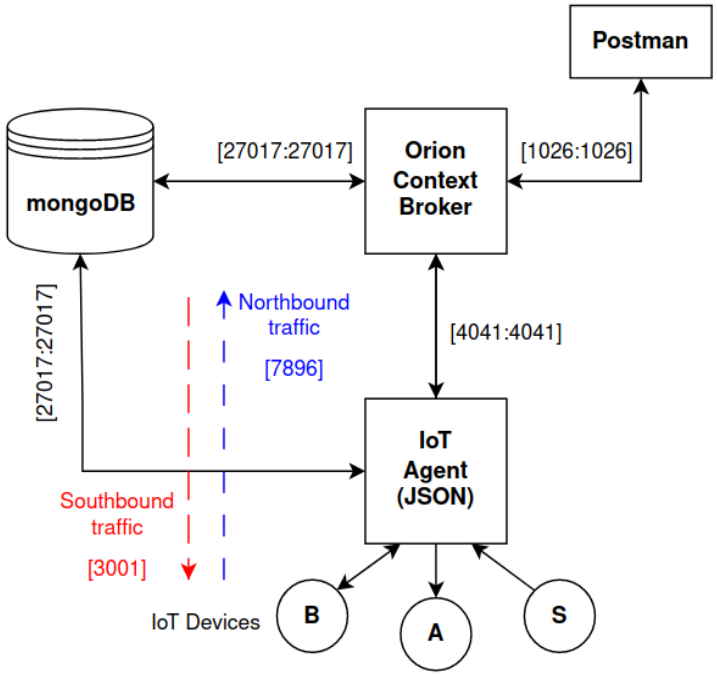


Figura 6.5 Arquitectura y puertos del entorno FIWARE.

Para realizar el despliegue del IoT Agent hay que añadir un nuevo apartado de configuración al archivo *docker-compose.yml.* A la configuración existente se añade un nuevo servicio.

```
iot-agent:
   image: fiware/iotagent-json:1.26.0
   hostname: iot-agent
   container_name: fiware-iot-agent
```

Como también guarda la información en la base de datos Mongo, se debe lanzar después de que mongo-db se inicie.

```
   depends_on:
     - mongo-db
```

Emplea la misma red que los dos componentes anteriores.

```
   networks:
     - default
```

Para sus comunicaciones, emplea estos dos puertos. Por el 4041 se comunica con Orion y por el puerto 7896 recibe la comunicación entrante de dispositivos y sensores.

```
   ports:
      - "4041:4041"
      - "7896:7896"
```

Por último, se configuran diversas variables de entorno. Son las siguientes y la mayoría se comprende de forma clara a qué se refiere.

```
environment:
      - "IOTA_CB_HOST=orion"
      - "IOTA_CB_PORT=1026"
      - "IOTA_NORTH_PORT=4041"
      - "IOTA_REGISTRY_TYPE=mongodb"
      - "IOTA_LOG_LEVEL=DEBUG"
      - "IOTA_TIMESTAMP=true"
      - "IOTA_CB_NGSI_VERSION=v2"
      - "IOTA_AUTOCAST=true"
      - "IOTA_MONGO_HOST=mongo-db"
      - "IOTA_MONGO_PORT=27017"
      - "IOTA_MONGO_DB=iotagent-json"
      - "IOTA_HTTP_PORT=7896"
      - "IOTA_PROVIDER_URL=http://iot-agent:4041"
```

Una vez configurado el archivo modificado, se ejecuta empleando los mismos comandos que en el apartado 6.1.

Para verificar que el IoT Agent recibe peticiones correctamente, se solicita mediante una petición HTTP de tipo GET su información propia. Al obtener respuesta se verifica su funcionamiento.

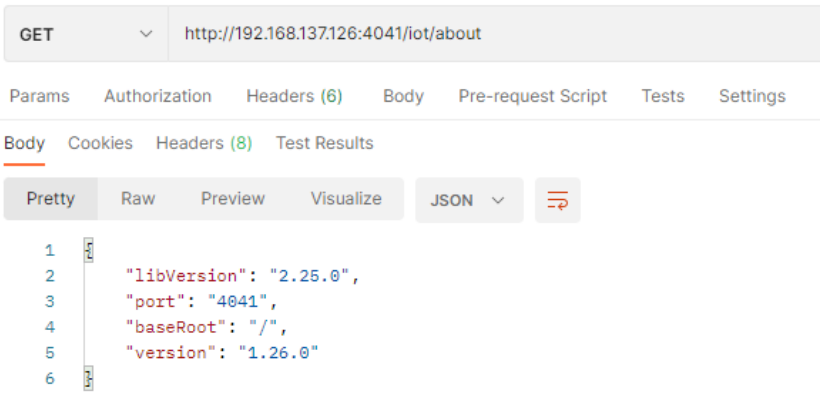


Figura 6.6 Comprobación del funcionamiento del IoT Agent.

6.4. Aprovisionamiento

Para conseguir que el IoT Agent funcione correctamente se debe configurar siguiendo las siguientes pautas. Hay que indicar el protocolo de comunicación, la ubicación de Orion, seguridad y lenguaje.

6.4.1. Grupo de servicio

Este es el primer paso de configuración del IoT. Se va a informar al IoT Agent de que un dispositivo va a enviar mensajes al puerto IOTA_HTTP_PORT, donde está escuchando comunicaciones Northbound.

Se lleva a cabo una petición de tipo POST al destino *http://192.168.137.126:4041/iot/services* que incluye las cabeceras explicadas anteriormente y un cuerpo que contiene los siguientes parámetros.

Dentro del campo *services* se incluye:

- *apikey:* se define la clave que tendrá que contener un mensaje proveniente de un dispositivo para ser aceptado. Lo conveniente sería colocar una apikey más larga y compleja para mayor seguridad, sin embargo, para facilitar el desarrollo y pruebas, en este caso se emplea una palabra sencilla y fácil de recordar como: *apikey*.

- *cbroker:* dirección en la cual se encuentra el Orion Context Broker, en este caso *http://192.168.137.126:1026*.

- *entity_type:* se rellena con *Thing* para dispositivos desconocidos.

- *resource:* se informa al IoT Agent de que el destino al cual el dispositivo IoT va a enviar la información es */iot/json*.

```
{
    "services": [
        {
            "apikey": "apikey",
            "cbroker": "http://192.168.137.126:1026",
            "entity_type": "Thing",
            "resource": "/iot/json"
        }
    ]
}
```

6.4.2. Dispositivo

En este apartado se realiza la configuración necesaria para que la información enviada por el dispositivo sea reconocida por el IoT Agent y sirva para actualizar los valores de los atributos *temperature* y *humidity* de la entidad creada en Orion. De esta forma el IoT Agent es capaz de reconocer los parámetros enviados en JSON por el sensor y actualizar los atributos correspondientes.

Se lleva a cabo una petición de tipo POST al destino *http://192.168.137.126:4041/iot/devices* que incluye las cabeceras [3] explicadas anteriormente y un cuerpo que contiene los siguientes parámetros:

- *device_id:* nombre del dispositivo, luego servirá para enviar la información.
- *entity_name:* nombre de la entidad que se va a actualizar.
- *entity_type:* tipo de la entidad.
- *timezone:* zona horaria.
- *attributes:* En este campo se incluyen los atributos que van a ser actualizados por la información entrante, para relacionarlas con el dato que le llega al IoT Agent. El dato enviado es de tipo JSON y se compone de dos pares key-value. {"t":22,"h":14}, por ello se debe relacionar *t* con el atributo temperatura y *h* con el atributo humedad.
    - *object_id:* identificador del par JSON que trae la información relevante para este atributo. En este caso, el dispositivo envía un dato tipo JSON {“t”: ”valor”, “h”: ”valor”}. Los identificadores son “t” para temperatura y “h” para humedad.
    - *name:* nombre del atributo en la entidad.
    - *type:* tipo de dato.

```
{
    "devices": [
        {
            "deiced": "DHT11",
            "entity_name": "urn:ngsi-ld:Sensor:001",
            "entity_type": "Sensor",
            "timezone": "Europe/Berlin",
            "attributes": [
                {
                    "object_id": "t",
                    "name": "temperature",
                    "type": "Integer"
                },
                {
                    "object_id": "h",
                    "name": "humidity",
                    "type": "Integer"
                }
            ]
        }
    ]
}
```

[3] Es un error habitual no colocar las cabeceras en las peticiones.

6.5. Envío de datos a la plataforma FIWARE

Una vez completado el paso anterior, lo siguiente es comenzar a enviar datos a la dirección y puerto por el que el IoT Agent escucha la comunicación Northbound, es decir, incluir en el *script* de la placa ESP32 el destino adecuado:

http://192.168.137.126:7896/iot/json?i=<device_id>&k=apikey

- ***192.168.137.126:*** dirección IP en la que está corriendo el IoTA.
- ***7896:*** puerto por el que el IoTA escucha comunicaciones Northbound (provenientes de los dispositivos).
- ***/iot/json:*** destino indicado en apartado *resource.*
- ***?i=<device_id>:*** identificador de dispositivo que debe coincidir con el campo que tiene este mismo nombre descrito en el Apartado 6.4.2.
- ***&k=apikey:*** clave para identificar al dispositivo que envía la petición designada en el campo *apikey.*

Se modifica el código de Arduino para que las peticiones HTTP apunten a este destino, y se reduce el dato a {"t":22,"h":14}, lo que conlleva cambios en estas partes con respecto al código empleado en la prueba de comunicación:

```
//Your Domain name with URL path or IP address with path
String serverName = "http://192.168.137.126:7896/iot/json?i=DHT11&k=apikey ";
DynamicJsonDocument doc(1024);
doc["t"]= t;
doc["h"]= h;
serializeJson(doc, json);
Serial.println(json);
int httpResponseCode = http.POST(json);
```

De esta forma se obtiene por pantalla al observar el monitor serie el siguiente mensaje*: HTTP Response code: 200*, indicando que se lleva a cabo la comunicación de forma correcta. Se confirma la recepción de la petición POST por parte de la plataforma FIWARE cuando se modifica la entidad.

Para visualizar el cambio en el valor de la entidad, se lleva a cabo una petición de tipo GET mediante Postman como la que se muestra en la Figura 6.7.

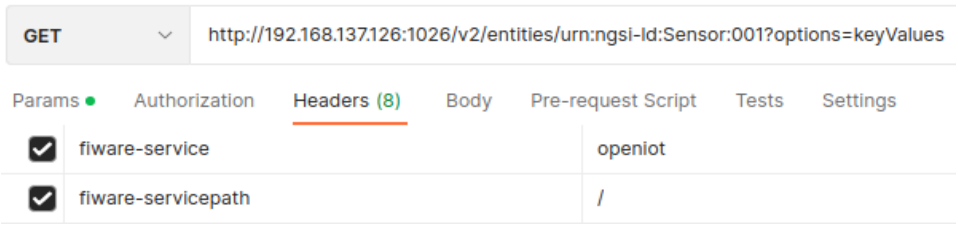


Figura 6.7 Petición GET para observar cambios en los valores de la entidad.

La respuesta obtenida es la deseada, con los valores actualizados para cada atributo del sensor con los datos más recientes enviados por el ESP32. Se verifica la correcta comunicación entre el dispositivo IoT y la plataforma FIWARE

6.6. Datos persistentes

Se verifica el correcto funcionamiento del sistema de comunicación entre la placa ESP32 y la plataforma FIWARE. El siguiente paso es crear una base de datos donde se almacene toda la información recogida por el sensor para posteriormente poder acceder a ella y consultarla para emplear esta información de manera inteligente.

6.6.1. Arquitectura

Con la arquitectura actual, la plataforma es capaz de guardar la información de configuraciones y entidades de la plataforma, sin embargo, los datos recogidos por el sensor todavía no se pueden almacenar. En la gestión de información de contexto lo verdaderamente relevante es el estado en tiempo real de las entidades y sus atributos, por lo que hay que añadir un componente capaz de almacenar estos datos. Más concretamente, se añade el componente Draco para enviar la información a una base de datos MySQL cuando el contexto se actualiza.

Draco no se limita a una tecnología específica de bases de datos, por lo que se puede emplear la más conveniente para la aplicación. Se va a emplear MySQL por ser además de la empleada en la empresa, la de conexión más directa con PowerBI.

La arquitectura de la plataforma está ahora completa, incluyendo Draco y MySQL. El resultado final puede verse en la Figura 6.8.

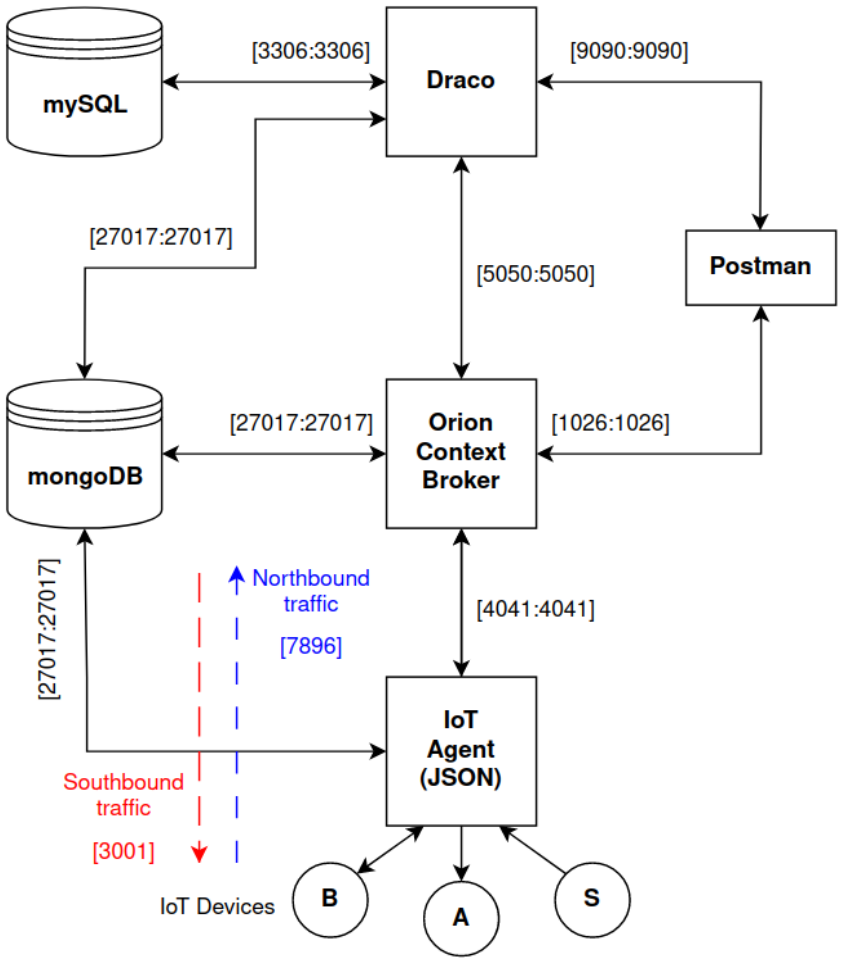


Figura 6.8 Diagramas y puertos de la arquitectura al completo.

6.6.2. Configuración

Al igual que con el resto de los componentes, se debe incluir la configuración de estos dos últimos servicios en el archivo *docker-compose.yml*

Comenzando con *draco,* en primer lugar, se escribe la imagen a utilizar y se nombra también el contenedor.

```
  draco:
    image: ging/fiware-draco:1.1.0
    container_name: draco
```

Se establecen prioridades en el arranque. Como Draco guarda la información en MySQL, esta se debe iniciar antes.

```
    depends_on:
      - mysql-db
```

Se especifica una variable de entorno para definir el puerto en el que se va a abrir el panel web.

```
    environment:
      - "NIFI_WEB_HTTP_PORT=9091"
```

Se configuran los puertos de igual manera. Por defecto está el 9090. No influye para nada.

```
    ports:
      - "9091:9091"
      - "5050:5050"
```

Comprobación del estado de salud del contenedor

```
    healthcheck:
      test: curl --fail -s http://localhost:9091/nifi-api/system-diagnostics ||
exit 1
```

Lo mismo ocurre para MySQL. En primer lugar, se escribe la imagen a utilizar, se le asigna al servicio un nombre de anfitrión arbitrario, y se nombra también el contenedor.

```
  mysql-db:
    image: mysql:5.7
    hostname: mysql-db
    container_name: db-mysql
    restart: always
```

Se especifica el puerto de funcionamiento

```
    ports:
      - "3306:3306"
```

Se define la red/es interna en las que va a trabajar la aplicación, en este caso la que se crea por defecto al levantar el contenedor.

```
networks:
  - default
```

Por último, se configuran dos variables de entorno que son las credenciales de acceso a la base de datos.

```
environment:
  - "MYSQL_ROOT_PASSWORD=123"
  - "MYSQL_ROOT_HOST=%"
```

6.6.3. Ver la base de datos MySQL

Una vez creado el nexo entre Orion, Draco y MySQL, para ver la base de datos se debe acceder a un cliente de MySQL en una terminal de la MV. Para ello es necesario ejecutar el siguiente comando:

sudo docker exec -it db-mysql mysql -h mysql-db -P 3306 -u root -p123

donde *-P* es el puerto, *-u* es el usuario y *-p* la contraseña, definidas en la configuración de MySQL en el archivo *docker-compose.yml*.

```
amartinez@amartinez-VirtualBox:~/fiware$ sudo docker exec -it db-mysql mysql -h mysql-db -P3306 -u root -p123
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.41 MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

Figura 6.9 Ejecutar un cliente MySQL

Para mostrar las bases de datos se escribe: *show databases;* Si el servicio no está corriendo, se obtendrá como respuesta estas cuatro bases de datos que existen por defecto.

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)
```

Figura 6.10 Bases de datos por defecto. Draco inactivo.

Al iniciar el servicio y escribir de nuevo el comando anterior:

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| openiot            |
| performance_schema |
| sys                |
+--------------------+
```

Figura 6.11 Nueva base de datos "openiot" al activar el servicio.

Ahora aparece una nueva base de datos llamada *openiot (*nombre seleccionado en el flujo NiFi), que contiene una tabla para cada entidad registrada en Orion. Para mostrar las tablas de la base de datos se escribe: *show tables from openiot;*

```
mysql> show tables from openiot;
+--------------------------------+
| Tables_in_openiot              |
+--------------------------------+
| urn_ngsi-ld_Sensor_001_Sensor |
+--------------------------------+
1 row in set (0.00 sec)
```

Figura 6.12 Mostrar las tablas de la base de datos "openiot".

La tabla que aparece tiene como nombre: <entity_id><entity_type>. Para ver la tabla se escribe: *select * from `urn_ngsi-ld_Sensor_001_Sensor` limit 10* (para sólo obtener 10 entradas de la tabla). Importante colocar las comillas invertidas ya que la *entity_id* contiene un guion y esto produce un error. De esta forma se coloca el nombre de la tabla como cadena mediante las dos comillas invertidas (`…`).

```
mysql> select*from `urn_ngsi-ld_Sensor_001_Sensor` limit 10;
+--------------+--------------------+------------------+----------------------+------------+-------------+----------+--------------------------+
| recvTimeTs   | recvTime           | fiwareServicePath | entityId             | entityType | attrName    | attrType | attrValue                |
+--------------+--------------------+------------------+----------------------+------------+-------------+----------+--------------------------+
| 1680529034772 | 04/03/2023 13:37:14 |                  | urn:ngsi-ld:Sensor:001 | Sensor     | name        | Text     | DHT11                    |
| 1680529034772 | 04/03/2023 13:37:14 |                  | urn:ngsi-ld:Sensor:001 | Sensor     | temperature | Integer  | 25                       |
| 1680529034772 | 04/03/2023 13:37:14 |                  | urn:ngsi-ld:Sensor:001 | Sensor     | humidity    | Integer  | 17                       |
| 1680529034772 | 04/03/2023 13:37:14 |                  | urn:ngsi-ld:Sensor:001 | Sensor     | TimeInstant | DateTime | 2023-04-03T13:37:14.742Z |
| 1680529034774 | 04/03/2023 13:37:14 |                  | urn:ngsi-ld:Sensor:001 | Sensor     | name        | Text     | DHT11                    |
| 1680529034774 | 04/03/2023 13:37:14 |                  | urn:ngsi-ld:Sensor:001 | Sensor     | temperature | Integer  | 25                       |
| 1680529034774 | 04/03/2023 13:37:14 |                  | urn:ngsi-ld:Sensor:001 | Sensor     | humidity    | Integer  | 17                       |
| 1680529034774 | 04/03/2023 13:37:14 |                  | urn:ngsi-ld:Sensor:001 | Sensor     | TimeInstant | DateTime | 2023-04-03T13:37:14.742Z |
| 1680529069984 | 04/03/2023 13:37:49 |                  | urn:ngsi-ld:Sensor:001 | Sensor     | name        | Text     | DHT11                    |
| 1680529069984 | 04/03/2023 13:37:49 |                  | urn:ngsi-ld:Sensor:001 | Sensor     | temperature | Integer  | 25                       |
+--------------+--------------------+------------------+----------------------+------------+-------------+----------+--------------------------+
10 rows in set (0.00 sec)
```

Figura 6.13 Tabla que contiene los datos enviados por el sensor.

Se puede hacer búsquedas concretas para un atributo en concreto, o un intervalo de tiempo, pero se va a emplear otra herramienta más potente y visual para hacer las peticiones y mostrar datos.

6.7. Suscripciones

Orion cuenta con un sistema de suscripción mediante el cual, realizando una petición de tipo POST al destino *http://192.168.137.126:1026/v2/subscriptions* con las cabeceras ya mencionadas y un cuerpo determinado, se puede suscribir un destino *url* de manera que cuando se produzca un cambio en la información de contexto, dicha dirección reciba un aviso HTTP POST.

Esto es necesario para el funcionamiento de dos componentes, Draco y Node-RED.

La estructura de una suscripción básica en JSON se muestra y comenta a continuación.

```
{
    "description": "Notificar sobre …",
    "subject": {
        "entities": [
            {
                "id": "…",
                "type": "…"
            }
        ],
        "condition": {
            "attributes": [ "…" ],
            "expression": {
               "q": "… > X"
            }
         }
    },
    "notification": {
        "http": {
            "url": "https://direccion_notificación "
        }
    },
    "throttling": 5
}
```

La estructura de la suscripción consta de cuatro apartados principales que contienen a su vez subapartados.

- *description* informa e identifica una suscripción en concreto.

- *subject* contiene dos subcampos, *entities* y *condition,* en los que se especifica la entidad o entidades cuyo cambio va a producir la notificación, y a un mayor nivel, los atributos en concreto que van a producir este aviso, con la posibilidad de definir unas reglas.

- *notification* contiene el subcampo *http,* el cual a su vez contiene un valor *url* que indica el destino al cual se envían las notificaciones.

- *throttling* define el ratio con el que se muestran los cambios, en este caso cada 5 segundos se enviará una notificación, sin atender a lo que ocurra en el transcurso de este tiempo.

Se puede observar que en la petición no se especifica el tipo de base de datos que se emplea. La suscripción es la misma para todos los tipos de bases de datos, la dependencia queda definida en el archivo *docker-compose.yml.*

6.7.1. Draco

Es necesario informar a Draco de los cambios en el contexto y proporcionarle un punto de unión con el resto de la arquitectura. Siguiendo el esquema de la Figura 6.8, Draco se comunica con Orion por el puerto 5050.

Se realiza por tanto una petición POST al destino */v2/subscriptions* del OCB para suscribir a Draco a los cambios en la información de contexto, de forma que la base de datos MySQL se actualice en tiempo real con la información del sensor y esté disponible para mostrarse en el sistema de control en el menor tiempo posible.

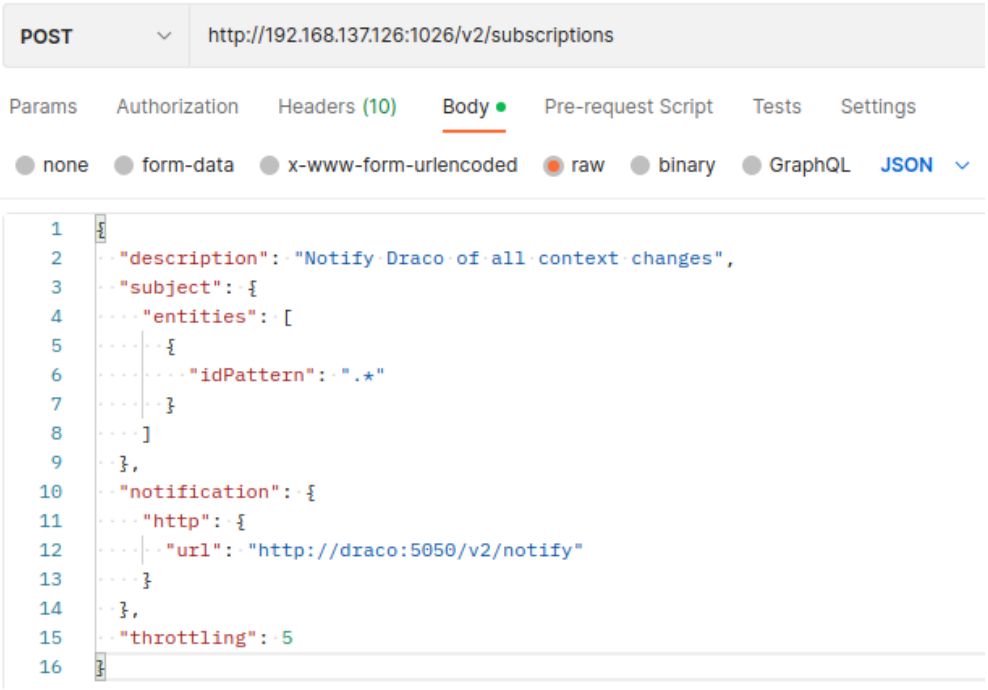


Figura 6.14 Suscribir Draco a los cambios de contexto.

Se incluyen las cabeceras empleadas en el aprovisionamiento de servicio y dispositivo. Los campos empleados son los siguientes. En el campo *entities* se indica con *"idPattern": ".*"* que cualquier cambio en cualquier entidad va a producir una notificación. Esto tiene sentido ya que se trata del gestor de la base de datos y se quiere almacenar toda la información que se produce.

6.7.2. Node-RED

Se va a mostrar la configuración de la suscripción de Node-RED a pesar de que todavía no se ha explicado en profundidad su funcionalidad. Se va a emplear para gestionar las alarmas y avisos.

```
{
    "description": "Notificar a Node-RED si humedad mayor que 50%",
    "subject": {
        "entities": [
            {
                "idPattern": ".*",
                "type": "Sensor"
            }
        ],
        "condition": {
            "attributes": [ "humidity" ],
            "expression": {
               "q": "humidity>50"
            }
         }
    },
    "notification": {
        "http":{
            "url": "http://192.168.137.126:1880/alarma"
        },
        "attrs": [ ]
    }
}
```

Se incluyen las cabeceras empleadas en el aprovisionamiento de servicio y dispositivo. Los campos empleados son los siguientes. En el campo *entities* se indica con *"idPattern": ".*"* que cualquier cambio en cualquier entidad va a producir una notificación.

En el campo *condition* se especifica el atributo *humidity* para el cual se establece una condición, que la notificación se produzca cuando se produzca un cambio en la información de contexto que implique una humedad mayor que 50 en este caso. Esto es sólo un ejemplo para entender las posibilidades que esto ofrece.

En el campo de notificación, se coloca un destino que apunta a Node-RED, concretamente a una pestaña encargada de recibir peticiones HTTP denominada *prueba.*

7. Visualización Power BI

Hasta este momento, se ha logrado desarrollar una plataforma FIWARE para la comunicación entre un sensor ambiental y una base de datos. En este apartado se va a implementar la herramienta que permite visualizar estos datos y obtener información relevante del análisis estadístico de los mismos.

Para ello se emplea este conocido programa de *Business Intelligence* de Microsoft, por ser gratuito y uno de los más empleados para estas tareas por sus amplias funcionalidades e interesantes características.

7.1. MySQL Workbench

Como se ha explicado anteriormente, los datos recogidos por el sensor DHT11 se almacenan en una base de datos MySQL a la cual se debe acceder con Power BI. Existe lo que en principio puede parecer un inconveniente, y es que Power BI sólo puede correr sobre Windows, por lo que no se puede abrir en la máquina Ubuntu en la que se levantan los contenedores.

Por ello, aunque la red esté bien configurada, surge la duda de si Power BI se puede sincronizar con una base de datos ejecutada en otra máquina, en este caso en la máquina virtual Ubuntu. Para verificar que se puede acceder desde una aplicación ejecutada en el PC anfitrión se prueba primero con MySQL Workbench.

Con este objetivo, y también con el de tener un portal con un interfaz mejor para navegar por los menús de MySQL, se instala y se configura MySQL Workbench.

7.2. Despliegue y configuración

A continuación, se procede a cargar los datos de la *database demo* en Power Bi. Para ello en primer lugar se descarga Power BI Desktop en el PC Host Windows desde la página oficial y de manera gratuita. Se instala de manera sencilla y al abrir el programa aparece el siguiente menú.

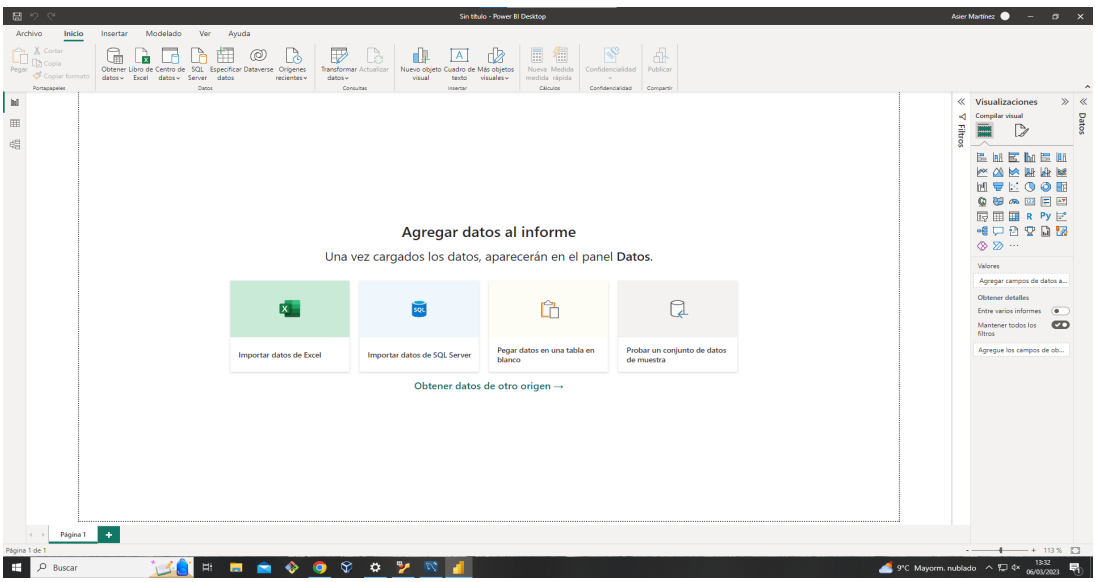


Figura 7.1 Menú de inicio Power BI.

Como la base de datos MySQL se ubica en la máquina virtual Ubuntu, habrá que especificar puerto y dirección IP en algún momento además de las credenciales de acceso. Sin embargo, va a ser necesario un paso previo; la instalación de un componente adicional, un conector para poder gestionar datos MySQL, por lo que se selecciona la opción *Importar datos de SQL Server* y aparece el siguiente aviso.

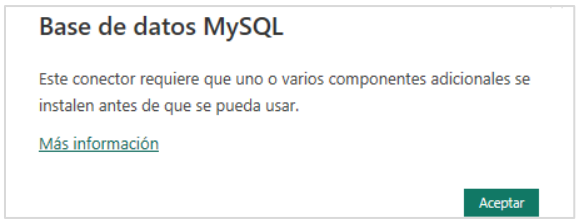


Figura 7.2 Instalación de conector adicional.

Al hacer clic en *Más información* se accede al siguiente enlace [30] donde se debe descargar el conector de datos necesario.

7.3. Configuración de las fuentes de datos

Una vez instalado el conector, al acceder nuevamente a Power BI y seleccionar *Obtener datos de otro origen.* Aparece el menú de selección mostrado en la Figura 7.3.

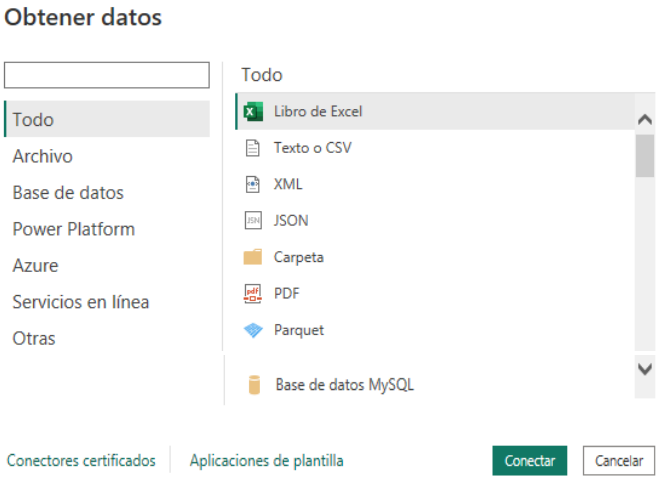
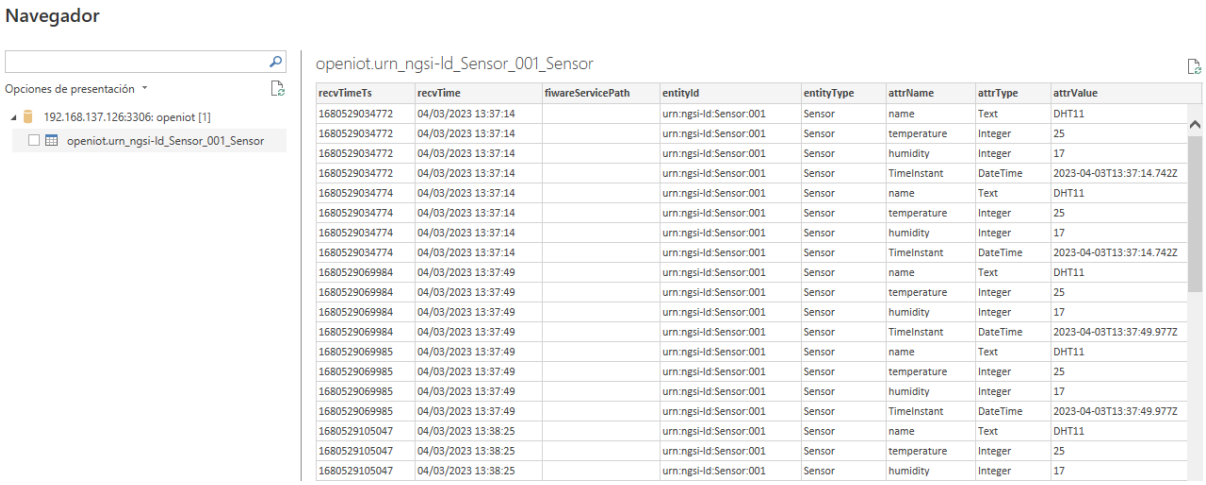


Figura 7.3 Menú de obtención de datos.

Se selecciona la opción *Base de datos MySQL,* se accede a la pestaña mostrada en la Figura 7.4.

Figura 7.4 Conectar base de datos MySQL.

En el espacio *Servidor* se escribe la dirección IP de la máquina Ubuntu junto con el puerto por el que se accede, según la arquitectura, el 3306. En el espacio *Base de datos* se especifica el nombre de la base de datos que se quiere abrir, en este caso, *demo*. Aceptar para acceder al menú de la Figura 7.5.



recvTimeTs	recvTime	fiwareServicePath	entityId	entityType	attrName	attrType	attrValue
1680529034772	04/03/2023 13:37:14		urn:ngsi-ld:Sensor:001	Sensor	name	Text	DHT11
1680529034772	04/03/2023 13:37:14		urn:ngsi-ld:Sensor:001	Sensor	temperature	Integer	25
1680529034772	04/03/2023 13:37:14		urn:ngsi-ld:Sensor:001	Sensor	humidity	Integer	17
1680529034772	04/03/2023 13:37:14		urn:ngsi-ld:Sensor:001	Sensor	TimeInstant	DateTime	2023-04-03T13:37:14.742Z
1680529034774	04/03/2023 13:37:14		urn:ngsi-ld:Sensor:001	Sensor	name	Text	DHT11
1680529034774	04/03/2023 13:37:14		urn:ngsi-ld:Sensor:001	Sensor	temperature	Integer	25
1680529034774	04/03/2023 13:37:14		urn:ngsi-ld:Sensor:001	Sensor	humidity	Integer	17
1680529034774	04/03/2023 13:37:14		urn:ngsi-ld:Sensor:001	Sensor	TimeInstant	DateTime	2023-04-03T13:37:14.742Z
1680529069984	04/03/2023 13:37:49		urn:ngsi-ld:Sensor:001	Sensor	name	Text	DHT11
1680529069984	04/03/2023 13:37:49		urn:ngsi-ld:Sensor:001	Sensor	temperature	Integer	25
1680529069984	04/03/2023 13:37:49		urn:ngsi-ld:Sensor:001	Sensor	humidity	Integer	17
1680529069984	04/03/2023 13:37:49		urn:ngsi-ld:Sensor:001	Sensor	TimeInstant	DateTime	2023-04-03T13:37:49.977Z
1680529069985	04/03/2023 13:37:49		urn:ngsi-ld:Sensor:001	Sensor	name	Text	DHT11
1680529069985	04/03/2023 13:37:49		urn:ngsi-ld:Sensor:001	Sensor	temperature	Integer	25
1680529069985	04/03/2023 13:37:49		urn:ngsi-ld:Sensor:001	Sensor	humidity	Integer	17
1680529069985	04/03/2023 13:37:49		urn:ngsi-ld:Sensor:001	Sensor	TimeInstant	DateTime	2023-04-03T13:37:49.977Z
1680529105047	04/03/2023 13:38:25		urn:ngsi-ld:Sensor:001	Sensor	name	Text	DHT11
1680529105047	04/03/2023 13:38:25		urn:ngsi-ld:Sensor:001	Sensor	temperature	Integer	25
1680529105047	04/03/2023 13:38:25		urn:ngsi-ld:Sensor:001	Sensor	humidity	Integer	17

Figura 7.5 Base de datos en bruto.

Como se puede ver, aparece la tabla de interés. A continuación, se van a transformar los datos para eliminar columnas que no sean útiles y hacer un primer filtrado de la información. De manera muy sencilla, se obtiene la tabla de la Figura 7.6.

	timeStamp	attrName	attrValue	Temperatura	Humedad
1	04/05/2023 10:43:30	humidity	13	null	13
2	04/05/2023 10:43:30	name	DHT11	null	null
3	04/05/2023 10:43:30	temperature	24	24	null
4	04/05/2023 10:43:30	TimeInstant	2023-04-05T08:43:30.193Z	null	null
5	04/05/2023 10:42:55	humidity	13	null	13
6	04/05/2023 10:42:55	name	DHT11	null	null
7	04/05/2023 10:42:55	temperature	24	24	null
8	04/05/2023 10:42:55	TimeInstant	2023-04-05T08:42:55.121Z	null	null
9	04/05/2023 10:42:20	humidity	13	null	13
10	04/05/2023 10:42:20	name	DHT11	null	null
11	04/05/2023 10:42:20	temperature	24	24	null
12	04/05/2023 10:42:20	TimeInstant	2023-04-05T08:42:20.060Z	null	null
13	04/05/2023 10:41:45	humidity	14	null	14
14	04/05/2023 10:41:45	name	DHT11	null	null
15	04/05/2023 10:41:45	temperature	24	24	null

Figura 7.6 Base de datos formateada.

En primer lugar, se eliminan las columnas inservibles, como las vacías o las que contienen un mismo dato repetido. Se cambia el formato de la tabla de tiempos a formato fecha para poder sumar dos horas, necesario para adaptar la zona horaria del dato que se recibe. Se crean dos tablas para cada variable con el objetivo de poder mostrar la información correctamente.

Hasta ahora se ha comprobado que es posible la conexión e intercomunicación de todos los elementos del proyecto y se completa la estructura comunicativa de la plataforma. A partir de este punto, se va a tratar la información de forma que resulte útil para el usuario, por medio de visualización de datos y el sistema de alarmas.

7.4. Dashboard

Una vez conectada la base de datos se dispone de la información relevante del sensor. La actualización se realiza en este proyecto de forma manual, sin embargo, es posible actualizar la información de la base de datos en tiempo real por medio de una suscripción de pago que mejora las capacidades del software Power BI.

Al filtrar y mejorar la calidad de la información, se puede crear un panel de control que contenga información que pueda resultar relevante para el usuario. De esta forma, en el apartado *Panel* se procede a la creación de este, añadiendo distintos elementos como se muestra en la Figura 7.7.

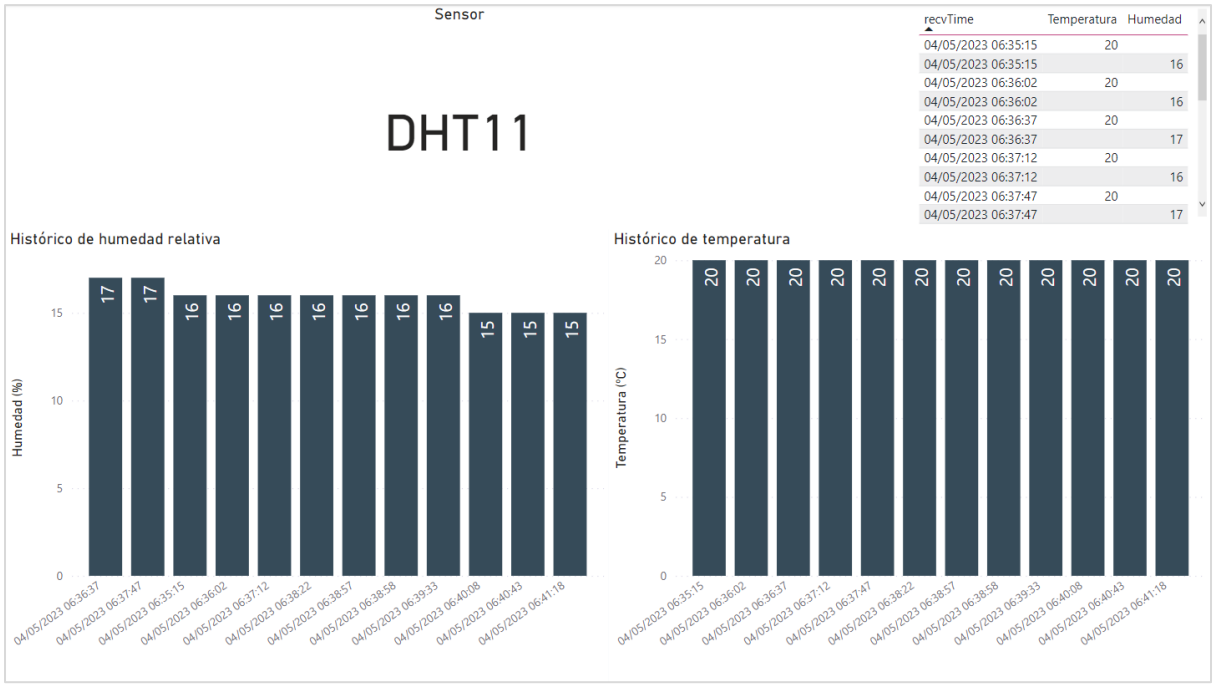


Figura 7.7 Panel para visualización de la información.

De esta forma, se puede ver de un vistazo el sensor cuya información se está mostrando y un histórico de los valores registrados. Existen multitud de opciones de configuración, se ha elegido esta por ser la más visual, conteniendo la información relevante de manera sencilla.

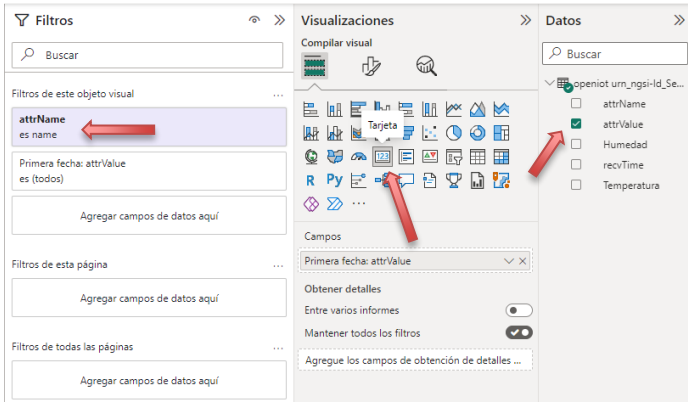


Figura 7.8 Creación de panel con filtros.

Se emplean diferentes filtros para seleccionar la información que se desea mostrar en cada caso, como se puede ver en la Figura 7.8, donde se selecciona un panel de tipo tarjeta, se introduce la columna *attrValue* y se filtra con la condición *attrName = name*

Donde:

- *attrValue* es el valor del atributo
- *attrName* es el nombre del atributo

Con esta configuración se va a crear un panel que muestra "DHT11" ya que es el valor del atributo cuando el nombre es *name.*

Para crear un gráfico que muestre el histórico de diferentes medidas en el tiempo se selecciona la opción de gráfico de barras y se asigna una columna a cada eje. Concretamente, la columna *Temperatura* al eje Y (con la opción de promedio ya que sólo hay un valor) y *recvTime* en el eje X.

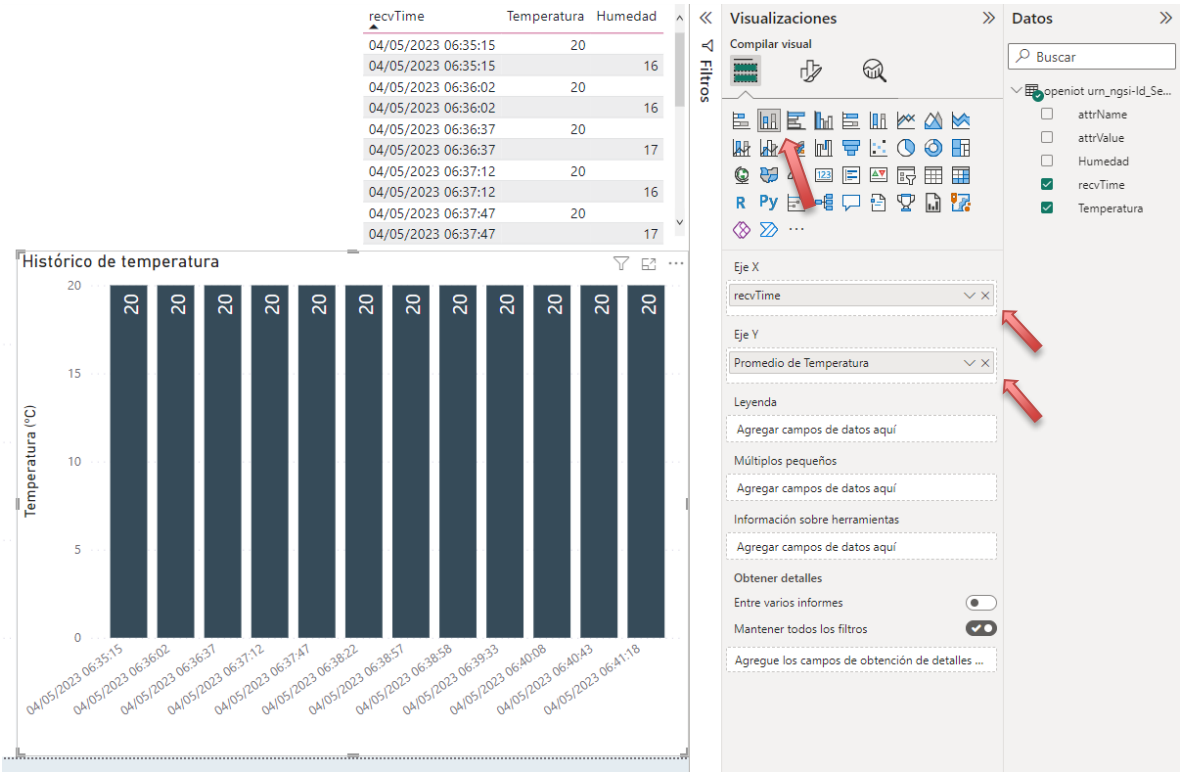


Figura 7.9 Creación de panel de histórico.

Una vez hecho esto, se realizan pequeños cambios para mostrar los números en el interior de la barra y mostrar correctamente la fecha en la parte inferior, son muy sencillos y no tienen mucha importancia.

En los datos mostrados no se puede apreciar ninguna variación de la temperatura, los datos son correctos y se corresponden con la realidad medida en ese momento, como se pude comprobar en la tabla que está justo encima y muestra los valores de temperatura y humedad en cada instante medido.

8. Sistema de alertas y notificaciones

Uno de los requisitos importantes fijados al principio del proyecto fue conseguir implementar un sistema funcional de alertas. La idea principal es mostrar mediante ejemplos que funcionan, la gran cantidad de posibilidades que existen. En este apartado, se va a describir el proceso seguido para llevar a cabo este desarrollo partiendo de una plataforma FIWARE con datos persistentes.

Existen varias formas de crear un sistema de avisos teniendo como base la arquitectura FIWARE. En un primer momento se intentó implementar el componente Perseo [31], un procesador de eventos complejos capaz de gestionar gran cantidad de información al mismo tiempo por un sistema de reglas basado en Esper. Sin embargo, se ha implementado Node-RED por ser una alternativa más visual y con más funcionalidades.

La instalación y configuración de Node-RED se detalla en el Anexo A.5. Una vez instalado y corriendo, se accede al panel de Node RED para comenzar con la creación del flujo.

8.1.1. Recepción de los datos en Node-RED

Para enviar la información desde Orion a Node-RED se emplea el método de suscripción que ha sido comentado en el Apartado 6.7.2. De esta forma, si el valor del parámetro humedad supera el valor 50, la suscripción se activa, y envía información al destino indicado, en este caso un bloque de recepción de peticiones HTTP POST en Node-RED.

Se muestra en la Figura 8.1 el flujo completo, para tener una visión general antes de la explicación por partes.

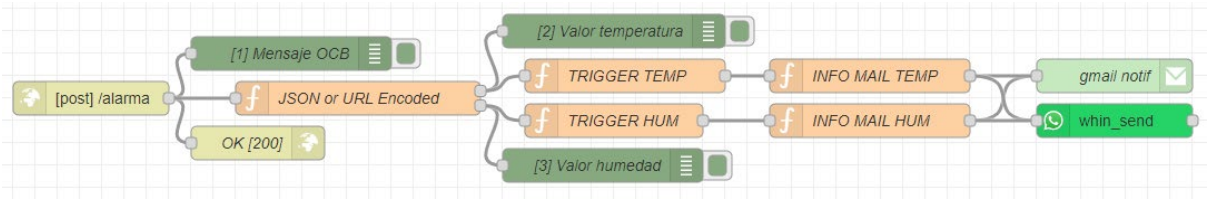


Figura 8.1 Flujo de Node-RED completo.

Ahora, se muestran en la Figura 8.2 los bloques que constituyen el inicio del flujo. El primero de todos ellos es un destino para peticiones POST en la dirección */alarma.* Con el bloque *[1] Mensaje OCB* se muestra en la ventana de *debug* el contenido (*payload)* del mensaje recibido. Se envía un código 200 de respuesta para indicar que el mensaje se ha recibido correctamente. El mensaje recibido es de tipo JSON, y se muestra en la Figura 8.3.

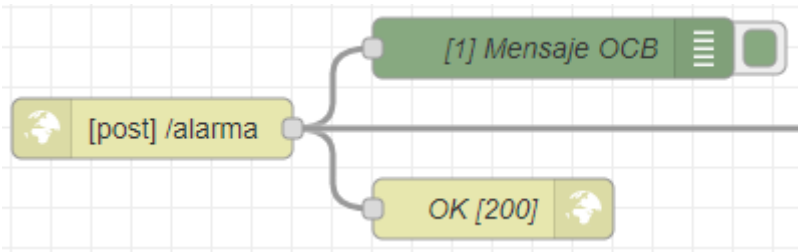


Figura 8.2 Recepción del mensaje HTTP de Orion.

Figura 8.3 Mensaje JSON recibido por Node-RED.

Como se puede ver, el mensaje contiene una clave *subscriptionId,* que es la clave que identifica a la suscripción que se ha ejecutado dentro de la plataforma FIWARE.

Haciendo desde Postman un GET a las suscripciones para mostrarlas todas se puede ver que la suscripción de Node-RED se identifica como:

```
"id": "643545b079581140dfb42981",
"description": "Notificar a Node-RED
"status": "active",
"subject": {
```

El mensaje contiene la id de la entidad que ha activado la suscripción, el tipo y el resto de los atributos, entre los que destacan el sello de tiempo, la humedad y temperatura, con sus respectivos tipos y valores.

Los bloques a continuación deben ser capaces de extraer la información relevante de este mensaje, es decir, el valor de la temperatura y de la humedad.

Así pues, los bloques que se muestran en la Figura 8.4 se colocan a continuación de los mostrados en la Figura 8.2 y su función es la siguiente. En primer lugar, el bloque *JSON or URL Encoded* sirve para extraer el valor de cada atributo seleccionado (se puede observar que tiene dos salidas, una para la temperatura y otra para la humedad).

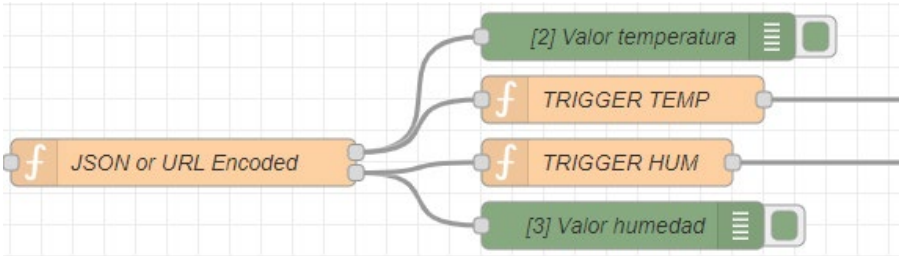


Figura 8.4 Bloques para la extracción de los valores.

La información que muestran en el *debug* los bloques [2] y [3] se muestra en la Figura 8.5.

Figura 8.5 Valores de temperatura y humedad extraídos del mensaje JSON y mostrados en el debug.

Los otros dos bloques que se muestran en la Figura 8.6, *TRIGGER TEMP* y *TRIGGER HUM* contienen el código necesario para hacer saltar las alarmas.

Figura 8.6 Contenido de ambos bloques.

El funcionamiento de ambos códigos es similar. Se define una variable de tipo *boolean* con el nombre *alarm_flag*, la cual indica con su estado si la alarma se debe activar o no. En un primer momento se le asigna un valor falso, y en caso de que se reciba un dato que supere el límite máximo tanto de humedad como de temperatura, se activa el *flag* y se da la orden para proceder al envío de la alerta, continuando con el flujo hacia los bloques mostrados en la Figura 8.7.

El segundo "if" sirve para desactivar la alarma, la condición que plantea es: si *alarm_flag* está activo y el dato recibido (proveniente del sensor) indica un valor normal (en caso de la temperatura menor de 23ºC y para la humedad menor que 30), el *flag* de la alarma se desactiva.

Por último, la parte final del flujo se encarga de introducir el contenido del mensaje en los bloques de envío de correo electrónico y mensaje de Whatsapp. En la Figura 8.7 se puede ver la estructura que permite esta funcionalidad.

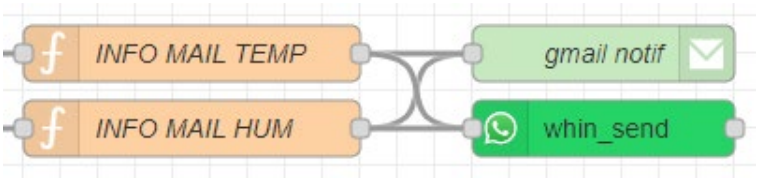


Figura 8.7 Últimos bloques del flujo.

El contenido de los bloques INFO MAIL TEMP e INFO MAIL HUM se muestran en la Figura 8.8.

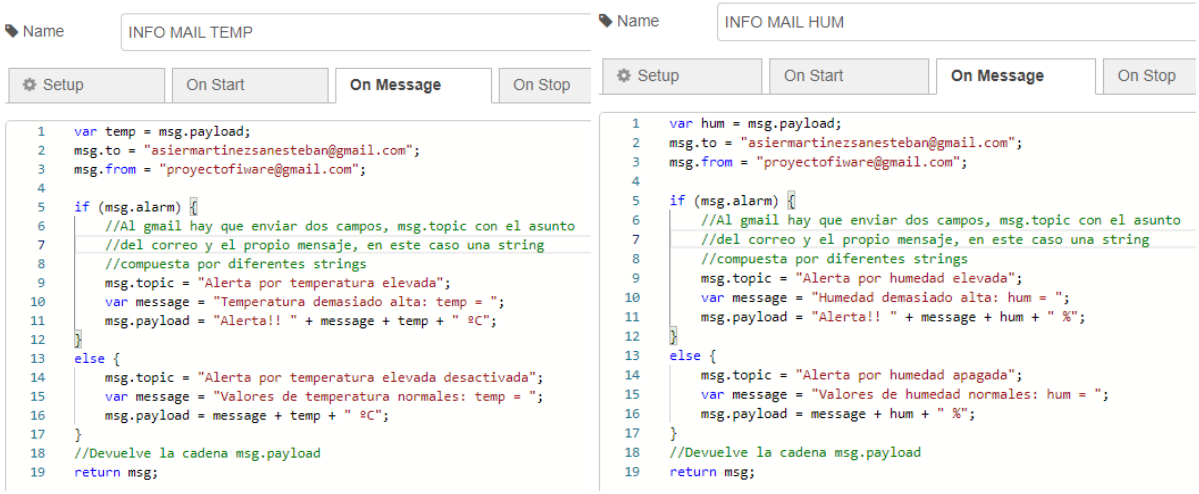


Figura 8.8 Contenido de ambos bloques.

En estos bloques se define el contenido que se va a insertar en los bloques finales *gmail notif y whin_send.* Al conectar la salida de estos dos bloques a un bloque *debug* se muestra por pantalla el contenido, que es el mismo contenido que se escribe tanto en el mensaje de Whatsapp como en el correo electrónico.

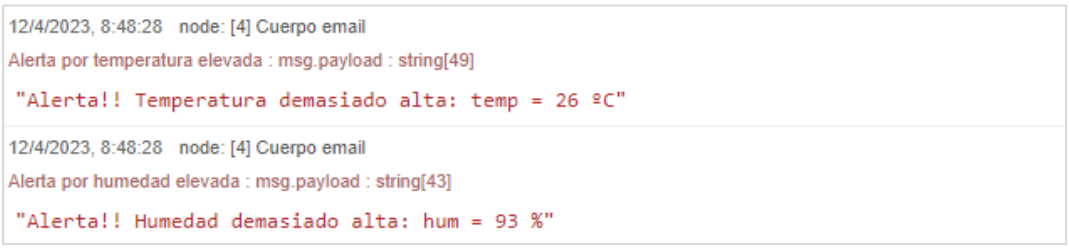
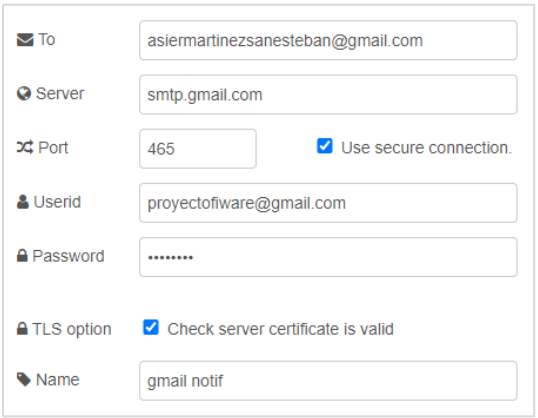


Figura 8.9 Mensaje obtenido en el debug.



El interior del bloque *gmail notif* contiene las credenciales de la cuenta que envía el correo, el puerto y el servidor, así como la dirección que recibirá el correo. Es importante activar la opción TLS y de conexión segura.

Figura 8.10 Contenido del bloque gmail notif.

Para poder enviar mensajes vía Whatsapp se emplea una librería de Node-RED llamada Whin [32], que es una API para el envío de mensajes Whatsapp. Se requiere de una cuenta premium ya que la básica incluye solamente unos pocos mensajes, que fueron gastados haciendo pruebas. Una de las pruebas realizadas para comprobar el envío de mensajes WhatsApp a través de Node-Red, como se puede ver en la Figura 8.11.

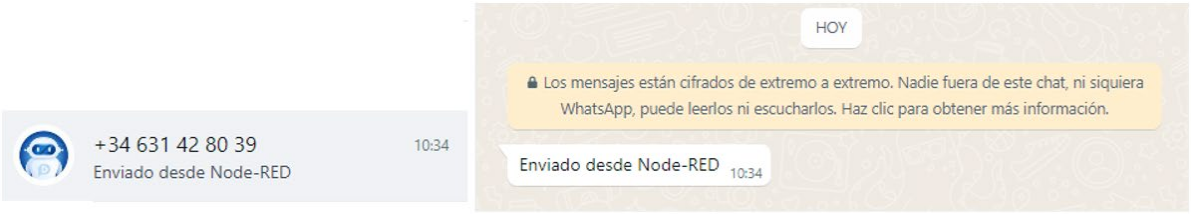


Figura 8.11 Prueba de envío de mensajes Whatsapp a través de Node-RED.

Finalmente, el resultado obtenido es una alerta enviada al correo electrónico personal, desde la cuenta creada para el proyecto y con el asunto y mensaje establecidos en los bloques finales de la Figura 8.7.

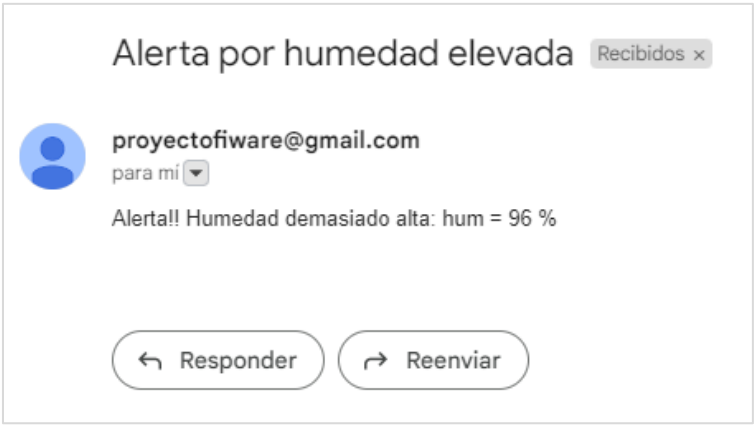


Figura 8.12 Alerta recibida al correo electrónico.

Es importante destacar que para emplear el SMTP de Gmail es necesaria una clave específica de aplicación. Para ello se debe verificar la cuenta y solicitar la clave, que se asigna de manera inmediata.

9. Resultados y discusión

En este apartado se pretende mostrar al completo el correcto funcionamiento de la plataforma. Se han realizado diferentes pruebas para verificar la plataforma en su conjunto, las cuales se documentan a continuación junto con los resultados obtenidos.

9.1. Pruebas realizadas

A lo largo del documento se han recopilado las diferentes pruebas que verifican el funcionamiento de las diferentes partes y justifican el avance a fases posteriores del proyecto. Esto se puede ver en el apartado *5.4 Prueba de comunicación* en el que se explica en detalle el proceso llevado a cabo para validar el script de Arduino que posibilita el envío de información con el formato correcto y protocolo HTTP.

En la parte final del apartado *6.5 Envío de datos a la plataforma FIWARE* se realiza una pequeña prueba para constatar el correcto envío y recepción de los datos desde el dispositivo IoT a la plataforma FIWARE, que consiste en realizar dos peticiones tipo GET para mostrar el estado de la entidad mientras el sensor está tomando datos, para ver cómo los valores de la entidad cambian para corresponderse con los datos obtenidos más recientes.

Para el desarrollo del sistema de alertas se han realizado múltiples pruebas, en un primer momento para probar el correcto funcionamiento de los bloques de envío de mensajes y más tarde para corroborar que la información actualizada del contexto llega a Node-RED y se disparan las alertas, todas ellas con un resultado positivo en poco tiempo.

Durante el desarrollo del proyecto se han realizado múltiples pruebas de muchas formas, las más significativas para reflejar el avance del proyecto ya se han explicado en profundidad

Con el objetivo de realizar una verificación final paso por paso del funcionamiento de la plataforma, se realizan diversas pruebas en las diferentes partes de la comunicación.

1. **Envío y recepción de los datos provenientes del sensor.** Con esta prueba se verifica que el script de Arduino que corre en la placa ESP32 es el adecuado, además de comprobar la correcta recepción de los datos en la plataforma FIWARE, lo cual garantiza el correcto despliegue de los componentes implicados (Orion, Mongo y el IoT Agent).

2. **Correcto almacenamiento de la información.** Se verifica la correcta integración de Draco, el estado de la suscripción y el funcionamiento de la base de datos.

3. **Visualización de datos actualizados.** Se comprueba la conexión de Power BI con MySQL y un primer análisis de los datos recibidos.

4. **Avisos cuando ocurre un evento.** Se comprueba el funcionamiento de Node-RED mediante la suscripción, la configuración del flujo y el funcionamiento del aviso por correo.

9.2. Resultados obtenidos

En este apartado se comenta el desarrollo de las diferentes pruebas y los resultados obtenidos.

9.2.1. Envío y recepción de los datos provenientes del sensor.

En primer lugar, se debe desplegar la plataforma FIWARE, iniciando los diferentes contenedores simultáneamente mediante el comando *sudo docker-compose -p fiware up -d* ejecutado en la consola de la máquina Linux (será necesario activar el punto de acceso WiFi del equipo anfitrión). Tras esperar unos minutos, se verifica que los contenedores se han levantado correctamente con el comando *sudo docker ps.* Si algún contenedor no está corriendo, se debe lanzar el primer comando de nuevo.

Una vez desplegada la plataforma, se procede al volcado del código en la placa ESP32 por medio de Arduino IDE. Este proceso tarda unos segundos. A continuación, se abre el monitor serie para ver cómo se conecta la placa ESP32 a la WiFi y con qué IP, y acto seguido comenzará el envío de datos. Se muestra por el monitor serie el dato que se envía (formato JSON) así como el código de respuesta obtenido del receptor, en este caso, HTTP 200 [4] indicando que la comunicación es correcta.

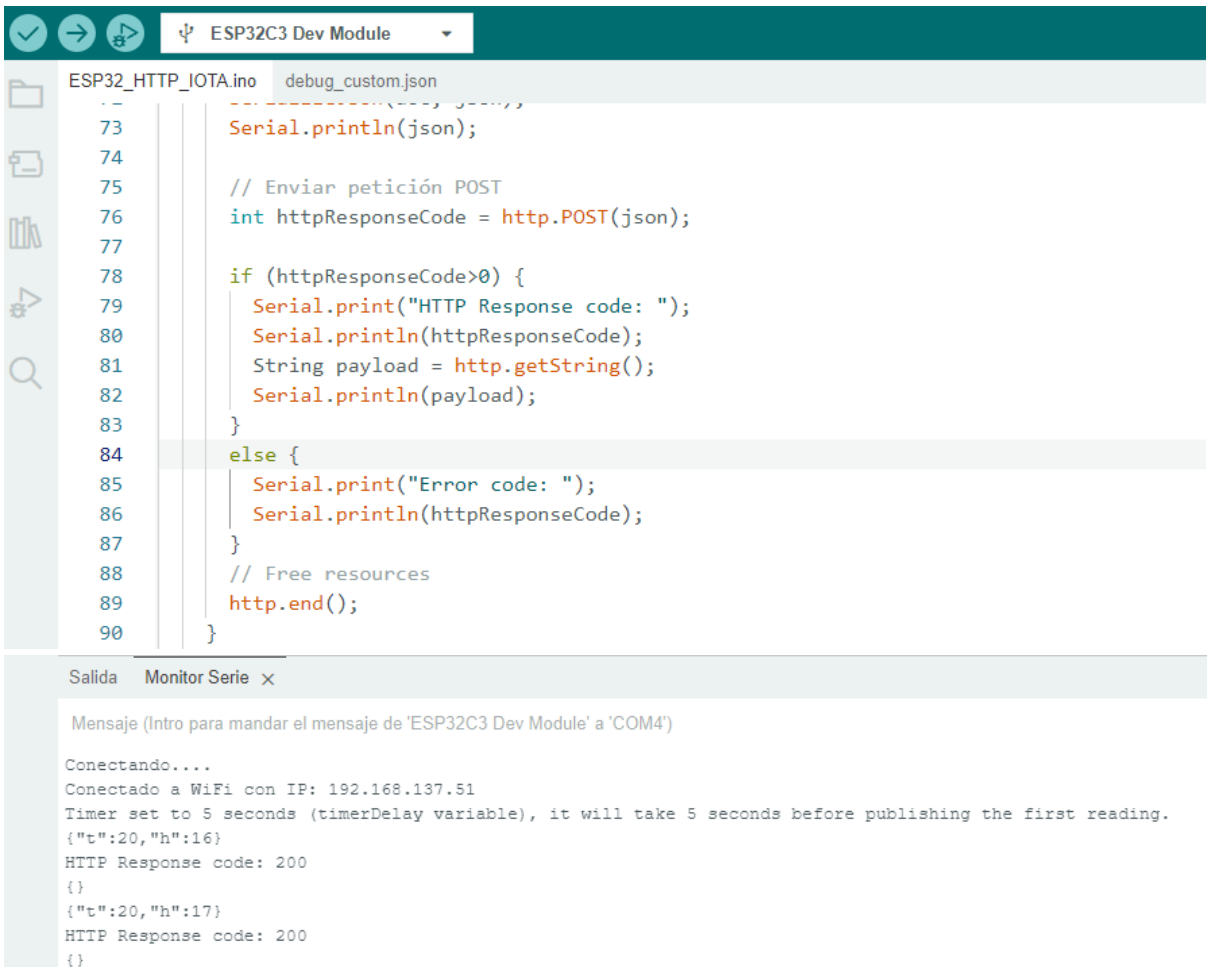


Figura 9.1 Información en el monitor serie confirma que el envío y la recepción son correctos.

[4] HTTP 200 es un código de estado de respuesta que indica que una solicitud HTTP ha sido procesada correctamente por el servidor y que se ha devuelto una respuesta exitosa al cliente.

Para completar esta prueba, se verifica la actualización de la entidad con los nuevos valores de temperatura y humedad provistos por el sensor. Haciendo una petición GET para mostrar la entidad se obtiene la información mostrada en la Figura 9.2.

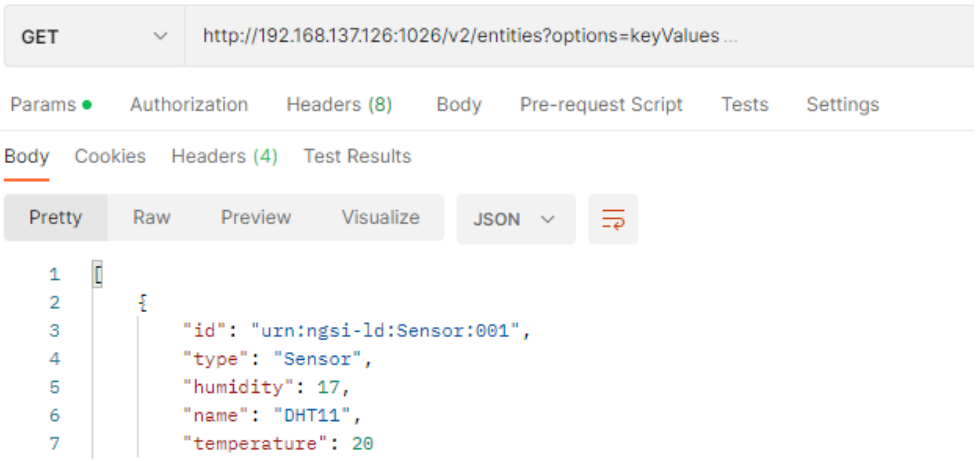


Figura 9.2 Entidad con valores actualizados.

9.2.2. Correcto almacenamiento de la información

Para esta prueba, con la plataforma desplegada y verificada la correcta recepción de los datos, se abre MySQL Workbench y se siguen los pasos recogidos en el Anexo A.7 para crear una nueva conexión. Se verifica que la base de datos existe al encontrarla en el apartado *schemas* de este programa.

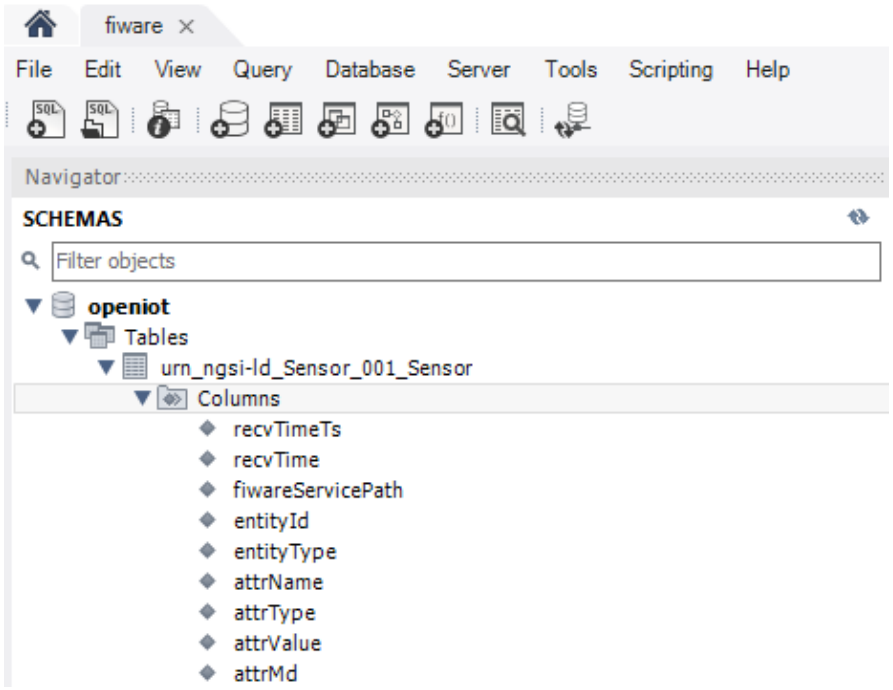


Figura 9.3 MySQL Workbench para verificar creación y existencia de la base de datos.

9.2.3. Visualización de datos actualizados

Una vez realizada la conexión de Power BI con la base de datos MySQL, los datos se acondicionan por primera vez y quedan disponibles para su uso en el panel. Sin embargo, al desactivar la plataforma y levantarla más tarde, o en otro momento; es preciso conocer si volviendo a actualizar los datos, se guardan todos ellos.

recvTime	attrName	attrValue	Temperatura	Humedad
04/11/2023 11:35:36	humidity	33		*33*
04/11/2023 11:35:36	TimeInstant	2023-04-11T11:35:36.577Z		
04/11/2023 11:35:36	temperature	23	*23*	
04/11/2023 11:35:36	name	DHT11		
04/05/2023 10:39:24	temperature	22	*22*	
04/05/2023 10:39:24	humidity	22		*22*
04/05/2023 10:39:24	TimeInstant	2023-04-05T10:39:24.657Z		
04/05/2023 10:39:24	name	DHT11		

Figura 9.4 Datos actualizados.

Como se puede ver en la Figura 9.4, los primeros datos son del día 5 de abril y los más recientes son del 11 de abril. Entre ambos periodos la plataforma no ha estado siempre activa. Esto quiere decir que al actualizar los datos desde Power BI, se conserva el histórico de días anteriores y se incluyen los nuevos datos generados.

9.2.4. Avisos cuando ocurre un evento

Para comprobar el funcionamiento del sistema de alertas, la prueba más sencilla consiste en elevar la humedad cerca del sensor y esperar la recepción del correo. Para ello, basta con soplar unos segundos en la dirección del sensor.

Cuando la medida de humedad es superior al 50%, este dato se recibe en la plataforma FIWARE y se activa la suscripción de Node-RED por cumplirse la condición establecida. De esta forma, se recibe una petición HTTP POST en el flujo de Node-RED, que finaliza con el envío de un correo electrónico desde una cuenta dada, a una dirección establecida.

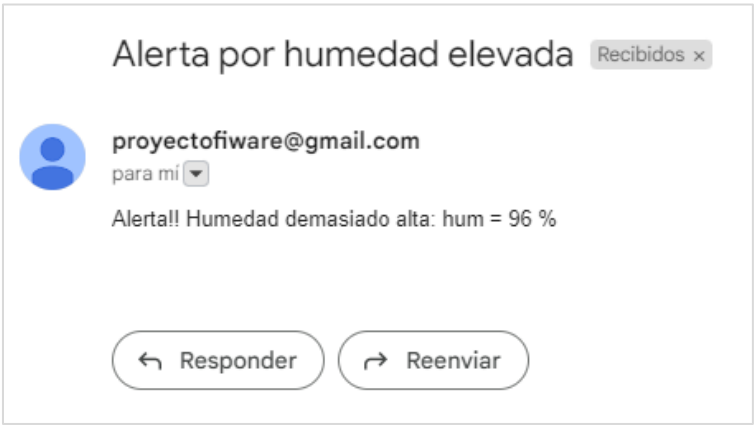


Figura 9.5 Recepción de correo electrónico.

De esta forma, se verifica la capacidad de la plataforma de producir las alertas que se requieran, siendo este un mero ejemplo para demostrar esta funcionalidad.

9.3. Tiempos

Se considera necesario comentar el tiempo que transcurre desde que se genera el dato hasta que se recibe la alerta que dispara dicho dato.

En el tiempo que tarda el usuario en recibir un correo electrónico por haberse disparado una alerta es la suma de varios tiempos, como se muestra en la Figura 9.6.

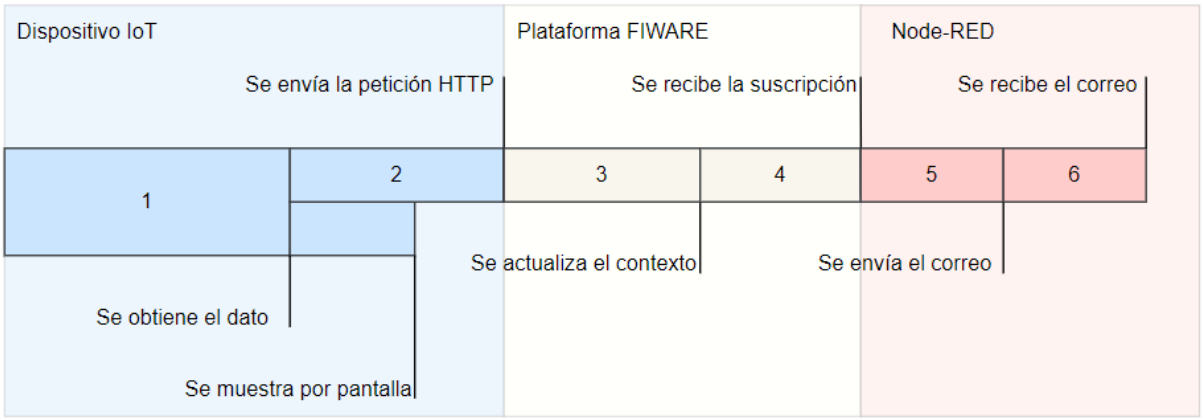


Figura 9.6 Diagrama de tiempos.

1. Tiempo de adquisición del dato: el tiempo de respuesta de un sensor DHT11 oscila entre 6 y 30 segundos.
2. Tiempo que tarda en realizar las tareas el script de Arduino
3. Tiempo de procesado en plataforma FIWARE: es el tiempo que transcurre desde que el sensor envía el dato (vía WiFi) hasta que la entidad se actualiza.
4. Tiempo de llegada a Node-RED: Desde que la entidad toma un valor hasta que Node-RED recibe ese dato si el valor cumple las condiciones.
5. Tiempo de procesado de Node-RED
6. Tiempo de llegada a la bandeja de entrada: comprende el procesamiento del dato y la alerta, y el envío del mensaje.

Como los tiempos intermedios son difíciles de medir, se va a hacer un análisis más general, desde que el dato se muestra en el puerto serie hasta que llega el correo electrónico. Haciendo esta prueba, se puede observar que el tiempo total transcurrido entre estos dos eventos oscila entre 1 y 2 segundos.

Con esta observación, teniendo en cuenta el tiempo de adquisición más desfavorable (30 s) y el tiempo de recepción del correo desde que se muestra el dato (2 s), el tiempo que tarda el usuario en enterarse de la alerta es, teóricamente, como máximo 32 segundos. Tiempo suficiente si hablamos de controles de temperatura ambiente. Para mediciones de temperaturas en procesos específicos se debería cambiar el sensor a otro con un tiempo de adquisición más bajo, puesto que el principal causante del retraso (94%) es este parámetro.

9.4. Revisión de seguridad

La ciberseguridad es un asunto capital a la hora de ofrecer un servicio web estable y de calidad. Existen formas diferentes de hacer una plataforma más segura frente a injerencias de terceros, tráfico no deseado y en definitiva ataques que atenten contra la integridad de esta.

Al tratarse de una plataforma creada en una red privada, en un mismo equipo y sin conexión con el exterior, por muy vulnerable que ésta sea no existe ningún riesgo. Como es una plataforma de pruebas, para tener una mayor agilidad y facilitar el desarrollo se han obviado numerosas contraseñas y verificaciones de seguridad las cuales van a ser revisadas en este apartado para que puedan ser tenidas en cuenta en un futuro.

- Contraseña WiFi en el propio script: No se debe incluir las credenciales de acceso a la red WiFi en el propio programa. Se debe crear una variable que haga referencia a otro archivo sobre el que se tenga más control.
- Apikey de dispositivo IoT: Para asegurar que sólo los dispositivos IoT autorizados pueden enviar peticiones HTTP a la plataforma, se debe establecer una apikey robusta. Al tratarse de un entorno de pruebas, la apikey empleada era “apikey”, para facilitar las cosas, sin embargo, se debe cuidar este aspecto.
- Acceso a Node-RED: configurar credenciales para acceder al portal de Node-RED, para evitar que cualquiera pueda realizar cambios.
- Acceso a Mongo DB: configurar credenciales de acceso para mongoDB
- Acceso a MySQL a través de NiFi: configurar credenciales de acceso seguras para el acceso a MySQL.

La seguridad y autenticación es de gran importancia al desarrollar cualquier tipo de servicio en línea enfocado al público. Es uno de los requisitos principales de una supuesta fase de producción de esta plataforma. Será necesario aprovechar todas las medidas de seguridad mencionadas así como llevar a cabo una buena gestión de los puertos para garantizar la ciberseguridad. Es conveniente realizar auditorías de seguridad regularmente y adoptar buenas prácticas en el desarrollo de software.

9.5. Requisitos para fase de producción

Para escalar la plataforma a una supuesta fase de producción de manera que los datos puedan ser mostrados y estar en funcionamiento continuo para un cliente, sería necesario implementar una serie de medidas. Se entiende que para escalar una plataforma de este estilo es necesario un mayor volumen de datos, por ejemplo, de un barrio o una ciudad; es decir, integrar una gran cantidad de sensores que permitan controlar diferentes parámetros de interés como el tráfico, la contaminación del aire, la temperatura etc.

Será necesario contar con dos plataformas, una de producción más accesible y visible, y otra de pruebas, en la que se irá desarrollando la propia plataforma, arreglando fallos e implementando nuevas funcionalidades en un entorno seguro, cerrado al público y de uso privado. Una vez que se depuren las soluciones, se implementarán en la plataforma de desarrollo.

Monitoreo y escalado automático: se recomienda implementar herramientas de monitoreo para controlar el rendimiento de las plataformas en tiempo real. Existe la opción de configurar alertas para detectar posibles problemas y establecer mecanismos de escalado automático que puedan aumentar o disminuir la capacidad de la infraestructura según la demanda instantánea.

Pruebas y optimización: Realiza pruebas exhaustivas de rendimiento y carga para identificar cuellos de botella y áreas de mejora en tu plataforma Fiware. Optimiza el código, la configuración y los recursos según los resultados de las pruebas para asegurar un rendimiento óptimo y una experiencia de usuario fluida.

10. Conclusiones y líneas futuras

En este apartado se realiza una revisión general del trabajo realizado a lo largo del desarrollo del proyecto, aciertos, fallos y mejoras que quedan pendientes para una segunda fase.

10.1. Conclusiones

A modo de conclusión, los objetivos expuestos al comienzo del proyecto se han cumplido, incluso superando las expectativas iniciales en cuanto a las funcionalidades logradas. Más concretamente, la recepción de mensajes vía Whatsapp y correos electrónicos es a simple vista algo trivial, pero son la muestra de que toda la plataforma funciona como debe y realmente cumple su función.

Durante el desarrollo del trabajo han surgido múltiples problemas que ralentizaban el avance del proyecto, siendo el principal la configuración de la red de comunicación entre el equipo host, la máquina virtual y la placa, que se consiguió solucionar con el responsable de equipos informáticos de la empresa, que fue clave para comprender el funcionamiento de las redes y entender de forma práctica cómo configurar las comunicaciones.

El principal reto no cumplido es la integración del componente FIWARE Perseo [31] para gestionar las alertas. Se ha usado Node-RED por ser más visual, práctico y sencillo de manejar no sin antes dedicar una gran cantidad de horas a intentar poner en marcha Perseo. La mentoría con el personal del Fiware Space de la Diputación de Badajoz [33] fue de gran ayuda, primero intentando hacer funcionar Perseo[5] y después recomendando la alternativa de Node-RED.

Como se ha ido revisando a lo largo de todo el documento, la intención con este trabajo es mostrar la simplicidad y efectividad de este tipo de plataformas, los problemas que resuelve su implementación y el paso a paso de esta. Las funcionalidades logradas van a ser implementadas de forma similar en la plataforma de la empresa, concretamente el sistema de alertas por correo y la implementación de Power BI, con lo que el trabajo de investigación y desarrollo tiene una aplicación real.

[5] La Diputación de Badajoz cuenta con un amplio equipo de informáticos que han conseguido implementar Perseo, sin embargo, el tutor comenta que no ha visto a nadie más hacerlo funcionar.

10.2. Líneas futuras

Este proyecto es un pequeño ejemplo que permite mostrar la escalabilidad y el potencial de este tipo de arquitecturas, donde la integración y gestión de una gran cantidad de dispositivos IoT de diferentes fabricantes y tipos puede resultar algo más sencilla.

La **primera mejora** a futuro sería implementar el **protocolo MQTT** [34] para el envío de los datos desde la placa ESP32, sustituyendo al actual protocolo empleado, HTTP. El protocolo MQTT (Message Queuing Telemetry Transport) presenta varias ventajas:

- Uso del ancho de banda eficiente. MQTT es un protocolo liviano, cualidad especialmente importante en aplicaciones IoT, donde los dispositivos pueden estar conectados a través de redes inalámbricas de baja velocidad.
- Tolerancia a fallos: MQTT tiene garantía de entrega de mensajes. Los mensajes no se pierden en caso de fallas en la red o en los dispositivos.
- Escalabilidad: MQTT es altamente escalable, puede manejar un gran número de dispositivos y clientes. Los clientes MQTT pueden suscribirse a varios temas y recibir sólo algunos mensajes.
- Seguridad: MQTT admite autenticación y encriptación, lo que significa que los datos están protegidos de accesos no autorizados.

MQTT es un protocolo eficiente, seguro y escalable, lo que lo hace una opción popular para la implementación de soluciones de IoT.

La **segunda mejora** es escalar la plataforma añadiendo más sensores de diferentes tipos para aumentar el volumen de datos y la riqueza de la información mostrada, sin embargo, esto es algo que requiere un motivo y un proyecto más grande de sensorización, como puede ser el control de la temperatura, humedad, presión y calidad del aire en diferentes puntos de una ciudad.

- La parte compleja de este crecimiento es la instalación y el despliegue técnico necesario, así como una buena planificación de los recursos y estandarización.
- La parte sencilla es crear las diferentes entidades e incluirlas en la plataforma, así como gestionar las suscripciones e información de cada uno de estos porque sería replicar el proceso realizado con un sensor, para varios sensores/actuadores.

Otras mejoras a futuro que son interesantes para mejorar la plataforma son:

- **Integración de Perseo**: acceder y modificar el código fuente del componente para poder trabajar con él.
- **Actualización de los datos automática en Power BI** mediante una cuenta premium. No depender de una actualización manual de los datos.
- **Sistema de toma de decisiones** mediante Whatsapp, mediante una suscripción de pago en el servicio Whin. Establecer una comunicación con un usuario responsable que permita tomar decisiones mediante Whatsapp ante alertas producidas por el sistema, por ejemplo, frente a un aumento de temperatura que pueda ser peligroso, tomar la decisión de activar la ventilación.

11. Referencias

[1] S. Sarkar y S. Sarkar, *Internet of Things*, vol. 9, n.º 4. Elsevier, 2016. Accedido: 10 de mayo de 2023. [En línea]. Disponible en: http://dx.doi.org/10.1016/b978-0-12-805395-9.00011-3

[2] D. Kyriazis, T. Varvarigou, D. White, A. Rossi, y J. Cooper, «Sustainable smart city IoT applications: Heat and electricity management & Eco-conscious cruise control for public transportation», *IEEE International Symposium and Workshops on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, p. 1, 2013, doi: 10.1109/WoWMoM.2013.6583500.

[3] «Developers Catalogue – FIWARE». https://www.fiware.org/catalogue/ (accedido 10 de mayo de 2023).

[4] «NGSI Context Management», Accedido: 31 de marzo de 2023. [En línea]. Disponible en: http://www.openmobilealliance.org/UseAgreement.html.

[5] Read The Docs, «NGSI-LD How to - Fiware-DataModels». https://fiware-datamodels.readthedocs.io/en/stable/ngsi-ld_howto/index.html (accedido 31 de marzo de 2023).

[6] Diputación de Badajoz, «Curso Medio Fiware».

[7] «JSON». https://www.json.org/json-es.html (accedido 3 de mayo de 2023).

[8] «¿Qué Es MongoDB? | MongoDB». https://www.mongodb.com/es/what-is-mongodb (accedido 10 de mayo de 2023).

[9] «IoT Agent (Ultralight) ». https://fiware-tutorials.readthedocs.io/en/stable/iot-agent/ (accedido 10 de mayo de 2023).

[10] Read The Docs, «Fiware Draco». https://fiware-draco.readthedocs.io/en/latest/ (accedido 31 de marzo de 2023).

[11] «Apache NiFi Documentation». https://nifi.apache.org/docs.html (accedido 31 de marzo de 2023).

[12] Read The Docs, «Fiware-cygnus». https://fiware-cygnus.readthedocs.io/en/latest/ (accedido 31 de marzo de 2023).

[13] «Apache Flume Documentation». https://flume.apache.org/ (accedido 31 de marzo de 2023).

[14] «Contenedores, imágenes y registros de Docker | Microsoft Learn». https://learn.microsoft.com/es-es/dotnet/architecture/microservices/container-docker-introduction/docker-containers-images-registries (accedido 3 de mayo de 2023).

[15] «Postman API Platform | Sign Up for Free». https://www.postman.com/ (accedido 10 de mayo de 2023).

[16] «Qué es REST». https://desarrolloweb.com/articulos/que-es-rest-caracteristicas-sistemas.html (accedido 10 de mayo de 2023).

[17] «Node-RED - Wikipedia, la enciclopedia libre». https://es.wikipedia.org/wiki/Node-RED (accedido 3 de mayo de 2023).

[18] «Software de código abierto - Wikipedia, la enciclopedia libre». https://es.wikipedia.org/wiki/Software_de_c%C3%B3digo_abierto (accedido 3 de mayo de 2023).

[19] Joyent y Joyent, «Node.js», *Github*, Accedido: 9 de abril de 2023. [En línea]. Disponible en: https://github.com/joyent/node/tags?after=v0.0.4

[20] «Temperatura y humedad DHT11 | Arduino». http://ceca.uaeh.edu.mx/informatica/oas_final/OA4/temperatura_y_humedad_dht11.html (accedido 31 de marzo de 2023).
[21] «ESP32 - Wikipedia». https://es.wikipedia.org/wiki/ESP32 (accedido 31 de marzo de 2023).
[22] «Espressif – Monolitic». https://www.monolitic.com/espressif/ (accedido 31 de marzo de 2023).
[23] «ESP32-C3-DeKiiM-1».
[24] Luis Llamas, «Ficheros Json con Arduino Json». https://www.luisllamas.es/arduino-json/ (accedido 3 de abril de 2023).
[25] «Modo: Red NAT | Tutorial de VirtualBox». https://www.fpgenred.es/VirtualBox/modo_red_nat.html (accedido 31 de marzo de 2023).
[26] Random Nerd Tutorials, «ESP32 HTTP GET and POST with Arduino IDE ». https://randomnerdtutorials.com/esp32-http-get-post-arduino/ (accedido 3 de abril de 2023).
[27] «The Official YAML Web Site». https://yaml.org/ (accedido 3 de abril de 2023).
[28] «Compose file versions and upgrading». https://docs.docker.com/compose/compose-file/compose-versioning/ (accedido 3 de abril de 2023).
[29] «Volumes». https://docs.docker.com/storage/volumes/ (accedido 3 de abril de 2023).
[30] «MySQL Download Connector». https://dev.mysql.com/downloads/connector/net/ (accedido 4 de abril de 2023).
[31] «Perseo Context-aware Complex Event Processing». https://fiware-perseo-fe.readthedocs.io/en/latest/ (accedido 10 de mayo de 2023).
[32] «whin-whatsapp - Node-RED». https://flows.nodered.org/node/@inutil-labs/node-red-whin-whatsapp (accedido 11 de mayo de 2023).
[33] «Fiware Space · Badajoz Es Más · Diputación de Badajoz». https://fiware.space/ (accedido 10 de mayo de 2023).
[34] «MQTT - The Standard for IoT Messaging». https://mqtt.org/ (accedido 10 de mayo de 2023).
[35] «Downloads – Oracle VM VirtualBox». https://www.virtualbox.org/wiki/Downloads (accedido 3 de abril de 2023).
[36] «Get Ubuntu | Download | Ubuntu». https://ubuntu.com/download#download (accedido 3 de abril de 2023).
[37] «Descargas Visual C++ Redistributable ». https://learn.microsoft.com/es-es/cpp/windows/latest-supported-vc-redist?view=msvc-170 (accedido 3 de abril de 2023).
[38] «Software | Arduino». https://www.arduino.cc/en/software (accedido 4 de abril de 2023).
[39] «HTTPClient · GitHub». https://github.com/espressif/arduino-esp32/tree/master/libraries/HTTPClient/src (accedido 4 de abril de 2023).
[40] «Running Node-RED locally : Node-RED». https://nodered.org/docs/getting-started/local (accedido 4 de abril de 2023).
[41] «Download Postman | Get Started for Free». https://www.postman.com/downloads/ (accedido 10 de mayo de 2023).
[42] «Download MySQL Workbench». https://dev.mysql.com/downloads/workbench/ (accedido 4 de abril de 2023).
[43] «Download Microsoft® Power BI ». https://www.microsoft.com/es-ES/download/details.aspx?id=58158 (accedido 4 de abril de 2023).

Anexo A. Instalación y despliegue

En este primer anexo se va a detallar el proceso de instalación de cada software requerido. Quedan excluidos los componentes fiware ya que su instalación, como se ha comentado, se lleva a cabo de manera sencilla mediante Docker y la configuración del archivo *docker-compose.yml*

A.1 Instalación de una máquina virtual

Para comenzar, se va a instalar una máquina virtual Linux en el PC Host Windows.

Se requiere:

- Instalación programa VirtualBox o similar
- Descargar imágenes de Ubuntu y Windows
- Instalación de Visual C++
- Configuración de red

1. El primer paso es la instalación de VirtualBox o una aplicación similar que permita generar máquinas virtuales en el PC. En este caso se opta por VirtualBox por ser de las más conocidas y haber trabajado con ella antes. Accediendo a la página oficial [35], descargar la primera opción.

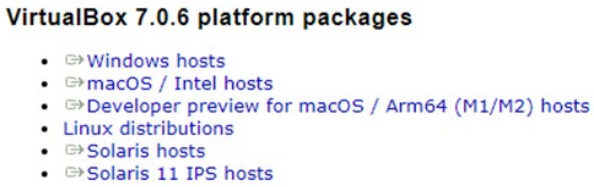


Figura A. 1. Descargar VirtualBox.

2. Una vez hecho esto, es necesario descargar la imagen del SO que se quiera emplear, en este caso se trata del archivo *Ubuntu 22.04.1-desktop-amd64.iso* (versión más reciente). Para ello se accede a la página oficial de Ubuntu [36], donde aparece el siguiente menú desplegable.

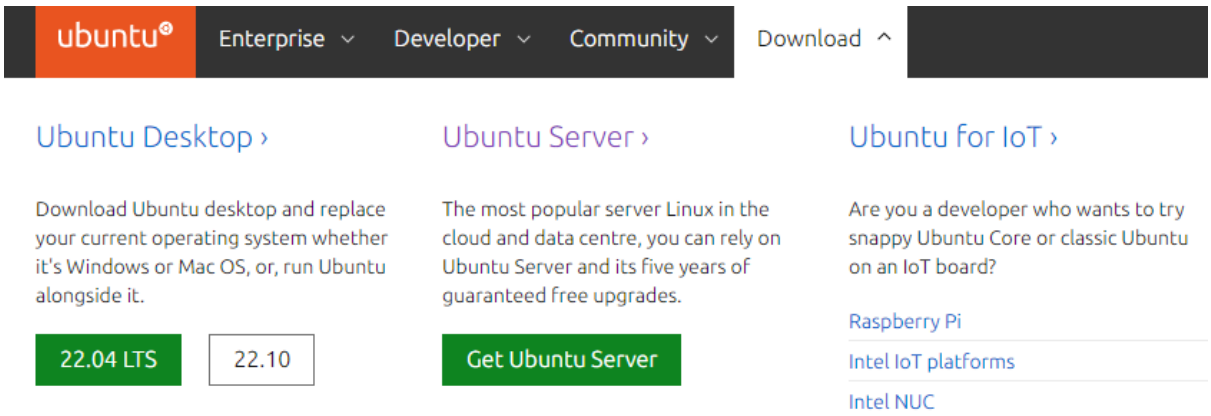


Figura A. 2. Descargar imagen de Ubuntu.

Como se puede ver, aparecen dos opciones de descarga, *Ubuntu Desktop* y *Ubuntu Server*. La versión *Desktop* presenta un escritorio, resulta más familiar para un usuario recién llegado de Windows

y sin demasiados conocimientos en informática. La versión *Server* es únicamente una consola de comandos Linux, lo que dificulta de primeras el manejo de la máquina para el usuario promedio.

Navegar por los archivos en formato consola puede resultar en ocasiones complicado, por lo que se opta por la versión Desktop, que además cuenta con navegador y otras aplicaciones que facilitan el manejo al usuario.

3. El siguiente paso consiste en la instalación de *Visual C++*, concretamente la primera opción disponible en la página oficial de descargas [37].

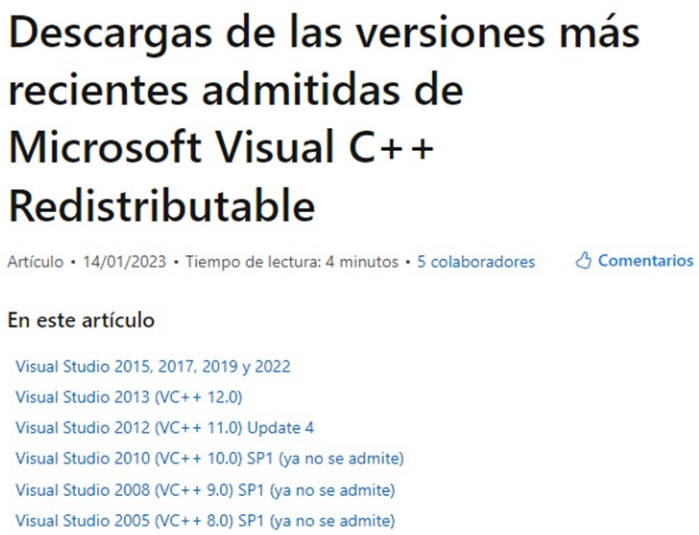


Figura A. 3. Instalación Visual C++ 2022.

Una vez completadas todas las instalaciones, para crear una máquina virtual se debe abrir VirtualBox y seleccionar la opción *Máquina > Nueva…* para acceder al siguiente menú.

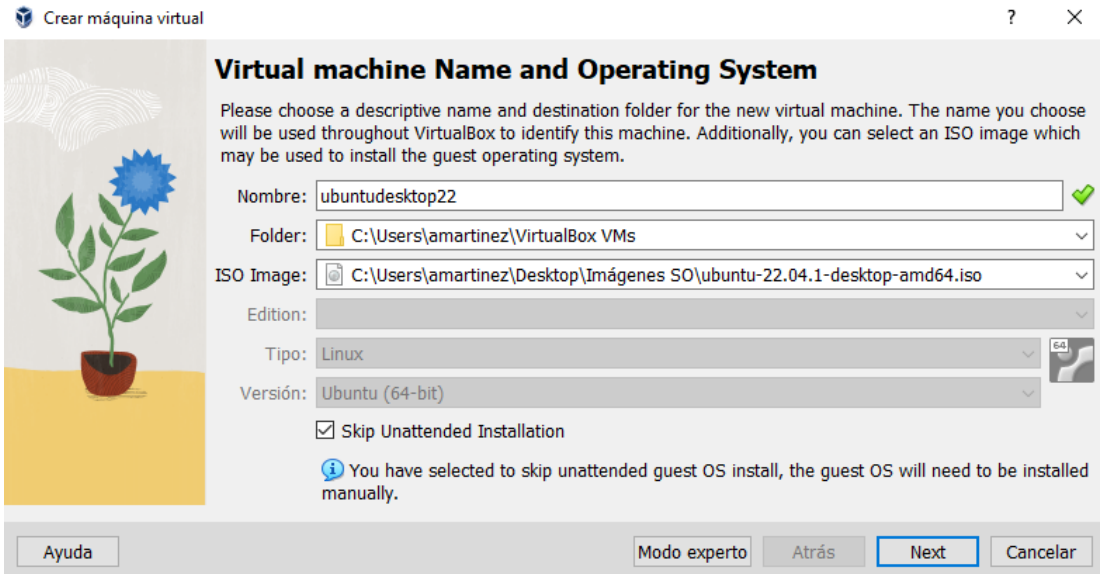


Figura A. 4. Crear una nueva máquina virtual en VirtualBox.

Los campos que aparecen se rellenan de la siguiente forma.

- *Nombre:* elegido por el usuario de manera arbitraria, en este caso se va a llamar *ubuntudesktop22* para reconocer de forma sencilla su SO.
- *Folder:* destino en el cual se va a guardar la máquina virtual que se va a crear, en este caso se deja sin cambios. Modificar según las preferencias del usuario.
- *ISO Image:* imagen del sistema operativo que va a correr la máquina virtual. En este caso se selecciona la imagen de Ubuntu descargada previamente, concretamente el archivo *Ubuntu 22.04.1-desktop-amd64.iso* (almacenado en la carpeta *Imágenes SO* dentro de *Desktop*).
- *Tipo* y *Versión:* se seleccionan de manera automática.
- *Skip Unattended Installation:* Seleccionar esta opción.

Tras configurar todos los apartados, seleccionar *Next* y proceder a la asignación de recursos. Tras finalizar este proceso comenzará la creación de la máquina e instalación del SO. Cuando finalice, el usuario podrá comenzar a configurar la máquina virtual e instalar el resto de los programas.

A.2 Instalación de Docker

La instalación de Docker y Docker-compose se va a llevar a cabo en la máquina virtual Linux, con sistema operativo a partir de la imagen Ubuntu 22.04.1-desktop-amd64.iso. Para ello se sigue el procedimiento siguiente. [FUENTE]

1. Desinstalar versiones anteriores

```
sudo apt-get remove docker docker-engine docker.io docker-compose containerd
```

2. Instalar usando el repositorio *apt.* Antes de instalar *Docker Engine* por primera vez en una máquina, hay que configurar el repositorio de Docker para poder instalar después Docker de este repositorio

   2.1 Configurar repositorio

   a) Actualizar el índice de paquete *apt.*

```
sudo apt-get update
```

   b) Instalar paquetes para permitir que *apt* use un repositorio sobre HTTPS.

```
sudo apt-get install \
  ca-certificates \
  curl \
  gnupg \
  lsb-release
```

c) Añadir la *GPG key* oficial de Docker.

```
sudo mkdir -m 0755 -p /etc/apt/keyrings
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
```

d) Configurar el repositorio

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

2.2 Instalar *Docker Engine*

e) Actualizar el índice de paquete *apt.*

```
sudo apt-get update
```

f) Instalar la última versión.

```
 sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
compose-plugin
```

g) Verificar funcionamiento ejecutando comandos propios de Docker.

MANEJO BÁSICO

A continuación, se recopilan los comandos más empleados en el entorno Docker (en este proyecto en concreto)

- *sudo docker images:* listar todas las imágenes instaladas en el host.
- *sudo docker rm <container_id>:* eliminar contenedor.
- *sudo docker rmi <image_id>:* eliminar imágenes.
- *sudo nano docker-compose.yml:* crear y editar el archivo *docker-compose.yml*
- *sudo docker-compose -p fiware up -d:* levantar contenedor compuesto ubicado en directorio fiware y ejecutarlo en segundo plano (-d, *detached*).
- *sudo docker ps:* listar las imágenes que están corriendo en el momento y ver su estado.
- *sudo docker-compose down:* "bajar" el contenedor compuesto, cesar su funcionamiento
- *sudo docker logs -f <image>:* muestra los logs de una imagen activa.
- *sudo docker-compose logs -f --tail=50:* muestra los logs del contenedor compuesto.

A.3 Instalación de Draco

La instalación de Draco se lleva a cabo de manera automática tras incluir su configuración en el archivo *docker-compose.yml* por lo que la única dificultad reside en hacer eso de manera correcta, como se ha explicado en apartados anteriores. Sin embargo, para llevar los datos hacia la base de datos MySQL se debe configurar el flujo en el portal Apache NiFi ubicado en este caso, (el puerto dependerá de la configuración elegida) en *http:///192.168.137.126:9091/nifi/.*

Una vez en el sitio, en la barra de herramientas superior se selecciona la opción *Template.*

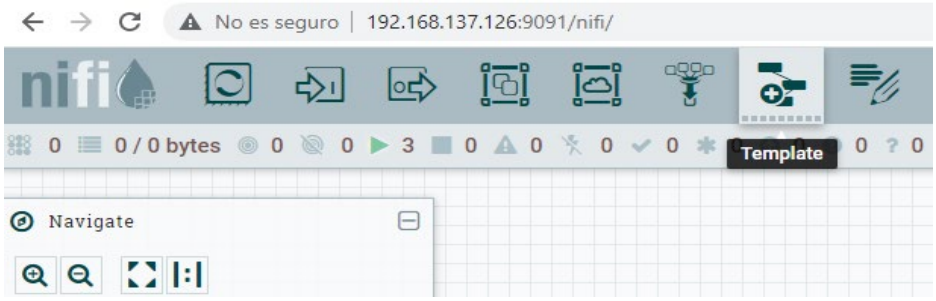


Figura A. 5 Portal Apache NiFi. Opción Template

A continuación, se despliega un menú con las diferentes plantillas, de donde se elige MYSQL_TUTORIAL.

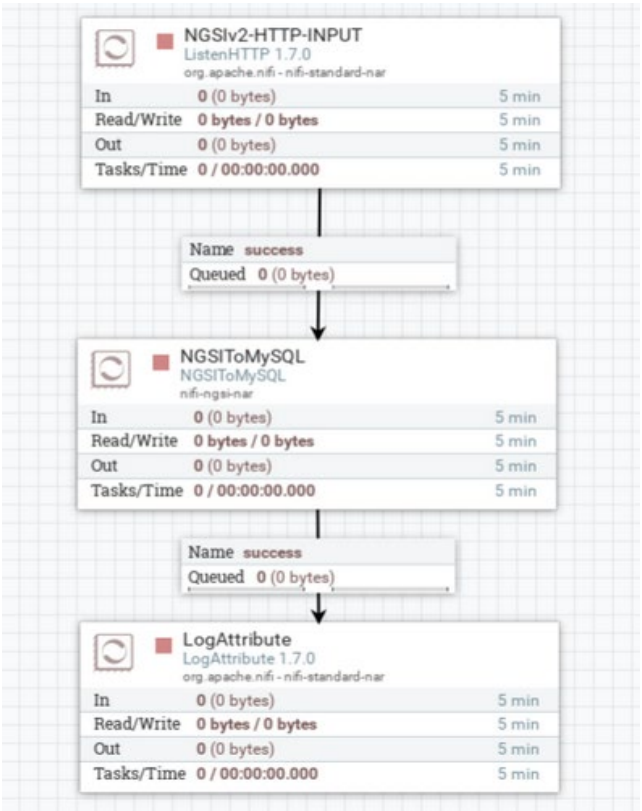


Figura A. 6 Template MYSQL_TUTORIAL.

Se debe configurar la plantilla para poder realizar la conexión. Para ello, haciendo clic derecho en la cuadrícula se selecciona la opción *Configure.*

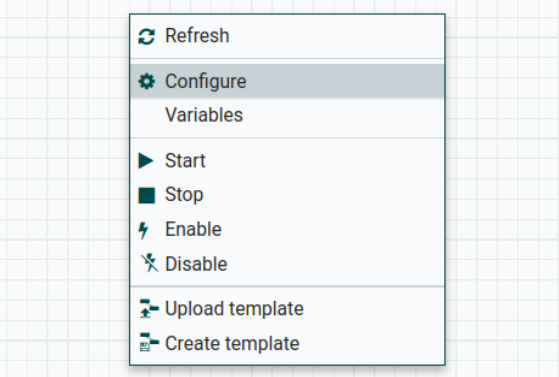


Figura A. 7 Configuración general.

Aparece el siguiente menú donde se puede ver el controlador *DBCPConnectionPool.* Se hace clic en el icono de la tuerca para acceder a la configuración del mismo y cambiar usuario y contraseña (mismas credenciales que en la configuración de docker-compose.yml).

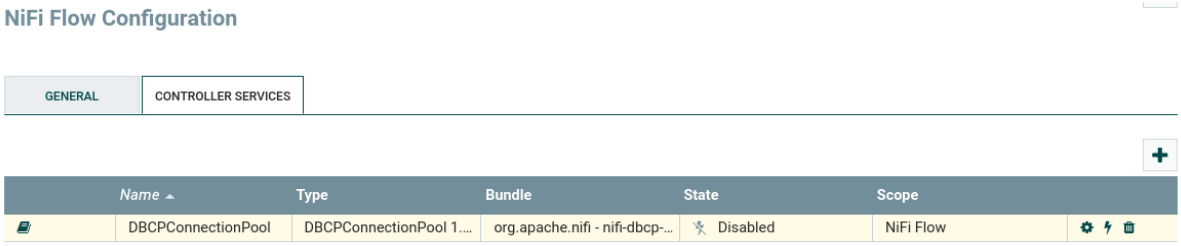


Figura A. 8 Configuración - Controller services.

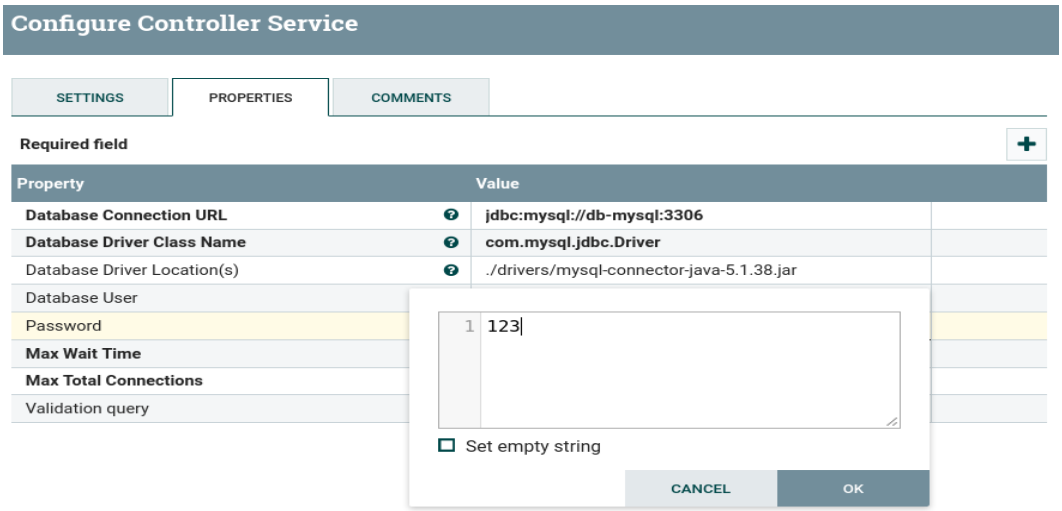


Figura A. 9 Credenciales como los de la configuración MySQL.

Tras aceptar y aplicar los cambios, se lanza el controlador haciendo clic en el icono del rayo.

Figura A. 10 Lanzar el controlador.

Tras aceptar y aplicar los cambios se vuelve al panel principal. Se selecciona el bloque *NGSIToMySQL* y haciendo clic derecho se accede a su configuración:

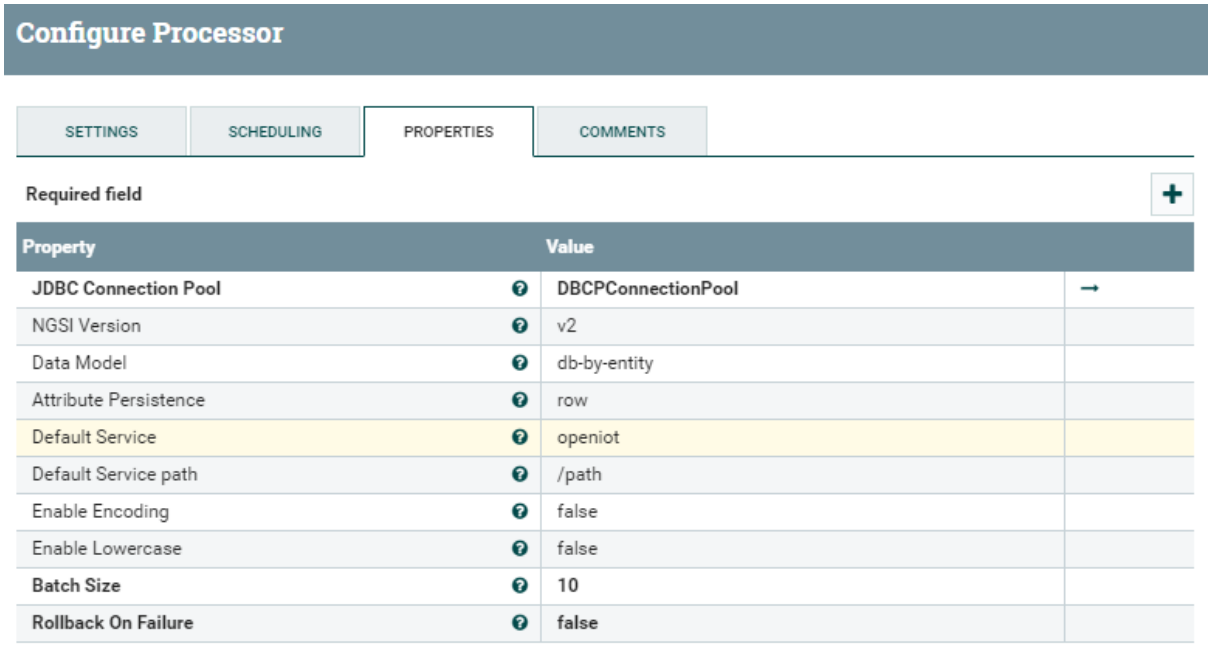


Figura A. 11 Se configura el procesador NGSIToMySQL.

En este menú es importante comprobar que la versión de NGSI empleada es la *v2*. Se configura también el campo *Default Service* para dar nombre a la base de datos que se va a crear para almacenar este flujo de datos, en este caso *openiot.*

Una vez hecho esto, se aceptan y aplican los cambios y se vuelve al panel principal, donde se activan los nodos con el botón *Start* que se muestra en la siguiente figura y los iconos que aparecen junto al nombre de cada bloque cambian a verde.

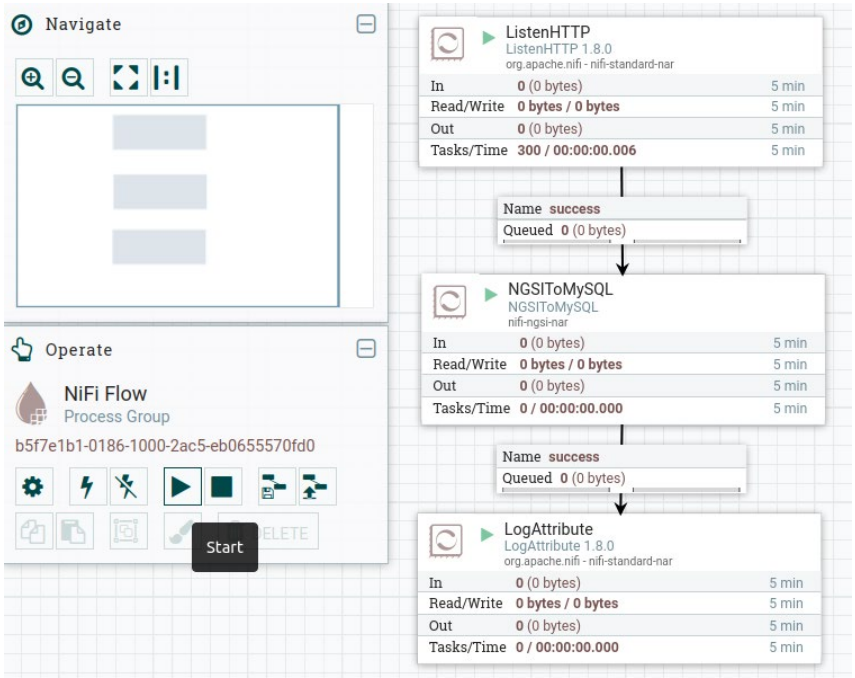


Figura A. 12 Flujo lanzado.

En este momento Draco está listo para recibir y direccionar el flujo de datos proveniente del sensor hacia la base de datos *MySQL* con el nombre *openiot*. Para visualizar estos datos, se puede hacer click derecho en cada bloque del flujo y seleccionar la opción *View data provenance*.

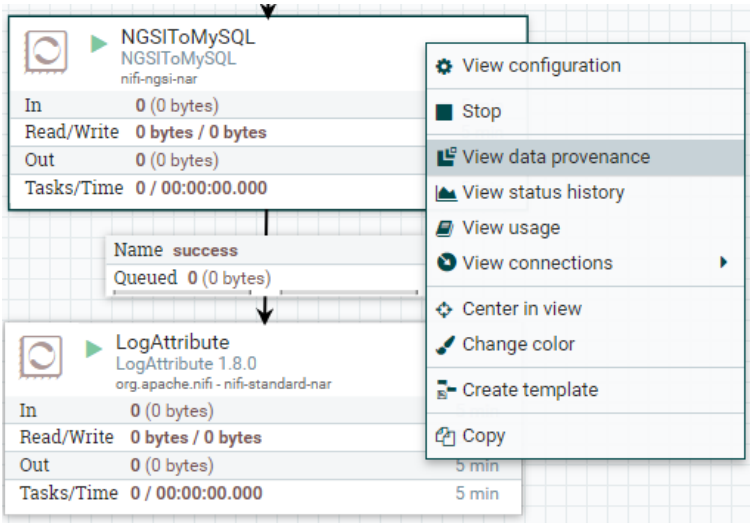


Figura A. 13 Ver flujo de datos.

De esta forma se accede al siguiente panel de control, donde se puede observar el flujo de datos proveniente del sensor por medio de Orion.

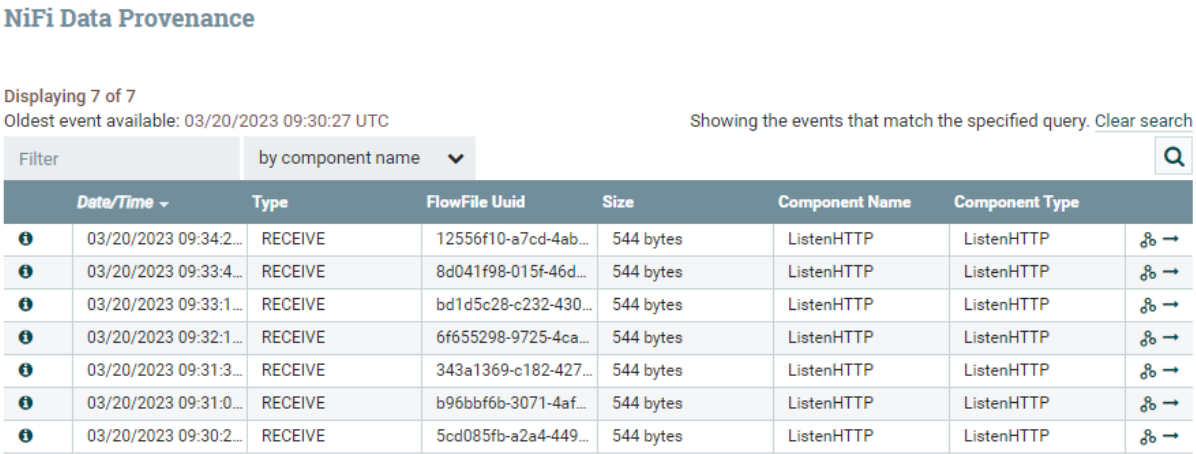


Figura A. 14 Ver datos.

La hora de recepción del mensaje está en formato UTC, siendo la franja horaria aquí en España UTC+2 (por el horario de verano), lo que supone un retraso de una hora entre la mostrada en el registro y la real. Esto se debe tener en cuenta a la hora de trabajar con el campo del día y la fecha.

A.4 Instalación de Arduino IDE

A.4.1 Gestión de Placas

El proceso de instalación del software Arduino IDE resulta sencillo, sin embargo, la configuración de la placa empleada puede ocasionar algún problema. En este apartado se describe con precisión los pasos necesarios para conectar correctamente la ESP32 con el PC. Cabe remarcar que existen otras formas de programar y cargar el software en la ESP32, pero se ha elegido Arduino IDE por ser la más conocida y utilizada.

Se lleva a cabo la instalación de Arduino IDE desde su página oficial de descarga [38] de manera gratuita. En este caso, se selecciona la opción de descarga para Windows 10 y versiones superiores, 64 bits. Una vez iniciado el software, se accede a *Archivo* > *Preferencias* y en el menú a continuación se debe verificar que la *Ruta del Sketchbook* sigue la estructura que se muestra.

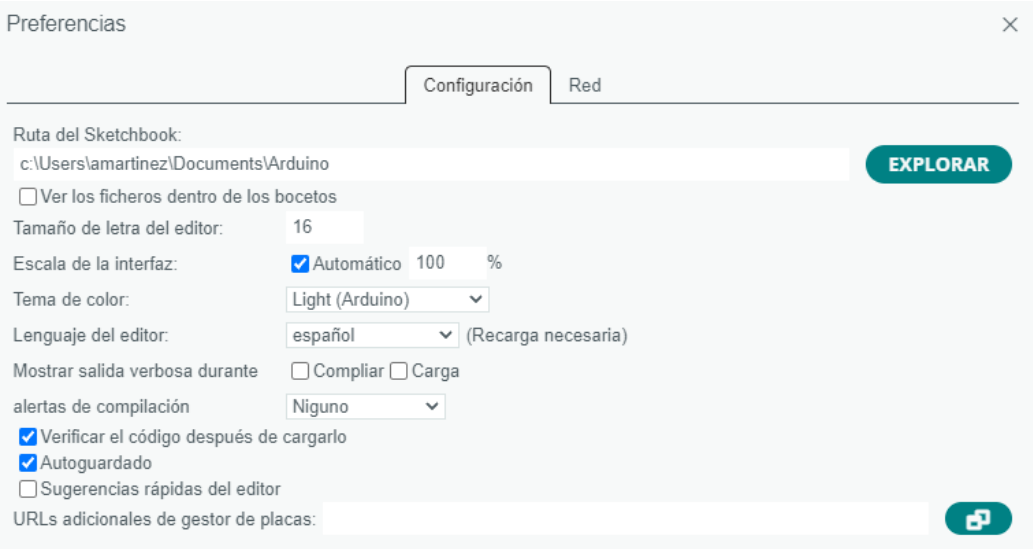


Figura A. 15 Configuración de las preferencias.

Se puede observar que el campo *URLs adicionales de gestor de placas* está vacío. Por defecto, Arduino trae consigo las URLs de sus placas propias, sin embargo, al querer emplear este software para configurar placas de otro fabricante distinto, se deben incluir en este campo las URLs siguientes para la gestión de la placa que se va a utilizar (ESP32 C3 DevKitM 1).

Por este motivo, hacer clic en el icono que aparece en la imagen abajo a la derecha e incluir las siguientes direcciones.

- https://dl.espressif.com/dl/package_esp32_index.json
- https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_dev_index.json
- https://resource.heltec.cn/download/package_heltec_esp32_index.json

Figura A. 16 Insertar URL`s adicionales

Para salir y guardar los cambios se debe pulsar "ACEPTAR" en ambas pantallas.

Una vez realizado el paso anterior, accediendo a *Herramientas > Placa > Gestor de placas* es posible acceder a los archivos que se encuentran en las URLs indicadas anteriormente, que contienen los paquetes necesarios para poder trabajar con el modelo de ESP32 elegido.

Escribir en el buscador "esp32" e instalar el paquete cuyo nombre es *ESP32 de Espressif Systems.*

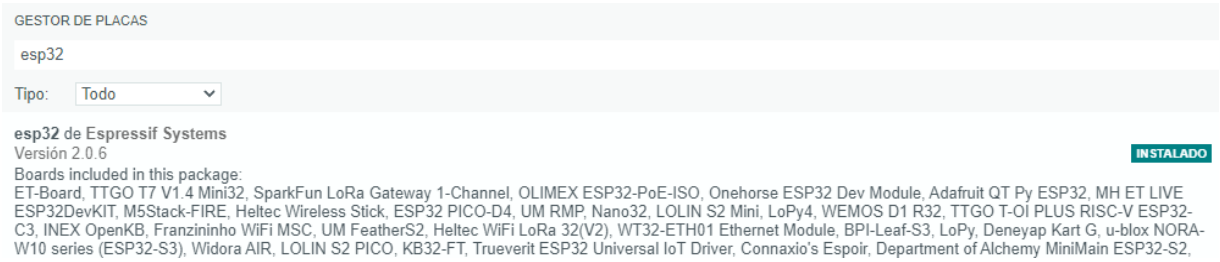


Figura A. 17 Gestor de placas.

Una vez instalado el complemento anterior, se puede seleccionar la placa accediendo a *Herramientas > Placas > esp32 > ESP32 C3 Dev Modul.*

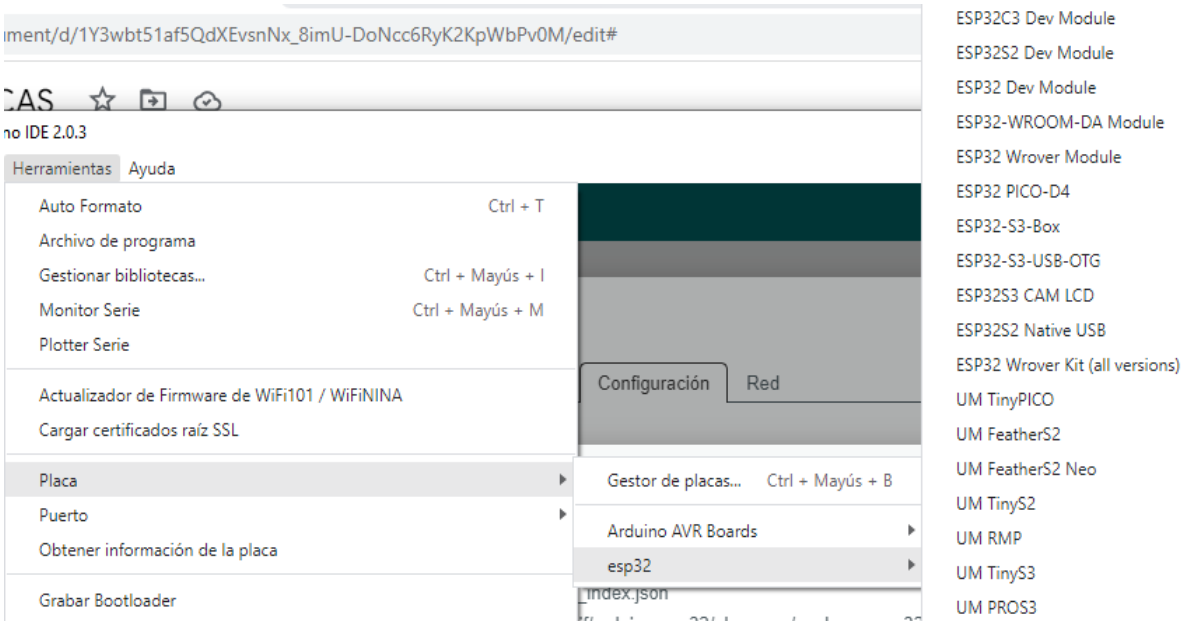


Figura A. 18 Selección de placa

Al conectar la ESP32 al PC mediante el cable USB - microUSB, Arduino asigna un puerto automáticamente a esta nueva conexión con el nuevo dispositivo, por lo que, para poder comunicar con la placa, sólo queda seleccionar dicho puerto en *Herramientas > Puerto > COM( ).*

De esta forma, la ESP32 queda configurada y lista para recibir información desde el PC.

A.4.2 Librerías necesarias

En este apartado se van a listar las diferentes librerías empleadas en el código final, cómo se instala cada una de ellas y dónde se ubica, ya que en ocasiones puede resultar algo complicado por no seguir todas el mismo proceso.

Existen dos tipos de librerías diferenciados al acceder a *Sketch > Incluir biblioteca,* las *Bibliotecas de Arduino (bibliotecas propias, BP)* y las *Bibliotecas aportadas (BA)*.Como su propio nombre indica, las primeras vienen instaladas de serie con Arduino IDE y las últimas son las que descarga el usuario.

Para las siguientes funcionalidades se requieren diferentes librerías:

- Datos con sensor DHT11: *DHT_sensor_library & Adafruit_Unified_Sensor (BA)*
- Construir mensajes en lenguaje JSON: *ArduinoJson (BA)*
- Comunicación con protocolo HTTP: *HTTPClient.h (BA)*
- Conexión WiFi *WiFi (BP)*

Para descargar las dos primeras librerías hay que acceder a *Herramientas > Gestionar bibliotecas* y escribir en el buscador cada una de ellas. Concretamente, para este proyecto se han empleado:

Figura A. 19 Librerías empleadas.

Sin embargo, para descargar la librería del protocolo HTTP es necesario realizar otro proceso. El primer paso es acceder al siguiente enlace de GitHub (repositorio de código) HTTPClient/src [39] donde aparecerá la siguiente pantalla.

Figura A. 20 Descargar librería para HTTP.

Una vez aquí, hay que abrir el archivo *HTTPClient.h* y copiar su contenido en un nuevo *Sketch* de Arduino, para después guardarlo en una carpeta de nombre (en este caso) *HTTPClient Port* ubicada en la *Ruta del Sketchbook [REFERENCIA]* y destino *libraries:* c:\Users\amartinez\Documents\Arduino\libraries.

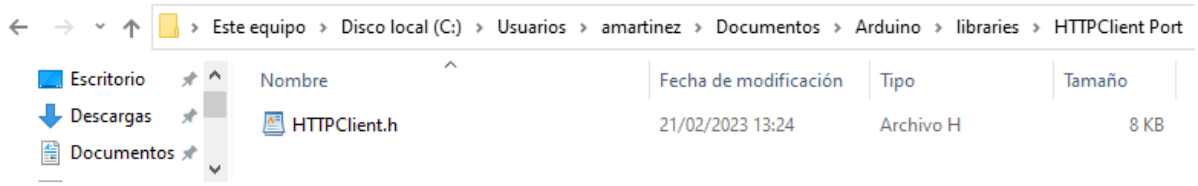


Figura A. 21. Ubicación librería HTTPClient.h

Por último, la librería para la conexión WiFi viene instalada por defecto en Arduino IDE. Se puede ver accediendo a *Sketch > Incluir biblioteca.*

De esta forma, se tienen todas las librerías necesarias para que el *script* funcione correctamente. Se deben incluir en el código de la siguiente manera.

```
#include                                                    <WiFi.h>
#include                                                <HTTPClient.h>
#include                                                     <DHT.h>
#include                                                <ArduinoJson.h>
#include <ArduinoJson.hpp>
```

A.5 Instalación de Node-RED

En este apartado se va a detallar el proceso de descarga y configuración de Node-RED, ya que existen pequeños detalles que de no reconocerlos a tiempo pueden ser grandes inconvenientes.

La instalación de Node-RED se va a llevar a cabo en la máquina virtual Linux, con Ubuntu 22.04.1-desktop. Para ello, se accede en primer lugar a la página oficial de Node-RED [40] , donde de las tres opciones que sugiere, se opta por la primera, instalar usando *npm* (comando de *node.js*).

1. Ejecutar el siguiente comando para proceder a la instalación con *npm.*

```
sudo npm install -g --unsafe-perm node-red
```

2. Es probable que la máquina no tenga *node.js* instalado, pero la propia consola de comandos hace de guía para la instalación.
3. Al instalar *node.js* y poder utilizar el comando *npm,* ejecutar de nuevo el comando [1] y la instalación se completará.

Por defecto, Node-RED se ejecuta en la IP local: 127.0.0.0, sin embargo, para poder acceder al portal desde el PC host Windows (mayor comodidad), se debe cambiar la IP en la que se lanza el sitio de Node-RED.

Para ello se debe acceder al archivo *settings.js,* ubicado, como se puede ver en la información que aparece en la consola al ejecutar *node-red* en: /home/<user>/.node-red

```
20 Mar 13:26:48 - [info] Dashboard version 3.4.0 started at /ui
20 Mar 13:26:48 - [info] Settings file  : /home/amartinez/.node-red/settings.js
```

Figura A. 22. Ubicación archivos de configuración.

Sin embargo, al acceder a la *Carpeta personal* del usuario no se podrá ver la carpeta *.node-red* si no está activada la opción *Mostrar archivos ocultos.*

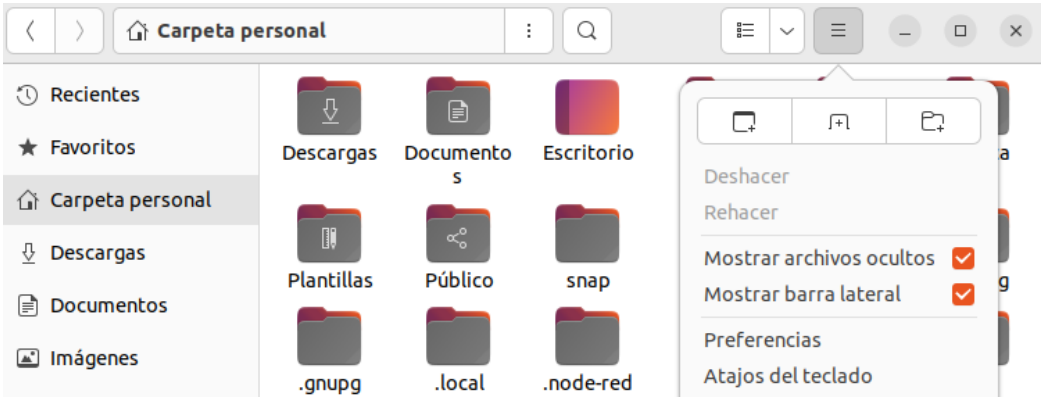


Figura A. 23. Carpeta personal en Ubuntu.

Una vez dentro de la carpeta, se abre con un editor de texto el archivo *settings.js* para proceder a personalizarlo. Primero se busca el apartado *Server Settings*.

```
/**********************************************************************
 * Server Settings
 *  - uiPort
 *  - uiHost
 *  - apiMaxLength
 *  - httpServerOptions
 *  - httpAdminRoot
 *  - httpAdminMiddleware
 *  - httpNodeRoot
 *  - httpNodeCors
 *  - httpNodeMiddleware
 *  - httpStatic
 *  - httpStaticRoot
 **********************************************************************/

    /** the tcp port that the Node-RED web server is listening on */
    uiPort: process.env.PORT || 1880,

    /** By default, the Node-RED UI accepts connections on all IPv4 interfaces.
     * To listen on all IPv6 addresses, set uiHost to "::",
     * The following property can be used to listen on a specific interface. For
     * example, the following would only allow connections from the local machine.
     */
    uiHost: "192.168.137.126",
```

Figura A. 24. Apartado Server Settings de archivo settings.js.

La línea que se debe modificar es la 151. Por defecto aparece como: //uiHost:”127.0.0.0”, sin embargo, se debe des-comentar y colocar la IP de la máquina virtual, que es accesible para el PC Host.

Otra configuración interesante se encuentra en el apartado *Security*. En este apartado se explica cómo configurar un usuario y contraseña para proteger el acceso al portal de Node-RED. Para activarlo se debe des-comentar el código que contiene las credenciales y personalizarlas. En este proyecto no se usa por comodidad.

```
/**********************************************************************
 * Security
 *  - adminAuth
 *  - https
 *  - httpsRefreshInterval
 *  - requireHttps
 *  - httpNodeAuth
 *  - httpStaticAuth
 **********************************************************************/

    /** To password protect the Node-RED editor and admin API, the following
     * property can be used. See http://nodered.org/docs/security.html for details.
     */
    //adminAuth: {
        //type: "credentials",
        //users: [{
            //username: "admin",
            //password: "1234",
            //permissions: "*"
        //}]
    //},

```

Figura A. 25. Apartado Security de archivo settings.js.

Al ejecutar nuevamente el comando *node-red* en la consola, entre los *logs* generados aparece la línea siguiente indicando que el portal Node-RED está corriendo en la IP configurada y en su puerto predeterminado, el 1880.

```
20 Mar 13:26:48 - [info] Server now running at http://192.168.137.126:1880/
```

Figura A. 26. Log de inicio de servidor.

A.5.1 Librerías necesarias

Node-RED incluye multitud de opciones y funcionalidades interesantes de base, pero éstas pueden ser ampliadas por librerías y complementos desarrollados por los propios usuarios (de código abierto). Se instalan de manera sencilla desde el propio panel de Node-RED, al cual se accede escribiendo en el buscador la dirección en la cual se está ejecutando el servidor, en este caso http://192.168.137.126:1880.

Para gestionar las librerías se selecciona el icono de las tres barras de la esquina superior derecha de la imagen y en el menú desplegable se selecciona la opción *Manage palette.*

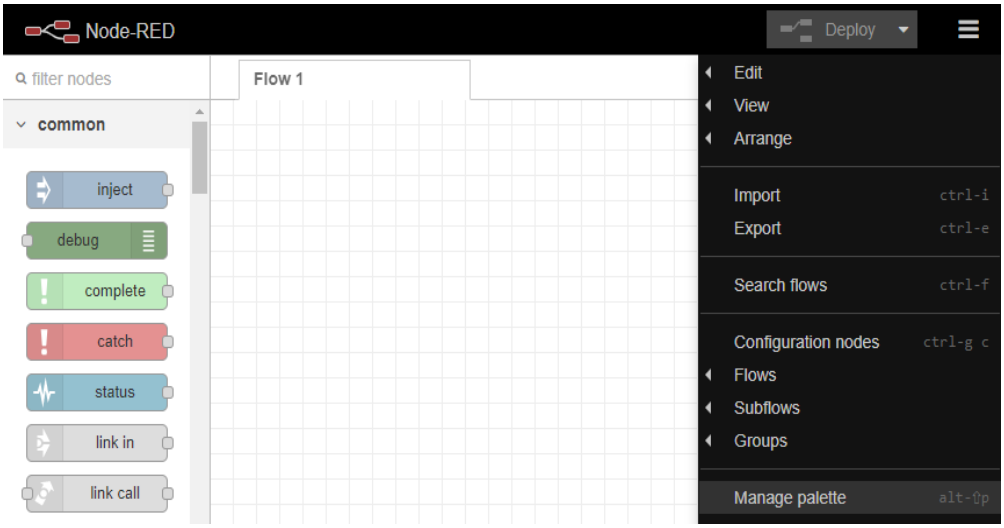


Figura A. 27. Seleccionar opción Manage palette.

Aparece el siguiente menú donde en el buscador *(search modules)* se pueden encontrar los diferentes complementos o librerías a instalar.

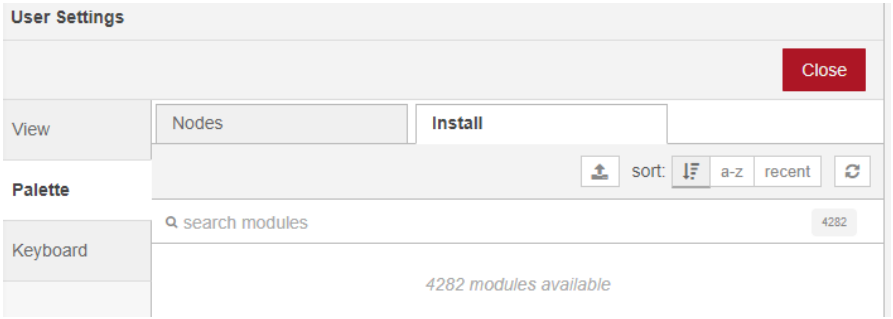


Figura A. 28. Librerías Node-RED

Los módulos descargados y empleados en este proyecto son los siguientes.

- Representación gráfica de valores: *node-red-dashboard*
- Conexión con FIWARE *node-red-contrib-snap4city-user*
- Nodos para Twitter *node-red-node-twitter*

A.6 Instalación de Postman

La instalación de Postman es muy sencilla, basta con acceder a su página oficial de descarga[41] y elegir en este caso, su aplicación de escritorio para Windows (64 bits). Seguir el proceso guiado para la instalación y se creará automáticamente un acceso directo al portal de Postman en el escritorio.

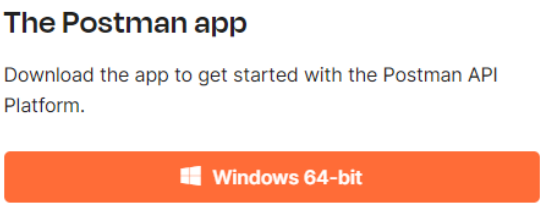


Figura A. 29. Instalación de Postman.

A.7 Instalación de MySQL Workbench

A continuación, se recoge el proceso completo de instalación y configuración del software MySQL Workbench, empleado para gestionar de manera sencilla las bases de datos de MySQL. El primer paso es por tanto instalar el cliente, desde la web oficial[42].

Una vez instalado el programa, se abre y aparece el siguiente menú.

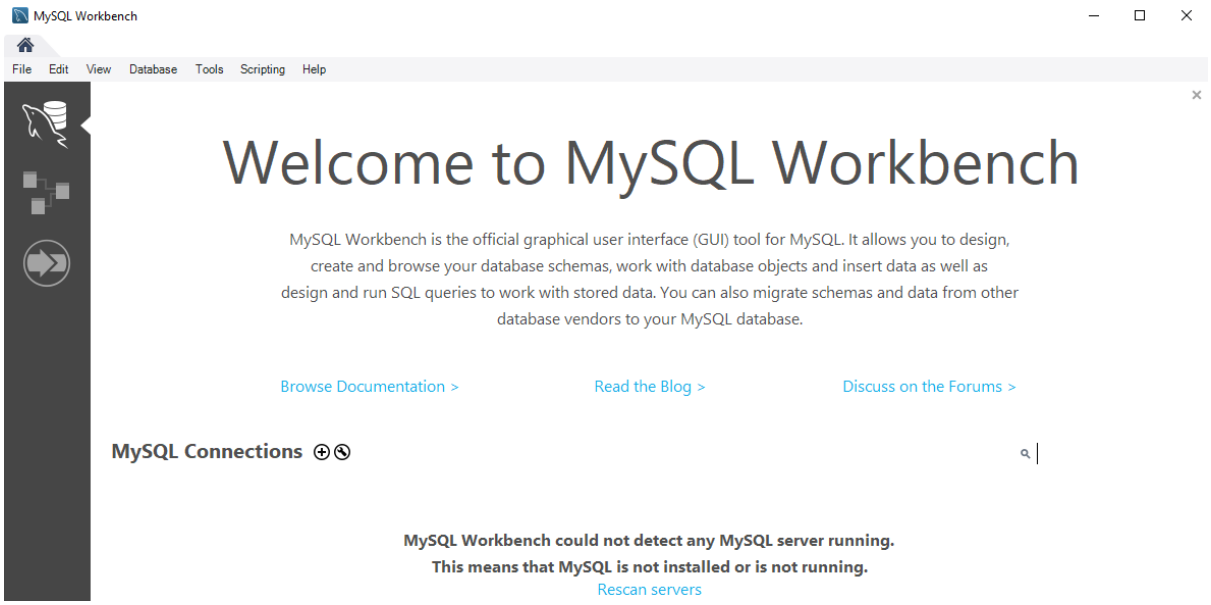


Figura A. 30. Menú de inicio de MySQL Workbench.

Seleccionando el icono + a la derecha de *MySQL Connections* se despliega el siguiente menú para la configuración de una nueva conexión con una base de datos. Se configura como se muestra para ver la base de datos ubicada en el puerto 3306 de la IP 192.168.137.126. Se introducen las credenciales y se aplica la configuración.

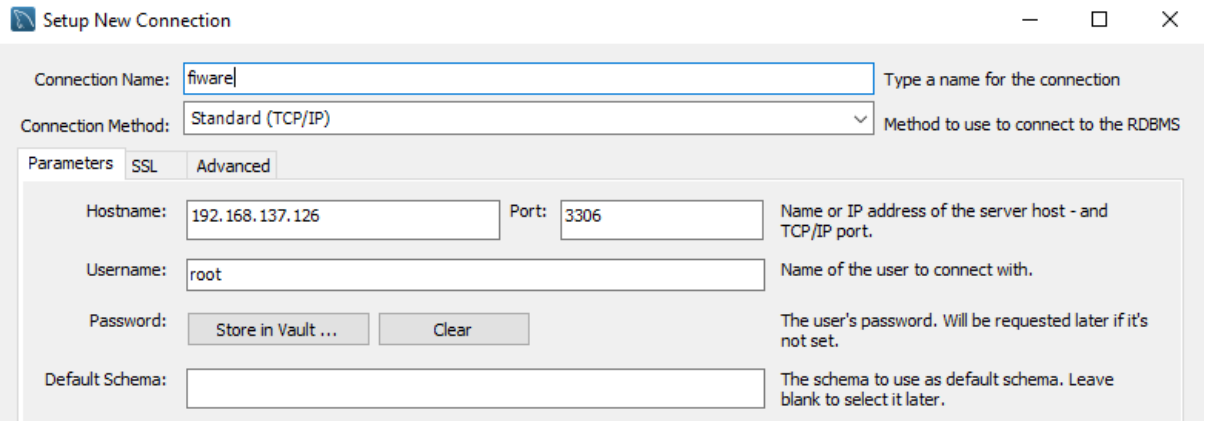


Figura A. 31. Configurar una nueva conexión.

Una vez hecho esto, se selecciona esta conexión y aparece el siguiente menú que muestra los *schemas* entre los que se encuentra la base de datos *openiot.* Se accede a las tablas, donde la única opción es *urn_ngsild_Sensor_001_Sensor.* Se muestran también las diferentes columnas.

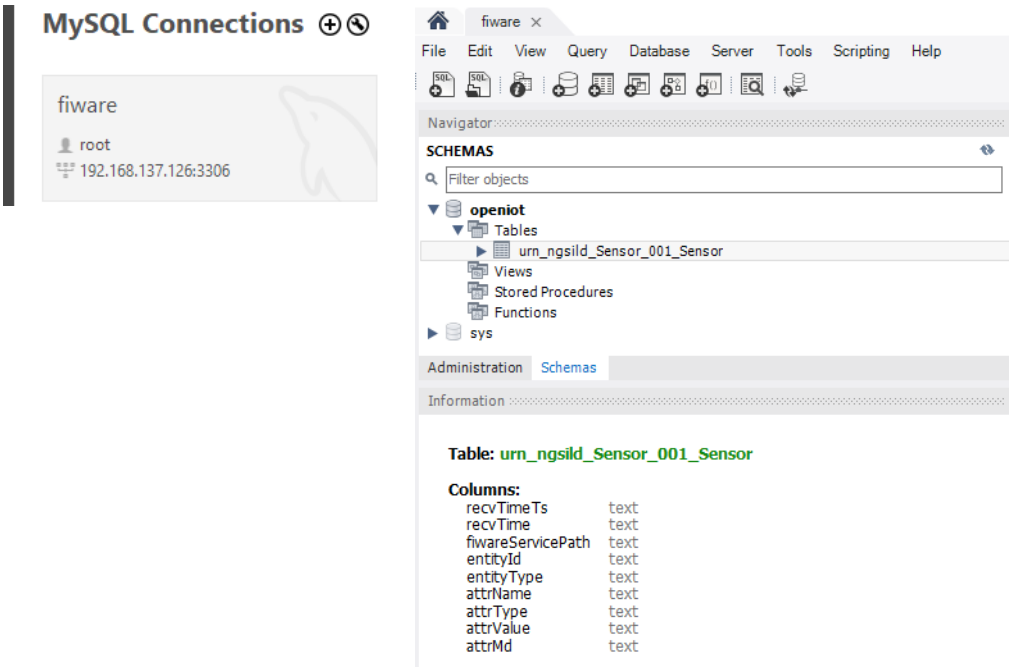


Figura A. 32. Conexión con base de datos configurada con éxito.

A.8 Instalación de Power BI

La instalación de Power BI es muy sencilla, basta con acceder a su página oficial de descarga [43] y elegir en este caso, su aplicación de escritorio para Windows (64 bits). Pulsar el botón *Descargar* y un instalador se descarga automáticamente. Seguir los pasos para generar un acceso directo al programa en el escritorio.

Anexo B. Scripts de Arduino

En este apartado se recogen todos los *scripts* empleados a lo largo del proyecto, separados en las diferentes funciones principales para que puedan ser utilizados de manera individual y facilitar la comprensión de estos.

B.1 Recogida de datos con sensor DHT11

A continuación, se muestra el script necesario para recoger datos del sensor DHT11. Se incluye, como se puede observar, la librería *DHT.h* cuyo proceso de instalación ha sido explicado en el Anexo A.4

```
#include <DHT.h>

#define DHTPIN 7
#define DHTTYPE DHT11

DHT dht(DHTPIN,DHTTYPE);

void setup() {
  Serial.begin(115200);
  dht.begin();
}

void loop() {
  delay(5000);
  int h = dht.readHumidity();              // Humedad relativa
  int t = dht.readTemperature();           // Temperatura ºC
  // Comprobar si ha habido algún error en la lectura
  if (isnan(h) || isnan(t)) {
    Serial.println("Error obteniendo los datos del sensor DHT11");
    return;
  }
  Serial.print("La humedad medida es: "); Serial.print(h); Serial.println("%");
  Serial.print("La temperatura medida es: "); Serial.print(t); Serial.println("ºC");
}
```

Output que se puede ver en *Monitor Serie.*

Figura B. 1 Output al ejecutar el script anterior, obteniendo valores del sensor DHT11.

Si se observa el recuadro que aparece en la esquina superior izquierda de la captura de pantalla, se puede leer *115200 baud*. En primer lugar, para que no haya problemas, es importante que este número coincida con el especificado en el campo *Serial.begin(115200)*, dentro de *void setup()*.

Sin embargo, para entender qué son los baudios (*baud* en inglés), se definen como una unidad de medida usada en telecomunicaciones, que representa el número de símbolos por segundo en un medio de transmisión analógico o digital [FUENTE].

Más concretamente, Arduino emplea el protocolo RS-232 para la comunicación serial con otros dispositivos. Al cargar información en la placa de desarrollo, se hace mediante el puerto serie, que emplea este protocolo. También es común mostrar información en la pantalla del ordenador, mediante el Monitor Serie de Arduino IDE.

Es por ello que en la instrucción de inicialización del puerto serie, *Serial.begin(),* se debe especificar la velocidad de comunicación serial que se desea dependiendo de la aplicación, y este debe coincidir con la "velocidad de lectura" del monitor serie, como se ha mencionado anteriormente.

B.2 Implementación lenguaje JSON

```
#include <ArduinoJson.h>
#include <ArduinoJson.hpp>
#include <DHT.h>
#define DHTPIN 7
#define DHTTYPE DHT11
DHT dht(DHTPIN,DHTTYPE);

void setup() {
  Serial.begin(115200);
  dht.begin();
}

void loop() {
  delay(5000);
  int h = dht.readHumidity();              // Humedad relativa
  int t = dht.readTemperature();           // Temperatura ºC
  if (isnan(h) || isnan(t)) {
    Serial.println("Error obteniendo los datos del sensor DHT11");
    return;
  }

  String json;
  DynamicJsonDocument doc(1024);
  doc["t"]= t;
  doc["h"]= h;
  serializeJson(doc, json);
  Serial.println(json);
}
```

Output que se puede ver en *Monitor Serie.*

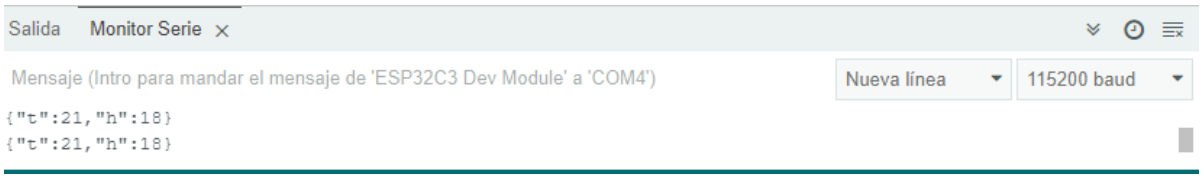


Figura B. 2 Output al ejecutar el script anterior, implementando JSON.

B.3 Conexión WiFi y comunicación HTTP

```
#include <WiFi.h>
#include <HTTPClient.h>

const char* ssid = "XXXX";
const char* password = "XXXX";

String serverPath = "http://192.168.137.126:7896/iot/json?i=DHT11&k=apikey";

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  Serial.print("Conectando...");
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Conectado a WiFi con IP: "); Serial.println(WiFi.localIP());
}

void loop() {
  http.begin(serverPath.c_str());
  http.addHeader("Content-Type", "application/json");
  int httpResponseCode = http.POST("mensaje");
  if (httpResponseCode>0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
    String payload = http.getString();
    Serial.println(payload);
  }
  else {
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
  }
  http.end();
}
```

Output que se puede ver en *Monitor Serie.*

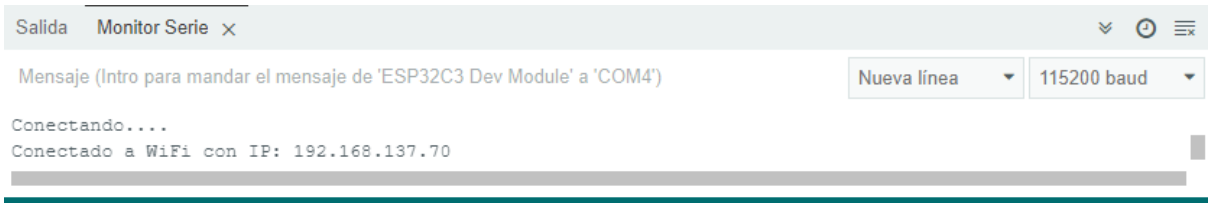


Figura B. 3 Output al ejecutar el script anterior, IP asignada por conexión a WiFi.

En este script no se lleva a cabo ninguna petición HTTP. En caso de querer ejecutar una petición de tipo GET, se debe modificar la parte del código correspondiente como se indica en los comentarios de este.

B.4 Script completo

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <DHT.h>
#include <ArduinoJson.h>
#include <ArduinoJson.hpp>

#define DHTPIN 7
#define DHTTYPE DHT11
DHT dht(DHTPIN,DHTTYPE);

//Definir credenciales para acceso WiFi
const char* ssid = "XXXX";
const char* password = "XXXX";

//Introducir destino
String serverName = "http://192.168.137.126:7896/iot/json?i=DHT11&k=apikey";

//Variables que almacenan tiempo en milisegundos, tipo unsigned long.
unsigned long lastTime = 0;
unsigned long timerDelay = 30000;   // Timer a 30 segundos (30000)

void setup() {
  Serial.begin(115200);
  dht.begin();
  WiFi.begin(ssid, password);
  Serial.println("Conectando...");
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Conectado a WiFi con IP: ");
  Serial.println(WiFi.localIP());

  Serial.println("Timer set to 5 seconds (timerDelay variable), it will take 5 seconds
before publishing the first reading.");
}

void loop() {
  //Enviar un post cada 10 minutos
  if ((millis() - lastTime) > timerDelay) {
    //Comprobar estado de la conexión WiFi
    if(WiFi.status()== WL_CONNECTED){
      HTTPClient http;
      String serverPath = serverName;
      delay(5000);
      int h = dht.readHumidity();
      int t = dht.readTemperature();
      if (isnan(h) || isnan(t)) {
      Serial.println("Error obteniendo los datos del sensor DHT11");
      return;
      }
      http.begin(serverPath.c_str());
```

```
      http.addHeader("Content-Type", "application/json");

      // Crear dato con formato JSON
      String json;
      DynamicJsonDocument doc(1024);
      doc["t"]= t;
      doc["h"]= h;
      serializeJson(doc, json);
      Serial.println(json);

      int httpResponseCode = http.POST(json);

      if (httpResponseCode>0) {
        Serial.print("HTTP Response code: ");
        Serial.println(httpResponseCode);
        String payload = http.getString();
        Serial.println(payload);
      }
      else {
        Serial.print("Error code: ");
        Serial.println(httpResponseCode);
      }
      http.end();
    }
    else {
      Serial.println("WiFi Disconnected");
    }
    lastTime = millis();
  }
}
```

Output que se puede ver en *Monitor Serie.*

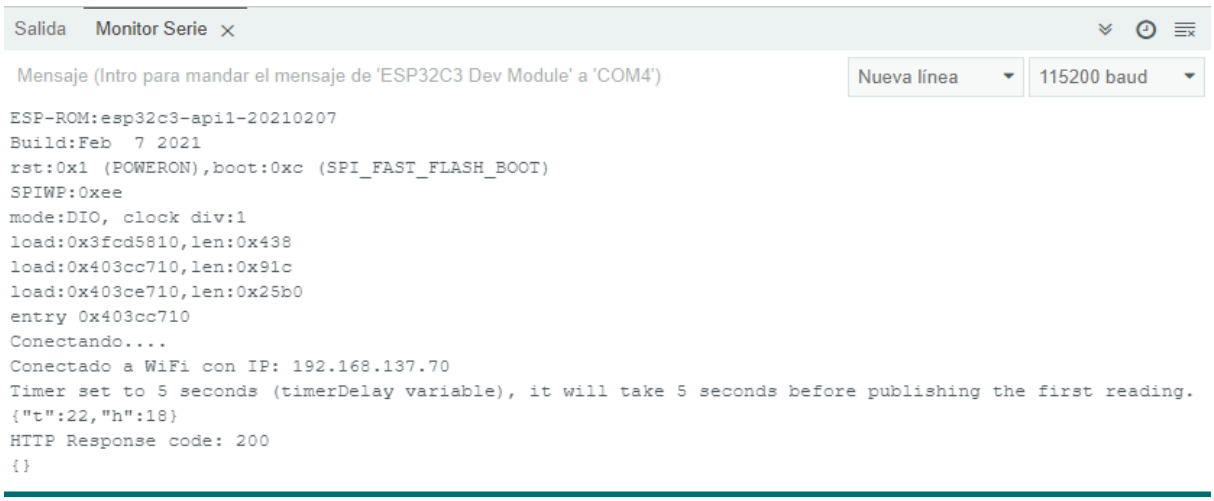


Figura B. 4 Output al ejecutar el script completo, comunicación efectiva.

El resultado al ejecutar el script completo es el deseado. Si la WiFi se encuentra accesible y la plataforma FIWARE está levantada y configurada correctamente como se ha mostrado, la comunicación es efectiva y se actualiza el valor de las entidades.

Anexo C. Scripts Node-RED

En este apartado se incluye el código que contienen los diferentes flujos creados en Node-RED para que resulte sencillo reproducir las pruebas realizadas. Basta con copiar y pegar el código.

C.1 *Script* para prueba de comunicación


```
[{"id":"ccedd8b841592462","type":"tab","label":"Flow
1","disabled":false,"info":"","env":[]},{"id":"2e6d1bd9dba597e2","type":"http
in","z":"ccedd8b841592462","name":"","url":"update-
sensor","method":"post","upload":false,"swaggerDoc":"","x":360,"y":240,"wires":[["4bd6877c83a8300b","7b
44c72f780196d9","b07a28c48105402b"]]},{"id":"4bd6877c83a8300b","type":"http
response","z":"ccedd8b841592462","name":"","statusCode":"200","headers":{},"x":560,"y":180,"wires":[]},
{"id":"7b44c72f780196d9","type":"debug","z":"ccedd8b841592462","name":"debug
1","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"false","statusVal":"","statusType
":"auto","x":560,"y":240,"wires":[]},{"id":"b07a28c48105402b","type":"json","z":"ccedd8b841592462","na
me":"","property":"payload","action":"obj","pretty":true,"x":550,"y":300,"wires":[["a13eb5f007f5a424"]]},{"
id":"a13eb5f007f5a424","type":"function","z":"ccedd8b841592462","name":"JSON or URL
Encoded","func":"var msg0 = { payload: msg.payload.sensor };\nvar msg1 = { payload: msg.payload.temp
};\nvar msg2 = { payload: msg.payload.hum };\n\nreturn [msg0, msg1,
msg2];","outputs":3,"noerr":0,"initialize":"","finalize":"","libs":[],"x":730,"y":300,"wires":[["5629d68daf878
1df"],["d08566ec534595ec"],["b68a3aa86ed7a4c0"]]},{"id":"5629d68daf8781df","type":"ui_text","z":"ccedd
8b841592462","group":"2b7ac01b.fc984","order":1,"width":0,"height":0,"name":"","label":"Sensor
Name","format":"{{msg.payload}}","layout":"row-
spread","className":"","x":1000,"y":280,"wires":[]},{"id":"d08566ec534595ec","type":"ui_gauge","z":"cced
d8b841592462","name":"","group":"2b7ac01b.fc984","order":2,"width":0,"height":0,"gtype":"gage","title":"
Temperature","label":"ºC","format":"{{value}}","min":0,"max":"38","colors":["#00b500","#e6e600","#ca38
38"],"seg1":"","seg2":"","diff":false,"className":"","x":950,"y":320,"wires":[]},{"id":"b68a3aa86ed7a4c0","
type":"ui_gauge","z":"ccedd8b841592462","name":"","group":"2b7ac01b.fc984","order":3,"width":0,"height
":0,"gtype":"gage","title":"Humidity","label":"%","format":"{{value}}","min":0,"max":"100","colors":["#00
80ff","#0062c4","#002f5e"],"seg1":"","seg2":"","diff":false,"className":"","x":920,"y":360,"wires":[]},{"id"
:"2b7ac01b.fc984","type":"ui_group","name":"SENSORS","tab":"99ab8dc5.f435c","order":1,"disp":true,"wi
dth":"6","collapse":false},{"id":"99ab8dc5.f435c","type":"ui_tab","name":"HTTP","icon":"dashboard","order
":1,"disabled":false,"hidden":false}]
```


C.1 *Script* final Node-RED


```
[{"id":"ccedd8b841592462","type":"tab","label":"Alerta
GMAIL","disabled":false,"info":"","env":[]},{"id":"a8427805aa5e9a38","type":"inject","z":"ccedd8b841592
462","name":"Campos del
gmail","props":[{"p":"payload"},{"p":"topic","vt":"str"}],"repeat":"","crontab":"","once":false,"onceDelay":0
.1,"topic":"Prueba Node-RED","payload":"Node-RED envía
correctamente","payloadType":"str","x":130,"y":460,"wires":[["7157d15323ee35f3"]]},{"id":"b35fb70a6b45
807c","type":"whin_send","z":"ccedd8b841592462","name":"","auth":"c6bee93baac17d0d","x":270,"y":580,
"wires":[[]]},{"id":"40d6d28930d24c17","type":"inject","z":"ccedd8b841592462","name":"Texto","props":[{
"p":"payload"}],"repeat":"","crontab":"","once":false,"onceDelay":0.1,"topic":"","payload":"{\"text\":
\"Enviado desde Node-
RED\"}","payloadType":"json","x":90,"y":580,"wires":[["b35fb70a6b45807c"]]},{"id":"7157d15323ee35f3"
,"type":"e-
mail","z":"ccedd8b841592462","server":"smtp.gmail.com","port":"465","secure":true,"tls":true,"name":"asie
rmartinezsanesteban@gmail.com","dname":"gmail
notif","x":330,"y":460,"wires":[]},{"id":"1abebdf5d1881ac5","type":"comment","z":"ccedd8b841592462","n
ame":"Envío de correo
CORRECTO","info":"","x":140,"y":420,"wires":[]},{"id":"d39f9d9946ce47f4","type":"comment","z":"ccedd
8b841592462","name":"Envío de mensaje Whatsapp
CORRECTO","info":"","x":180,"y":540,"wires":[]},{"id":"46b7b2552cad0d53","type":"inject","z":"ccedd8b
841592462","name":"hum =
25","props":[{"p":"payload"}],"repeat":"","crontab":"","once":false,"onceDelay":0.1,"topic":"","payload":"25
","payloadType":"num","x":520,"y":400,"wires":[["b72bbd4e2d9d994b"]]},{"id":"9fc8ff1f37853864","type"
```

:"inject","z":"ccedd8b841592462","name":"hum=50","props":[{"p":"payload"}],"repeat":"","crontab":"","once":false,"onceDelay":0.1,"topic":"","payload":"50","payloadType":"num","x":520,"y":440,"wires":[["b72bbd4e2d9d994b"]]},{"id":"02be5b7aa768d055","type":"inject","z":"ccedd8b841592462","name":"hum=85","props":[{"p":"payload"}],"repeat":"","crontab":"","once":false,"onceDelay":0.1,"topic":"","payload":"85","payloadType":"num","x":520,"y":520,"wires":[["b72bbd4e2d9d994b"]]},{"id":"a0e1b65b59a58543","type":"inject","z":"ccedd8b841592462","name":"hum=70","props":[{"p":"payload"}],"repeat":"","crontab":"","once":false,"onceDelay":0.1,"topic":"","payload":"70","payloadType":"num","x":520,"y":480,"wires":[["b72bbd4e2d9d994b"]]},{"id":"b72bbd4e2d9d994b","type":"function","z":"ccedd8b841592462","name":"Trigger Alarm","func":"var payload=msg.payload;\nvar alarm_flag=context.get(\"alarm_flag\");\nif(typeof alarm_flag==\"undefined\")\nalarm_flag=false;\n\nif (payload>50 && !alarm_flag)\n{\n alarm_flag=true;\n msg.alarm=1;\n context.set(\"alarm_flag\",alarm_flag);\n return msg;\n}\nif (payload<=30 && alarm_flag)\n{\n alarm_flag=false;\n msg.alarm=0;\n context.set(\"alarm_flag\",alarm_flag);\n return msg;\n}\n","outputs":1,"noerr":0,"initialize":"","finalize":"","libs":[],"x":740,"y":460,"wires":[["969b01c3a4b7bfaf","5d4d9ad6f88a621f","614ef9909b2f5e92"]]},{"id":"c0a9345be5e34d5c","type":"comment","z":"ccedd8b841592462","name":"Pruebas para el bloque trigger con 1 variable CORRECTO","info":"","x":850,"y":420,"wires":[]},{"id":"969b01c3a4b7bfaf","type":"debug","z":"ccedd8b841592462","name":"[3] Valor humedad","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"payload","targetType":"msg","statusVal":"","statusType":"auto","x":1110,"y":460,"wires":[]},{"id":"5d4d9ad6f88a621f","type":"function","z":"ccedd8b841592462","name":"","func":"var hum = msg.payload;\nmsg.to = \"asiermartinezsanesteban@gmail.com\";\nmsg.from = \"proyectofiware@gmail.com\";\n\nif(msg.alarm)\n{\n //Al gmail hay que enviar dos campos, msg.topic con el asunto\n//del correo y el propio mensaje, en este caso una string compuesta \n//por diferentes strings \n msg.topic = \"Alerta por humedad elevada\";\n var message = \"Humedad demasiado alta: hum = \";\n \n msg.payload = \"Alerta!! \" + message + hum + \" %\";\n}\nelse\n{\n msg.topic = \"Alerta por humedad apagada\";\n var message = \"Valores de humedad normales: hum = \";\n msg.payload = message + hum + \" %\";\n} \n//Devuelve la cadena msg.payload\nreturn msg;","outputs":1,"noerr":0,"initialize":"","finalize":"","libs":[],"x":920,"y":500,"wires":[["4c9119b12a43b16d"]]},{"id":"614ef9909b2f5e92","type":"ui_gauge","z":"ccedd8b841592462","name":"","group":"929ddc155b83e08f","order":3,"width":0,"height":0,"gtype":"gage","title":"Humidity","label":"%","format":"{{value}}","min":0,"max":"100","colors":["#0080ff","#0062c4","#002f5e"],"seg1":"","seg2":"","diff":false,"className":"","x":900,"y":540,"wires":[]},{"id":"836b2a592f10a6f8","type":"http in","z":"ccedd8b841592462","name":"","url":"alarma","method":"post","upload":false,"swaggerDoc":"","x":90,"y":180,"wires":[["ef6fb3199ecb1937","98a9c0d87046837b","276598acfb43942c"]]},{"id":"ef6fb3199ecb1937","type":"http response","z":"ccedd8b841592462","name":"OK [200]","statusCode":"200","headers":{},"x":240,"y":220,"wires":[]},{"id":"98a9c0d87046837b","type":"debug","z":"ccedd8b841592462","name":"[1] Mensaje OCB","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"payload","targetType":"msg","statusVal":"","statusType":"auto","x":270,"y":140,"wires":[]},{"id":"276598acfb43942c","type":"function","z":"ccedd8b841592462","name":"JSON or URL Encoded","func":"var hum = { payload: msg.payload.data[0].humidity.value };\nvar temp = { payload: msg.payload.data[0].temperature.value };\n\nreturn [temp, hum];","outputs":2,"noerr":0,"initialize":"","finalize":"","libs":[],"x":330,"y":180,"wires":[["d4480d67088a2887","ed3e7b9650301f5a"],["62131c7cad8dc46c","03eb6666197555a3"]]},{"id":"62131c7cad8dc46c","type":"debug","z":"ccedd8b841592462","name":"[3] Valor humedad","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"payload","targetType":"msg","statusVal":"","statusType":"auto","x":550,"y":240,"wires":[]},{"id":"d4480d67088a2887","type":"debug","z":"ccedd8b841592462","name":"[2] Valor temperatura","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"payload","targetType":"msg","statusVal":"","statusType":"auto","x":560,"y":120,"wires":[]},{"id":"03eb6666197555a3","type":"function","z":"ccedd8b841592462","name":"TRIGGER HUM","func":"var payload = msg.payload;\nvar alarm_flag = context.get(\"alarm_flag\");\nif (typeof alarm_flag == \"undefined\")\n alarm_flag = false;\n\nif (payload > 50 && !alarm_flag) {\n alarm_flag = true;\n msg.alarm = 1;\n context.set(\"alarm_flag\", alarm_flag);\n return msg;\n}\nif (payload <= 30 && alarm_flag) {\n alarm_flag = false;\n msg.alarm = 0;\n context.set(\"alarm_flag\", alarm_flag);\n return msg;\n}","outputs":1,"noerr":0,"initialize":"","finalize":"","libs":[],"x":560,"y":200,"wires":[["c8c7e5a760364f53"]]},{"id":"ed3e7b9650301f5a","type":"function","z":"ccedd8b841592462","name":"TRIGGER TEMP","func":"var payload = msg.payload;\nvar alarm_flag = context.get(\"alarm_flag\");\nif (typeof alarm_flag == \"undefined\")\n alarm_flag = false;\n\nif (payload > 25 && !alarm_flag) {\n alarm_flag =

true;\n msg.alarm = 1;\n context.set(\"alarm_flag\", alarm_flag);\n return msg;\n}\nif (payload <= 23 && alarm_flag) {\n alarm_flag = false;\n msg.alarm = 0;\n context.set(\"alarm_flag\", alarm_flag);\n return msg;\n}","outputs":1,"noerr":0,"initialize":"","finalize":"","libs":[],"x":570,"y":160,"wires":[["f824211c3fb48d17"]]},{"id":"c8c7e5a760364f53","type":"function","z":"ccedd8b841592462","name":"INFO MAIL HUM","func":"var hum = msg.payload;\nmsg.to = \"asiermartinezsanesteban@gmail.com\";\nmsg.from = \"proyectofiware@gmail.com\";\n\nif (msg.alarm) {\n //Al gmail hay que enviar dos campos, msg.topic con el asunto\n //del correo y el propio mensaje, en este caso una string \n //compuesta por diferentes strings \n msg.topic = \"Alerta por humedad elevada\";\n var message = \"Humedad demasiado alta: hum = \";\n msg.payload = \"Alerta!! \" + message + hum + \" %\";\n}\nelse {\n msg.topic = \"Alerta por humedad apagada\";\n var message = \"Valores de humedad normales: hum = \";\n msg.payload = message + hum + \" %\";\n}\n//Devuelve la cadena msg.payload\nreturn msg;","outputs":1,"noerr":0,"initialize":"","finalize":"","libs":[],"x":790,"y":200,"wires":[["65431d82334706f8","2abdb8df6905d64b"]]},{"id":"f824211c3fb48d17","type":"function","z":"ccedd8b841592462","name":"INFO MAIL TEMP","func":"var temp = msg.payload;\nmsg.to = \"asiermartinezsanesteban@gmail.com\";\nmsg.from = \"proyectofiware@gmail.com\";\n\nif (msg.alarm) {\n //Al gmail hay que enviar dos campos, msg.topic con el asunto\n //del correo y el propio mensaje, en este caso una string \n //compuesta por diferentes strings \n msg.topic = \"Alerta por temperatura elevada\";\n var message = \"Temperatura demasiado alta: temp = \";\n msg.payload = \"Alerta!! \" + message + temp + \" ºC\";\n}\nelse {\n msg.topic = \"Alerta por temperatura elevada desactivada\";\n var message = \"Valores de temperatura normales: temp = \";\n msg.payload = message + temp + \" ºC\";\n}\n//Devuelve la cadena msg.payload\nreturn msg;","outputs":1,"noerr":0,"initialize":"","finalize":"","libs":[],"x":790,"y":160,"wires":[["65431d82334706f8","2abdb8df6905d64b"]]},{"id":"65431d82334706f8","type":"e-mail","z":"ccedd8b841592462","server":"smtp.gmail.com","port":"465","secure":true,"tls":true,"name":"asiermartinezsanesteban@gmail.com","dname":"gmail notif","x":1010,"y":160,"wires":[]},{"id":"4c9119b12a43b16d","type":"debug","z":"ccedd8b841592462","name":"[4] Cuerpo email","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"payload","targetType":"msg","statusVal":"","statusType":"auto","x":1100,"y":500,"wires":[]},{"id":"95aefcea1f3247ae","type":"twitter out","z":"ccedd8b841592462","twitter":"","name":"Tweet","x":350,"y":700,"wires":[]},{"id":"f183474ee7ecef16","type":"inject","z":"ccedd8b841592462","name":"Publicar un Tweet","props":[{"p":"payload"}],"repeat":"","crontab":"","once":false,"onceDelay":0.1,"topic":"","payload":"Prueba Node-RED","payloadType":"str","x":130,"y":700,"wires":[["95aefcea1f3247ae"]]},{"id":"437d51cf5712186c","type":"comment","z":"ccedd8b841592462","name":"Publicación de Tweet CORRECTO","info":"","x":160,"y":660,"wires":[]},{"id":"1b5849a9dd7ad694","type":"json","z":"ccedd8b841592462","name":"","property":"payload","action":"obj","pretty":true,"x":150,"y":300,"wires":[[]]},{"id":"2abdb8df6905d64b","type":"whin_send","z":"ccedd8b841592462","name":"","auth":"c6bee93baac17d0d","x":1010,"y":200,"wires":[[]]},{"id":"c6bee93baac17d0d","type":"whin_config","name":"whin","apikey":"12f275427amsh4577dc2bae12a24p1ca10ajsn9888b38f43b1"},{"id":"929ddc155b83e08f","type":"ui_group","name":"Default","tab":"d499198df2f5572c","order":1,"disp":true,"width":"6","collapse":false,"className":""},{"id":"d499198df2f5572c","type":"ui_tab","name":"Home","icon":"dashboard","disabled":false,"hidden":false}]

Anexo D. Configuración de archivo *docker-compose.yml*

En este apartado se incluye el código completo para la configuración del archivo *docker-compose.yml*, que ha sido explicado anteriormente parte por parte, para facilitar su manejo.

```
version: "3.4"
services:
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ORION ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  orion:
    image: fiware/orion-ld:1.2.0-PRE-1295
    hostname: orion
    container_name: fiware-orion
    ports:
      - "1026:1026"
    networks:
      - default
    depends_on:
      - mongo-db
    command: -dbhost mongo-db -logLevel DEBUG
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ MONGO-DB ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  mongo-db:
    image: mongo:4.2
    hostname: mongo-db
    container_name: db-mongo
    ports:
      - "27017:27017"
    networks:
      - default
      - perseo
    volumes:
      - ./mongodata:/data/db
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ IOT-AGENT ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  iot-agent:
    image: fiware/iotagent-json:1.26.0
    hostname: iot-agent
    container_name: fiware-iot-agent
    depends_on:
      - mongo-db
    networks:
      - default
    expose:
      - "4041"
      - "7896"
    ports:
      - "4041:4041"
      - "7896:7896"
    environment:
      - "IOTA_CB_HOST=orion"
      - "IOTA_CB_PORT=1026"
      - "IOTA_NORTH_PORT=4041"
      - "IOTA_REGISTRY_TYPE=mongodb"
      - "IOTA_LOG_LEVEL=DEBUG"
```

```
      - "IOTA_TIMESTAMP=true"
      - "IOTA_CB_NGSI_VERSION=v2"
      - "IOTA_AUTOCAST=true"
      - "IOTA_MONGO_HOST=mongo-db"
      - "IOTA_MONGO_PORT=27017"
      - "IOTA_MONGO_DB=iotagent-json"
      - "IOTA_HTTP_PORT=7896"
      - "IOTA_PROVIDER_URL=http://iot-agent:4041
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ DRACO~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  draco:
    image: ging/fiware-draco:1.1.0
    container_name: draco
    depends_on:
      - mysql-db
    environment:
      - "NIFI_WEB_HTTP_PORT=9091"
    ports:
      - "9091:9091"
      - "5050:5050"
    healthcheck:
      test: curl --fail -s http://localhost:9091/nifi-api/system-diagnostics || exit 1
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ MYSQL ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  mysql-db:
    restart: always
    image: mysql:5.7
    hostname: mysql-db
    container_name: db-mysql
    ports:
      - "3306:3306"
    networks:
      - default
    volumes:
      - ./mysqldata:/data/db
    environment:
      - "MYSQL_ROOT_PASSWORD=123"
      - "MYSQL_ROOT_HOST=%"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ NETWORKS & VOLUMES ~~~~~~~~~~~~~~~~~~~~~~~~
networks:
    perseo:
    default:
volumes:
  mongo-db: ~
  mysql-db: ~
```

Para **levantar** los tres contenedores ahora sólo es necesario ejecutar el comando:

sudo docker-compose -p fiware up -d.

Donde:

- ***docker-compose:*** es el nombre del archivo *.yml.*
- ***-p fiware:*** porque el archivo se encuentra dentro de este fichero.
- ***up:*** es la acción de levantar.
- ***-d:*** significa detached, o en segundo plano, para poder seguir usando la consola.

Se puede ejecutar *sudo docker ps* para **comprobar** qué contenedores están funcionando. Cuando se quiera **parar** de correr los contenedores, se ejecuta el comando: *sudo docker-compose down*