Diseño, desarrollo y prototipado de un Gateway M2M multiprotocolo para aplicaciones IoT

Aarón Cancio Gómez
Grado en Ingeniería Informática
Arduino

Antoni Morell Pérez
Pere Tuset Peiró

Junio de 2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Diseño, desarrollo y prototipado de un Gateway M2M multiprotocolo para aplicaciones IoT*
Nombre del autor:	*Aarón Cancio Gómez*
Nombre del consultor/a:	*Antoni Morell Pérez*
Nombre del PRA:	*Pere Tuset Peiró*
Fecha de entrega (mm/aaaa):	*06/2018*
Titulación:	*Grado en Ingeniería Informática*
Área del Trabajo Final:	*Arduino*
Idioma del trabajo:	*Castellano*
Palabras calve	*Arduino, M2M, IoT*

Resumen del Trabajo (máximo 250 palabras):

El presente trabajo trata sobre el diseño, desarrollo y prototipado de un Gateway M2M multiprotocolo para aplicaciones IoT basado en Arduino, Raspberry PI y el uso de software libre. El objetivo que se persigue es permitir la interconexión, comunicación bidireccional y adquisición de información procedente de nodos con diferentes protocolos de comunicación mediante el uso de conectores, gestionar la persistencia de la información en una base de datos local y permitir la interoperabilidad mediante el uso de una API REST y la publicación de los datos en un servicio en la nube.

Abstract (in English, 250 words or less):

The present project is about the design, development and prototyping of a multiprotocol M2M Gateway for IoT applications based on Arduino, Raspberry PI and the use of open source software. The objective is to offer interconnection, bidirectional communication and acquisition of information from nodes that use different communication protocols through the use of connectors, manage the persistence of information in a local database and allow interoperability through the use of a REST API and the publication of data in a cloud service.

AGRADECIMIENTOS

Quiero dar las gracias a los consultores y profesores que tanto me han aportado durante estos años, porque gracias a su dedicación y verdadera pasión por la docencia han conseguido ir empujándome a través de un camino no exento de dificultades hasta llegar al que parece ser el final de este viaje apasionante.

También quiero darle las gracias de corazón a mi madre por su amor, apoyo incondicional y sus increíbles esfuerzos por darme la mejor educación posible. Le agradezco a mi hermano su cariño, su apoyo y su infinita fe en mí. Finalmente le doy las gracias a mi pareja por su amor, su amistad, su increíble paciencia y esa mirada que hace que me sienta capaz de cualquier cosa.

ÍNDICE DE CONTENIDOS

1

INTRODUCCIÓN

1.1 ANTECEDENTES

Los sistemas M2M han sido tradicionalmente sistemas aislados, redes acotadas de telemetría a disposición de una compañía y cuya información y su explotación respondían a una finalidad muy concreta. El IoT, en cambio, supone una ampliación dramática del contexto y convierte la red acotada en la red universal, Internet, llevando la información a la nube para explotar los datos para el uso de nuevas aplicaciones. La idoneidad de una u otra filosofía o la necesidad de uno u otro contexto dependerán de las necesidades de cada caso.

Tanto a nivel M2M como a nivel IoT, existen una plétora creciente de dispositivos, estándares, protocolos y aplicaciones con diferentes enfoques o para cubrir diferentes necesidades. Esta cantidad de opciones es idónea para tener una solución apta para cualquier escenario que se nos pueda plantear pero también hace que se presente un doble problema en estos sistemas: la interconexión y la interoperabilidad.

Ante una situación en la que se debe optar por un contexto concreto y ceñirse a un protocolo o aplicación para que todo funcione como debe, parece oportuno tener un sistema que ofrezca un

contexto mixto, a nivel local o de Internet, y que permita la conexión con algunos de los protocolos más populares. Esta es el germen de la idea para la creación del sistema a diseñar y desarrollar en este proyecto.

1.2 OBJETIVOS Y MOTIVACIÓN

El objetivo principal del proyecto es diseñar, desarrollar y prototipar un sistema basado en tecnologías libres y abiertas que permita la comunicación e interconexión con múltiples dispositivos y protocolos, y ofrezca otras capas de servicio como persistencia e interoperabilidad. Como objetivos adicionales se persigue que la arquitectura sea modular y el funcionamiento sea transparente de manera que la incorporación de nuevos protocolos, dispositivos o servicios sea relativamente sencilla.

Los objetivos a nivel personal están intrínsecamente vinculados con la motivación: este proyecto es relativamente ambicioso en cuanto al número de tecnologías, lenguajes y distintos tipos de hardware y supone un desafío importante teniendo en cuenta que no estoy familiarizado de antemano con todas ellas. El objetivo es profundizar en el conocimiento de estas tecnologías y lenguajes y conseguir que trabajen entre ellas. El reto a nivel de aprendizaje y la posibilidad de obtener un producto final plenamente funcional son las dos motivaciones más importantes.

1.3 ENFOQUE Y METODOLOGÍA

Al comenzar el proyecto se llevará a cabo un proceso de investigación de los diferentes protocolos y tecnologías existentes así como de soluciones actuales al problema presentado. Tras recopilar la información necesaria para la toma de decisiones respecto a la solución que se va a proponer se elaborará un diseño conceptual a alto nivel y se llevará a cabo la planificación del trabajo. Tras ello se trasladará el diseño a una arquitectura concreta que emplee las tecnologías y componentes disponibles y se definirá un entorno de prototipado concreto. Con el diseño final realizado se dividirá el trabajo en las diferentes tareas a realizar: integración del hardware de los nodos y el Gateway, desarrollo del software de adquisición y comunicación de los nodos y el Gateway y de las capas de servicio de este último. Al finalizar cada tarea se llevarán a cabo una serie de pruebas de verificación antes de avanzar a la siguiente tarea y al finalizar el proyecto se llevará a cabo una batería completa de pruebas de todo el sistema explotando todas las funcionalidades implementadas.

1.4 PLANIFICACIÓN DEL TRABAJO

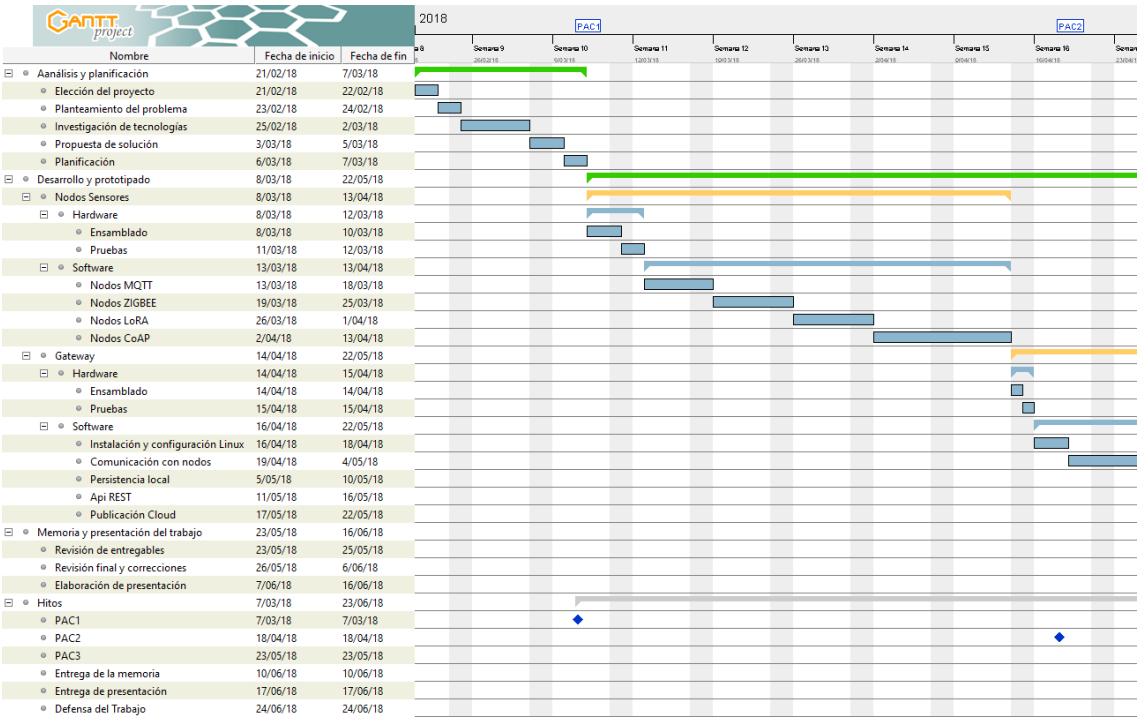


Ilustración 1: Planificación del trabajo, primera mitad del cuatrimestre

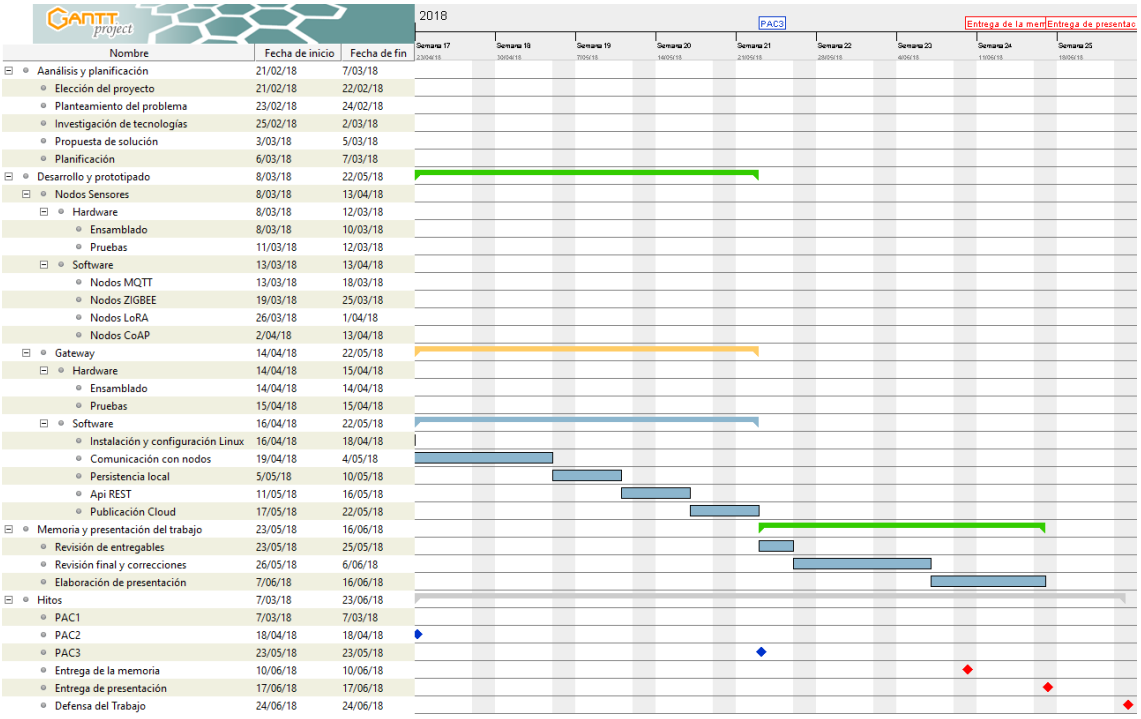


Ilustración 2: Planificación del trabajo, segunda mitad del cuatrimestre

1.5 PRODUCTOS OBTENIDOS

Al finalizar el proyecto se espera obtener un Gateway plenamente operativo con todas las características propuestas. Junto con el Gateway se obtendrá un módulo de Python que se encargue de toda la gestión del Gateway. También se obtendrá un archivo SQL para la creación de la base de datos y un script PHP encargado de gestionar la API RESTfull. Además de esto se obtendrá una librería en C++ para Arduino integrable en cualquier proyecto que permita de menara sencilla incorporar a la red gestionada por el Gateway cualquier dispositivo basado en Arduino.

1.6 ESTRUCTURA DE LA MEMORIA

La memoria se ha estructurado en diferentes capítulos que cubren conceptual y cronológicamente todo el ciclo de vida del proyecto, desde la identificación del problema y la concepción de la solución hasta el producto final

Introducción: en este apartado se incluye la información necesaria para introducir el contenido del proyecto, darle contexto y ponerlo en perspectiva. Se plantean aspectos básicos como los objetivos, la motivación o la planificación.

Tecnología y arquitectura: en esta sección se hace una breve descripción de los entornos, tecnologías y componentes que forman parte del proyecto y que son, en definitiva, objeto del problema que se plantes. Posteriormente se expone la solución conceptual propuesta basada en capas de servicio, se esboza el diseño tecnológico de la misma y se proponen los componentes básicos que se utilizarán para su implementación

Desarrollo y prototipado: en este capítulo se describe el proceso de implementación de la solución con los nodos del entorno de prototipado en primer lugar y el Gateway después. Se muestra el uso, montaje e interconexión del hardware elegido así como la utilización de librarías y herramientas para el desarrollo de las diferentes partes que componen el software del sistema. Finalmente se evidencian las pruebas realizadas al producto final.

Análisis de resultados: aquí se recogen las conclusiones y valoraciones, se analiza el producto final y su viabilidad y se expone lo que se ha aprendido a lo largo de la realización del proyecto.

2

TECNOLOGÍA Y ARQUITECTURA

2.1 INTERNET OF THINGS Y REDES M2M

A pesar de que el Internet de las Cosas (Internet of Things) y las redes M2M (Machine To Machine) son a conceptos que se ven juntos muchas veces, hay que dejar claro que son cosas distintas. Las redes M2M son redes enfocadas a la comunicación entre máquinas o dispositivos habitualmente denominados nodos, sin un usuario detrás en ningún extremo de la comunicación. Desde este punto de vista podríamos decir que el Internet de las cosas es una forma de red M2M y no sería totalmente incierto. Sin embargo el paradigma de ambos conceptos difiere de base. Las redes M2M son algo que lleva mucho tiempo entre nosotros, especialmente a nivel Industrial y su enfoque siempre ha sido compartir información, normalmente lecturas de sensores y mensajes de actuadores, entre distintos nodos de una red, pero generalmente de forma local (al menos a nivel lógico) y restringida, para su consumo o procesado de forma acotada. El Internet de las cosas, sin embargo, da una vuelta de tuerca al concepto ampliando los horizontes del mismo a nivel de diversidad, espacialidad, y disponibilidad. Su razón de ser es que la mayoría de dispositivos electrónicos son susceptibles de convertirse en un nodo de la red más grande de todas, Internet. Estos nodos son capaces de ofrecer información de forma masiva que normalmente se pondrá a disposición de todo el mundo y será accesible públicamente. Esto abre las puertas a nuevas formas de explotar esta información y al surgimiento de nuevas áreas como la Industria 4.0 o las Smart Cities.

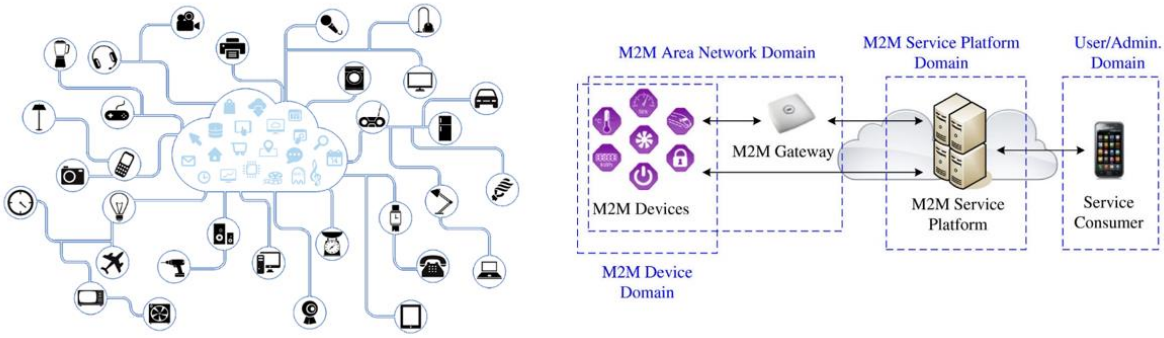


Ilustración 3: Internet Of Things vs Redes Machine 2 Machine3

2.1.1 NODOS

Un nodo, de forma muy básica, es un sistema capaz de recolectar información de diversa índole y comunicársela a otro nodo de la red. Esta sería una definición de mínimos pero es habitual encontrar que estos nodos tienen, en realidad, otras características y funciones útiles como la capacidad de procesar y/o almacenar de diversas formas la información que recolecta, la capacidad de interactuar con el entorno por medio de actuadores o la comunicación bidireccional con otros sistemas de la red. La complejidad de un nodo puede escalar hasta un sistema computacional completo aunque lo más habitual es que sean sistemas pequeños, simples, con limitada capacidad de cálculo, de bajo consumo energético y, algo muy relevante, de bajo coste, ya que es común encontrar redes con decenas e incluso centenares de nodos. Los componentes básicos de un nodo son:

- **Controlador**: habitualmente en formato de placa integrada que incluye CPU, memoria y buses de diversas características
- **Transceptor**: el dispositivo que sirve para que el controlador se pueda conectar a una red y comunicarse con otros nodos, normalmente de tipo inalámbrico.
- **Sensores**: encargados de recolectar la información, se conectan al controlador mediante uno de los buses disponibles. Existen una infinitud de sensores de todo tipo de características y capacidades.
- **Actuadores**: son dispositivos capaces de transformar la señal eléctrica del controlador en la activación de un proceso, es decir, hacen que se generen cambios en el entorno promovidos desde el controlador de manera automática. Como con los sensores, también hay de diversos tipos y capacidades

- **Fuente de alimentación**: debido al pequeño consumo que suelen tener estos dispositivos es normal que estén alimentados por baterías que pueden llegar a durar meses y años. También es común empezar a encontrar alimentación fotoeléctrica y la combinación de ambas.

Para el proyecto todos los nodos del entorno de prototipado están basados en diferentes modelos de placas Arduino. Todos ellos cuentan con un transceptor inalámbrico, un sensor, un transceptor y una fuente de alimentación.

2.1.2 GATEWAY

Un Gateway es un dispositivo físico (también puede ser un programa de software) que sirve como punto de conexión entre la nube y los controladores, sensores y dispositivos inteligentes. Los Gateway también pueden denominarse también puertas de enlace inteligente o niveles de control. Todos los datos que se trasladan a la nube o vienen de ella pasan por el Gateway. Un Gateway también proporciona un lugar para pre-procesar los datos generados por los sensores en el borde antes de enviarlos a la nube minimizando el volumen de datos enviados a la nube, así como el impacto de los tiempos de respuesta y el coste de transmisión de red.El Gateway también puede proporcionar seguridad adicional a la red y los datos debido a que al gestionar la información en ambas direcciones puede protegerlos de filtraciones y ataques externos maliciosos.

2.1.3 PROTOCOLOS DE COMUNICACIÓN

2.1.3.1 WIFI

WIFI siglas de Wireless Fidelity o fidelidad inalámbrica, es una tecnología para la interconexión de forma inalámbrica de dispositivos electrónicos. El término se acuñó en 1999 por una asociación (sin ánimo de lucro) de varias empresas para utilizar una nueva tecnología de red inalámbrica, la WIFI Alliance. Está tecnología está regulada por la especificación para redes inalámbricas IEEE 802.11 y en él se describen las normas a seguir por los fabricantes de dispositivos inalámbricos para que todos ellos sean compatibles. A lo largo de los años se han ido creado diferentes estándares para la optimización del ancho de banda o para especificar componentes mejores para asegurar la compatibilidad y la seguridad. En el estándar de la IEEE se establecen las capas OSI cubiertas por la arquitectura WIFI y estas son la capa de enlace de datos y la capa física.

2.1.3.2 MQTT

Este protocolo fue inventado por Arlen Nipper y Andy Stanford-Clark en 1999. Sus siglas vienen del inglés Message Queue Telemetry Transport o transporte de telemetría de cola de mensajes. Es un protocolo de mensajería de publicación y suscripción simple, basado en el principio cliente/servidor. Es muy ligero, especialmente diseñado para dispositivos con restricciones y redes de bajo ancho de banda, alta latencia o poco confiables. Sus principios de diseño buscan minimizar el ancho de banda de la red y los requisitos de recursos del dispositivo a la que intenta garantizar la fiabilidad con cierto grado de seguridad de la entrega. Es ideal para la comunicación machine-to-machine (M2M), para el internet de las cosas (IoT), para la comunicación de sensores y se puede utilizar en la mayoría de dispositivos empotrados con pocos recursos (CPU, RAM,..). Desde 2014, existe un estándar MQTT a cargo del organismo OASIS y su versión más actual es la 3.1.1.MQTT se sitúa en las capas superiores (aplicación) del modelo OSI apoyándose normalmente en TCP/IP lo que hace que los participantes de la aplicación MQTT deban tener una pila TCP/IP.

Los actores que participan en una red MQTT son los clientes y los servidores (denominados *brokers*). Los clientes pueden recolectar información del medio como los sensores y sistemas embebidos o ser aplicaciones ejecutando librerías MQTTA que interactúan con los datos. Pueden ser publicadores y subscriptores de mensajes, a la vez que pueden controlar y configurar con comandos los sensores a su cargo si son nodos de sensores. Siempre se conectan con un tercer participante, el *broker* de mensajes.

El broker es un servicio que implementa el protocolo y que establece la comunicación a nivel de aplicación entre los clientes. Hace de intermediario entre los productores y los consumidores de información y es el responsable de recibir los mensajes de tipos específicos (denominados *topics*). MQTT se basa en que alguien (aplicación o nodo) publica un mensaje con el identificador de un *topic* específico y el *broker* distribuye el mensaje a todos los clientes (aplicaciones o dispositivos) que le constan como subscritos a ese *topic*. Esos clientes reciben y consumen los datos de esos mensajes. Con este protocolo la única dirección IP y puerto que se necesita saber es la del *broker* de mensajes.

Para garantizar la entrega de mensajes de forma correcta a los nodos, MQTT implementa hasta 3 niveles de *QoS* y cuanto más alto es el nivel más fiable es la transmisión, pero hay mayor consumo de ancho de banda y mayor latencia. El servidor también es capaz de mantener mensajes después de haber sido enviados a los nodos subscritos de manera que, si hay nuevas subscripciones a los *topics* de esos mensajes, estos se envían a los nuevos clientes.

2.1.3.3 COAP

El CoAP (*Constrained Application Protocol*) fue creado por la Internet *Engineering Task Force* (ITEF) y definido en el estándar RFC 7252 que especifica como los dispositivos de cómputo de bajo consumo pueden funcionar en IoT. Es un protocolo a nivel de aplicación diseñado para nodos y redes con recursos limitados y para aplicaciones M2M ofreciendo un funcionamiento similar al HTTP. Se basa en la arquitectura REST (*representational state transfer*) en la que los clientes envían peticiones a los servidores y éstos responden con una representación del recurso solicitado.

Sus características principales son que permite el descubrimiento de recursos alojados en los diferentes servidores de la red, es compatible con arquitecturas basadas en web, tiene un soporte a las comunicaciones multicast e intercambio asíncrono de mensaje. En el modelo de interacción de CoAP los mensajes se envían de forma asíncrona usando el protocolo UDP e implementa mensajes confirmables (ACK sobre UDP), pero no implementa niveles de calidad de servicio (QoS), por lo que las aplicaciones son las que deben implementar este tipo de mecanismos.

Los mensajes de este protocolo tienen la misma estructura, una cabecera de tamaño fijo, un número variable de opciones y un payload (siendo no obligatorios estos dos últimos). Hay cuatro tipos de mensajes: Confirmable (CON) que requieren confirmación de recepción (ACK), No confirmable (NON) cuando no es imperativa la confirmación usados en aplicaciones con transmisiones regulares, Acknowledgement (ACK) para la confirmación de un CON o en respuesta a una petición de tipo GET, y RESET (RST) que es un mensaje dado en respuesta a un mensaje CON o NON recibido en el que el receptor no es capaz de procesar aunque el contenido sea correcto (suele ocurrir cuando algún de los dispositivos se reinicia).

Cuando se envía una petición a través de un mensaje CON o NON se hace para ejecutar un método sobre un recurso. Los métodos soportados son GET, POST, PUT y DELETE. El tipo de respuesta (reflejada en el campo code del mensaje) puede ser de tres clases: Success, cuando la petición fue recibida, entendida y aceptada correctamente; Client Error, cuando la petición contiene una sintaxis errónea o no puede ser correctamente tratada por el servidor; Server Error, cuando el servidor puede tratar una petición aparentemente correcta.

2.1.3.4 ZIGBEE

ZigBee es una especificación basada en IEEE 802.15.4 para un conjunto de protocolos de comunicación de alto nivel utilizados para crear redes de área personal con radios digitales pequeños y de baja potencia con usos como la domótica, la recopilación de datos de dispositivos médicos y

otras necesidades de bajo ancho de banda y baja potencia. Está diseñado para proyectos a pequeña escala que necesitan conexión inalámbrica, por lo que se puede decir que ZigBee es una red ad-hoc inalámbrica de baja potencia, baja velocidad de datos y de proximidad. El estándar fue diseñado por la ZigBee Alliance para aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximizar la vida útil de las baterías de los dispositivos.

Está constituido por diferentes capas al igual que el modelo OSI. La capa de más bajo nivel es la capa física (PHY), y la siguiente es la capa de acceso al medio (MAC), capas descritas en el estándar IEEE 802.15.4–2003 que define el nivel físico y el control de acceso al medio de redes inalámbricas de área personal con tasas bajas de transmisión de datos. Este estándar trabaja sobre las bandas ISM de uso no regulado, dónde se definen hasta 16 canales en el rango de 2.4 GHz. La siguiente capa es la capa de red (NWK) cuyo principal objetivo es el de permitir el correcto uso del subnivel MAC y ofrecer una interfaz adecuada para su uso por parte de la capa de aplicación. En esta capa se encuentran los métodos para iniciar la red, unirse a ella, enrutar paquetes dirigidos a otros nodos, proporcionar medios que garanticen la entrega del paquete al destinatario final y filtrar cifrar y autenticar paquetes recibidos. Es en esta capa donde se implementan las distintas topologías de red que soporta (árbol, estrella y *mesh network*). A continuación, se encuentra la capa de soporte a la aplicación, responsable de mantener el rol del nodo en la red, de filtrar paquetes a nivel de aplicación, de mantener la relación de grupos y dispositivos con los que la aplicación interactúa y de simplificar el envío de datos a los nodos de la red. La ZigBee Alliance es la que define la capa de red y la de soporte a la aplicación. Por último, tenemos la capa de aplicación que es el nivel más alto definido por la especificación. Es la interfaz entre el nodo ZigBee y sus usuarios. Aquí nos encontramos con la mayor parte de los componentes definidos por la especificación como los objetos de dispositivo ZigBee (ZDO), los procedimientos de control y los objetos de aplicación definidos por cada uno de los fabricantes.

Según su papel en la red, tenemos tres tipos de dispositivos ZigBee:

- **Coordinador** (*Coordinator*, ZC). Debe existir uno por red, y se encarga de controlar la red y los caminos que deben seguir los dispositivos para conectarse entre ellos. Es el primer dispositivo en la red y almacena toda la información de esta debiendo estar siempre alimentado.
- **Router** (*Router*, ZR). Es el encargado de interconectar dispositivos separados en la topología de la red, además de ofrecer un nivel de aplicación para la ejecución de código de usuario.
- **Dispositivo final** (*End Device*, ZED). Este dispositivo se comunica con su nodo padre (el coordinador o un *router*), pero no puede transmitir información destinada a otros dispositivos.

Puede estar "dormido" la mayor parte del tiempo, aumentando la vida media de sus baterías. Sus requerimientos de memoria son mínimos por lo que son los más baratos.

2.1.3.5 LORA

LoRa es una tecnología patentada de modulación de radio propiedad de Semtech Corporation. LoRa proporciona conectividad inalámbrica de largo alcance mediante el uso de varias técnicas de espectro expandido. LoRa es solo tecnología de capa física/de enlace. Por tanto, es una solución cerrada y patentada que permite enviar datos a velocidades extremadamente bajas a rangos extremadamente largos. Sus características principales es que tiene una alta tolerancia a las interferencias, una alta sensibilidad para recibir datos, un bajo consumo, un gran alcance de 10 a 20km, una baja transferencia de datos y sus frecuencias de trabajos son 868Mhz en Europa, 915 en América y 433 en Asia. Por todo esto, esta tecnología es buena para conexiones a grandes distancias, para lugares con poca cobertura móvil, para redes privadas de sensores y actuadores.

LoRaWAN es un protocolo de red que utiliza la tecnología Lora para administrar y comunicar dispositivos LoRa. Al incluir la capa de red, permite enviar la información a cualquier estación base que ya esté conectada a una plataforma en la nube. Se compone de dos partes principalmente, los Gateways y los nodos. Los Gateways se encargan de recibir y enviar la información de los nodos que son los dispositivos finales. Sus características principales es que tiene topología en estrella, encriptación AES 128, soporto para tres clases de nodos, administración de dispositivos, redes públicas y privadas y baja transferencia de datos.

<table>
<tr><th>CAPA OSI</th><th>LoRa</th><th>ZigBEE</th><th>CoAP</th><th>MQTT</th></tr>
<tr><td>APLICACIÓN</td><td colspan="4">Aplicación</td></tr>
<tr><td>PRESENTACIÓN</td><td rowspan="4">LoRaWan</td><td rowspan="4">ZigBee</td><td rowspan="2">CoAP</td><td rowspan="2">MQTT</td></tr>
<tr><td>SESIÓN</td></tr>
<tr><td>TRANSPORTE</td><td>UDP</td><td>TCP</td></tr>
<tr><td>RED</td><td colspan="2">IPv4, IPv6</td></tr>
<tr><td>ENLACE A DATOS</td><td rowspan="2">IEEE 802.15 4g</td><td rowspan="2">IEEE 802.15 4</td><td colspan="2">MAC</td></tr>
<tr><td>FÍSICA</td><td colspan="2">IEEE 802.11</td></tr>
</table>

Tabla 1: Modelo OSI en capas de LoRa, ZigBEE, CoAP y MQTT

2.2 ESPECIFICACIONES Y REQUISITOS

Como se ha expuesto previamente de forma no exhaustiva, el objetivo del proyecto es crear una combinación de hardware y software que actúe como Gateway multiprotocolo para redes de nodos basados en Arduino y que a su vez ofrezca ciertos servicios adicionales. Bajando al detalle de las especificaciones podemos precisar los siguientes requisitos que se han de implementar en la solución:

REQUISITOS FUNCIONALES

- El sistema ha de implementar una ontología que identifique a las entidades de manera jerárquica y univoca dentro de la misma red.
- La comunicación entre Gateway y nodos ha de tener carácter bidireccional y ha poder establecerse mediante diferentes interfaces y protocolos
- Los nodos y el Gateway tienen que poder llevar a cabo las mismas acciones independientemente de su protocolo o modelo de comunicación (publicación / suscripción, cliente / servidor…)
    - Los nodos deben poder informar periódicamente al Gateway del valor de uno o varios parámetros, han de poder describirse ante el Gateway y también poder responder a peticiones de lectura o asignación procedentes del Gateway.
    - El Gateway debe poder descubrir los nodos conectados, ha de poder recibir las publicaciones periódicas y también poder enviar peticiones de lectura o asignación a los nodos.
- Todos los mensajes que se intercambien en el sistema deben tener formato JSON
- Se debe poder almacenar las lecturas publicadas así como la estructura jerárquica de la red en una base de datos relacional donde permanecerán para consumo frecuente durante un periodo de tiempo acotado.
- Las lecturas publicadas por los nodos también se han de almacenar en una base de datos no relacional sin periodo límite.
- Se ha de proporcionar una API RESTfull con capacidad para leer y actualizar entidades que actúe de interfaz única sobre el Gateway y los nodos.
- Se han de poder publicar las lecturas un servicio en la nube especializado en IoT que ofrezca capacidades de visualización
- específicas.

REQUISITOS NO FUNCIONALES

- La incorporación de un nuevo nodo a la red no ha de requerir de ninguna configuración en el Gateway y la comunicación se debe establecer de forma automática.
- Se han te utilizar tecnologías y software con licencias libres y gratuitas.
- El Gateway ha de ser compacto y tener un consumo de energía contenido

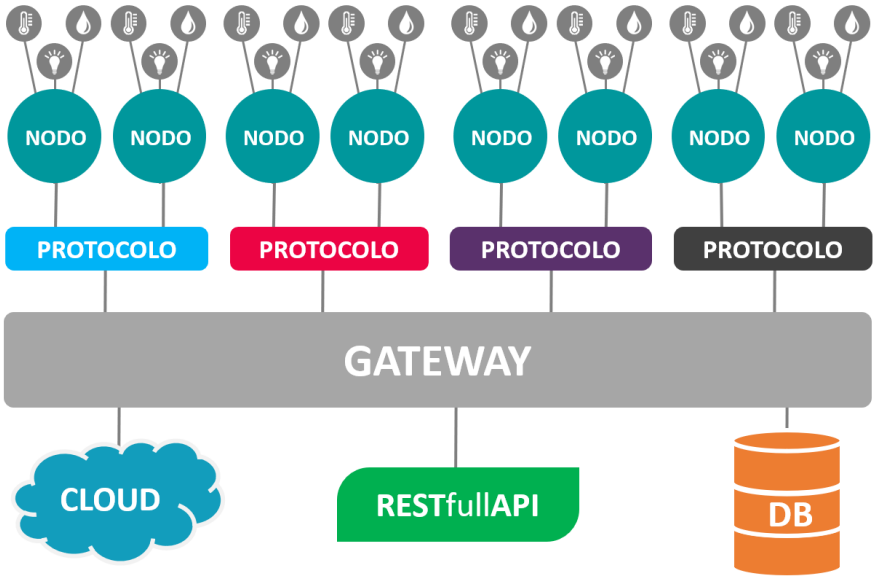


Ilustración 4: Diagrama conceptual del sistema

2.3 DISEÑO CONCEPTUAL

La pieza central de la solución que hemos de diseñar es el Gateway. Para cumplir con los requisitos el Gateway tiene que poder contar con diferentes interfaces de comunicaciones y usar varios protocolos así como implementar diferentes capas de servicios y todo ello en un tamaño reducido. Esto nos hace decantarnos por una SBC (ordenador de placa reducida) ya que, a priori, nos permitirá alcanzar todos los objetivos marcados. Existen muchos modelos de SBC en el mercado: Arduino Yun, Beaglebone Black, Intel Galileo, Hummingboard, ODroid, Orange Pi y Raspberry Pi entre muchas otras. Debido al soporte existente, su precio contenido y a la gran comunidad que existe detrás, optamos por usar la Raspberry Pi.

Para el entorno de prototipado se utilizarán dos nodos por cada protocolo que queramos implementar, en este caso ZigBee, LoRa, MQTT y CoAP. Esto hace un total de ocho nodos, todos ellos conformados por diferentes placas Arduino y componentes e interfaces adicionales. La elección del número de nodos se debe a que probar simultáneamente dos nodos por protocolo parece el mínimo

para garantizar la interoperabilidad y adecuada concurrencia del sistema. La idea es que el Gateway pueda escalar a un número mucho mayor de nodos por lo que para que el funcionamiento se evalúe correctamente y se pueda considerar que es un sistema viable los ocho nodos han de poder trabajar simultáneamente en el dominio sin que se produzca ningún problema ni suponga una carga significativa para el Gateway.

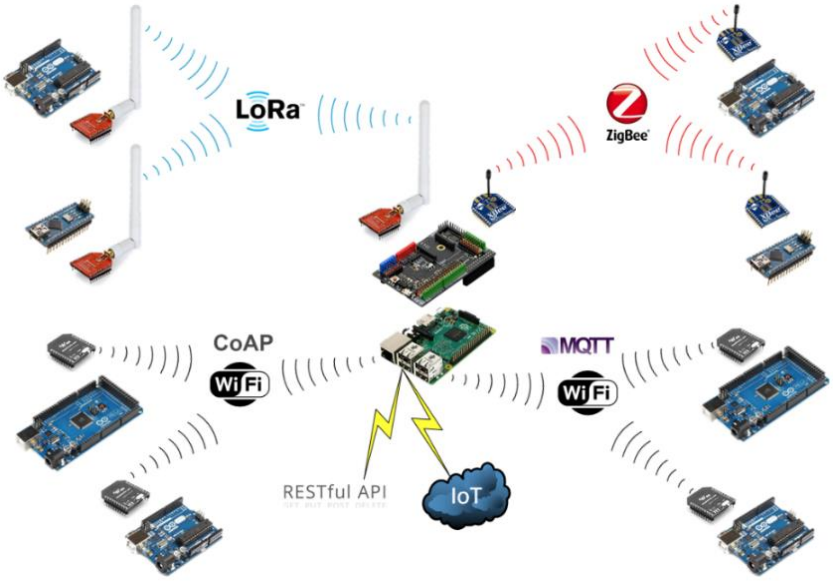


Ilustración 5: representación gráfica de la red de nodos y Gateway

2.3.1 ONTOLOGÍA Y JERARQUÍA DE ENTIDADES

Uno de los puntos más importantes del sistema consiste en poder identificar cada una de sus entidades de manera unívoca en todo el dominio y que esto se produzca independientemente del protocolo o formato de comunicación subyacente. La complicación de conseguir este objetivo radica en que algunos de los protocolos no implementan ningún tipio de etiquetado y se comportan poco más o menos como interfaces serie remotas y otros, sin embargo, implementan las capas de sesión y presentación con lo que ya tienen un sistema de identificación propio. La solución por la que optamos es crear una jerarquía inspirada por los protocolos MQTT y CoAP pero con una estructura más restrictiva. De esta forma podemos utilizar la ontología subyacente de estos protocolos para sustentar la nuestra sin demasiadas complicaciones y simplificamos su implementación sobre el resto de protocolos a la par que podemos traducirla a la API REST casi de manera automática. De esta forma tendremos tres entidades jerárquicas que, separadas por barras y combinadas, generan un tópico (término prestado también de MQTT) único en el dominio. Los tópicos son las entidades sobre las que se harán las operaciones de lectura y escritura. La jerarquía de entidades que definimos queda de la siguiente manera:

- **LOCALIZACIÓN**: identificación o agrupación física o lógica donde se encuentra el nodo. Debe ser única dentro del dominio.
- **NOMBRE DEL NODO**: nombre que permite identificar al nodo. Debe ser único dentro de la misma localización.
- **PARÁMETRO**: identificador del sensor, actuador o registro dentro un nodo. Debe ser único dentro del mismo nodo.

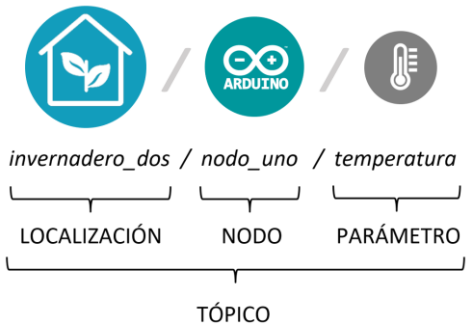


Ilustración 6: Jerarquía de entidades del dominio

2.3.2 COMUNICACIÓN ENTRE NODOS Y GATEWAY

2.3.2.1 ACCIONES, COMANDOS Y PASO DE MENSAJES

Las especificaciones respecto a las acciones que deben poder a llevar a cabo los nodos y el Gateway están muy definidas e implican una suerte de combinación entre el modelo de comunicación de publicación-suscripción (*publish-subscribe)* y el modelo de solicitud-respuesta (*request-response).* La implementación de este modelo mixto se realizará de diferentes formas en función del protocolo de comunicación subyacente.

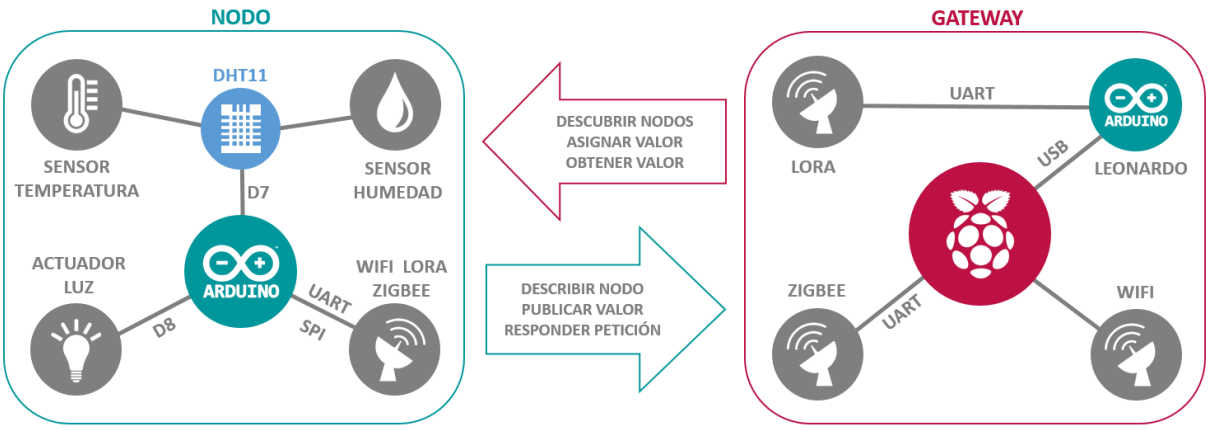


Ilustración 7: Estructura de intercambio de mensajes nodo <-> Gateway

PUBLICACIÓN - SUSCRIPCIÓN: cuando el nodo se conecta al dominio o cuando recibe una solicitud de tipo *DISCOVERY* envía un mensaje al Gateway de tipo *DESCRIBE* en el que reporta todos los tópicos que tiene disponibles. Automáticamente el Gateway queda "suscrito" a todos ellos. A partir de ese momento el nodo "publica" las lecturas de los parámetros mediante un mensaje de tipo *PUBLISH* sin esperar ningún tipo de confirmación por parte del Gateway.

SOLICITUD - RESPUESTA: el Gateway puede enviar dos tipos de solicitudes a los nodos conectados al dominio: una petición de información del valor de un parámetro mediante una mensaje tipo *GET* o una petición de asignación de un valor a un parámetro mediante un mensaje tipo *SET*. En ambos casos el Gateway espera una respuesta del nodo que se materializará como un mensaje de tipo *RESPOND* con el valor del parámetro solicitado o de tipo *ACK* informando del éxito o fracaso de la asignación.

ACCIÓN	ORIGEN	DESTINO	DESCRIPCCIÓN
DISCOVER	GATEWAY	NODO	*El Gateway emite un multicast solicitando a todos los nodos conectados que se describan*
DESCRIBE	NODO	GATEWAY	*El nodo envía al Gateway su localización, su nombre y los sensores y actuadores que tiene*
PUBLISH	NODO	GATEWAY	*El nodo informa al Gateway de la lectura o valor de un sensor o el status de un actuador (best effort delivery)*
SET	GATEWAY	NODO	*El Gateway solicita a un nodo que asigne un valor a un parámetro (actuador o registro)*
ACK	NODO	GATEWAY	*El nodo responde a una solicitud SET informando del éxito o fracaso de la solicitud*
GET	GATEWAY	NODO	*El Gateway solicita a un nodo que le informe de la lectura de un parámetro (sensor o estado de un actuador)*
RESPOND	NODO	GATEWAY	*El nodo informa al Gateway de la lectura de un sensor o el status de un actuador que ha solicitado mediante GET*

Tabla 2: Tipologías de mensajes nodo <-> Gateway

Los mensajes que se intercambiarán nodos y Gateway estarán conformados por una cadena en formato JSON con hasta cuatro campos que estarán presentes o no en función del protocolo y la acción. Los campos que componen el mensaje son autodescriptivos como se puede ver a continuación:

- **COMMAND**: contiene la acción que se pretende llevar a cabo
- **TOPIC**: contiene el tópico objeto del mensaje
- **VALUE**: contiene el valor del tópico o de la respuesta del mensaje
- **ID**: contiene un identificador único del mensaje

ZIGBEE Y LORA	MQTT Y COAP

<pre>{
"COMMAND":"RESPOND",
"TOPIC":"GREENHOUSE/TWO/HUMIDITY",
"VALUE": 65,
"ID":"1D462Y530983G2"
}</pre>
	<pre>{
	"VALUE": 65,
	"ID":"1D462Y530983G2"
	}</pre>

Tabla 3: estructura de mensajes en formato JSON

En el caso de los protocolos MQTT y COAP, ellos tienen mecanismos propios del protocolo que podemos reutilizar para identificar el comando y el tópico por lo que no es necesario incluirlos en el propio mensaje. Esta implementación debe estar presente de manera similar tanto en los nodos como en el Gateway.

La comunicación entre los nodos y el Gateway se llevará a cabo sobre los cuatro protocolos elegidos y mediante diferentes interfaces de comunicaciones. Transferir el modelo de intercambio de mensajes e implementar las acciones requeridas sobre los diferentes protocolos implica hacer uso de los mecanismos y el modelo de comunicación que estos utilizan para construir sobre ellos el nuevo modelo común propuesto que convierta en transparente el protocolo que lo sustenta. La aproximación que se ha elegido es distinta para cada protocolo salvo para LoRa y ZigBee que, al carecer de mecanismos utilizables en las capas superiores, pueden compartir la mayor parte de la implementación

2.3.2.2 PROTOCOLOS LORA Y ZIGBEE

Tanto LoRa como ZigBee tienen un modelo de comunicación de tipo solicitud – respuesta. En ambos protocolos, la carga o *payload* contendrá el mensaje con la estructura completa codificada en JSON. De esta manera tenemos en el mensaje toda la información necesaria para procesar las diferentes acciones.

2.3.2.3 PROTOCOLO MQTT

El protocolo MQTT tiene un modelo de comunicación de tipo publicación-suscripción lo que implica que haya que encajar las acciones de tipo solicitud-respuesta dentro de este modelo. Para ello se han de establecer una serie de tópicos conocidos por los nodos y por el Gateway que sirvan para identificar las acciones a realizar. Se utilizará un tópico para la petición y otro para la respuesta. El destinatario de la petición se suscribirá al tópico de la acción y el que origine la petición, fundamentalmente el Gateway, se suscribirá al tópico de respuesta de esa acción.

ACCIÓN	TOPICO	SUSCRIBE	PUBLICA
DISCOVER	***NODES/DISCOVER***	*Nodo*	*Gateway*
DESCRIBE	***NODES/DESCRIBE***	*Gateway*	*Nodo*
PUBLISH	*LOCATION/NODE/SENSOR*	*Gateway*	*Nodo*
SET	*LOCATION/NODE/ACTUATOR/**SET***	*Nodo*	*Gateway*
ACK	*LOCATION/NODE/ACTUATOR/**ACK***	*Gateway*	*Nodo*
GET	*LOCATION/NODE/SENSOR/**GET***	*Nodo*	*Gateway*
RESPOND	*LOCATION/NODE/SENSOR/**RESPOND***	*Gateway*	*Nodo*

Tabla 4: Adaptación del modelo publicación-suscripción a petición-respuesta en MQTT

2.3.2.4 PROTOCOLO COAP

El protocolo CoAP tiene un modelo de comunicación de tipo solicitud – respuesta. En este protocolo se establecen una serie de recursos que podemos asimilar con los tópicos y las acciones que se llevan a cabo sobre ellos que están marcadas por el método de la petición (GET, PUT, POST…). Estas peticiones son equiparables a las acciones y comandos que se ha decidido implementar con lo que la carga del mensaje no tendrá que contener está información.

ACCIÓN	RECURSO	MÉTODO	ORIGEN	DESTINO
DISCOVER	*/.WELL-KNOWN*	*GET*	*Gateway*	*Nodo*
DESCRIBE	*LOCATION/NODE/SENSOR*	*POST*	*Nodo*	*Gateway*
PUBLISH	*LOCATION/NODE/SENSOR*	*PUT*	*Nodo*	*Gateway*
SET	*LOCATION/NODE/ACTUATOR/*	*PUT*	*Gateway*	*Nodo*
ACK	*LOCATION/NODE/ACTUATOR/**ACK***	*PUT*	*Nodo*	*Gateway*
GET	*LOCATION/NODE/SENSOR/*	*GET*	*Gateway*	*Nodo*
RESPOND	*LOCATION/NODE/SENSOR/**RESPOND***	*PUT*	*Nodo*	*Gateway*

Tabla 5: Adaptación del modelo de peticiones CoAP

2.3.3 NODOS

Los nodos estarán formados por diferentes placas Arduino, una interfaz de comunicaciones y al menos un sensor y un actuador. En el momento de su conexión, el nodo deberá ejecutar la acción de describe para informar al Gateway de su presencia y parámetros. Durante la ejecución de su programa deberá publicar las lecturas de sus sensores a intervalos regulares preestablecidos y a la vez será capaz de responder apropiadamente a los diferentes comandos que reciba desde el Gateway. Todos los nodos, independientemente de su tecnología y protocolo de comunicación, han

de implementar la ontología del sistema, el modelo de intercambio de mensajes y todas las acciones y comandos especificados. Deberán compartir también una estructura común de programa con mínimas diferencias de manera que un programa pueda ser portado con pocos cambios a cualquier otro nodo. A tal efecto se desarrollara una librería común que implemente la mayor parte de las funcionalidades y agregue una capa de abstracción al sistema haciendo el desarrollo sobre cualquier interfaz y protocolo algo casi transparente.

2.3.4 GATEWAY

Como se ha comentado con anterioridad, el Gateway estará basado en una Raspberry Pi 3. Esta SBC ya cuenta con una interfaz Wi-Fi integrada por lo que para cumplir los requisitos habrá que dotarla también de interfaces ZigBee y LoRa. El núcleo del Gateway y encargado de gestionar prácticamente toda la funcionalidad del mismo será una aplicación en Python. La elección de Python viene motivada por el gran soporte que posee en Raspberry Pi y al hecho de que al tratarse de un lenguaje interpretado los ciclos de desarrollo son más rápidos. Para el diseño del sistema a nivel funcional y tecnológico se ha planteado una arquitectura en diferentes capas en las que se distribuyen las distintas responsabilidades del sistema.

2.3.4.1 CAPA DE ADQUISICIÓN E INTERCONEXIÓN

En esta capa se incluyen los diferentes conectores específicos para cada protocolo que se encargarán de gestionar la comunicación bidireccional con los nodos. Aquí nos encontramos las interfaces de hardware, las aplicaciones y librerías de software necesarias para gestionar el hardware y el protocolo y la aplicación principal que gestione todas ellas

2.3.4.2 CAPA DE CONOCIMIENTO Y PERSISTENCIA

En esta capa se ofrecen los servicios de almacenamiento y procesado de los datos procedentes de la captación. En el diseño propuesto se utilizarán como DMBS relacional MariaDB y MongoDB como no relacional.

2.3.4.3 CAPA DE INTEROPERABILIDAD

En esta capa se sitúan los servicios que permiten que entidades externas al propio sistema puedan operar con el mismo. Para ello se establece una API REST que ofrece la lectura ya actualización de entidades a la red local. El encargado de la interacción con el cliente de la API será un script PHP ejecutándose en un servidor web Apache. También se utilizará un servicio especializado en IoT en la nube, en nuestro caso SensorCloud

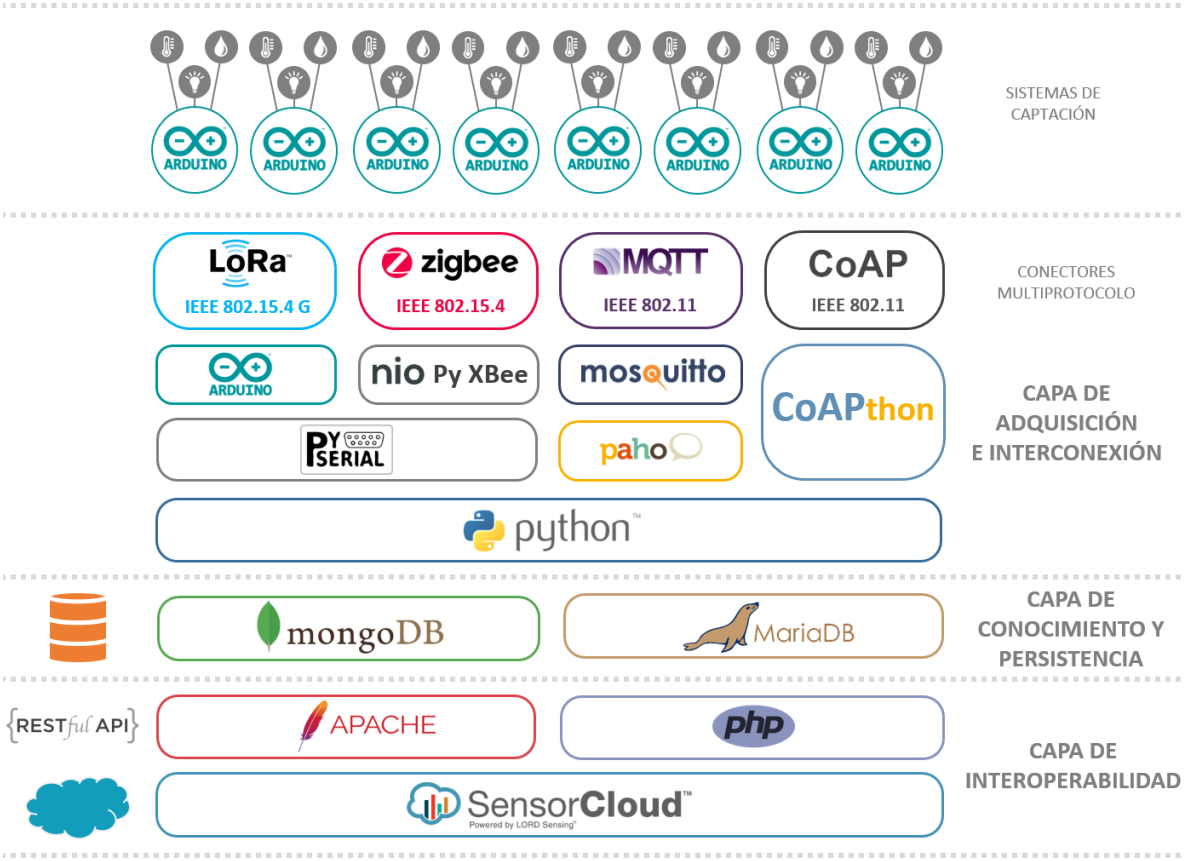


Ilustración 8: Arquitectura en capas de la solución propuesta

3

DESARROLLO Y PROTOTIPADO

3.1 NODOS

3.1.1 COMPONENTES COMUNES

3.1.1.1 HARDWARE

Todos los nodos comparten una serie de componentes de hardware comunes que hacen las funciones de sensores y actuadores: un sensor DHT 11 y un diodo LED.

Ilustración 9: DHT11, Diodo Led, Resistencia 100KΩ

SENSOR DE TEMPERATURA Y HUMEDAD DHT11

Para medir la temperatura y la humedad se utilizará el sensor DHT11, un sensor digital de bajo coste que ofrece ambas lecturas con una respuesta rápida. La precisión del sensor no es muy elevada pero para el propósito del proyecto cumple perfectamente su función. La versión elegida va montada sobre su propia PCB y cuenta con tres pines: VDD+, GND- y DATA. En la PCB lleva integrada una resistencia de 5.1kΩ entre VDD+ y DATA necesaria para su funcionamiento.

Resolución	Rango Temperatura	Rango Humedad	Precisión Temperatura	Precisión Humedad	Tiempo de respuesta
16 Bit	0 - 50 °C	20% - 90% RH	± 2°C	± 4% HR	2s

Tabla 6: Características del sensor DHT11

DIODO EMISOR DE LUZ

Se trata de un sencillo LED de dos terminales conectado en serie a una resistencia. La función de este LED es asumir el rol de un actuador (como podría ser un relé o una electrobomba).

CONEXIONES Y CABLEADO

Las conexiones del sensor DHT11 y del diodo son muy sencillas y comunes a todas las placas y nodos. Se utiliza el pin D7 para controlar el sensor DHT y el pin D8 para el diodo LED.

ARDUINO	DHT11	DIODO
VCC	+5V	
GND	GND	CÁTODO
D7	SIGNAL	
D8		ÁNODO (Ω)

Tabla 7: Conexiones entre Arduino, diodo LED y sensor DHT11

3.1.1.2 SOFTWARE

LIBRERÍA NODE

Se trata de una librería Arduino desarrollada para este proyecto en lenguaje C++ que implementa los métodos encargados de gestionar los diferentes comandos y acciones que representan el núcleo de la funcionalidad del sistema. También incluye el modelo de intercambio de mensajes común a todos los nodos así como métodos de codificación y descodificación y otros métodos auxiliares (conversión de tipos, etc…).

Para el diseño de la librería se ha implementado una jerarquía de clases partiendo de una clase abstracta que mantiene las estructuras de datos, la ontología de los nodos y los métodos comunes. De esta clase heredan el resto de clases que implementan los métodos y dependencias de cada protocolo y que son las que se importarán en el sketch de Arduino.

- Clase **Node:** es la superclase abstracta de la librería. Implementa los atributos y métodos comunes a todas las clases. Como atributos de la clase tenemos el nombre del nodo, la

localización, el protocolo y las tres funciones callback de los comandos GET, SET y SUSCRIBE. A nivel de métodos implementa las funciones de asignación de los atributos así como otros métodos para el tratamiento de mensajes normalizados y tópicos. También incorpora varios métodos auxiliares para la conversión de tipos, útiles para dar soporte a distintos tipos de sensores.

- Clase **NodeUART**: esta clase abstracta derivada de la clase Node implementa atributos y métodos comunes a los protocolos que utilizan la UART (puerto serie) para conectar la interfaz de comunicación, es decir, LoRa y ZigBee, ya que gran parte de la funcionalidad es similar para ambos protocolos. Esta clase no implementa atributos. Entre los métodos que incluye se encuentran la codificación y decodificación de los mensajes en JSON, los métodos de los comandos PUBLISH, RESPOND, ACK Y DESCRIBE y los métodos virtuales *tx* y *rx* que se han de implementar en las subclases ya que son específicos para cada protocolo. Son los encargados del envío y recepción de mensajes.
- Clase **NodeZIGBEE**: esta clase deriva de la clase NodeUART y presenta una dependencia con la librería XBee-Arduino de Andrew Rapp encargada de gestionar el protocolo. Implementa los atributos para contener el cliente del protocolo y objetos auxiliares inyectados en el constructor. Como métodos implementa las funciones de inicialización y de control del bucle y los métodos *tx* y *rx* según el contrato de la clase padre.
- Clase **NodeLORA**: esta clase deriva de la clase NodeUART y presenta una dependencia con la librería Arduino-LoRa de Sandeep Mistry encargada de gestionar el protocolo. Implementa los atributos que contienen el cliente del protocolo y objetos auxiliares inyectados en el constructor. Como métodos implementa las funciones de inicialización y de control del bucle y los métodos *tx* y *rx* según el contrato de la clase padre.
- Clase **NodeMQTT**: esta clase deriva directamente de la clase Node y presenta una dependencia con la librería Pubsubclient de Nicholas O'Leary encargada de la gestión del protocolo. Implementa como atributo el cliente del protocolo. Como métodos implementa las funciones de inicialización y de control del bucle y los métodos *publish y subscribe* así como otros métodos auxiliares para la reconexión al bróker o la generación del payload en JSON. Un método importante es *subscribe_handle* que es el encargado de la gestión de los tópicos a los que se suscribe el nodo, es decir, a los mensajes entrantes y de sus respuestas.
- Clase **NodeCOAP**: esta clase deriva directamente de la clase Node y presenta una dependencia con la librería Microcoap de Toby Jaffey y Pilgrim Beart encargada de la gestión del protocolo. Implementa como atributo el cliente del protocolo. La implementación es muy similar a la de

MQTT compartiendo casi toda la estructura y nomenclatura y variando sólo la implementación del cliente gestionado por la librería Microcoap

Ilustración 10: Diagrama de clases de la librería Node para Arduino

SKETCH ARDUINO

Gracias al uso de la librería Node, la estructura de los programas Arduino que se implementen en los diferentes nodos así como gran parte del código de los mismos será compartido por todos ellos, simplificando el proceso de desarrollo, clarificando el código, y facilitando la incorporación de nuevos nodos sin apenas esfuerzo. Las variaciones sobre la estructura presentada serán mínimas y vendrán condicionadas fundamentalmente por algunos aspectos específicos de cada protocolo. El programa de cada nodo quedaría dividido en tres partes:

Inclusión de librerías, constantes, variables globales y objetos

En esta sección se incluyen las librerías necesarias (sensor, cliente de protocolo y nuestra librería Node. Acto seguido se definen las constantes del programa: pines asignados, nombre de la

localización y el nodo y nombres de los parámetros (sensores y actuadores). También se incluiría cualquier otra constante como los parámetros relacionados con la conexión si es que los hubiera (direcciones, contraseñas…). Tras las constantes declaramos las variables globales para controlar los intervalos de lectura no bloqueantes (mediante el uso de un *timer*). Finalmente se lleva a cabo la instanciación de clases, en nuestro caso la clase que gestiona el sensor DHT, las clases que gestionan el protocolo y la subclase de Node correspondiente al protocolo a la cual le inyectaremos la clase o clases cliente en el constructor.

```
#include <NodeZIGBEE.h>
#include <XBee.h>
#include <DHT.h>
```
Inclusión de dependencias

```
#define L_PIN 8
#define DHTPIN 7
#define DHTTYPE DHT11
```
Definición de constantes

```
#define LOCATION "GREENHOUSE"
#define NODE "ONE"
```
Definición de constantes: ontología

```
#define TEMPERATURE "TEMPERATURE"
#define HUMIDITY "HUMIDITY"
#define LIGHT "LIGHT"
```
Definición de constantes: parámetros

```
long lastSendTime = 0;
long interval = 60000;
```
Variables para la lectura a intervalos no bloqueante

```
DHT dht(DHTPIN, DHTTYPE);
```
Instancia de la clase que gestión sensor DHT

```
XBee xbee = XBee();
ZBRxResponse rx = ZBRxResponse();
ModemStatusResponse msr = ModemStatusResponse();
NodeZIGBEE node(xbee, rx, msr);
```
Instancia de la clase que gestión el protocolo

Instancia de la subclase de Node según el protocolo

Funciones Callback

En esta sección se deben declarar y desarrollar las funciones callback que se pasarán a la instancia de la subclase Node en la inicialización. La función de estas clases es dar respuesta a las diferentes acciones o peticiones que lleguen provenientes del Gateway y que variarán dependiente de las características de cada nodo. En nuestro caso, dado que a nivel de parámetros (sensores y actuadores) todos los nodos son exactamente iguales, esta parte del código será idéntica para todos los nodos pero en un entorno real será normal que cada nodo cuente con diferentes sensores y actuadores por lo que estas funciones deberán implementarse de acuerdo a las características específicas de cada uno. Las tres funciones que han de estar presentes en cualquier nodo son:

Callback para el comando GET: esta función será invocada por el programa cuando el nodo reciba una petición de tipo GET procedente del Gateway. La función debe devolver el valor actual del parámetro que se la facilita como argumento por tanto debe implementar los mecanismos para obtener ese valor para cada parámetro y retornarlo.

Callback para el comando SET: esta función será invocada por el programa cuando el nodo reciba una petición de tipo SET procedente del Gateway. La función ha de asignar el valor que se le pase como argumento al parámetro (actuador) que le es igualmente facilitado.

Callback para el comando DESCRIBE: esta función será invocada por el programa cuando el nodo reciba una petición de tipo DISCOVER procedente del Gateway así como en la conexión inicial del nodo. La función ha de describir cada sensor y actuador presentes en el nodo mediante invocaciones a la función "describir parámetro" con el nombre del parámetro y el tipo como argumentos.

```
char* get_parameter(char* parameter) {
  if(!strcmp(parameter, LIGHT)){
    return Node::to_char_buff(digitalRead(L_PIN));
  }else if (!strcmp(parameter, TEMPERATURE)){
    float temperature = dht.readTemperature();
    if  (!isnan(temperature)) {
      return Node::to_char_buff(dht.readTemperature());
    }
  }else if(!strcmp(parameter, HUMIDITY)){
    float humidity = dht.readHumidity();
    if  (!isnan(humidity)) {
      return Node::to_char_buff(dht.readHumidity());
    }
  }
  return NULL;
}
```

Función callback para el comando GET

```
bool set_parameter(char* parameter, char *value) {
    if(!strcmp(parameter, LIGHT)){
        bool v = value[0] == '1' ? HIGH : LOW;
        digitalWrite(L_PIN, v);
        return true;
    }
    return false;
}
```

Función callback para el comando SET

```
void describe_callback() {
   node.describe_parameter(TEMPERATURE);
   node.describe_parameter(HUMIDITY);
   node.describe_parameter(LIGHT, 1);
}
```

Función callback para el comando DESCRIBE

Inicialización y bucle principal

Finalmente tenemos la sección de inicialización y bucle principal del programa. En la inicialización (*setup*) se seguirá un mismo esquema para todos los nodos aunque puede haber pequeñas variaciones dependiendo del protocolo. Inicialmente se configurará el pin del actuador y su estado. Después se inicializará el objeto DHT encargado de la gestión del sensor. Posteriormente se realizará la inicialización del Serial (y/o, en función del nodo, del cliente WiFi). Tras esto inicializamos el objeto Node que instanciamos previamente, al cual le inyectaremos ahora el cliente (serial o wifi) y que será quien se encargue de gestionar el grueso de la comunicación de forma transparente. Tras la inicialización se configurarán la localización y nombre del nodo y se le pasarán las funciones callback.

En el bucle principal se encuentra el timer que controla los intervalos de ejecución de forma no bloqueante. Dentro del mismo se lleva a cabo una lectura de los parámetros de temperatura y

humedad procedentes del sensor y se publica su valor enviándolos al Gateway. Fuera del timer se encuentra la llamada a la función loop de la instancia Node. Esta llamada es crítica puesto que gracias a ella y con independencia del protocolo el nodo se mantienes escuchando a la espera de la llegada de mensajes procedentes del Gateway.

```
void setup() {
  pinMode(L_PIN, OUTPUT);
  digitalWrite(L_PIN, LOW);
  dht.begin();
  Serial.begin(9600);
  node.init(Serial);
  node.set_location(LOCATION);
  node.set_name(NODE);
  node.describe_callback(describe_callback);
  node.get_parameter_callback(get_parameter);
  node.set_parameter_callback(set_parameter);
  delay(3000);
}
```

Configuración e inicialización del programa

```
void loop() {
  if (millis() - lastSendTime > interval) {
    float humidity = dht.readHumidity();
    float temperature = dht.readTemperature();
    if (isnan(humidity) || isnan(temperature)) {
    }else{
      node.publish(TEMPERATURE, temperature);
      node.publish(HUMIDITY, humidity);
    }
    lastSendTime = millis();
  }
  node.loop();
}
```

Bucle principal del programa

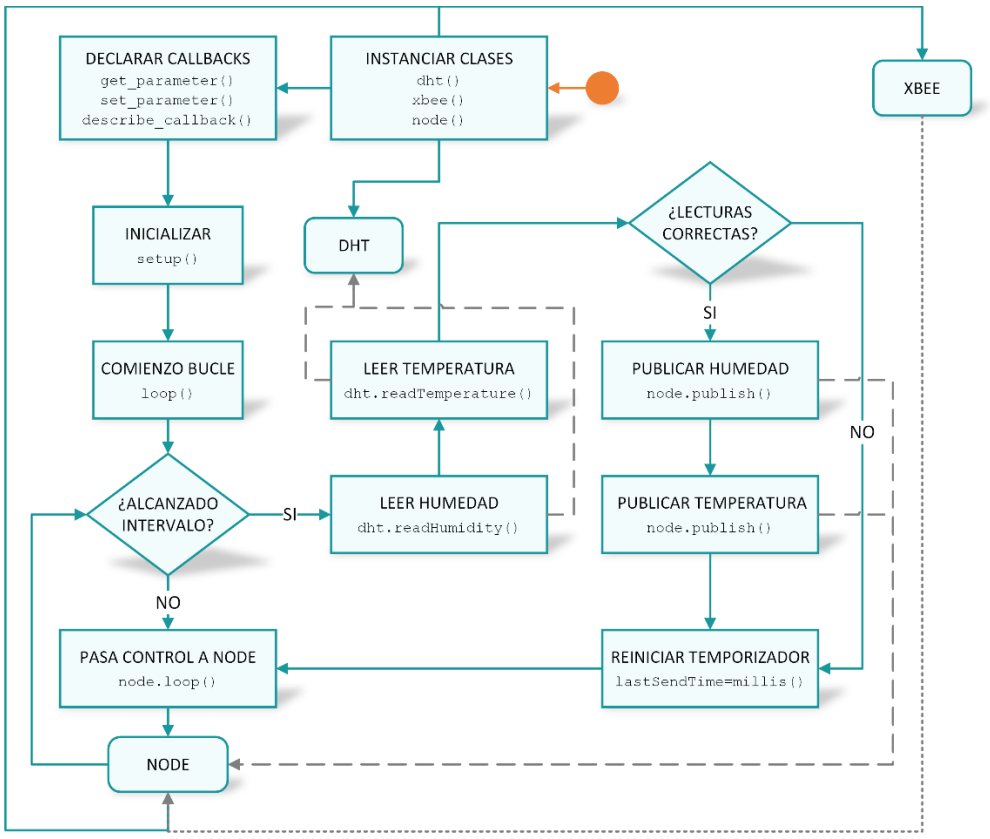


Ilustración 11: Diagrama de actividad genérico de los nodos

3.1.2 NODOS ZIGBEE

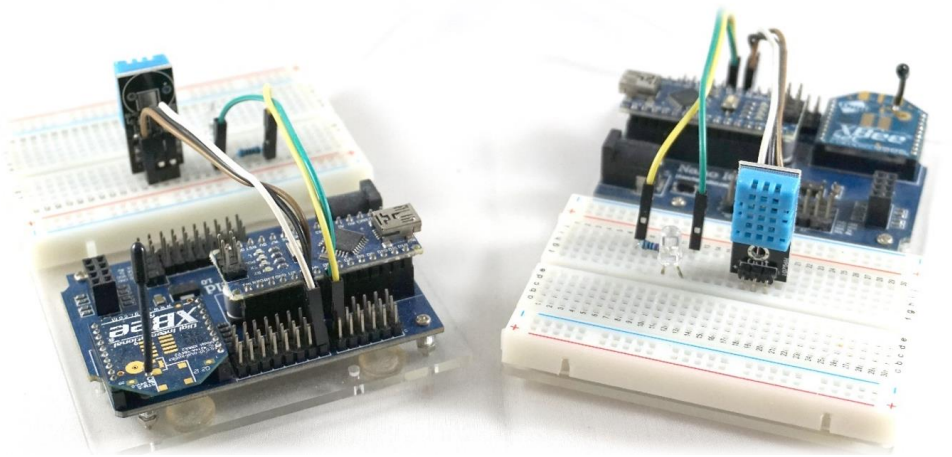


Ilustración 12: Nodos ZigBee

3.1.2.1 HARDWARE

Se describe a continuación el hardware distintivo que se emplea en los dos nodos del entorno de prototipado que implementarán el protocolo ZigBee

ARDUINO NANO

El Arduino Nano es una placa de microcontrolador basado en el ATmega328P. Es básicamente un clon del Arduino Uno pero con un factor de forma mucho más pequeño. Esta reducción hace que también disminuyan el número de entradas y salidas pero, en esencia, esta placa podría ser un reemplazo directo del Arduino original.

Procesador	*Voltaje Operación Entrada*	*Velocidad de CPU*	*I/O Analógico*	*I/O PWM Digital*	*EEPROM [kB]*	*SRAM [kB]*	*Flash [kB]*	*UART*
ATmega328P	*5 V / 7-9 V*	*16 MHz*	*8/0*	*14/6*	*1*	*2*	*32*	*1*

Tabla 8: Características de Arduino Nano

NANO I/O SHIELD

Esta placa permite la conexión fácil de un módulo XBee con una placa Arduino Nano. Para ello incluye un socket donde conectar los pines del Arduino Nano y otro con el factor de forma del XBee. Esta conexión se realiza a través de la UART (Tx/Rx). Además de esto la placa incluye un regulador de

potencia para alimentar al módulo XBee ya el Arduino Nano opera a 5v mientras que el módulo lo hace a 3.3v.

XBEE S2

El módulo S2 de Digi es una interfaz de comunicación sobre protocolo 802.15.4 que opera en los 2.4 GHz e implementa la pila de protocolo ZigBee. Este módulo pertenece a la serie 2 de Xbee y puede operar en modo transparente o mediante comandos API. También está preparado para soportar redes de tipo Mesh. Tiene un rango de hasta 60 metros en interior y 1200 metros en exterior

Alimentación	*Velocidad de transf.*	*Potencia de salida*	*Alcance*	*Pines ADC*	*Pines Digitales*	*Encript.*	*Modos*
3.3V 40mA	*250kbps*	*2mW (+3dBm)*	*60-1200 m*	*6 10-bit*	*8*	*128 bit*	*AT o API*

Tabla 9: Características del XBee S2 XB24-ZB

Para utilizar el módulo Xbee antes hay que configurarlo mediante el uso de la aplicación XCTU que proporciona Digi de manera gratuita. La aplicación permite configurar decenas de parámetros, hacer diferentes pruebas, enviar mensajes y otras funciones adicionales pero lo fundamental para lo que se va a utilizar es para cargar en los Xbees los firmwares adecuados y configurar los parámetros básicos. Para ambos nodos tenemos que configurar el Xbee en modo *router* o *end device.* Optamos por configurarlos como *routers* utilizando el protocolo ZigBee y en modo Api. para ello lo que hacemos es cargarle el firmware "ZigBee Router API" en su última versión: 23A7.

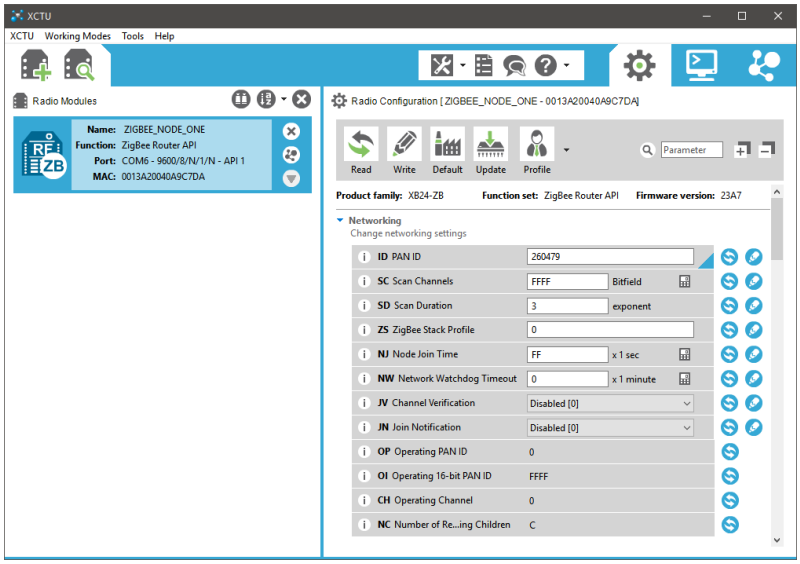


Ilustración 13: Digi XCTU, Pantalla de configuración de Xbee

Hecho esto podemos configurar el parámetro más importante, el PAN ID, que identificará nuestra red de forma unívoca. Les asignamos el PAN ID: 260479. También configuramos el parámetro API ENABLE a 2 para utilizar el modo API con escape de caracteres.

Ilustración 14: Arduino Nano, Nano I/O Shield, Módulo Xbee

CONEXIONES Y CABLEADO

La conexión entre la placa Arduino y el módulo ESP8266 se realiza mediante el Escudo Bee. En realidad se trata de una sencilla conexión al puerto serie (UART) de Arduino junto a un regulador de potencia en medio para salvar la diferencia de 5v a 3,3v.

ARDUINO	ESCUDO BEE		ESP8266
VCC	*+5V*	*+3,3V*	*+3,3V (1)*
GND	*GND*		*GND (10)*
RX	*RX*		*RX (2)*
TX	*TX*		*TX (3*

Tabla 10: Conexiones de nodo CoAP

3.1.2.2 SOFTWARE

Se describe a continuación el software que implementarán los dos nodos ZigBee, destacando las dependencias de librerías externas así como la estructura y contenido diferencial del sketch respecto al modelo general que se obtiene del uso de la librería Node que hemos desarrollado.

SKETCH ARDUINO

El programa de los nodos ZigBee es exactamente el que se ha mostrado como sketch común. Presenta una dependencia de la librería XBee-Arduino de Andrew Rapp

3.1.3 NODOS LORA

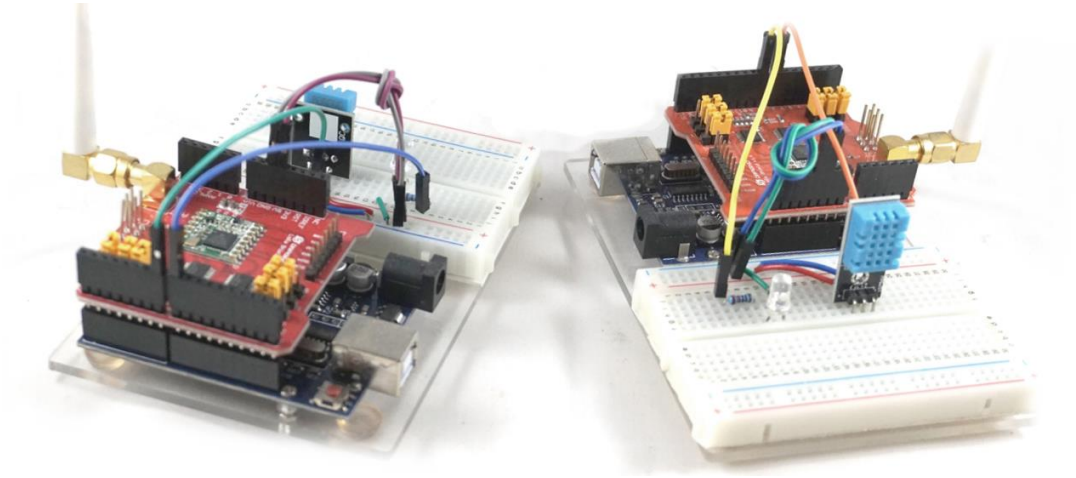


Ilustración 15: Nodos LoRa

3.1.3.1 HARDWARE

Se describe a continuación el hardware distintivo que se emplea en los dos nodos del entorno de prototipado que implementarán el protocolo LoRa

ARDUINO UNO

El Arduino Uno es una placa de microcontrolador basado en el ATmega328P. Es el diseño de referencia Arduino original y el más popular a pesar de haber sido superado en muchos aspectos por muchas otras placas.

Procesador	*Voltaje Operación Entrada*	*Velocidad de CPU*	*I/O Analógico*	*I/O PWM Digital*	*EEPROM [kB]*	*SRAM [kB]*	*Flash [kB]*	*UART*
ATmega328P	*5 V / 7-12 V*	*16 MHz*	*6/0*	*14/6*	*1*	*2*	*32*	*1*

Tabla 11: Características de Arduino Uno

DRAGINO LORA SHIELD

El Draguino LoRa Shield es un escudo Arduino que integra un transceptor de radio LoRa basado en el chip Semtech SX1276/SX1278. Esta interfaz permite comunicaciones a distancias muy grandes, llegando con facilidad por encima de los 6 kilómetros. Este escudo se caracteriza por su gran sensibilidad, y un amplificador integrado de 20 dBm. Soporta modos (G) FSK para WMBus y IEEE802.15.4.g.

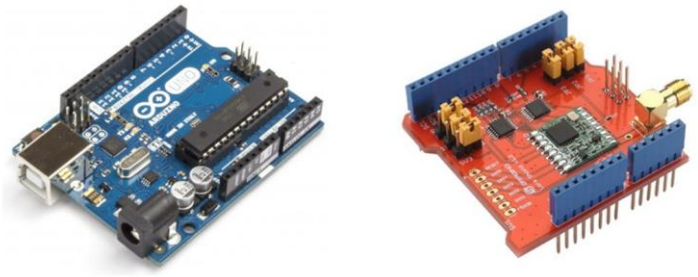


Ilustración 16: Arduino Uno, Dragino LoRaShield

CONEXIONES Y CABLEADO

La conexión del módulo Lora SX1276 al Arduino se lleva a cabo mediante el puerto SPI. La ventaja de utilizar un escudo con el módulo de radio incorporado es que éste está conectado a los pines adecuados mediante pistas en la PCB y además incluye el correspondiente regulador de tensión para pasar de 5V a 3,3V. Todo ello hace que la conexión sea un proceso fácil y rápido, lo que resulta ideal para el prototipado.

ARDUINO	*LORASHIELD*			SX1276
GND	*GND*			*GND*
VCC	*5V*	*REGULADOR*	*3,3V*	*3,3V*
NSS				D10
MISO				MISO
MOSI				MOSI
SCK				SCK
RESET				N9
DI0				D2

Tabla 12: Conexiones de nodo CoAP

3.1.3.2 SOFTWARE

Se describe a continuación el software que implementarán los dos nodos LoRa, destacando las dependencias de librerías externas así como la estructura y contenido diferencial del sketch respecto al modelo general que se obtiene del uso de la librería Node que hemos desarrollado.

SKETCH ARDUINO

El programa de los nodos LoRa es prácticamente igual al modelo general explicado junto a la librería Node y, por tanto, al de los nodos Zigbee. Presenta una dependencia de la librería Arduino-LoRa de Sandeep Mistry. A diferencia de lo que sucede con los nodos ZigBee, tenemos que configurar el

direccionamiento dentro del propio programa. En nuestro caso los nodos tendrán asignadas las direcciones 0xAA y 0xBB

3.1.4 NODOS WIFI MQTT

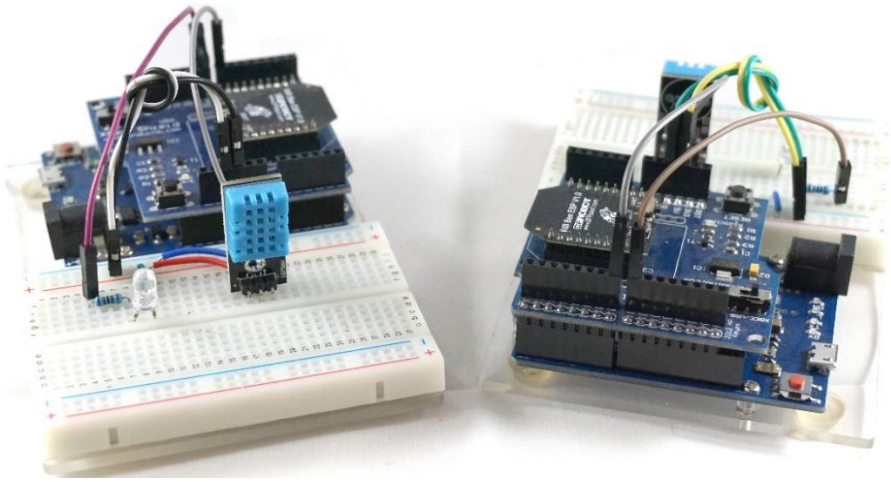


Ilustración 17: Nodos Wifi MQTT

3.1.4.1 HARDWARE

Se describe a continuación el hardware distintivo que se emplea en los dos nodos del entorno de prototipado que implementarán el protocolo MQTT

ARDUINO LEONARDO

El Arduino Leonardo es un microcontrolador basado en el ATmega32u4 que, en general, presenta características muy similares al Uno, el microcontrolador Arduino original. Difiere de la mayoría de placas Arduino en que el procesador incluye comunicación USB integrada, lo que elimina la necesidad de un segundo procesador para esta tarea y permite que cuando se conecta a un ordenador se muestre como un puerto Serial / COM o incluso como un teclado y ratón. Esta característica resulta muy útil, por ejemplo, para poder imprimir por el Serial conectado al ordenador mientras el Leonardo hace uso de la UART tradicional.

Procesador	*Voltaje Operación Entrada*	*Velocidad de CPU*	*I/O Analógico*	*I/O PWM Digital*	*EEPROM [kB]*	*SRAM [kB]*	*Flash [kB]*	*UART*
ATmega32U4	*5 V/7-12 V*	*16 MHz*	*12/0*	*20/7*	*1*	*2.5*	*32*	*1*

Tabla 13: Características de Arduino Leonardo

ESCUDO XBEE

Este escudo permite la conexión sencilla de un módulo XBee con una placa Arduino sin necesidad de cableado. En su placa incluye un socket con el factor de forma del XBee y sus pines se conectan por las pistas de la placa a los pines correspondientes del Arduino. Esta conexión se realiza a través de la UART (Tx/Rx). Además de esto la placa incluye un regulador de potencia para alimentar al módulo XBee ya que la mayoría de placas de Arduino operan a 5v mientras que el módulo lo hace a 3.3v. Cuanta con un switch SPDT con el que se puede elegir conectar el XBee con los pines de la UART(D0 y D1) o con los pines D2 y D3 -en caso de necesitar liberar la UART-. Aunque este escudo se desarrolló inicialmente para los módulos XBee, han surgido en el mercado muchos otros módulos de comunicaciones con el mismo factor de forma pero con diferentes tecnologías (ej: WiFly, Bluetooth) que pueden hacer uso de este tipo de escudos siempre y cuando se respete la posición y función de los pines y se haga uso de la conexión UART.

ESP8266 WIFIBEE

El ESP8266 WifiBee es un módulo de comunicación Serial a Wifi con el factor de forma de Xbee que integra la pila de protocolo TCP/IP. Puede operar en modo AP+STA y es totalmente configurable mediante comandos AT. Este módulo se puede utilizar con Arduino y otros microcontroladores o de forma independiente puesto que es programable, cuenta con un potente procesador de 32 bits y gran capacidad de almacenamiento por sí mismo.

Alimentación	*Frecuencia*	*Potencia de salida*	*Protocolos*	*Seguridad*	*CPU*
3.3V <240Ma	*2,4 GHz*	*+19.5dBm*	*802.11b/g/n*	*WPA WPA2*	*32 Bit*

Tabla 14: Características del WifiBee ESP8266

Para que el módulo ESP8266 sea compatible con la librería Pubsubclient se ha modificar su firmware y actualizarlo a las versiones: SDK 1.5.4 y AT 1.1.0. La actualización de del firmware se lleva a cabo con un software programador específico, en nuestro caso con el de Vowstar que proporciona el fabricante. La configuración para la programación es sencilla y sólo requiere indicarle las imágenes en formato .bin que se desean programar y las posiciones en la memoria donde se han de quemar.

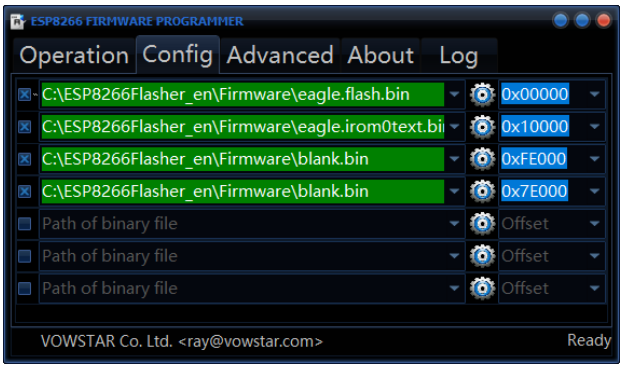


Ilustración 18: Vowstar Firmware Programmer, Pantalla de flasheo de firmware ESP8266

CONEXIONES Y CABLEADO

El conexionado de los nodos MQTT es exactamente igual que el de los nodos ZigBee ya que el módulo ESP8266 hace un uso del puerto serie similar al del Xbee.

ARDUINO	*ESCUDO BEE*			*ESP8266*
VCC	*+5V*	*REGULADOR*	*+3,3V*	*+3,3V (1)*
GND	*GND*			*GND (10)*
TX	*TX*			*D.OUT (2)*
RX	*RX*			*D.IN (3)*

Tabla 15: Conexiones de nodo MQTT

Ilustración 19: Arduino Leonardo, Escudo Bee, ESP8266 WifiBee

3.1.4.2 SOFTWARE

Se describe a continuación el software que implementarán los dos nodos MQTT, destacando las dependencias de librerías externas así como la estructura y contenido diferencial del sketch respecto al modelo general que se obtiene del uso de la librería Node que hemos desarrollado.

SKETCH ARDUINO

El sketch que gobierna los nodos MQTT es muy similar al modelo general explicado junto a la librería Node. Presenta dependencias con las librerías WifiEsp de Bruno Portaluri y PubSubClient de Nocholas

O'Learey. En este caso, al utilizar como interfaz un ESP8266, primero se instancia el cliente Wifiesp que se inyecta al constructor del cliente MQTT y éste a su vez se le inyecta al constructor de la clase NodeMQTT. Además de esto hay que llevar a cabo toda la configuración de la red WiFi y la configuración e inicialización de la conexión que se realiza mediante la función *setup_wifi*. La última diferencia es la función *subscribe_callback* que, como su nombre indica, establece la función callback que gestiona la recepción de mensajes de los tópicos a los que está suscrito. El resto del programa es prácticamente idéntico a lo que ya se ha visto.

3.1.5 NODOS WIFI COAP

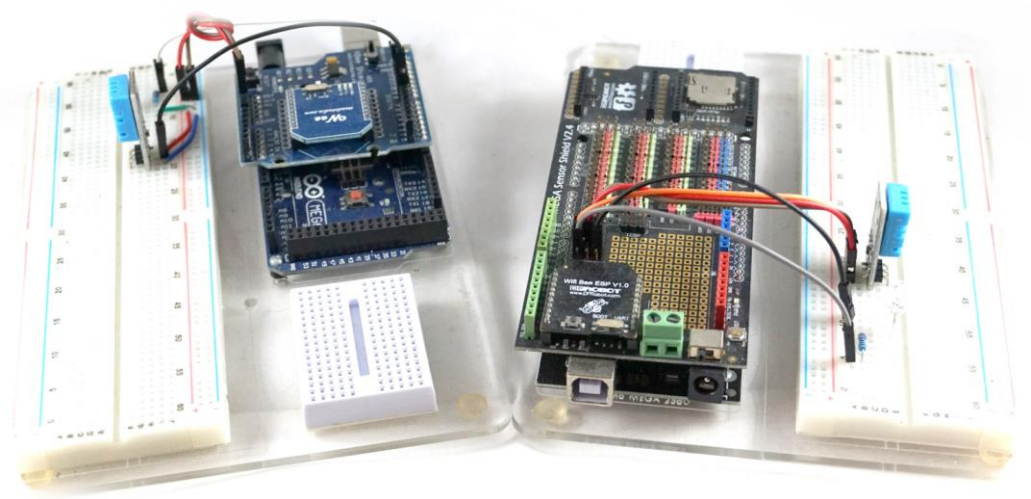


Ilustración 20: Nodos WiFi CoAP

3.1.5.1 HARDWARE

Se describe a continuación el hardware distintivo que se emplea en los dos nodos del entorno de prototipado que implementarán el protocolo CoAP

ARDUINO MEGA 2560 (ATMEGA2560)

El Arduino Mega 2560 es una placa de microcontrolador basado en el ATmega2560 y representa una evolución sobre el anterior Arduino Mega original. Dentro del ecosistema Arduino es una de las placas más completas por su gran número de entradas y salidas así como por su abultada memoria flash en comparación con otras placas. Además las comunicaciones habituales como I2C, TWI o SPI, cuenta con 4 UARTs implementadas a nivel de Hardware. La gran cantidad de conexiones hace que su tamaño crezca, convirtiéndola junto a la Due en el Arduino de mayor tamaño de todos aunque su diseño le permite mantener la compatibilidad con todos los escudos compatibles con Arduino.

Procesador	Voltaje Operación Entrada	Velocidad de CPU	I/O Analógico	I/O PWM Digital	EEPROM [kB]	SRAM [kB]	Flash [kB]	UART
ATmega2560	5 V / 7-12 V	16 MHz	16/0	54/15	4	8	256	4

Tabla 16: Características de Arduino Mega 2560

ESCUDO XBEE

Este es el mismo escudo que se ha utilizado con anterioridad en los nodos MQTT

MÓDULO ESP8266 WIFI BEE

Se trata del mismo módulo que se utiliza en los nodos MQTT descrito anteriormente

Ilustración 21: Arduino Mega, Escudo Bee, ESP8266 WifiBee

CONEXIONES Y CABLEADO

El conexionado de los nodos CoAP es exactamente igual que el de los nodos MQTT ya que el hardware de comunicaciones es igual o muy similar.

ARDUINO	ESCUDO BEE		ESP8266
VCC	+5V	+3,3V	+3,3V (1)
GND	GND		GND (10)
RX	RX		RX (2)
TX	TX		TX (3

Tabla 17: Conexiones de nodo CoAP

3.1.5.2 SOFTWARE

Se describe a continuación el software que implementarán los dos nodos CoAP, destacando las dependencias de librerías externas así como la estructura y contenido diferencial del sketch respecto al modelo general que se obtiene del uso de la librería Node que hemos desarrollado.

SKETCH ARDUINO

El sketch que gobierna los nodos CoAP es similar al modelo general explicado junto a la librería Node a pesar de lo cual es el que presenta mayores diferencias. El programa presenta dependencias con la librería WifiEsp de Bruno Portaluri y la librería Microcoap de Toby Jaffey y Pilgrim Beart. Este caso es similar a los nodos MQTT ya que al utilizar como interfaz un ESP8266 hay que llevar a cabo los mismos pasos: primero se instancia el cliente Wifiesp aunque en este caso se instancia la versión UDP del mismo. Después se inyecta al constructor del cliente CoaP y éste a su vez se le inyecta al constructor de la clase NodeCOAP. También hay que llevar a cabo toda la configuración de la red WiFi y la configuración e inicialización de la conexión que se realiza mediante la función *setup_wifi.* El enfoque del protocolo y de la propia librería hacen que, a diferencia de lo que sucede en el resto de nodos, haya que implementar una función callback de respuesta para cada parámetro. Por lo demás el programa sigue el mismo patrón y estructura que el del resto de nodos.

3.2 GATEWAY M2M

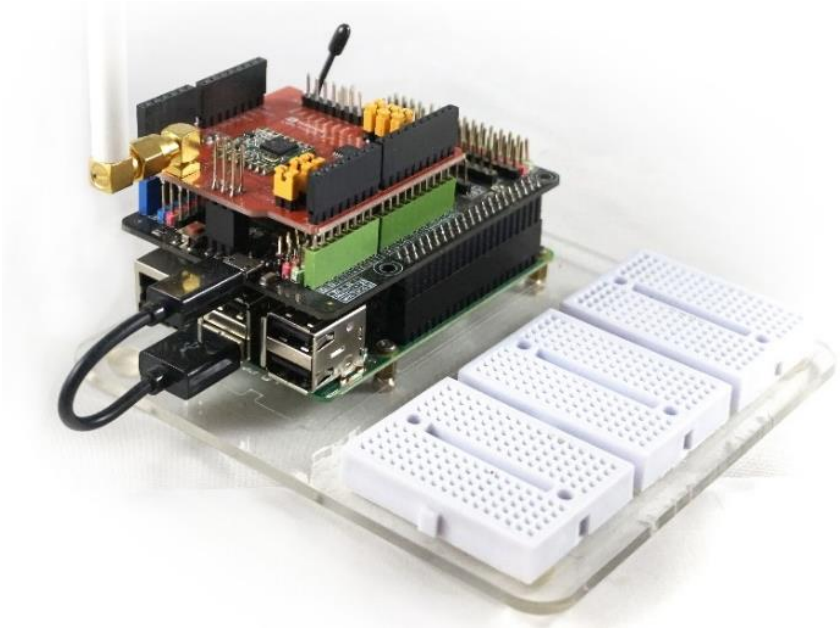


Ilustración 22: prototipo del Gateway

3.2.1 HARDWARE

Se describe a continuación el hardware que se utilizará en el prototipado del Gateway.

3.2.1.1 RASPBERRY PI 3 B

La Raspbery Pi 3 B es un computador de placa única (SBC) de arquitectura ARM y coste reducido que nació con fines educativos pero que ha superado con mucho sus objetivos y se ha convertido por derecho propio en uno de los SBCs multipropósito más exitosos. El modelo 3 B es la cuarta iteración de la Raspberry Pi A, el modelo original y, a diferencia de este, ya cuenta con unas características sumamente interesantes que le dan gran versatilidad como una CPU mucho más potente, mayor memoria RAM, un GPIO extendido y, sobre todo, mayor conectividad gracias a la incorporación de WiFi y Bluetooth.

Procesador	*CPU*	*Juego Instrucciones*	*GPU*	*Memoria*	*Conectividad*	*Periféricos*
Broadcom BCM2837	*1.2GHz quad core ARMv8*	*RISC 64 bits*	*Broadcom VideoCore IV*	*1 Gb*	*10/100 Ethernet, Wifi 802.11n Bluetooth 4.1*	*17xGPIO, SPI, I^2C, UART54*

Tabla 18: Características de Raspberry Pi 3B

3.2.1.2 ESCUDO EXPANSIÓN GPIO ARDUINO LEONARDO

Este escudo de expansión incorpora un socket con factor de forma Xbee, un reloj de tiempo real DS1307 RTC para la Raspberry Pi, un conversor bidireccional de 3.3v a 5v y un Arduino Leonardo completo integrado. La placa recibe su alimentación a través del GPIO y la conexión con el Arduino puede hacerse por el GPIO o mediante USB como en cualquier placa Arduino Es un escudo realmente versátil y cuyo enfoque es permitir que la Raspberry Pi utilice hardware inicialmente diseñado para Arduino. El puerto Xbee es asignable mediante switch a la Raspberry PI o al Arduino Leonardo.

3.2.1.3 DRAGINO LORA SHIELD

El escudo Draguino Lora Shield es exactamente el mismo empleado en los nodos LoRa.

3.2.1.4 XBEE S2

El módulo XBEE S2 es exactamente el mimo empleado en los nodos ZigBee. En este caso la configuración del módulo con la aplicación XCTU de Digi es distinta. Dado que su papel es actuar como Gateway configuraremos el Xbee como *coordinator* utilizando el protocolo ZigBee y en modo Api. Para ello le cargamos el firmware "ZigBee Coordinator Api" en su última versión: 21A7. Hecho esto podemos configurar el PAN ID al mismo que indicamos en los nodos: 260479. También configuramos el parámetro API ENABLE a 2 para utilizar el modo API con escape de caracteres.

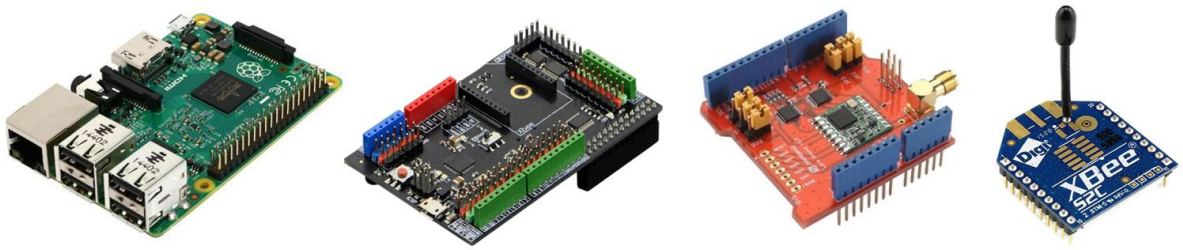


Ilustración 23: Raspberry Pi 3, Escudo Arduino Leonardo, Draguino LoRa Shield, Xbee

3.2.1.5 CONEXIONES Y CABLEADO

La conexión entre la Raspberry Pi el escudo de expansión se realizada mediante el GPIO. Para acceder al Arduino Integrado desde la Respberry Pi se podría configurar una conexión software serial configurando dos pines libres del GPIO de la Raspbery pero como tenemos la opción de conectarlo por USB optamos por esa opción. El módulo XBee se conecta a la Raspberry Pi mediante el UART a través del escudo de expansión seleccionando en el escudo que el acceso esté disponible para el GPIO y no para el Arduino Leonardo. El Draguino LoRa Shield, que utiliza la conexión SPI, se conecta al Arduino Leonardo integrado en el escudo de expansión.

3.2.2 SOFTWARE

3.2.2.1 RASPBIAN 9

La versión de Linux que instalaremos en la Raspberry Pi es Raspbian 9 (stretch) basada en la distribución Debian. El proceso de instalación consiste en grabar en una tarjeta MicroSD la imagen disponible para descarga en la web de Raspberry Pi.

Como queremos una instalación *headless*, esto es, sin pantalla ni teclado, con la imagen ya grabada en la tarjeta, creamos un archivo denominado “ssh” en la raíz de la partición *boot* para habilitar la sesión SSH que viene deshabilitada por defecto. Además de esto, creamos un archivo denominado “wpa_supplicant.conf” donde indicaremos los parámetros de la conexión Wi-Fi. Con esto conseguimos que en el primer arranque la Raspberry se conecte automáticamente a la red y esté disponible para establecer una sesión SSH. Una vez obtenida su IP nos podemos conectar y continuar el proceso de instalación.

Los siguientes pasos de configuración básica son triviales y pasan por lanzar la aplicación *raspi-config* donde podremos actualizar la propia aplicación, expandir el sistema de archivos, modificar opciones

de red, cambiar la contraseña, seleccionar la configuración local, y configurar puerto y periféricos entre otros. No entraremos en el detalle de estas configuraciones puesto que la mayoría resultan triviales. No obstante hay una configuración importante de cara a que la comunicación mediante Xbee y LoRa funcione correctamente. Debemos habilitar la interface serial y deshabilitar la posibilidad de hacer login a través de ella. Con esto podremos acceder al puerto serie sin problemas. Esta opción la configuramos dentro de *Interfacing Option* / *P6 Serial.*

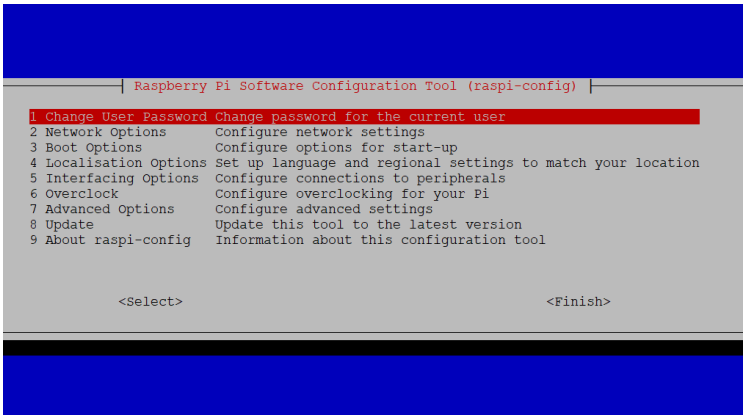


Ilustración 24: Captura de la aplicación raspi-config

3.2.2.2 APLICACIÓN PRINCIPAL

La aplicación encargada el Gateway estará desarrollada fundamentalmente en Python a excepción de toda la gestión web que se implementará en PHP. Para cumplir con los requisitos e implementar los casos de uso partimos de algunos conceptos o características que serán la base de la aplicación:

- **Estructura en capas**: se basa de las diferentes capas de responsabilidad que se propusieron en el diseño conceptual de la solución, de tal forma que tenemos la capa de adquisición, la de persistencia y la de interoperabilidad.
- **Formato de mensaje unificado**: cada mensaje que circule a través del Gateway ha de tener una serie de propiedades o campos comunes independientemente de su origen, destino o protocolo. Estos mensajes son similares (de hecho son sus generadores) a los que utilizamos en la comunicación entre los nodos y el Gateway pero se completan con propiedades adicionales como direccionamiento propio del protocolo. Dado que los mensajes entre Gateway y nodos se codifican en JSON y partimos de ellos, los mapeamos en diccionarios.
- **Enrutamiento de mensajes**: en función de la información de estos mensajes la aplicación sabrá qué hacer con los mismos, identificando el origen, destino y acción requerida, obrando en consecuencia. Esta es la pieza clave de la aplicación, constituida por una pequeña y sencilla función de ámbito global denominada que denominaremos *router.*

- **No bloqueante:** dado que la aplicación tiene que "escuchar" múltiples dispositivos con un grado de concurrencia que puede ser elevado y realizar acciones de manera asíncrona, se realizará un uso intensivo de hilos para que no se produzcan situaciones de bloqueo

Las diferentes funcionalidades de la aplicación se han agrupado en clases que pertenecen a alguna de las capas mencionadas. Cada una de estas clases agrupa los métodos y propiedades relacionados con su funcionalidad. A pesar de que a estas agrupaciones podríamos no haberlas considerado clases sino funciones del mismo espacio global (tal vez más propio de Python), por organización y legibilidad del código optamos por esta decisión de diseñó:

Ilustración 25: Diagrama de clases de la aplicación principal

- Capa de Adquisición
    - **Zigbee**: conector del protocolo ZigBee que se encarga de toda la comunicación el mismo.
    - **Lora**: conector del protocolo Lora que se encarga de toda la comunicación el mismo.
    - **Mqtt**: conector del protocolo Mqtt que se encarga de toda la comunicación el broker.
    - **Coap**: conector del protocolo CoAP y se encarga de toda la comunicación el mismo.
- Capa de persistencia
    - **Db**: se encarga de gestionar la interacción con las bases de datos SQL y NOSQL

- Capa de Interoperabilidad
    - **Cloud**: responsable de la interacción con el servicio Cloud
    - **Rest**: gestiona las peticiones REST y la comunicación con el script PHP mediante sockets

La aplicación en primer lugar lleva a cabo la declaración de las variables globales y la instanciación de las clases mencionadas. A partir de ahí, los constructores de cada una de las clases toman el control de sus responsabilidades creando los hilos pertinentes que se encargarán de "escuchar" las diferentes interfaces. En el siguiente diagrama se muestra una abstracción superficial de la actividad de la aplicación que, a pesar de no bajar al detalle, permite entender el flujo de la misma.

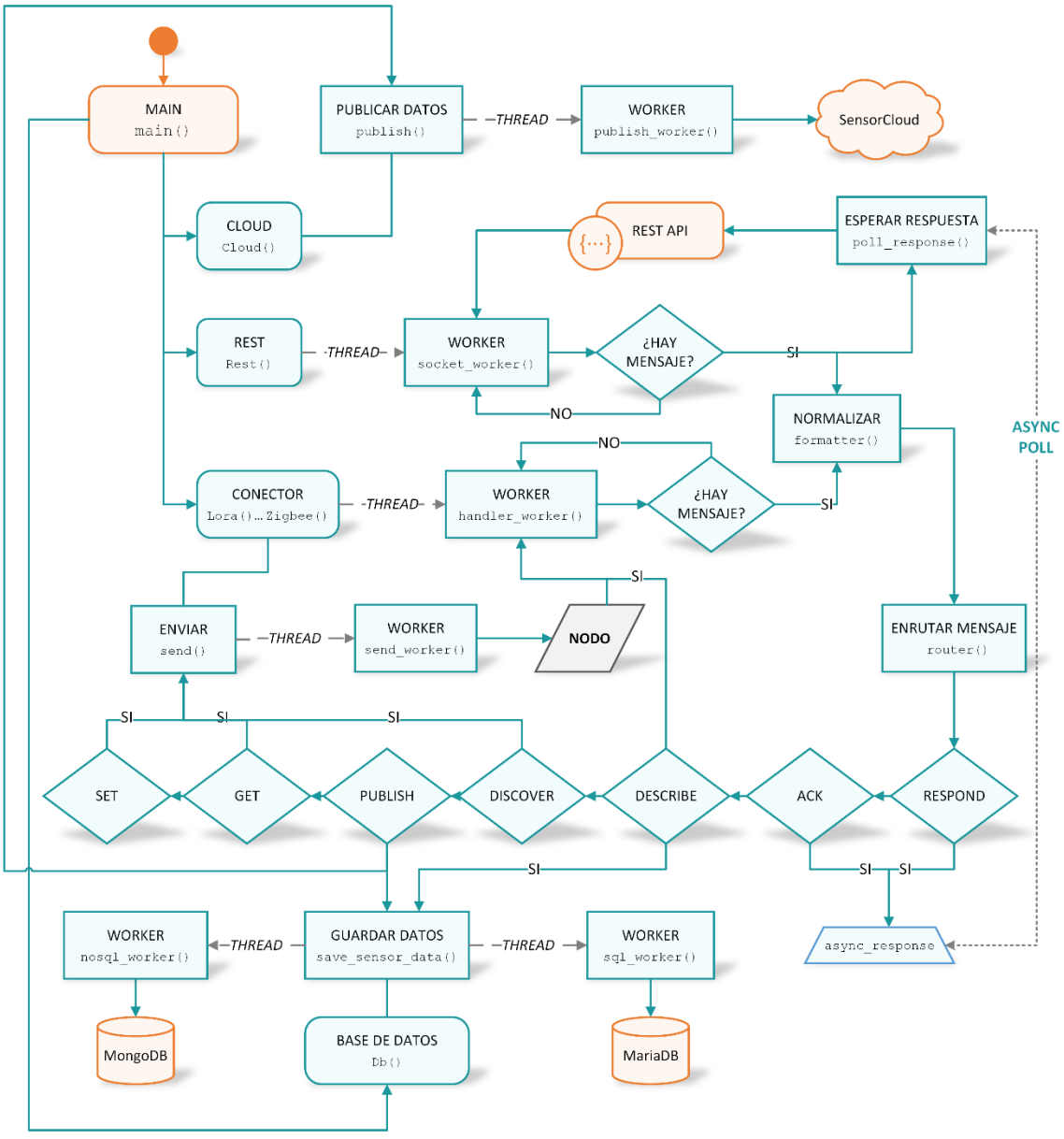


Ilustración 26: Diagrama de actividad global del Gateway

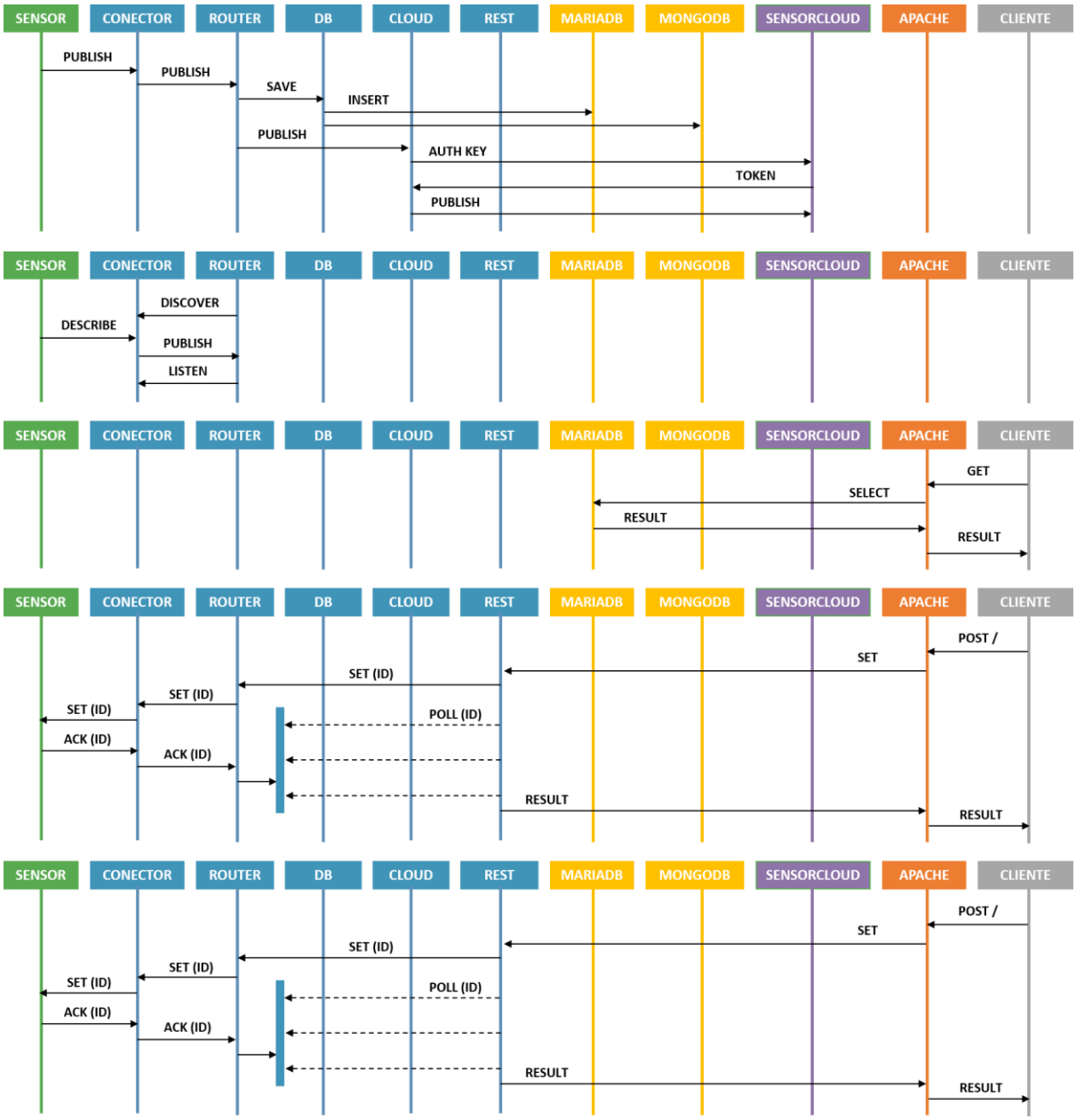


Ilustración 27: Diagramas de usos del Gateway

3.2.2.3 CAPA DE ADQUISICIÓN

Esta es sin duda la capa más relevante del Gateway puesto que se encarga de la interconexión y comunicación con los nodos y la adquisición de información. Sin esta capa el resto de capas y servicios no tienen sentido. Está formada por las cuatro conectores que se encargan de cada uno de los protocolos que se van a implementar en el Gateway. El comportamiento de todos los conectores es similar en el concepto y sólo difiere en la implementación por las características de la interface o de las librerías a implementar.

CONCTOR ZIGBEE

La comunicación con el módulo XBee de realiza mediante la clase ZigBee que tiene como dependencia la librería XBee. Dado que la conexión con el módulo se establece por el puerto serie "/dev/serial0", es decir, la UART del GPIO de la Raspberry Pi, la librería XBee presenta una dependencia con la librería PySerial que se encarga de gestionar las conexiones de tipo serial. La librería XBee se ejecuta en modo multi-hilo de forma síncrona o asíncrona por lo que podemos delegarle la creación de los hilos necesarios para que no sea bloqueante. La clase ZigBee además del constructor incorpora los siguientes métodos:

- **handler()**: función callback para gestionar los mensajes recibidos. Cuando recibe un mensaje lo formatea y se lo pasa al enrutador de la aplicación
- **formatter()**: función para dar formato a los mensajes entrantes. Decodifica el mensaje en JSON y le añade otras propiedades (direccionamiento, protocolo…)
- **send()**: función para el envío de mensajes
- **close()**: función que termina el cliente y cierra el puerto.

CONECTOR LORA

La conexión con el escudo Dragino LoRa shield se realiza a través del Arduino Leonardo del escudo de expansión de la Raspberry Pi como intermediario. Esto tiene dos implicaciones: por un lado hacemos que los dos extremos de la comunicación estén gestionados por Arduinos y, específicamente, por la misma librería; por otro lado hacemos que el protocolo sea transparente desde el punto de vista de la aplicación por lo que a todos los efectos se comportará como una conexión por el puerto serie. Debido a esto tendremos que implementar un sencillo sketch en el Arduino Leonardo que actué como un proxy de los mensajes en las dos direcciones.

Ya hemos comentado que desde el punto de vista de la aplicación, se gestiona como una conexión en el puerto serie "/dev/ttyACM0" que es donde irá conectado el Arduino. Para gestionar la comunicación por el puerto serie se utilizará la librería Lora que tiene como dependencia la librería PySerial que ya se ha utilizado en el conector XBee.

- **listen_worker ()**: función worker que escucha la llegada de mensajes en el puerto serial y cuando recibe alguno lo trata con la función handler()
- **handler()**: función callback para gestionar los mensajes recibidos. Cuando se le pasa un mensaje lo formatea con la función formatter() y se lo pasa al enrutador de la aplicación, esto es, la función router()

- **formatter()**: función para dar formato a los mensajes entrantes añadiendo al mensaje recibido otras propiedades (direccionamiento, protocolo…)
- **send()**: función que gestiona el envío de mensajes creando un hilo con la función worker
- **send_worker()**: función worker que ejecuta el envío de mensajes
- **close()**: función que termina el cliente y cierra el puerto.

CONECTOR MQTT

Tal como ya se ha visto, MQTT es un protocolo de tipo publicación / suscripción que necesita un Broker, esto es, un tipo de servidor que representa el punto central de la red y se encarga de centralizar y gestionar la trasmisión de mensajes en la red. La aplicación Python se comportará como un cliente más del bróker sólo que estará suscrita a los tópicos de todos los nodos y publicará determinados tópicos que permitan emular el modelo solicitud/respuesta. Cómo nuestro Broker decidimos usar Mosquitto, un bróker de la fundación Eclipse, por sus características, sencillez y amplio soporte. Tendremos que instalar el bróker y cliente para tener toda la funcionalidad. Su instalación es trivial y aunque se pueden establecer múltiples opciones de configuración, el bróker ya se puede utilizar sin necesidad de configurar nada. El complemento natural de Mosquitto es la librería Paho, un proyecto también de la fundación Eclipse. Se trata de una librería Python que implementa permite actuar como cliente del bróker y, por tanto, publicar y suscribirse a tópicos.

Las comunicaciones con los nodos MQTT estarán a cargo de la clase homónima que, como hemos explicado, tiene como dependencia la librería Paho. El cliente Paho se puede invocar en modo multi-hilo con lo que, al igual que sucedía con la librería XBee, podemos delegar esta tarea en la librería.

- **handler()**: función callback para gestionar los mensajes recibidos. Cuando se le pasa un mensaje lo formatea con la función formatter() y se lo pasa al enrutador de la aplicación, esto es, la función router()
- **formatter()**: función para dar formato a los mensajes entrantes añadiendo al mensaje recibido otras propiedades (direccionamiento, protocolo…)
- **send()**: función para el envío de mensajes
- **connect()**: función callback que se ejecuta al conectarse al bróker. Publica y se suscribe a los tópicos especiales que permiten el modo petición / respuesta.
- **close()**: función que termina el cliente

CONECTOR COAP

La comunicación con los nodos CoAP se realiza mediante la clase Coap que tiene como dependencia la librería CoAPthon. El funcionamiento de la librería difiere bastante de las otras. Esto es debido a que necesita que se declare una clase que implemente el servidor por un lado y clases que representen los recursos por otro lado e implementen entre sus métodos los manejadores a los distintos tipos de petición (GET, PUT, POST…) que se lleve a cabo sobre ese recuro. Para nuestra aplicación implementamos dos tipos de recursos: *child* y *basic*.

- **Server**: clase que implementa el servidor CoAP
- **Child:** clase de tipo recurso que permite que se creen recursos hijos
  - **render_GET()**: función para dar formato a los mensajes entrantes añadiendo al mensaje recibido otras propiedades (direccionamiento, protocolo…)
  - **render_POST()**: se encarga de la creación de recursos hijos
- **Basic:** clase tipo recurso que representa cada uno de los parámetros
  - **render_GET()**: función que devuelve el valor del recurso
  - **render_PUT()**: función que actualiza el valor de un recurso

Además de las clases internas necesarias para el funcionamiento de la librería CoAPthon nuestra clase implementa los siguientes métodos

- **listen_worker()**: función worker que ejecuta el servidor CoAP
- **formatter()**: función para dar formato a los mensajes entrantes. Decodifica el mensaje en JSON y le añade otras propiedades (direccionamiento, protocolo…)
- **send()**: función que gestiona el envío de mensajes creando un hilo con la función worker
- **send_worker()**: función worker que ejecuta el envío de mensajes

3.2.2.4 CAPA DE CONOCIMIENTO Y PERSISTENCIA

En la capa de servicios de persistencia lo que se pretende es que los datos se preserven, de tal forma que puedan ser consumidos y reutilizados con posterioridad. En el caso de nuestro Gateway hemos optado por mantener un sistema dual de persistencia con dos enfoques muy distintos, una base de datos relacional y otra no relacional.

MARIADB

Dentro de la capa de servicios de persistencia la base de datos SQL elegida para el Gateway es MariaDB. Esta base de datos se ha escogido por ser un clon de MySQL, con el gran soporte y

compatibilidad que esto conlleva, pero con un licenciamiento totalmente libre. Se trata de un clon binario de MySQL, es decir un reemplazo directo. Esta base de datos SQL se va a utilizar para almacenar dos tipos de información: entidades u objetos conectados según la jerarquía definida en la ontología y datos procedentes de las lecturas. Para mantener un tamaño adecuado que no penalice el rendimiento, en el caso de los datos procedentes de las lecturas se limitará el volumen de información a un histórico de treinta días. De esta forma mantenemos los datos recientes disponibles con el máximo de información para su consumo en un plazo breve desde su producción pero no sobrecargamos el Gatway en este sentido. Podríamos situar esta base de datos en otro servidor dedicado de la propia red sin ningún inconveniente pero con la limitación que hemos impuesto y el consumo objetivo que esperamos de la información almacenada no parece necesario mover la base de datos a una infraestructura externa al propio Gateway. Denominamos a la base de datos **iot** y estará compuesta por cinco tablas y dos vistas:

- **protocol**: tabla que almacena los diferentes protocolos soportados
- **location**: tabla que almacena las localizaciones
- **node**: tabla que almacena los nodos
- **sensor**: tabla que almacena los sensores
- **data**: tabla que almacena los datos procedentes de las lecturas
- **sensor_full**: vista que aglutina toda la información de los sensores
- **data_full**: vista que aglutina las lecturas con los datos más relevantes de los sensores

La estructura de la base de datos está reflejada en el siguiente esquema:

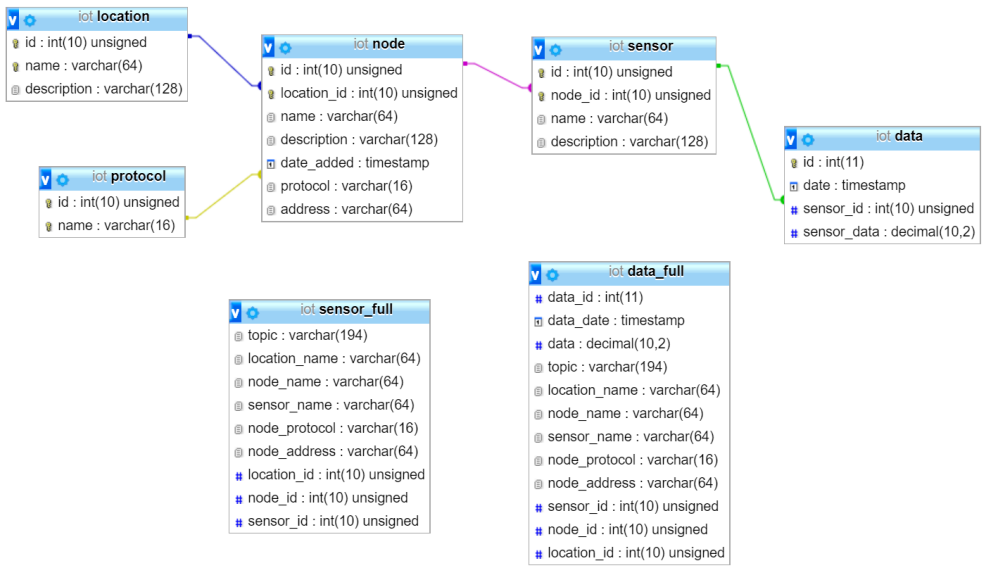


Ilustración 28: Esquema de la base de datos SQL

MONOGODB

En combinación con la base de datos relacional vamos a usar una base de datos no relacional para almacenar masivamente los datos recogidos por los sensores y poder explotarlos después en distintas aplicaciones. En este caso no establecemos limitación temporal alguna para el almacenamiento de los datos. Nos hemos decantado por MongoDB debido, entre otros factores, a sus características, popularidad y amplio soporte. El almacenamiento de las lecturas en formato documento que ofrece MongoDB nos permite guardar diferentes tipos de datos, incluso textuales, y permite que varíen a lo largo del tiempo. Esto aporta una flexibilidad que las bases de datos SQL no nos permiten tener.

Como hemos mencionado, MongoDB almacena los datos en colecciones de documentos. Una colección es similar a una tabla en una base de datos relacional. Con la misma analogía, un documento equivaldría a un registro o fila de la tabla. Los documentos se almacenan en formato BSON, un formato de JSON binario. En nuestro caso, para almacenar los datos usaremos una colección denominada **iot**. El formato de documentos de la colección que utilizaremos será el siguiente:

```
{
   "TOPIC": "<topic>",
   "VALUE": <value>,
   "DATETIME": <timestamp>
}
```

Ilustración 29: Formato de la colección iot en MongoDb

Aunque la Raspberry PI es capaz de ejecutar una instancia de MongoDB sin problemas, para -de nuevo- no sobrecargar el Gateway con ello y aprovechar mejor la capacidad y potencia de este tipo de base de datos hemos optado por utilizar el cluster en la nube que ofrece de forma gratuita el servicio MongoDB Atlas en formato DBaaS. El elegir un servicio desplegado localmente en el propio Gateway, en la red local o en la nube resulta transparente para la aplicación y bastará con cambiar la cadena de conexión para configurar la opción por la que nos decantamos.

Cluster MongoDB Atlas

Para crear la instancia en MongoDB Atlas nos registramos en su web *https://www.mongodb.com/cloud/atlas* y acto seguido, siguiendo los pasos de un asistente, podemos crear el primer cluster se servidores gratuito. Al cluster que creamos lo denominamos **UOC**. El servicio cuenta con un panel de control desde el que podemos ver el estado de nuestro cluster y controlar un gran número de aspectos del mismo. El siguiente paso es crear un usuario para la base

de datos y restringir o no su acceso a una IP o rango de IPs concreto. Hecho esto ya podemos obtener la cadena de conexión en el propio panel de control que incorporaremos a la configuración de la aplicación del Gateway.

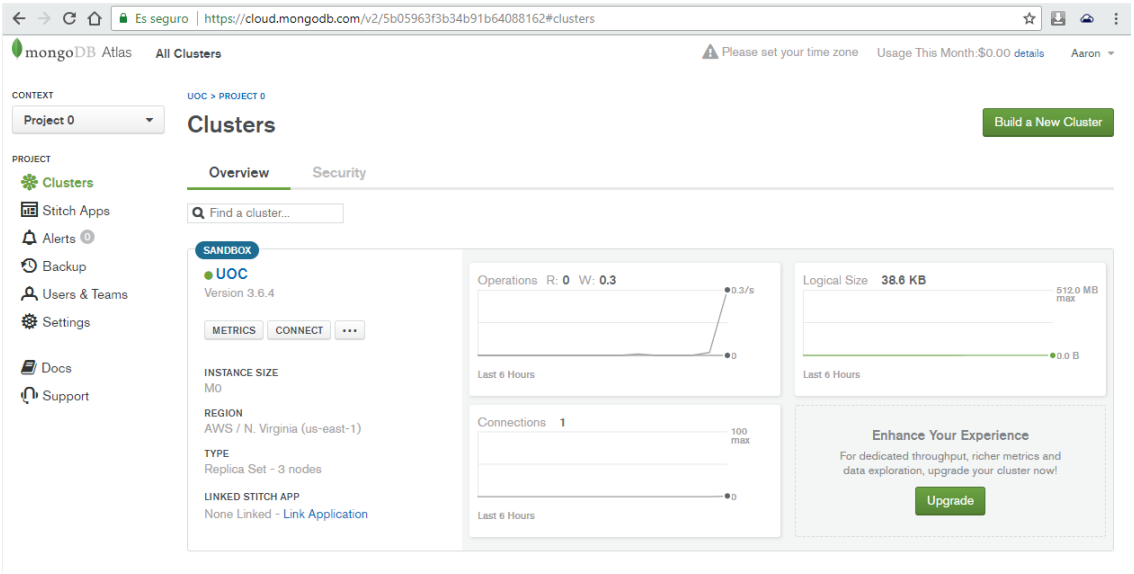


Ilustración 30: Panel de control de cluster MongoDB Atlas en la nube

La clase encargada de gestionar la persistencia es la clase Db. Esta clase presenta una dependencia con las librerías Mysql y Pymongo Esta clase tiene un punto de entrada único que es el método save_sensor_data. Este método crea dos hilos distintos con dos funciones worker, uno para la base de datos SQL y otro para la NOSQL. La función que gestiona la persistencia en MariaDB hace primero una consulta para averiguar si el sensor existe en la base de datos. En caso afirmativo hace la inserción de la lectura en la base de datos y termina. Si, por el contrario, el sensor no consta en la base de datos comienza un proceso que recorre el tópico en profundidad pasando jerárquicamente del parámetro al nodo y del nodo a la localización. Si no se produce un acierto en el nodo consulta la localización y si tampoco se produce un acierto comienza a realizar las inserciones. De esta manera cualquier nodo que no haya sido visto previamente por el Gatway queda registrado y las lecturas que produzca y se guarden en la base de datos están vinculadas a un nodo conocido e identificado.

En el caso de la función encargada de la persistencia en MongoDB, sólo se encarga de registrar las lecturas, su tópico y el timestamp de la lectura. Por ello lanza la inserción en el servidor sin verificar nada más. Se presenta a continuación el diagrama de actividad de la capa de persistencia, gestionado por la clase DB

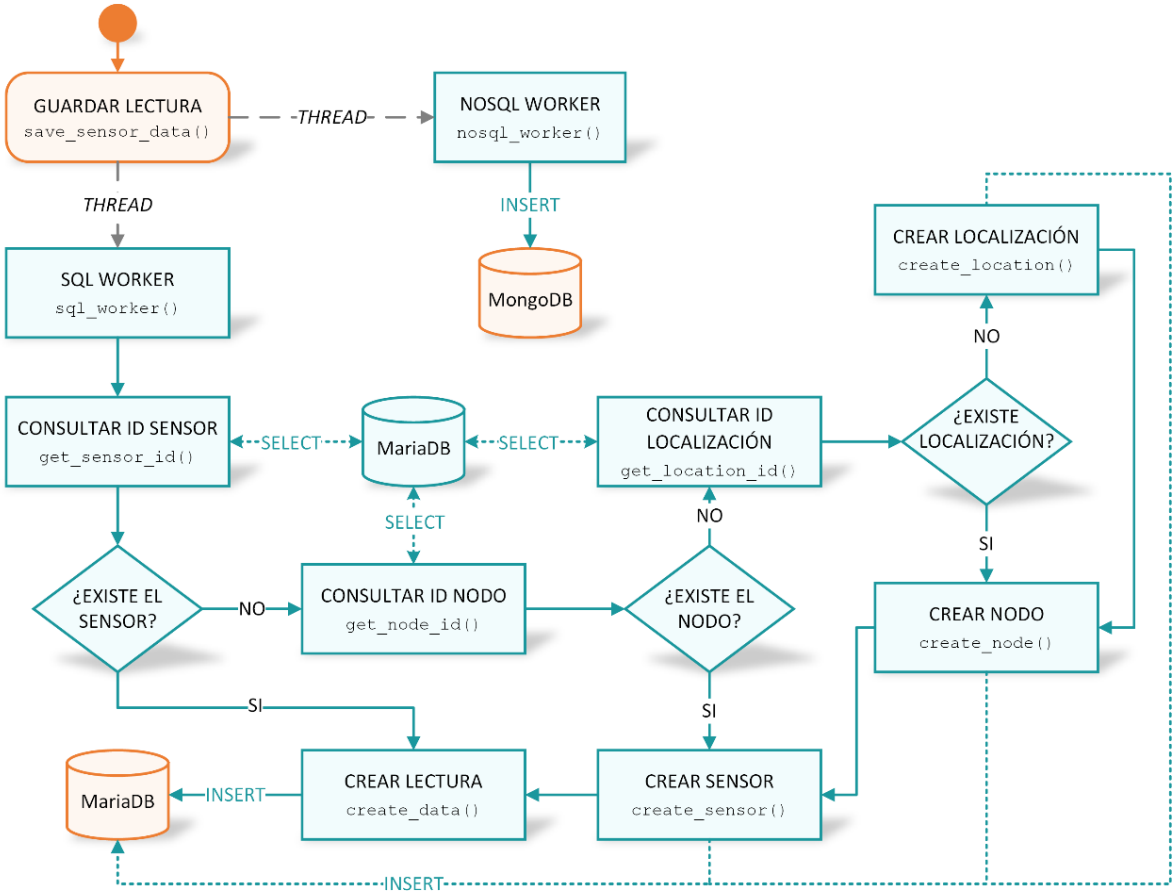


Ilustración 31: Diagrama de actividad de la capa de persistencia

3.2.2.5 CAPA DE INTEROPERABILIDAD

SERVICIO CLOUD

Siguiendo con la capa de interoperabilidad tenemos el servicio de publicación en la nube. La finalidad de este servicio es disponer de los datos recopilados por los diferentes nodos en una plataforma online especializada en el almacenamiento la visualización, la explotación y la gestión remota de este tipo de información y que ofrezca servicios adicionales como establecimiento de alarmas y notificación en tiempo real así como motores de análisis matemática. La plataforma seleccionada, SensorCloud, ofrece toda esta funcionalidad y más a un coste asequible en función del número de lecturas y transacciones mensuales.

La jerarquía ontológica de SensorCloud difiere ligeramente de la que se ha propuesto para nuestro Gateway con lo que hay que realizar una adaptación a la misma. En la siguiente tabla se recogen las equivalencias y nomenclatura.

ONTOLOGÍA	NIVEL 1	NIVEL 2	NIVEL 3
SENSORCLOUD	DEVICE	SENSOR	CHANNEL
GATEWAY	GATEWAY	LOCATION + NODE	PARAMETER

Tabla 19: Equivalencia de entidades jerárquicas con SensorCloud

En el plan gratuito SensorCloud sólo ofrece un dispositivo por lo que nuestra jerarquía lo equipara a nuestro Gateway y a partir de él desarrolla la ontología. La interacción con SensorCloud se lleva a cabo mediante una Api REST bastante sencilla en la que una vez negociada la autenticación se pueden llevar a cabo diferentes operaciones sobre las diferentes entidades. El propio servicio ofrece una guía para la implementación en Python que es la que adaptamos a nuestro caso.

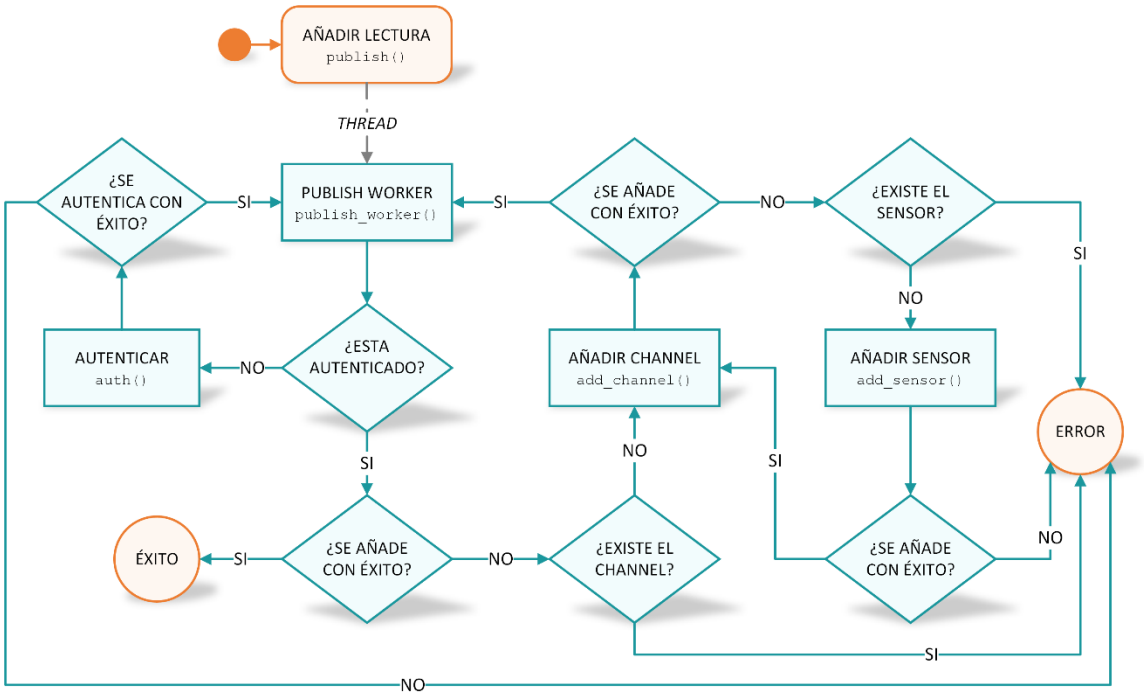


Ilustración 32: Diagrama de actividad de la publicación en servicio Cloud

La clase encargada de gestionar la interoperabilidad con el servicio SensorCloud es la clase Cloud. El punto de entrada único de esta clase es la función *publish* para añadir una lectura. El proceso que lleva a cabo es similar al de la capa de persistencia en cuanto a la recursividad jerárquica que lleva a cabo si el nodo no existe previamente. La función lanza un hilo con la función worker que lo primero que revisará es la autenticación. Superado este paso el proceso seguirá avanzando verificando que existen las entidades asociadas a la lectura que se pretende guardar. Todas aquellas entidades que no existan se irán creando hasta que finalmente nos permita almacenar la lectura y el hilo se extinga.

REST API

En la capa de servicios de interoperabilidad vamos a crear un servicio web basado en arquitectura REST que proporcione una API para la consulta y actualización de recursos. Denominamos recursos a cada uno de los objetos o entidades definidas jerárquicamente en la ontología, esto es: localización, nodo y parámetro. Estos recursos están identificados por una URI única que permite el acceso a los mismos. Sobre estos recursos y en función del tipo de entidad que representen permitiremos operaciones de consulta o lectura (GET) y de actualización (POST). Como operaciones complementarias se implementarán operaciones de listado y descubrimiento. Nuestra API no permitirá operaciones de creación (PUT) ni eliminación (DELETE) de recursos puesto que este tipo de operaciones quedan reservadas al backend del propio Gateway y no son admisibles del lado cliente.

OPERACIÓN	METODO	URI	PARAMETRO
DISCOVER	GET	/API/V1	
DISCOVER	GET	/API/V1/*<LOCATION>*	
DISCOVER	GET	/API/V1/*<LOCATION>*/*<NODE>*	
READ LAST	GET	/API/V1/*<LOCATION>*/*<NODE>*/*<PARAMETER>*	
READ NUM	GET	/API/V1/*<LOCATION>*/*<NODE>*/*<PARAMETER>*/*<1..1000>*	
READ NOW	GET	/API/V1/*<LOCATION>*/*<NODE>*/*<PARAMETER>*/STATE	
UPDATE	POST	/API/V1/*<LOCATION>*/*<NODE>*/*<PARAMETER>*/	:VALUE

Tabla 20: Relación de operaciones, métodos y URIs de la Api REST

La funcionalidad y responsabilidad de la API REST estará dividida en dos partes: un front-end gestionado por un script PHP corriendo sobre un servidor Apache y un back-end gestionado por la aplicación en Python. La comunicación entre ambas se realizará por medio de stream sockets de Unix. El script PHP se ejecuta sobre un servidor Apache configurado para que envíe todas las peticiones de URLS al mismo script. El script descompone la URL de forma que si hace referencia a un recurso válido puede avanzar. En función del método y tipo de petición el script hará una consulta directa a la base de datos MaríaDB para entregar información histórica o utilizará el socket para transmitir la solicitud a la aplicación en Python. Por su parte la aplicación encapsula la funcionalidad vinculada a la Api en la clase REST. Esta clase se encarga de la gestión multi-hilo del socket invocando una función worker que permanece a la espera de una conexión. Cuando llega la petición el mensaje es formateado, se le asigna un id único y es enviado al enrutador de la aplicación que envía la petición al nodo correspondiente. Acto seguido se almacena el id como clave de una entrada en un diccionario global denominado async_response. A la par se lanza otra función worker que se encarga

de consultar cíclicamente si existe una respuesta por parte del nodo hasta que esta llega o se produce un timeout. Si se produce una respuesta esta se almacena en el diccionario mencionado. Cundo la función worker que hace la consulta lo detecte enviará la respuesta a través del socket. Llegue o no la respuesta la entrada que tiene el id por clave es eliminado del diccionario. Devuelto el control al script PHP este entregará una respuesta válida en formato JSON o devolverá un error 404 si el recurso no existe o no hay respuesta del nodo y un error 500 si existe algún problema con el socket.

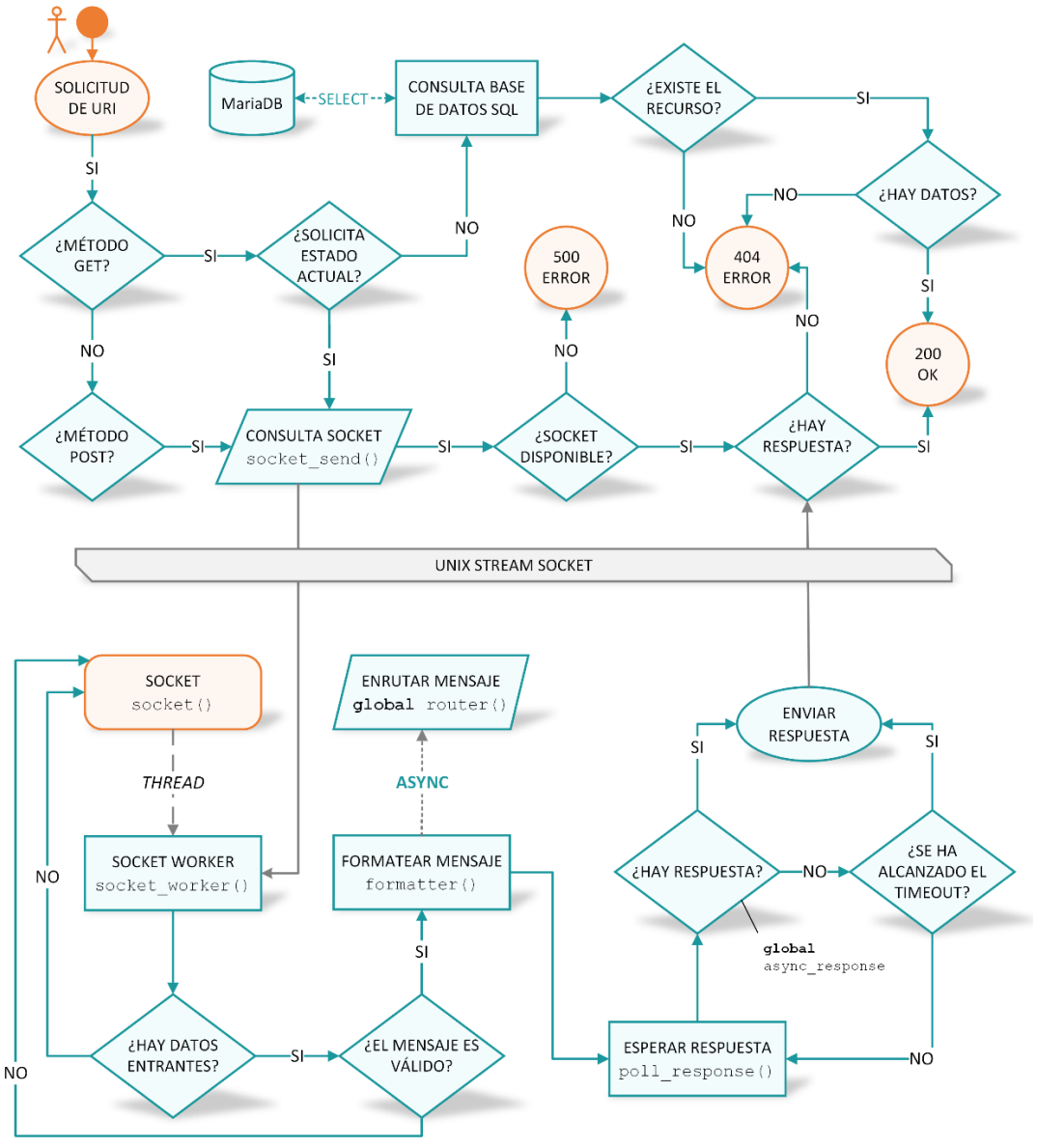


Ilustración 33: Diagrama de actividad de la Api REST

3.2.3 PRUEBAS FINALES

Una vez implementado todo el software y tras haber realizado el proceso de corrección de errores se lleva a cabo una batería de pruebas intensiva para verificar el correcto funcionamiento del sistema. Para la realización de las pruebas finales del sistema se mantienen encendidos todos los nodos del entorno de prototipado y se activan todos los servicios del Gateway durante, al menos, doce horas para simular un entorno real. Para verificar los resultados se utiliza la librería logger de Python y se diseminan marcadores por todo el código de la aplicación que nos ayuden a trazar la ejecución. El resultado se almacena en un archivo. Tras la ejecución de las pruebas un examen a conciencia del log vemos que aparentemente el funcionamiento ha sido correcto: los ocho nodos han mantenido la conexión y han publicado sus lecturas regularmente.

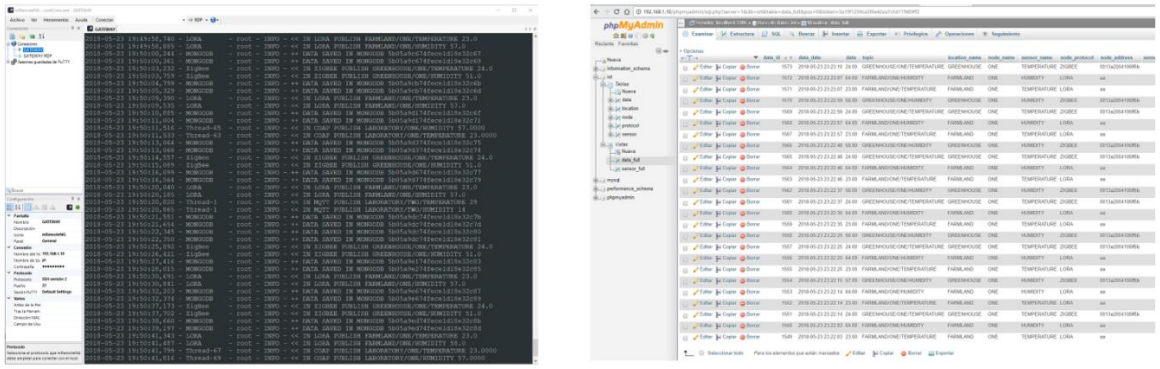


Ilustración 34: Salida por consola del logger del Gatway y muestra de la base de datos MariaDB

También verificamos que las lecturas se han guardado correctamente en la base de datos local MaríaDB y en MongoDB Atlas. Los resultados han sido igualmente satisfactorios con el servicio SensorCloud ya que se observa que las lecturas se han reportado regularmente. Finalmente realizamos una batería de pruebas en las que ponemos a pruebas la API REST. Para ello lanzamos sobre cada nodo diversas consultas históricas que son devueltas con éxito. Para finalizar ponemos a prueba las peticiones de valor actual de un sensor y la activación y desactivación de un actuador, en nuestro caso el led que posee cada nodo. Para lanzar las peticiones utilizamos la extensión “Advanced Rest Client” de Crhome. El resultado de todas las pruebas es satisfactorio y no se produce ningún fallo aparente. Tras esto se revisa de nuevo la salida del log de la aplicación para verificar que la ejecución ha sucedido según lo planeado y no se detectan fallos ni incongruencias. Por último comprobamos la carga de memoria y procesador de la Raspberry Pi con los ocho nodos conectados y vemos que apenas existe impacto. Con todas las pruebas realizadas se puede concluir que el sistema funciona correctamente.

4

ANÁLISIS DE RESULTADOS

4.1 PRODUCTO FINAL

Como broche final al producto terminado diseñamos e imprimimos en 3D una carcasa en PLA realizada específicamente para albergar los componentes del prototipo del Gateway.

Ilustración 35: Prototipo final del Gateway con carcasa impresa en 3D

4.2 VALORACIÓN ENCONÓMICA

En la siguiente tabla se refleja el detalle del coste de los componentes del Gateway cuyo elemento central es una Raspberry PI 3

Uds.	*COMPONENTE*	*PRECIO Ud.*	*TOTAL*
1	Raspberry Pi 3	30 €	30 €
1	Arduino Shield for Raspberry Pi	21 €	21 €
1	Escudo Draguino Lora	24 €	24 €
1	XBee S2	25 €	25 €
1	Cables	3 €	3 €
1	Protoboard Grande	6 €	6 €
	TOTAL		***109 €***

Tabla 21: Coste de componentes de hardware del Gateway

A continuación se detallan los costes del entorno de prototipado formado por los ocho nodos Arduino con sus diferentes placas, sensores, escudos e interfaces de comunicaciones.

Uds.	*COMPONENTE*	*PRECIO Ud.*	*TOTAL*
2	Arduino Uno (clon)	7 €	14 €
2	Arduino Mega 2560 (clon)	12 €	24 €
2	Arduino Leonardo (clon)	9 €	18 €
2	Ardunio Nano (clon)	8 €	16 €
3	Escudo Xbee	8 €	24 €
2	Escudo I/O Nano	6 €	12 €
1	Escudo I/O Mega	21 €	21 €
2	Escudo Draguino Lora	24 €	48 €
2	ZigBee S2	25 €	50 €
4	WiFiBee ESP8266	9 €	36 €
8	DHT11	2 €	16 €
8	Led	0,10 €	1 €
8	Resistencia	0,10 €	1 €
1	Cableado protoboard	3 €	3 €
6	Protoboard Pequeñas	4 €	24 €
2	Protoboard Grandes	6 €	12 €
	TOTAL		***320 €***

Tabla 22: Coste de componentes de hardware del entorno de prototipado

El coste total del entorno de prototipado de 320 € es una cifra nada despreciable. Hay que considerar, no obstante, que lo que más encarece el importe total son las interfaces de comunicaciones. El coste medio por nodo es de alrededor de 40 €. Este coste bajaría muy

fácilmente de los 10 € si se produjesen las PCBs con los componentes integrados en masa (500 unidades) en lugar del formato de interfaces y escudos de los prototipos.

HORAS	PERFIL	TASA	TOTAL
30	Gestor de proyecto	30 €	900 €
60	Arquitecto / Analista	25 €	1.500 €
180	Desarrollador	20 €	3.600 €
60	Documentador Técnico	15 €	900 €
330	**TOTAL**		**6.900 €**

Tabla 23: Coste de recursos humanos del proyecto

El coste en recursos humanos se ha calculado en función de las horas de dedicación aproximadas de los diferentes roles o perfiles que se han ido asumiendo en la realización del proyecto. Las tasas elegidas son cercanas a las del mercado laboral en ciudades grandes.

4.3 VIABILIAD DEL PRODUCTO

Si ponemos en perspectiva la información que tenemos se puede tener una idea bastante clara de la viabilidad del producto final. La relación coste/beneficio del Gateway desde un punto de vista funcional y económico lo sitúan en una posición mejor que la de otros productos comerciales que hay en el mercado. Su coste es bastante bajo, pudiendo tener una versión optimizada del equipo por menos de 60 €. La disponibilidad de los componentes de hardware es muy elevada y todo el software se basa en licencias abiertas y/o sin conste. Las plataformas sobre las que está montada la solución (Arduino y Raspberry PI) permiten la total configuración, modificación o incorporación de software y hardware lo que aporta una flexibilidad muy grande al producto.

A nivel funcional la solución cubre un amplio espectro de protocolos y servicios y resulta sencillo integrarla en un sistema o red existente para que interactúe con otros equipos y aplicaciones. Las pruebas realizadas parecen indicar que el producto es suficientemente robusto para un entorno real de explotación y la carga de trabajo con los ocho nodos del entorno de prototipado funcionando y todos los servicios activos apenas han supuesto impacto al procesador o memoria con lo que, a falta de pruebas concretas, podemos prever que el sistema escale sin problemas por encima de cincuenta nodos, lo que debería ser suficiente en gran cantidad de escenarios.

Su único hándicap evidente es que no se trata de un producto plug-and-play ya que hay tener unas mínimas competencias técnicas para desplegar la solución e implementar la programación de los nodos. A parte de esto, el resto de procesos apenas requiere intervención del usuario con lo que, en

este aspecto, se posiciona igual o mejor que otros productos comerciales. Teniendo en cuenta el coste, la disponibilidad, la flexibilidad, la funcionalidad, la fiabilidad y la escalabilidad podemos concluir que se trata de un producto plenamente viable y parece una solución idónea para una pequeña o mediana empresa o proyecto.

4.4 PUNTOS DE MEJORA

Debido a las restricciones temporales y la envergadura del proyecto hay una seria de aspectos que han quedado fuera del alcance del mismo así como otros puntos que se han identificado a lo largo de su desarrollo y cuya implementación mejoraría el producto final. Los más significativos son los siguientes:

Seguridad: no se han implementado los mecanismos de seguridad (autenticación, cifrado…) que deberían estar presentes en diferentes puntos del desarrollo y es un aspecto crítico que no debería pasarse por alto en un entorno de explotación real.

Pruebas: a pesar de que se ha sometido al sistema a una serie de pruebas consistentes no se han desarrollado test unitarios que permitieran realizar las pruebas de manera sistemática y exhaustiva.

Modularidad: con una pequeña refactorización del código se podría conseguir un sistema más modular que permitiese añadir o quitar conectores (y crearlos nuevos) en función de las necesidades particulares de cada entorno.

Interfaz web: a pesar de que el sistema tiene pocos requisitos de configuración sería interesante dotarlo de una interfaz web que permitiese modificar los parámetros de configuración, autenticación y permisos una vez implementados los mecanismos de seguridad, así como la edición de nodos y sensores lo que permitiría agregar información como unidades y metadatos.

4.5 CONCLUSIONES Y LECCIONES APRENDIDAS

Este trabajo ha supuesto un verdadero desafío en distintos ámbitos, siendo el mayor de todos, sin lugar a dudas, conseguir que los diferentes sistemas, tecnologías, protocolos, librerías, aplicaciones y lenguajes trabajasen entre ellos como engranajes de un mecanismo bien ajustado. Además de esto hay que mencionar las dificultades de trabajar con hardware de todo tipo y procedencia y sus implicaciones: revisar conexiones, controlar compatibilidades, actualizar firmwares… Ambos aspectos han hecho que la envergadura real del proyecto y la dimensión del trabajo planificado fuesen

bastante superiores a lo previsto inicialmente. Cumplir con la planificación inicial ha implicado aumentar las horas de dedicación previstas significativamente y ha supuesto un sobreesfuerzo importante.

He tenido que trabajar y poner en práctica múltiples competencias y disciplinas: gestión de proyectos, valoración de soluciones tecnológicas, análisis de problemas y capacidad de abstracción, investigación de nuevas tecnologías, diseño y desarrollo de aplicaciones, integración y reutilización de soluciones, diseño y uso de bases de datos, redes y sistemas de comunicaciones, fundamentos de electrónica, arquitectura de computadores y sistemas operativos.

Aparte de esto, el proyecto ha sido un reto apasionante, divertido, instructivo y enriquecedor. He podido aprender tanto que se hace difícil tratar de mencionarlo todo pero lo más importante, sin duda, es a no temer la falta de conocimiento o experiencia y confiar en tener una base suficientemente sólida como para utilizar cualquier herramienta al alcance. El hecho de tener que enfrentarte a un problema real de cierta envergadura, no una situación guiada ni en un entorno controlado o con resultados predecibles, hace que de verdad pongas a prueba las competencias desarrolladas y los conocimientos adquiridos a lo largo de los últimos años.

5

BIBLIOGRAFÍA

Carsten Bormann. CoAP. [en línea]. Carsten Bormann • TZI • 2014–2016. [fecha de consulta: 10/03/2018]. Disponible en: http://coap.technology

Zigbee Alliance. Varias secciones. [en línea]. 2017 Zigbee Alliance. [fecha de consulta: 10/03/2018]. Disponible en: http://www.zigbee.org

LoRa Alliance™. Varias secciones. [en línea]. 2018 LoRa Alliance™. [fecha de consulta: 11/03/2018]. Disponible en: https://lora-alliance.org

Mqtt.org community members. Varias secciones. [en línea]. 2018 GitHub, Inc. [fecha de consulta: 10/03/2018]. Disponible en: http://mqtt.org

Arduino Company. Varias secciones. [en línea]. 2018 Arduino. [fecha de consulta: 11/03/2018]. Disponible en: https://www.arduino.cc

RASPBERRY PI FOUNDATION. Varias secciones. [en línea]. RASPBERRY PI FOUNDATION 2015. [fecha de consulta: 12/03/2018]. Disponible en: https://www.raspberrypi.org

Python Software Foundation. paho-mqtt 1.3.1. Varias secciones. [en línea]. Oct 9, 2017. [fecha de consulta: 21/03/2018]. Disponible en: https://pypi.org/project/paho-mqtt

Eclipse Foundation. Eclipse Mosquitto An open source MQTT broker. [en línea]. Eclipse Mosquitto™ 2018. [fecha de consulta: 23/03/2018]. Disponible en: https://mosquitto.org

MariaDB Foundation. Varias secciones. [en línea]. 2018 MariaDB Foundation. [fecha de consulta: 25/03/2018]. Disponible en: https://mariadb.org

mongoDB. Varias secciones. [en línea]. 2018 MongoDB, Inc. [fecha de consulta: 29/03/2018]. Disponible en: https://www.mongodb.com

jrotah. A Simple Example (Arduino, MQTT, m2m.io). [en línea]. Tumblr 2015. [fecha de consulta: 05/04/2018]. Disponible en: http://m2mio.tumblr.com/post/30048662088/a-simple-example-arduino-mqtt-m2mio

Marechal, Sander. A simple unix/linux daemon in Python. [en línea]. jejik.com 2007. [fecha de consulta: 07/04/2018]. Disponible en: http://www.jejik.com/articles/2007/02/a_simple_unix_linux_daemon_in_python

Hellmann, Doug. asyncio — Asynchronous I/O, event loop, and concurrency tools. [en línea]. pymotw.com 2016. [fecha de consulta: 08/04/2018]. Disponible en: https://pymotw.com/3/asyncio

Diaz, Roberto. Comunicación entre Raspberry PI y Arduino por USB. [en línea]. Blog 2013. [fecha de consulta: 13/04/2018]. Disponible en: http://rdiaz.es/blog/comunicacion-entre-raspberry-pi-y-arduino-por-usb

Rijware Inc. Connecting the Last 100 Metres to the Cloud using Mosquitto. [en línea]. rijware 2014. [fecha de consulta: 14/04/2018]. Disponible en: http://rijware.com/mosquitto-bridge-to-secure-mqtt-broker

Bruce, James. Getting Started with OpenHAB Home Automation on Raspberry Pi. [en línea]. MakeUseOf 2015. [fecha de consulta: 16/04/2018]. Disponible en: https://www.makeuseof.com/tag/getting-started-openhab-home-automation-raspberry-pi

Kovatsch, Matthias y Vermillard, Julien. Hands-on with CoAP. [en línea]. Google Docs 2014. [fecha de consulta: 17/04/2018]. Disponible en: http://goo.gl/LLQ03w

Jaffey, Toby. MQTT and CoAP, IoT Protocols. [en línea]. Eclipse Org 2014. [fecha de consulta: 18/04/2018]. Disponible en: https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php

Stansbery, James. MQTT and CoAP: Underlying Protocols for the IoT. [en línea]. ElectronicDesign 2015. [fecha de consulta: 18/04/2018]. Disponible en: http://www.electronicdesign.com/iot/mqtt-and-coap-underlying-protocols-iot

Arouje, Sony. MQTT Communication with Arduino using ESP8266 ESP-01. [en línea]. Sony Arouje web 2016. [fecha de consulta: 18/04/2018]. Disponible en: https://sonyarouje.com/2016/03/15/mqtt-communication-with-arduino-using-esp8266-esp-01

HiveMQ. MQTT Essentials: Part 1 – Introducing MQTT. [en línea]. HiveMq Enterprise MQTT BROKER 2016. [fecha de consulta: 19/04/2018]. Disponible en: https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt

Johnson, Wes. MQTT manages devices, collects data. [en línea]. INDUSTRIAL EMBEDDED SYSTEMS 2010. [fecha de consulta: 20/04/2018]. Disponible en: http://industrial.embedded-computing.com/articles/mqtt-devices-collects-data

Smith, Oliver. MQTT, Mosquitto and PHP. [en línea]. oliversmith.io 2010. [fecha de consulta: 21/04/2018]. Disponible en: https://oliversmith.io/technology/2010/02/26/mqtt-mosquitto-and-php

Koster, Michael. MQTT - REST Bridge using the Smart Object API. [en línea]. Slideshare 2013. [fecha de consulta: 21/04/2018]. Disponible en: https://es.slideshare.net/michaeljohnkoster/mqtt-rest-bridge

Landoni, Boris. Creating a Network of Nodes with LoRa Shield. [en línea]. OpenElectronics 2016. [fecha de consulta: 22/04/2018]. Disponible en: https://www.open-electronics.org/creating-a-network-of-nodes-with-lora-shield

Glozic, Dejan. REST and MQTT: Yin and Yang of Micro-Service APIs. [en línea]. Dejan Glozic web 2014. [fecha de consulta: 22/04/2018]. Disponible en: https://dejanglozic.com/2014/05/06/rest-and-mqtt-yin-and-yang-of-micro-service-apis

User Alex. Basic Pi <-> Arduino Communication over USB. [en línea]. seeedstudio 2009. [fecha de consulta: 23/04/2018]. Disponible en: https://community.seeedstudio.com/project_detail.html?id=168

Janakiram MSV. Tutorial: Prototyping a Sensor Node and IoT Gateway with Arduino and Raspberry Pi – Part 1. [en línea]. thenewstack 2015. [fecha de consulta: 28/04/2018]. Disponible en: https://thenewstack.io/tutorial-configuring-a-sensor-node-and-iot-gateway-to-collect-and-visualize-data-part-2

Janakiram MSV. Tutorial: Configuring a Sensor Node and IoT Gateway to Collect and Visualize Data — Part 2. [en línea]. thenewstack 2015. [fecha de consulta: 25/04/2018]. Disponible en: https://thenewstack.io/tutorial-configuring-a-sensor-node-and-iot-gateway-to-collect-and-visualize-data-part-2

Rijware Inc. Using Arduinos to prototype IoT solutions. [en línea]. rijware 2014. [fecha de consulta: 29/04/2018]. Disponible en: http://rijware.com/using-arduinos-to-prototype-iot-solutions

6

ANEXOS

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

APLICACIONES UTILIZADAS

IDES Y ENTORNOS DE DESARROLLO

Arduino IDE 1.8.5	https://www.arduino.cc/en/main/software
Notepad ++ 7.5.6	https://notepad-plus-plus.org
JetBrains CLion 2018.1.3	https://www.jetbrains.com/clion/
JetBrains PyCharm 2018.1.3	https://www.jetbrains.com/pycharm

PROGRAMACIÓN DE FIRMWARE

Digi XCTU 6.3.14	http://www.digi.com/support
Vowstar Firmware Programmer	http://www.vowstar.com

SOFTWARE DE PRUEBAS

Copper 1.0.1	https://github.com/mkovatsc/Copper
Advanced Rest Client 10.0.12	https://github.com/jarrodek/ChromeRestClient

SOFTWARE DE COMUNICACIONES

mRemoteNG 1.75.7012.16814	https://mremoteng.org/
WinSCP 5.3	https://winscp.net/eng/download.php

OTRO SFOTWARE

Bouml 7.5	https://www.bouml.fr/
GanttProject 2.8.8	https://www.ganttproject.biz/
Microsoft Visio 16.0.9330	https://products.office.com/es-es/visio

LIBRERÍAS Y DEPENDENCIAS

NODOS

LIBRERÍA DHT

La librería DHT de Adafruit permite efectuar lecturas sobre un sensor de la familia DHT (11 o 22) de manera muy sencilla. Esta librería requiere la librería Adafruit Sensor Driver

- Librería: DHT sensor library 1.3.0
- Autor: Adafruit
- Licencia: MIT License
- Url: https://github.com/adafruit/DHT-sensor-library

LIBRERÍA XBEE

XBee-Arduino es una librería que permite a las distintas placas Arduino la comunicación mediante módulos XBee en modo API. Soporta la transmisión de la mayoría de tipos de paquetes: TX/RX, AT, I/O Samples y Modem Status.

- Librería: XBee-Arduino 0.6.0
- Autor: Andrew Rapp
- Licencia: GNU General Public License v3
- Url: https://github.com/andrewrapp/xbee-arduino

LIBRERÍA ARDUINO-LORA

Arduino-LoRa es una librería que permite a Arduino enviar y recibir datos mediante módulos inalámbricos LoRa. Esta librería es compatible con los chips de la familia Semtech SX1276/77/78/79.

- Librería: Arduino-LoRa
- Autor: Sandeep Mistry
- Licencia: MIT License
- Url: https://github.com/sandeepmistry/arduino-LoRa

LIBRERÍA WIFIESP

WifiEsp es una librería que permite a las distintas placas Arduino la conexión WiFi mediante el uso del módulo ESP8266. La librería se comporta como una capa de abstracción sobre los comandos AT

que implementa el firmware del módulo. Su API es compatible y casi idéntica a la de las librerías estándar WiFi y Ethernet de Arduino lo que permite su integración directa y sin modificaciones con muchas otras librerías. Para su correcto funcionamiento necesita una versión del firmware AT igual o superior a la 0.25.

- Librería: WifiEsp 2.2.1
- Autor: Bruno Portaluri
- Licencia: GNU General Public License v3
- Url: https://github.com/bportaluri/WiFiEsp

LIBRERÍA PUBSUBCLIENT

Pubsubclient es una librería que convierte a Arduino en un sencillo cliente MQTT. La librería permite el intercambio de mensajes mediante publicación y subscripción con un servidor o bróker MQTT. Tiene como dependencia la librería WiFi o Ethernet de Arduino

- Librería: Pubsubclient 2.4
- Autor: Nicholas O'Leary
- Licencia: MIT License
- Url: https://github.com/knolleary/pubsubclient

LIBRERÍA MICROCOAP

Microcoap es una librería que convierte a Arduino en un sencillo servidor CoAP. La librería permite el intercambio de mensajes con un cliente.

- Librería: Microcoap
- Autores: Toby Jaffey y Pilgrim Beart
- Licencia: MIT License
- Url: https://github.com/1248/microcoap

GATEWAY

LIBRERÍA XBEE

Esta librería permite la comunicación con XBee en modo Api a través del puerto serie.

- Librería: XBee 2.3.2
- Autor: Varios - Niolabs
- Licencia: MIT License

- Url: https://github.com/niolabs/python-xbee

LIBRERÍA PYSERIAL

Esta librería permite la comunicación con XBee en modo Api a través del puerto serie.

- Librería: PySerial 3.2.1
- Autor: Chris Liechti
- Licencia: BSD license, (C) 2001-2017
- Url: https://github.com/pyserial/pyserial

BROKER MOSQUITTO

Mosquitto es un Broker ligero que implementa las versiones 3.1 y 3.1.1 de MQTT. Se trata de un Broker apropiado para SBCs

- Aplicación: Mosquitto 1.4.15
- Autor: Eclipse Foundation
- Licencia: Eclipse Distribution License 1.0 (BSD), Eclipse Public License 1.0
- Url: https://github.com/eclipse/mosquitto

LIBRERÍA PAHO

Esta librería provee en Python un cliente que permite la comunicación con un Broker MQTT que implemente las versiones 3.1 o 3.1.1. Ofrece capacidad de publicación y suscripción.

- Librería: Paho 1.3.1
- Autor: Eclipse Foundation
- Licencia: Eclipse Distribution License 1.0 (BSD), Eclipse Public License 1.0
- Url: https://github.com/eclipse/paho.mqtt.python

LIBRERÍA COAPTHON

Esta librería implementa en Python el protocolo CoAP como cliente y servidor

- Librería: CoAPthon 4.0
- Autor: G.Tanganelli, C. Vallati, E.Mingozzi, "CoAPthon: Easy Development of CoAP-based IoT Applications with Python", IEEE World Forum on Internet of Things (WF-IoT 2015)
- Licencia: MIT License
- Url: https://github.com/Tanganelli/CoAPthon

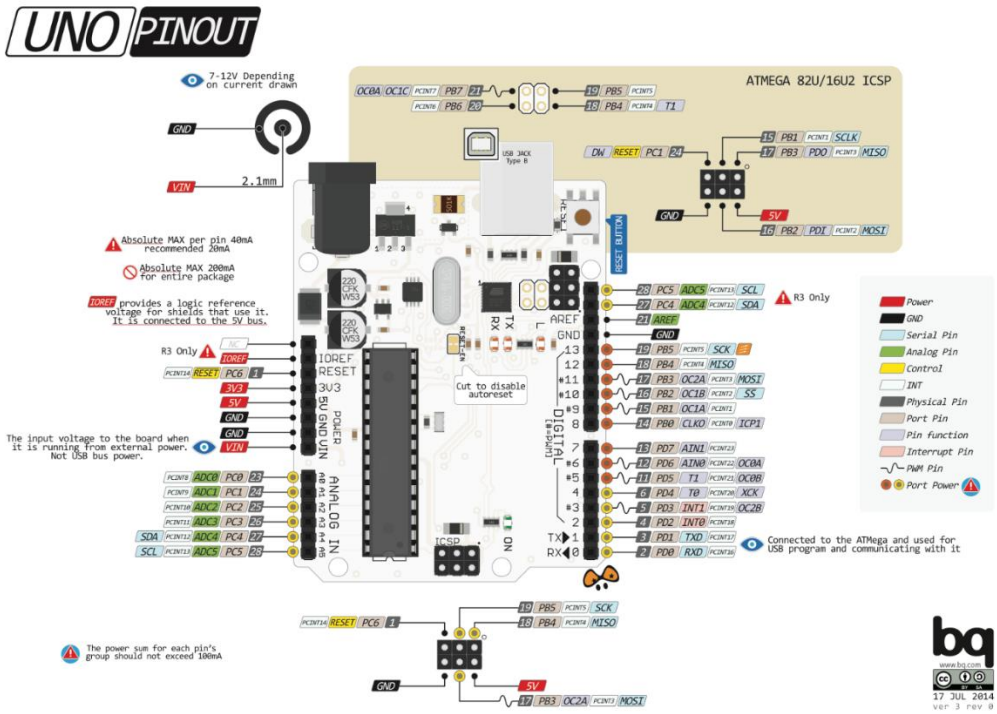
LIBRERÍA PYMONGO 3.6.1

Es una librería que implementa en Python el driver de la base de datos no relacional MongoDB

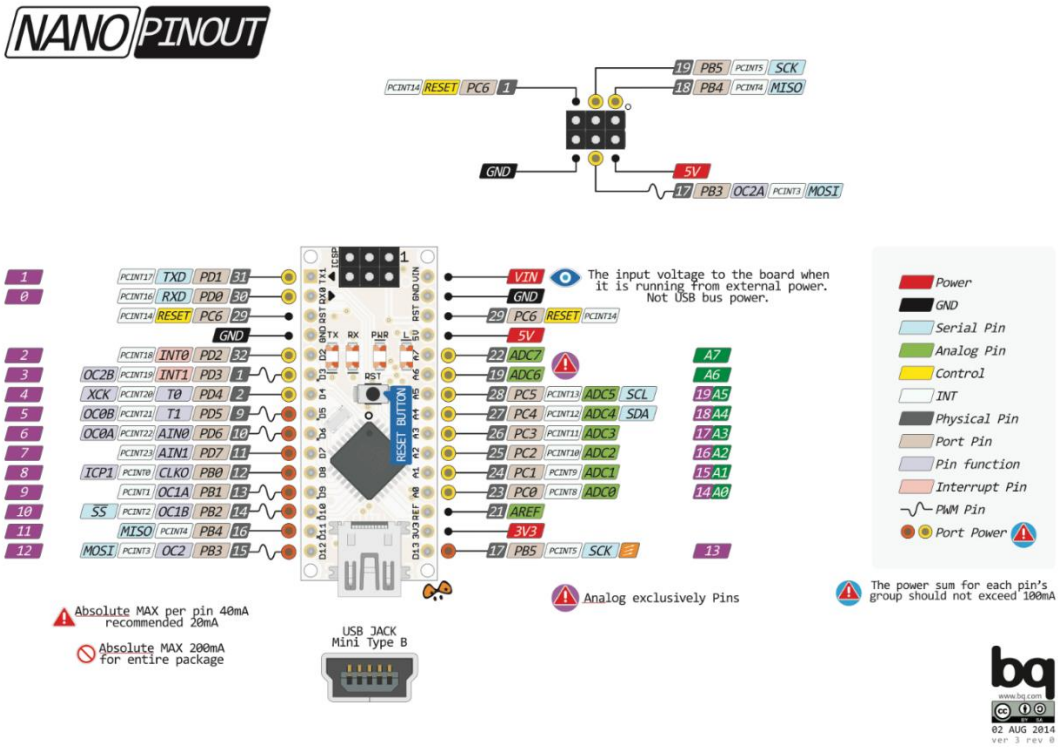
- Librería: Pymongo 3.61.1
- Autores: Mike Dirolf y Bernie Hackett
- Licencia: Apache
- Url: https://github.com/mongodb/mongo-python-driver

DIAGRAMAS DE HARDWARE

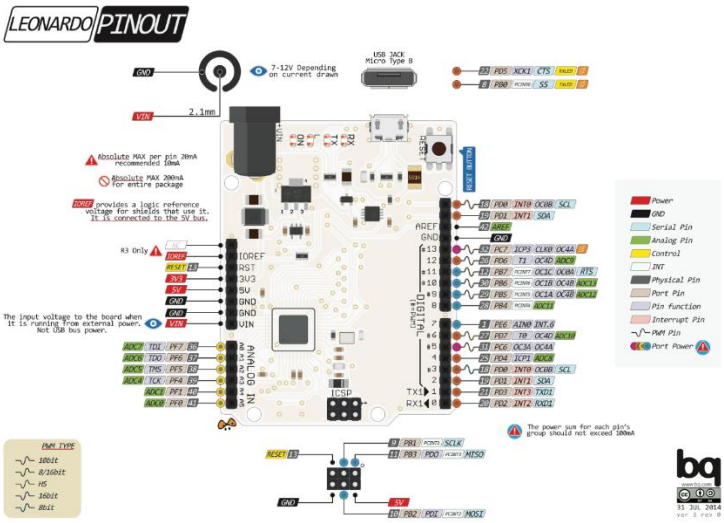
ARDUINO UNO



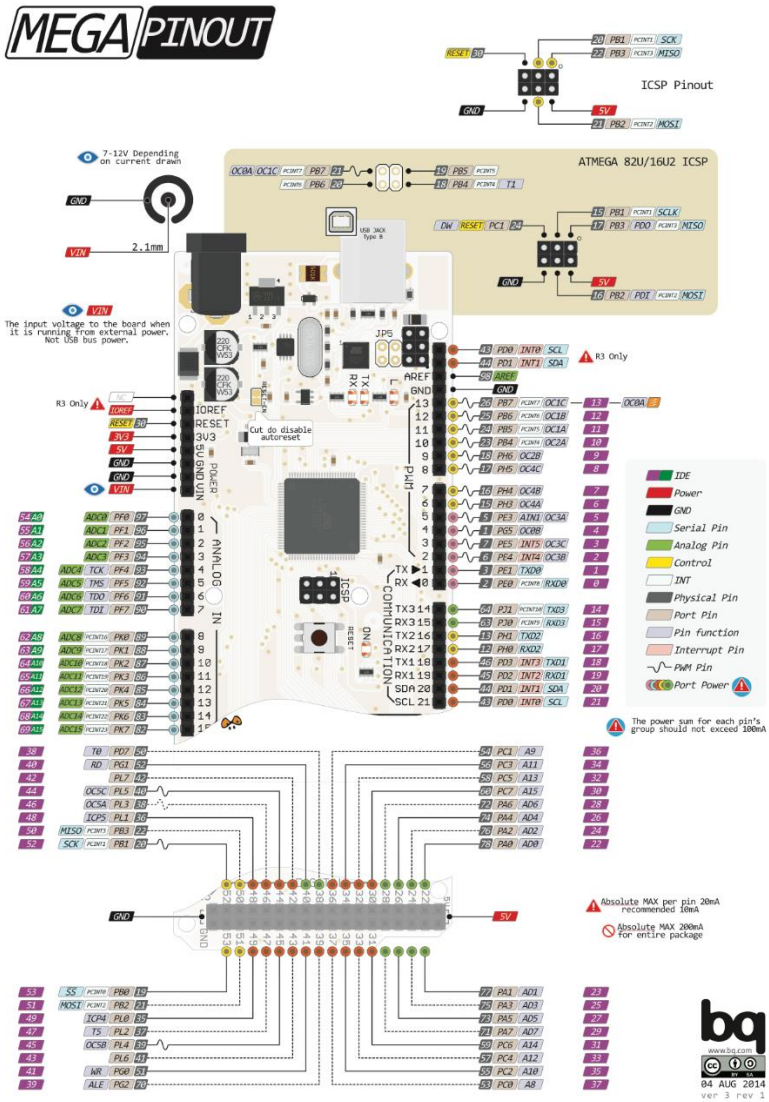
ARDUINO NANO



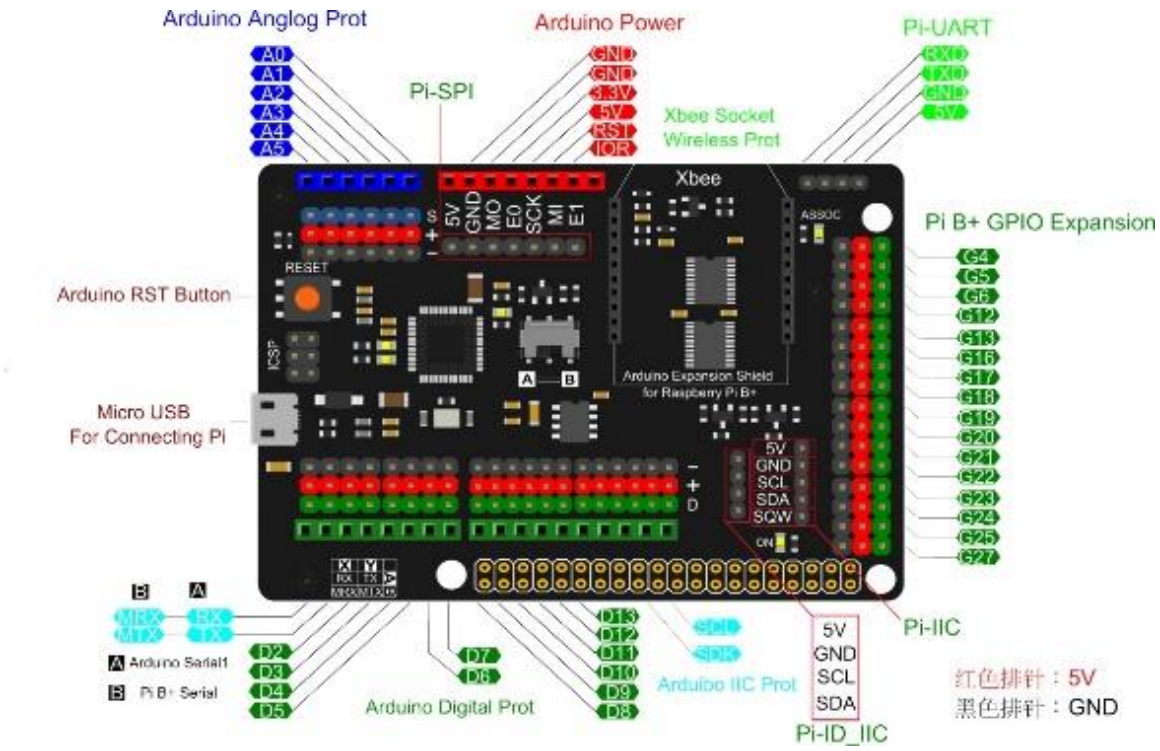
ARDUINO LEONARDO



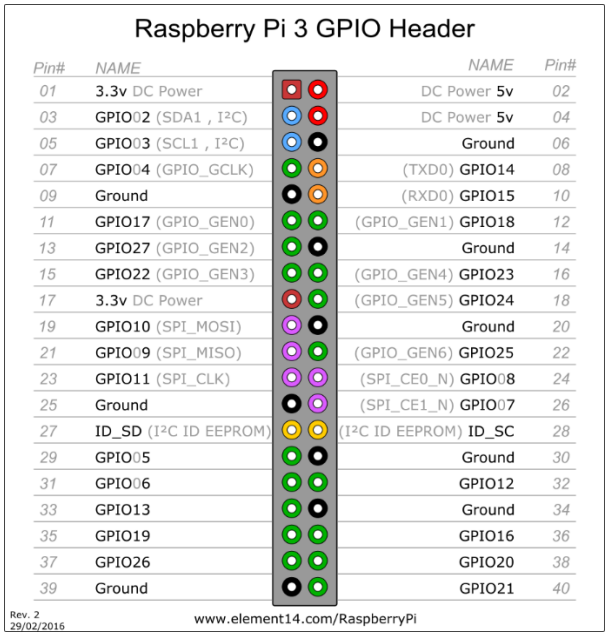
ARDUINO MEGA



ESCUDO ARDUINO PARA RASPBERRY PI



GPIO RASPBERRY PI 3B



Raspberry Pi 3 GPIO Header

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I²C)		DC Power 5v	04
05	GPIO03 (SCL1 , I²C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)		(I²C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Rev. 2
29/02/2016

www.element14.com/RaspberryPi