
# Máster Universitario en Software de Sistemas Distribuidos y Empotrados

*Desarrollo y despliegue automático de un sistema de sensorización y actuación 'smart agriculture' de acuerdo a principios DevOps y GitOps*

PROYECTO FIN DE MÁSTER

*David Cristóbal Pascual*

Enero de 2023


# Máster Universitario en Software de Sistemas Distribuidos y Empotrados

## *Desarrollo y despliegue automático de un sistema de sensorización y actuación 'smart agriculture' de acuerdo a principios DevOps y GitOps*

PROYECTO FIN DE MÁSTER

*Autor: David Cristóbal Pascual*

*Directores: Jessica Díaz y José Carlos Gamazo*

Enero de 2023

# Dedicatoria

Como siempre, a mis padres, Jesús y Yolanda. A Celia, por ayudarme y apoyarme en todos mis proyectos, en especial en este. Un trabajo que también es el vuestro.

# Agradecimientos

A Jessica y José Carlos por su mentoría, consejos y ayuda.

A mis abuelos Antonio y Florinda, por dejarme hacer de su huerto un campo de pruebas.

# Contenido

# Índice de diagramas

# Índice de ilustraciones

# Índice de esquemas

# Índice de tablas

# Glosario de términos

**Agricultura inteligente (*Smart Agriculture*):** es un enfoque para mejorar la eficiencia y la sostenibilidad de la producción agrícola a través del uso de tecnología y del análisis de datos.

**ARM (*Acorn RISC Machine*):** es una familia de procesadores con arquitectura RISC (*Reduced Instruction Set Computing*) utilizados en una gran variedad de dispositivos. Es conocida por su buena relación rendimiento/consumo.

**Automatización:** es el proceso de utilizar tecnología para realizar tareas de manera automática, sin la intervención humana.

**Cifrado:** es un proceso de codificación de datos para hacer que sean ilegibles a cualquier persona que no cuente con la clave de descifrado. Se utiliza para proteger la privacidad y seguridad de la información.

**Computación en el borde (*edge computing*):** es el procesamiento y análisis de los datos cerca de donde se generan, en lugar de la nube o en centros de datos. Permite una reducción en la latencia y un mayor rendimiento al poder procesar los datos en tiempo real.

**DevOps:** es una cultura y conjunto de prácticas que buscan mejorar la colaboración y comunicación entre los equipos de desarrollo y operaciones dentro de una organización. El principal objetivo es acelerar el ciclo de vida del desarrollo y entrega de aplicaciones y mejorar la calidad de estas.

**DNS (*Domain Name System*):** es un sistema utilizado para asociar nombres de dominio con las direcciones IP a las que representan.

**Docker:** es una plataforma de contenedores de código abierto que permite empaquetar y ejecutar aplicaciones de manera portable y consistente en cualquier entorno. Permiten empaquetar y distribuir aplicaciones con todas las dependencias y configuraciones necesarias.

**GitOps:** es la práctica de implementación y gestión que utiliza repositorios Git como fuente centralizada de verdad con la intención de automatizar despliegues y gestión de aplicaciones en entornos de producción.

**I²C (*Inter-Integrated Circuit*):** es un protocolo de comunicación de datos utilizado para conectar dispositivos en una red que únicamente necesita dos hilos: datos y reloj.

**IDE (*Integrated Development Environment*):** es un entorno de desarrollo que proporciona a sus usuarios un conjunto completo de herramientas para crear y depurar aplicaciones de forma rápida y eficiente.

**IoT (*Internet of Things*):** se refiere a la conexión de dispositivos electrónicos a Internet, incluyendo en ellos la capacidad de recopilar, compartir y tratar datos.

**JSON (*JavaScript Object Notation*):** es un formato de texto sencillo utilizado para el intercambio de datos. Es una alternativa a XML.

**Kubernetes:** es una plataforma de código abierto utilizada en el despliegue y gestión de aplicaciones en contenedores de forma sencilla.

**LoRa (*Long Range*):** es una tecnología de comunicación inalámbrica de bajo consumo y ancho de banda, largo alcance y que utiliza la banda ISM (868 MHz en Europa).

**MAC (*Media Access Control*):** es una dirección única que identifica a un dispositivo concreto dentro de una red. Se compone de ocho octetos hexadecimales.

**Microcontrolador:** es un dispositivo electrónico integrado que incluye microprocesador, memoria y periféricos en un solo chip.

**MQTT (*Message Queue Telemetry Transport*):** es un protocolo de comunicación de mensajería en tiempo real especialmente utilizado por dispositivos IoT.

***Pod*:** es la unidad básica de implementación y ejecución en Kubernetes. Unifica a uno o más contenedores que comparten una serie de recursos y se ejecutan en un mismo nodo.

**Sensorización:** es el proceso de recopilar datos y mediciones en un entorno o sistema utilizando para ello dispositivos llamados sensores.

# Resumen

La agricultura inteligente es un tema con un alto interés por su demostrada eficacia en la reducción de costes y el aumento de la producción de plantaciones. Internet de las Cosas (IoT) y, en particular, el tratamiento de los datos recabados por los sensores IoT en "el borde" (edge) mejora la calidad del servicio que estos sistemas pueden ofrecer. A pesar de las ventajas para acelerar la entrega de software que, enfoques y principios como DevOps y GitOps, han reportado en el desarrollo de sistemas distribuidos basados en la nube, su incorporación en el desarrollo y despliegue de sistemas IoT Edge es escasa. Algunas de estas ventas son: despliegues rápidos, confiables y automatizados, mejora de la calidad del software, mayor control de las versiones y detección anticipada ante cualquier desviación, una rápida vuelta a atrás frente a cualquier imprevisto o error en producción, y el registro de auditoría de los despliegues y ejecuciones.

Por lo tanto, en este TFM se propone la creación de un sistema de agricultura inteligente que regule el riego de una plantación tomando las decisiones conforme a los datos recabados y enviados por sensores situados en dicha plantación. Estos datos serán procesados por un computador situado en el borde que tomará la decisión de si es necesario iniciar el riego o detenerlo conforme a los datos que se han recogido en ese momento en la plantación. Esta decisión será transmitida a los actuadores que ejecutarán la instrucción habilitando o denegando el riego de la plantación. Los datos sensorizados se mostrarán mediante un panel web accesible por el agricultor en tiempo cuasi real y sin estar presente en la plantación. El ciclo software estará gestionado por varios flujos de integración continua y despliegue continuo (CI/CD) siguiendo los principios y buenas prácticas promovidas por DevOps y GitOps para, así, automatizar, acelerar y mejorar el despliegue de dicho software. La gestión y orquestación de los nodos edge se realizará haciendo uso de la plataforma Kubernetes, la cual se ha convertido en los últimos años en un estándar de facto para el despliegue de aplicaciones de nube nativas. En particular, al tratarse de un sistema IoT Edge, se contemplará el uso de implementaciones Kubernetes para dispositivos con restricciones computacionales, como KubeEdge, lo que facilitará la replicación y actualización de los nodos cuando sea necesario.

El hardware asociado a los dispositivos de la capa física está basado en el microcontrolador ESP32, en una Raspberry Pi 4B para la capa edge y una Raspberry Pi 3B para la visualización. El resto de los componentes se implementan en la nube con distintas soluciones. La comunicación entre la capa física y la de edge se lleva a cabo utilizando LoRa, y entre la capa edge y la de visualización mediante el protocolo MQTT. El funcionamiento de todos los dispositivos debe ser sencillo y modular para poder ser modificado o sustituido en caso de necesidad.

Los resultados y conclusiones del proyecto han sido positivos pues se han cumplido los principales requisitos impuestos al sistema con pocos fallos. El sistema es completo y funcional y se ha podido comprobar su correcto funcionamiento en campo. El uso de las tecnologías de desarrollo empleadas como parte de los principios DevOps y GitOps (CI/CD, Git, Kubernetes, virtualización, etc.) han permitido crear un sistema más robusto y mejor organizado, resistente a fallos y con un seguimiento mucho más sencillo. En definitiva, han permitido que el desarrollo de este sistema se lleve a cabo con mayor facilidad y rapidez.

*Palabras clave*

IoT, Agricultura Inteligente, DevOps, GitOps, Computación en el Borde, LoRa

# Abstract

Smart agriculture is a topic of high interest due to its proven effectiveness in reducing costs and increasing plantation production. Internet of Things (IoT) and, in particular, the processing of data collected by IoT sensors at the edge improves the quality of service that these systems can provide. The incorporation of DevOps and GitOps culture and principles is still scarce in the development and deployment of IoT Edge systems, despite the advantages for accelerating software delivery that have already been reported in the development of distributed cloud-based systems. Some of these sales are: fast, reliable, and automated deployments, improved software quality, greater version control and early detection of any deviations, fast rollback against any unforeseen events or bugs in production, and audit logging of deployments and executions.

Therefore, in this master's thesis, we propose the creation of an intelligent agriculture system that regulates the irrigation of a plantation taking decisions according to the data collected and sent by sensors located in the plantation. These data will be processed by a computer located at the edge of the plantation, which will decide whether to start or stop irrigation based on the data collected at that time in the plantation. This decision will be transmitted to the actuators that will follow the instruction by enabling or denying irrigation to the plantation. The data will be displayed on a web panel from which the farmer can consult them in real time and without being present at the plantation. The software cycle will be managed by several flows of continuous integration and deployment (CI/CD), following the principles and best practices promoted by DevOps and GitOps to automate, accelerate, and improve the deployment of the software. The management and orchestration of edge nodes will be performed using the Kubernetes platform, which has become in recent years the de facto standard for the deployment of cloud native applications. In particular, being an IoT Edge system, the use of Kubernetes implementations for computationally constrained devices, such as KubeEdge, will be contemplated, which will facilitate the replication and upgrade of the nodes when needed.

The hardware associated with the physical layer devices is based on the ESP32 microcontroller, the edge layer on a Raspberry Pi 4B, and the visualization on a Raspberry Pi 3. The rest of the components are in the cloud with different solutions. Communications between the physical layer and the edge are carried out using LoRa and between the edge and visualization using the MQTT protocol. The operation of all devices must be simple and modular, so that they can be modified or replaced if necessary.

The conclusions have been positive, as the main requirements imposed on the system have been met with few failures. The system is complete and functional, and it has been possible to verify its correct operation in the field. The use of the employed development technologies as part of the DevOps and GitOps principles (CI/CD, Git, Kubernetes, virtualization, etc.) has allowed us to create a more robust and better organized system, resistant to failures and with much easier monitoring. In short, they have allowed the development of this system to be carried out more easily and quickly.

## *Key words*

IoT, Smart Agriculture, DevOps, GitOps, Edge Computing, LoRa.

# 1. Introducción

En este primer apartado de la memoria, se presentará el contexto y motivaciones que han guiado el desarrollo de este proyecto.

## 1.1. Contexto del proyecto

### 1.1.1. Internet de las cosas (IoT)

El Internet de las Cosas, *Internet-of-Things* en inglés, es un marco conceptual que aprovecha la disponibilidad de dispositivos heterogéneos y soluciones de interconexión, así como de objetos físicos aumentados que proporcionan una base de información compartida a escala mundial, para apoyar el diseño de aplicaciones que implican en el mismo nivel virtual tanto a personas como a representaciones de objetos [1].

Es decir, hacemos que esos objetos que nunca han estado conectados ahora lo estén. Esto aumenta su coste en un primer momento; pero, a la larga, ese valor adicional se convierte en un ahorro o una mejora de la calidad del servicio prestado. Estos usualmente pequeños dispositivos, se distribuyen en el terreno, se conectan entre sí y con el resto de la red para proporcionar a sus usuarios datos o accesos que, sin esta tecnología, no serían posibles.

### 1.1.2. Agricultura inteligente

La necesidad de hacer más eficiente e inteligente la agricultura es cada vez más importante. La escasez de terreno y de recursos se hace más acuciante cada año que pasa. Frente al uso desmedido de recursos de la agricultura *tradicional*, se establece la agricultura *inteligente* como una alternativa que permite desperdiciar menos medios de los que disponen los agricultores. Principalmente, los aspectos diferenciadores de la agricultura inteligente son la automatización de procesos y la monitorización de datos, ambos, objeto de estudio y aplicación en esta Tesis Fin de Máster.

La incursión de las tecnologías IoT anteriormente mencionadas, está permitiendo la interconexión de cultivos, la obtención de nuevos datos y una forma de automatizar las actuaciones en el ámbito de la agricultura. Una cantidad importante de tecnologías están siendo empleadas en este ámbito, tecnologías que hace unos años eran impensables, pero que, gracias al avance tecnológico, ahora sí que lo son [2].

### 1.1.3. DevOps y GitOps

La cultura DevOps promueve la colaboración entre los dos ámbitos fundamentales de la ingeniería del software y la gestión del ciclo de vida de las aplicaciones: desarrollo y operaciones. La cultura DevOps integra ambos aspectos con el fin de mejorar y automatizar todo el proceso de desarrollo y despliegue de un producto software.

GitOps, por su parte, promueve los principios de la cultura DevOps, promoviendo su automatización a través de código (ej. infraestructura como código, IaC), de esta forma se mejora su gestión y modificación. Esto hace que sea posible que cada agente que interviene en la construcción del software tenga su función definida y un repositorio Git al que acudir y donde realizar sus cambios.

### 1.1.4. Virtualización

En software, la virtualización es una técnica que permite crear un entorno o máquina virtual (y no real) que simula un sistema informático dentro de otro sistema con el mismo nivel de abstracción. Esto permite que diversos sistemas compartan otro sistema donde todos ellos se encuentran alojados. En este proyecto se han empleado principalmente dos tipos de virtualización.

Virtualización de hardware (plataforma o servidor) consiste en la abstracción del hardware de una máquina para su aprovechamiento por diferentes instancias de sistemas operativos. Esto es aprovechado por las máquinas virtuales que son un software que simula un sistema informático completo dentro de otro. Permiten la ejecución de uno o varios sistemas físicos dentro de la misma máquina anfitriona, consintiendo el acceso a unos determinados recursos de esta última según su configuración que maneja el hipervisor que las gobierna. Las máquinas virtuales se comportan como máquinas físicas ya que cuentan con sus propios recursos y dispositivos (CPU, memoria, almacenamiento, entrada/salida, etc.). Permiten una mayor flexibilidad y aprovechamiento de los recursos. En este proyecto se han utilizado para alojar las herramientas de gestión del ciclo de vida de las aplicaciones (*Application Lifecycle Management*, ALM).

La virtualización a nivel de sistema operativo, también llamada virtualización basada en contenedores, al contrario que el anterior tipo de virtualización, la capa de virtualización se ejecuta sobre el núcleo del sistema operativo. Un contenedor aísla una aplicación con sus dependencias en lugar de todo un sistema completo (no contienen un sistema operativo completo) donde, como si se tratase de una máquina física, se ejecutase dicha aplicación. El contenedor no es sino una imagen de *contenedor*, que incluye todos los archivos y configuración que necesita la aplicación a ejecutar. Todo esto se indica en forma de código, por lo que su uso es ideal como parte de la cultura GitOps. Un ejemplo de plataforma de contenedores y tiempo de ejecución es Docker [3], que, por sí misma, no puede orquestar dichos contenedores. Para orquestar estos contenedores se utilizan herramientas especializadas en ello como Kubernetes, aunque actualmente el uso de Docker con Kubernetes no es recomendable y está considerado obsoleto [4], en favor de otras alternativas como containerd o CRI-O. Con esta herramienta, los desarrolladores pueden elegir cómo se comportarán los contenedores que componen el sistema utilizando para ello uno o varios ficheros de configuración y despliegue llamados *manifest*.

### 1.1.5. Computación en el borde

La computación en el borde o *edge computing* permite reducir las latencias de las cargas de trabajo relacionadas con el procesamiento, analítica, o filtrado de datos, entre otros, ya que estas cargas se ejecutan lo más cerca posible de la fuente de datos. De esta forma, se reduce el tráfico de red desde el borde a la nube u otras localizaciones. Por ello, para este proyecto, se ha decidido contar con una capa intermedia entre la sensorización y lo remoto (panel y datos): la capa *edge*, compuesta por nodos. Dichos nodos son computadores independientes que se sitúan en el *edge* y lo componen. De esta forma, se consiguen cumplir varios de los objetivos de este proyecto:

- Se reducen los costes de acceso a Internet ya que solo un nodo en la capa *edge* accederá a Internet y los datos que enviará estarán preprocesados y/o filtrados.

- Se mejora el rendimiento del sistema: se reduce el volumen de datos transmitidos hacia Internet y, por lo tanto, la latencia y el ancho de banda también serán menores.
- Mejora la seguridad y privacidad de los datos: se reduce el riesgo de pérdida o fuga de datos durante las transmisiones.
- Se reducen los costes operativos: con las reducciones anteriores, tanto las telecomunicaciones como el procesamiento posterior reducen su consumo.
- Si falla la comunicación del nodo con Internet, el procesamiento de datos y la toma de decisiones no se detiene. Se sigue manteniendo la capacidad de regar o dejar de hacerlo cuando sea necesario.

### 1.1.6. Sensores y actuadores

Estos son los dispositivos que interactúan con el medio en el que se encuentran. Los primeros, toman medidas y ayudan a identificar cuáles son las condiciones que les rodean. Los segundos, interactúan con dicho medio provocando cambios en él. El uso de ambos en un conjunto orquestado, permite que los cambios que realizan los actuadores se vean reflejados y sean iniciados por los sensores, permitiendo, así, una correlación entre los efectos realizados y percibidos.

## 1.2. Motivación del proyecto

La principal motivación de este proyecto es la mejora en el rendimiento de las plantaciones y de la calidad de vida de los agricultores que las trabajan. Tras una sequía sin precedentes que ha obligado a reducir la producción de alimentos a lo largo del año 2022 [5], el uso más racional de los recursos hídricos se ha convertido en una prioridad para todos. Por lo tanto, un sistema que permita un riego más preciso, que solo funcione cuando es necesario, es imprescindible para lograr este fin. Anteriormente, los sistemas de riego eran accionados por una persona que abría o cerraba el sistema manualmente y cuando le parecía que era necesario. Con datos y sistemas de riego automatizado, esto puede mejorarse y accionar el sistema de riego únicamente cuando sea necesario.

Como añadido a esto, la posibilidad de acceder a los datos necesarios para su monitorización, permite optimizar los parámetros de accionamiento del riego y faculta a obtener una correlación entre los resultados de las cosechas y los parámetros ambientales y de riego que se han producido a lo largo del tiempo de cultivo.

La mejora en el desarrollo y despliegue software que promueven la cultura DevOps y los principios GitOps permite una mejor experiencia de usuario, es decir, permite ofrecer al agricultor un servicio desplegado de forma confiable, con actualizaciones continuas que den valor a su negocio. Es imprescindible que este proceso sea transparente para el agricultor, manteniendo niveles adecuados de disponibilidad. De esta forma, se consigue un software útil, que no genera rechazo por parte del agricultor, fiable y que, pese a que pueda necesitar actualizaciones, estas no interrumpan la ejecución del sistema causando excesivas molestias.

## 1.3. La solución propuesta

Ante el problema mostrado anteriormente, se propone una solución que satisfaga los requisitos de las dos partes interesadas: el prestador y el receptor del servicio. El primero

es el desarrollador del servicio y, el segundo es el agricultor, que hace uso del servicio. Por ello, y para satisfacer a las dos partes, a continuación, se expondrán los requisitos e intereses de ambas partes y qué solución se les ha dado.

### 1.3.1. Desarrollador

Para el desarrollador, su interés principal es que el desarrollo del producto sea lo más predecible y controlado y que, además, el despliegue sea automatizado y desatendido, es decir, que no requiera de acciones por su parte que puedan bloquear y/o retrasar el despliegue de actualizaciones y cambios sobre el software. Para ello se ha implementado un sistema de Integración continua y Despliegue Continuo: CI/CD (*Continuous Integration/Continuous Deployment*). Este sistema está compuesto por dos herramientas: Azure DevOps (CI) y Argo CD (CD). Para facilitar los despliegues y mejorar la continuidad operacional (buena disponibilidad) se ha utilizado KubeEdge, que permite el despliegue de Kubernetes en la capa *edge*. De esta forma se obtiene un sistema que es capaz de preparar y realizar los despliegues de forma automática y que asigna a cada nodo *edge* una función concreta y lo previene ante los posibles fallos que puedan surgir.

### 1.3.2. Agricultor

El objetivo principal de este proyecto es brindar al agricultor de un sistema formado por componentes hardware –dispositivos– que permitan una instalación rápida y sencilla, así como de componentes software que ofrezcan una visualización clara y útil de los datos obtenidos. El principal interés es que toda la instalación pueda llegar a realizarse por cualquier persona, utilizando para ello sensores y actuadores *plug-and-play*, es decir, que solo sea necesario conectarlos para que empiecen a funcionar y cuya instalación en la plantación sea muy sencilla. Los nodos *edge* también deben ser sencillos de instalar requiriendo únicamente por parte del usuario una sencilla conexión y una instalación de igual facilidad. En el posible evento de rotura o desgaste, la sustitución de cualquier elemento debería ser igualmente simple. Por ello, los sensores, actuadores y nodos, vendrán preparados de serie para funcionar nada más sean conectados gracias a una base de datos y su correspondiente API, que gestionen su funcionamiento.

El sistema de riego será automático y basado en las mediciones realizadas por los sensores. Así, se conseguirá reducir la carga de trabajo del agricultor y se mejorará la irrigación del terreno, no quedando zonas sin regar lo suficiente o zonas inundadas.

Para la visualización de los datos se desplegará un panel *dashboard* que permitirá el almacenamiento, visualización y análisis de los datos. Para ello lo único que tendrá que realizar el agricultor es acceder a una aplicación web en la que podrá ver todos los datos recogidos por sus sensores y detectar si alguno falla, recibir alertas personalizadas, etc.

## 1.4. Objetivos

Esta Tesis Fin de Máster aborda el desarrollo de un proyecto de agricultura inteligente siguiendo la cultura y principios DevOps y GitOps aplicados a entornos IoT Edge con el objetivo de acelerar y mejorar la entrega del software. Los objetivos de forma más detallada son:

**OBJ1:** Desarrollo de un sistema de agricultura inteligente que permita la toma automática de decisiones gracias al sistema IoT Edge. Para ello, se instalará un conjunto de sensores

y actuadores en las plantaciones que tomarán datos ambientales y el control sobre el riego respectivamente. La visualización de los datos se realizará en un panel web al que podrá acceder el agricultor desde cualquier lugar en tiempo cuasi real.

**OBJ2:** Aplicación de los principios y buenas prácticas DevOps y GitOps para lo que se implementarán flujos de integración y despliegue continuos (CI/CD) y repositorios Git. Gracias a estos, se automatizará la actualización de los nodos IoT Edge y servicios cloud, manteniendo la trazabilidad y el control de cualquier cambio. El despliegue en dichos nodos se hará utilizando un clúster orquestado utilizando Kubernetes.

**OBJ3:** Aceleración de la entrega de valor mediante entregas continuas y facilidad de instalación del hardware. Esto se conseguirá aprovechando las buenas prácticas DevOps y GitOps junto con herramientas de apoyo, que por una parte permiten las entregas continuas y por otra la inicialización automática de todos los sistemas sin requerir una configuración manual por parte del instalador. El hardware estará diseñado para que su instalación en las plantaciones pueda ser realizada por cualquier usuario con un mínimo de experiencia.

# 2. Estado del Arte

La implementación de los temas tratados en este trabajo en su conjunto: agricultura inteligente, IoT Edge y, la cultura y principios DevOps y GitOps, no está muy extendida. La relación entre los dos primeros entre sí es habitual y evidente. De igual forma lo es la relación entre DevOps y GitOps, en especial en sistemas nativos *cloud*. Lo especialmente extraño y llamativo es la implementación y despliegue de sistemas IoT Edge utilizando los principios DevOps y GitOps. Por esta particularidad de los temas tratados, primero se reportarán los estados del arte de cada tema por separado y, para finalizar, se resaltará el estado de la técnica del uso conjunto.

## 2.1. Internet de las cosas (IoT)

Ahora mismo, el IoT se caracteriza por un aumento en la sensorización, monitoreo y control (de la misma forma que la agricultura inteligente). Cada vez es más habitual el uso de dispositivos conectados en un mayor número de aplicaciones. Las principales son [8][9]:

- Infraestructura inteligente
    - Horas y oficinas agradables
    - Centros industriales
    - Museos y gimnasios inteligentes
- Sanidad
    - Seguimiento
    - Identificación y autenticación
    - Recolección de datos
    - Sensorización
- Logística y cadenas de suministro
    - Logística
    - Conducción asistida
    - Venta de entradas móvil
    - Vigilancia medioambiental
    - Mapas con realidad aumentada
- Social y personal
    - Relaciones humanas
    - Peticiones históricas
    - Pérdidas
    - Robos
- Futurístico
    - Taxi robot
    - Información de la ciudad
    - Salas de juego mejoradas

Como puede verse, los usos que se le están dando al IoT son muy variados y sus finalidades van desde las puramente organizativas hasta el placer personal. Todas ellas están sufriendo un incremento en su uso y su investigación por la oferta económica que suponen para los desarrolladores.

Se plantean varios desafíos como la seguridad [10] donde se resalta que los dispositivos IoT al estar distribuidos en el espacio son más vulnerables a todo tipo de ataques. La

información que recaudan suele ser muy sensible por lo personalizada que puede llegar a estar y ser muy confidencial como los datos médicos o de relaciones. La heterogeneidad de los dispositivos juega en contra de la seguridad de estos, por ello genera homogeneidad favorece el mantenimiento de la seguridad y del control. La gran cantidad de agentes menos serios que se encuentran desarrollando dispositivos IoT también juega en contra de la seguridad de estos.

## 2.2. Agricultura inteligente

El aumento de la agricultura inteligente o de precisión en los últimos años es algo evidente, no solo en el ámbito de la investigación, también en el de la industria. El uso cada vez mayor de tecnologías de sensorización, monitoreo y control para recopilar y procesar los datos en tiempo real en la capa *edge*. Estas tecnologías incluyen sensores de campo, drones y otros vehículos autónomos y sistemas de información geográfica (GIS) [6]. Otro pilar fundamental en el que se apoya la agricultura inteligente es en la inteligencia artificial y la ciencia de datos para crear, así, soluciones basadas en datos.

- Actualmente, los espacios clave en los que se está aplicando la agricultura inteligente en la Unión Europea son los siguientes [7]: Se emplea principalmente en tierras de gran tamaño que sean arables.
- El componente más exitoso de la agricultura inteligente en tierras arables es el uso de CTF (*Controlled Traffic Farming*) que ha reducido el uso de maquinaria hasta en un 75%.
- En la agricultura tradicional, los fertilizantes y otras sustancias de control de plantaciones son aplicadas de forma uniforme, lo que conlleva una sobre aplicación en algunos puntos e infra aplicación en otros. Los métodos de la agricultura de precisión permiten la aplicación variable y relacionada con las necesidades de cada zona y plantación.
- La aplicación de la agricultura de precisión en frutales y vegetales es más reciente que la de tierras arables.
- Las altas ganancias y los altos riesgos de las plantaciones de frutales y vegetales hacen que este tipo de plantaciones sean ideales para este tipo de agricultura. Un buen ejemplo es la viticultura, donde la calidad de las uvas y de las plantaciones se pueden obtener gracias a la sensorización remota y otro tipo de instrumentos instalados en el campo. Esto permite obtener mejores vinos al no mezclar uvas de buena calidad con las de mala.
- En plantaciones de alto valor de frutales y vegetales, el riego de precisión se está desarrollando muy rápidamente con la intención de ahorrar agua, mejorar la calidad y aumentar la cantidad del producto. En este caso con el déficit en el riego se pueden lograr productos de mayor calidad y para esto se utilizan herramientas de sensorización a bordo de drones.

## 2.3. Computación en el borde (*edge computing*)

Con la expansión del IoT, la computación en el borde está experimentando un incremento notable. La utilización de cada vez más sensores y la necesidad de procesar esos datos y tomar decisiones lo más rápido posible, han hecho que la computación en el borde experimente un aumento en popularidad y uso. Con un número creciente de dispositivos conectados a Internet, se hace necesario el procesado y depurado de los datos antes de subirlo a la nube (es decir, lo más cerca posible de la fuente del dato), al contrario de lo

que se hace en un modelo tradicional de computación cloud donde el procesado se realiza en los propios servicios de nube y, por lo tanto, mucho más lejos.

Uno de los nuevos conceptos que están siendo estudiados y empleados es el *edge continuum* [11]. Este describe la distribución de recursos entre centros de procesamiento de datos (CPD o *data centers*) y los dispositivos situados en el *edge*. Como se puede ver en la Ilustración 1, la distribución de esta arquitectura va desde los dispositivos más cercanos al usuario (a la izquierda) a los CPD centralizados (a la derecha), de izquierda a derecha se puede ver la progresión desde un *edge* cercano a uno más distante. Con esta arquitectura, se consigue que las aplicaciones se puedan ejecutar en dispositivos lo más cercanos a los usuarios finales que sea posible. Esto permite una mayor eficiencia de latencia, disponibilidad y seguridad. En aplicaciones que requieren buenos niveles de las características anteriores este enfoque es especialmente notable.

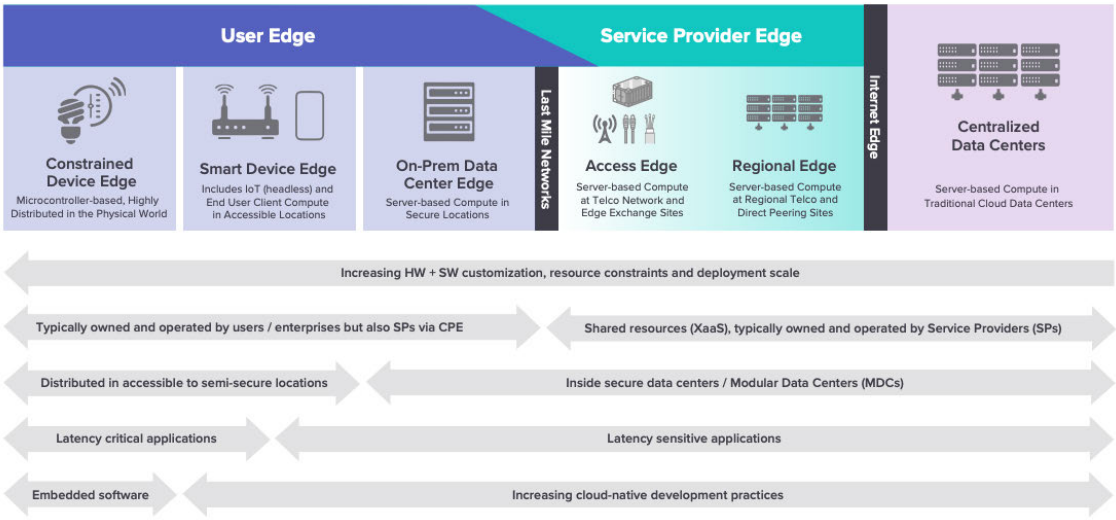



*Ilustración 1 Resumen de edge continuum. Fuente* [11]

En cuanto a la seguridad de la computación en el borde, al igual que la seguridad IoT, siempre ha sido un paradigma muy estudiado por su vulnerabilidad ante ataques, ya que el control de los dispositivos no puede ser tan alto como nos gustaría.

Otras áreas en las que la computación en el borde está expandiéndose son [12]:

- Movilidad
- Control de accesos y detección de entradas
- Sanidad
- Análisis de vídeo
- Conexión entre vehículos
- Análisis *big data*
- Control inteligente de instalaciones
- Monitoreo y control de instalaciones
- Sensorización
- Casas inteligentes
- Ciudades inteligentes

Como puede verse, en todas las categorías se necesitan unos tiempos de respuesta bajos y un procesamiento muy intensivo de una gran cantidad de datos que, por su naturaleza,

es mejor tratar en el *edge* para conseguir un mejor resultado y no saturar las telecomunicaciones.

El futuro no son dos capas aisladas: sensores y nube; sino varias: sensores, *edge*, *fog* y nube. De esta forma las latencias se reducen y mejora la capacidad de procesamiento en tiempo real, lo que permitirá tener una mejor calidad en el procesamiento y que los usuarios puedan aprovechar mucho mejor las capacidades reales del sistema que estén utilizando.

## 2.4. DevOps y GitOps

La cultura DevOps integra los mundos de desarrollo ("Dev") y operaciones ("Ops"), utilizando para ello desarrollo automatizado, despliegues y monitorización de la infraestructura. DevOps significa un cambio en la cultura hacia la colaboración entre: desarrollo, pruebas y operaciones, frente a la tradicional cultura de compartimentar estos roles en espacios separados y no colaborativos. Es un cambio en la forma de trabajar de los equipos, integrando los procesos de negocio de desarrollo, producción y operaciones utilizando para ello las tecnologías más adecuadas [13]. El principal objetivo es acelerar la entrega de valor y mejorar la organización en los equipos de desarrollo.

El estado de la técnica en materia DevOps se caracteriza por el aumento en el uso de herramientas y prácticas de integración y entrega continua (CI/CD) para ayudar en la automatización del desarrollo y despliegue de aplicaciones. Con este enfoque los desarrolladores no "arrojan" su trabajo a sus compañeros de trabajo para que estos lo desplieguen y se desentienden; trabajan en conjunto y ambos son informados de todos los resultados y posibles problemas [14]. Este enfoque cambia la forma en la que se habían desarrollado y desplegado aplicaciones en producción hasta el momento.

Por su parte, GitOps se caracteriza por un uso cada vez mayor de herramientas de control de versiones Git, a lo que se le tiene que sumar un uso también mayor de herramientas de orquestación de contenedores como Kubernetes [15]. De esta forma, se puede desplegar el código alojado en un repositorio de forma automática y controlar todos estos despliegues utilizando código para ello también, para esto se combina junto con DevOps.

En estos momentos, son tecnologías en constante crecimiento por la alta rentabilidad, reducción del esfuerzo y trazabilidad con las que ayudan a los equipos de desarrollo de las compañías que así lo implementan. Son culturas de trabajo muy acertadas y que, a lo largo de estos últimos años, han demostrado su gran valía y utilidad.

## 2.5. Enfoque integrado: Agricultura inteligente en IoT Edge aplicando DevOps & GitOps

Aplicaciones conjuntas de todos estos enfoques y tecnologías podemos encontrar pocas. Parte de la responsabilidad la tienen las compañías, que, aunque probablemente implementen todas o alguna de estas culturas, no lo hacen público. Resulta bastante evidente que son tecnologías que casan perfectamente y son muy útiles en su conjunto, pero, como es habitual, las empresas, de normal, no publican su *know-how*. Sin embargo, son muy conocidos y ya han sido mostrados los casos de uso de agricultura inteligente, junto con computación edge e IoT, dado que son tecnologías que van casi siempre a la par [6].

Sin embargo, en el ámbito académico y científico sí que existen algunos trabajos publicados:

El trabajo de Ramón López [16] ha inspirado y ayudado a la definición de este proyecto y es un claro ejemplo de la integración de todos estos principios, tecnologías y herramientas en sistemas que, en conjunto, permiten un mejor rendimiento y facilidad de desarrollo para los sistemas de agricultura inteligente. La arquitectura que propone R. López en su proyecto (ver Ilustración 2) es muy útil en los escenarios en los que este proyecto se enfoca, y en ella se basa parte de este TFM.

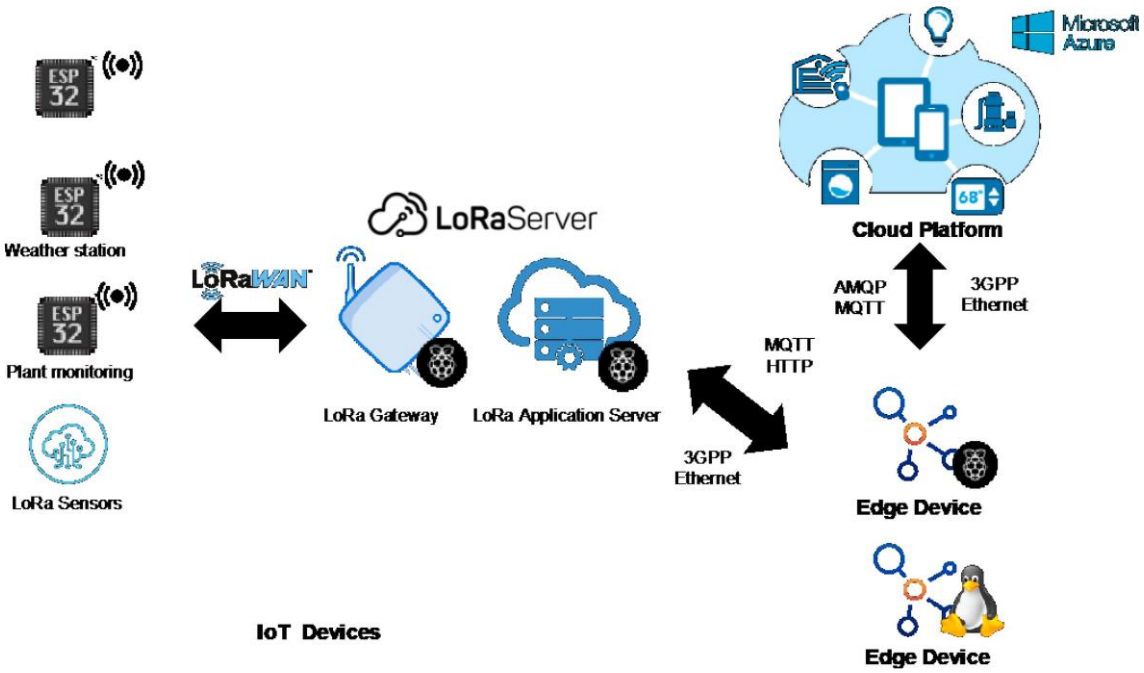



*Ilustración 2 Arquitectura de despliegue de una solución de agricultura de precisión. Fuente* [16]

La incorporación de la cultura DevOps al IoT es una perspectiva muy interesante para el desarrollo de este tipo de soluciones y, sin duda, es la tendencia, como así demuestra la evolución en los estudios realizados por Eclipse Foundation en los años 2020 [17] y 2022 [18]. De estos informes podemos extraer unas conclusiones muy reveladoras sobre la forma de trabajar en la industria y el aumento de cultura DevOps:

- En 2020, el uso de contenedores en el *edge* representaba el 36 % de los casos, siendo binarios y *scripts* el 66 % restante. En 2022 esta tendencia ha continuado y ha aumentado aún más la virtualización en el *edge* siendo actualmente un 49 % lo representado por contenedores y apareciendo máquinas virtuales en el segundo puesto con el 31 % de los usos, quedando binarios y *scripts* con un 27 y un 22 % respectivamente.
- El crecimiento de la virtualización en el *edge* ha sido tal que ahora sí que se hace referencia a qué tecnologías son las más utilizadas, siendo Docker (43 %), Fog05 de Eclipse y, otro tipo de Kubernetes, las más comunes.
- En la encuesta de 2020 sí que se habla de las formas más habituales de despliegue en dispositivos *edge*, siendo la más común las *over-the-air* (OTA) gestionadas desde entornos *cloud*, seguida desde entornos locales y, finalmente, las manuales utilizando una red cableada. Como puede verse, la automatización de despliegues es la solución más habitual. En 2022, aunque no se haya expuesto esta información, resulta evidente cuál es el método más habitual: *cloud*.

- Otro indicador de la aplicación de la cultura DevOps es que los despliegues en el informe de 2020 no figuran como una de las mayores preocupaciones de los desarrolladores IoT, pero en 2022 sí que lo hacen con un 20 % de los encuestados preocupados por esto. En 2021 la preocupación fue mayor, con un 31 % de los encuestados, pero esto también puede ser debido a la mejora y automatización de los despliegues al utilizar los principios DevOps y las herramientas de virtualización comentadas.

Por lo tanto, queda patente el uso de buenas prácticas DevOps por la industria en el ámbito del IoT Edge, aunque directamente no suelan revelarlo. Estas encuestas son un reflejo sesgado de la industria que participa en la comunidad *open source* siendo un 69 % de los usuarios desarrolladores *open source* y usuarios de herramientas *open source*.

# 3. Gestión del proyecto

## 3.1. Metodología

### 3.1.1. Diagrama de bloques/Paquetes de trabajo

En el Diagrama 1 pueden verse los diferentes paquetes de trabajo en los que está estructurado el desarrollo de este proyecto. Cada categoría principal se ha dividido en subcategorías que, a su vez, están divididas en tareas simples. El orden mostrado por dichas subcategorías es el real, pero el orden de las categorías principales no se ha seguido estrictamente, como se podrá ver en el apartado 3.1.2. Se ha intentado lograr una coherencia en cuanto a las categorías y los ítems principales de este proyecto y memoria para que su seguimiento sea más intuitivo. Por lo que, viendo el Diagrama 3, se puede entender correctamente esta clasificación.

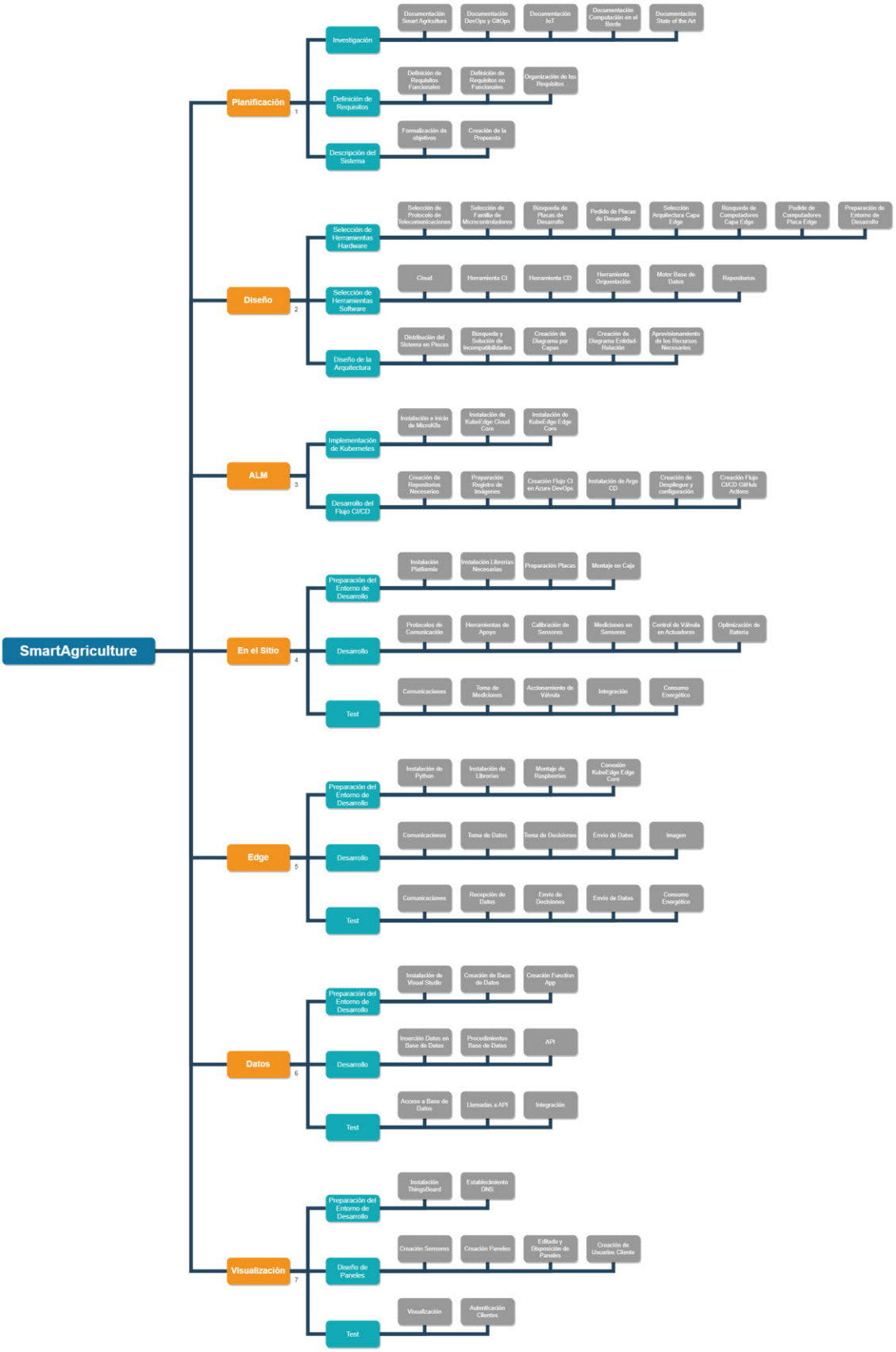



*Diagrama 1 Elementos de Trabajo*

### 3.1.2. Planificación

Este proyecto ha sido desarrollado entre los meses de septiembre y diciembre de 2022, los tiempos dedicados a cada tarea anterior y cuándo se han realizado se muestran en el siguiente diagrama.

Como puede verse en el Diagrama 2, la organización del proyecto se ha basado en la realización de las tareas expuestas en el capítulo 3.1.1. La duración de la mayor parte de dichas tareas ha sido de una semana, aunque algunas se han extendido más allá por su complejidad o la falta de dispositivos, que ha sido un problema durante cierta parte del proyecto. Estas tareas componen pequeños ciclos de desarrollo en los que se desarrolla una función a lo largo de una o dos semanas haciendo así que el desarrollo sea trazable y dividido en partes.

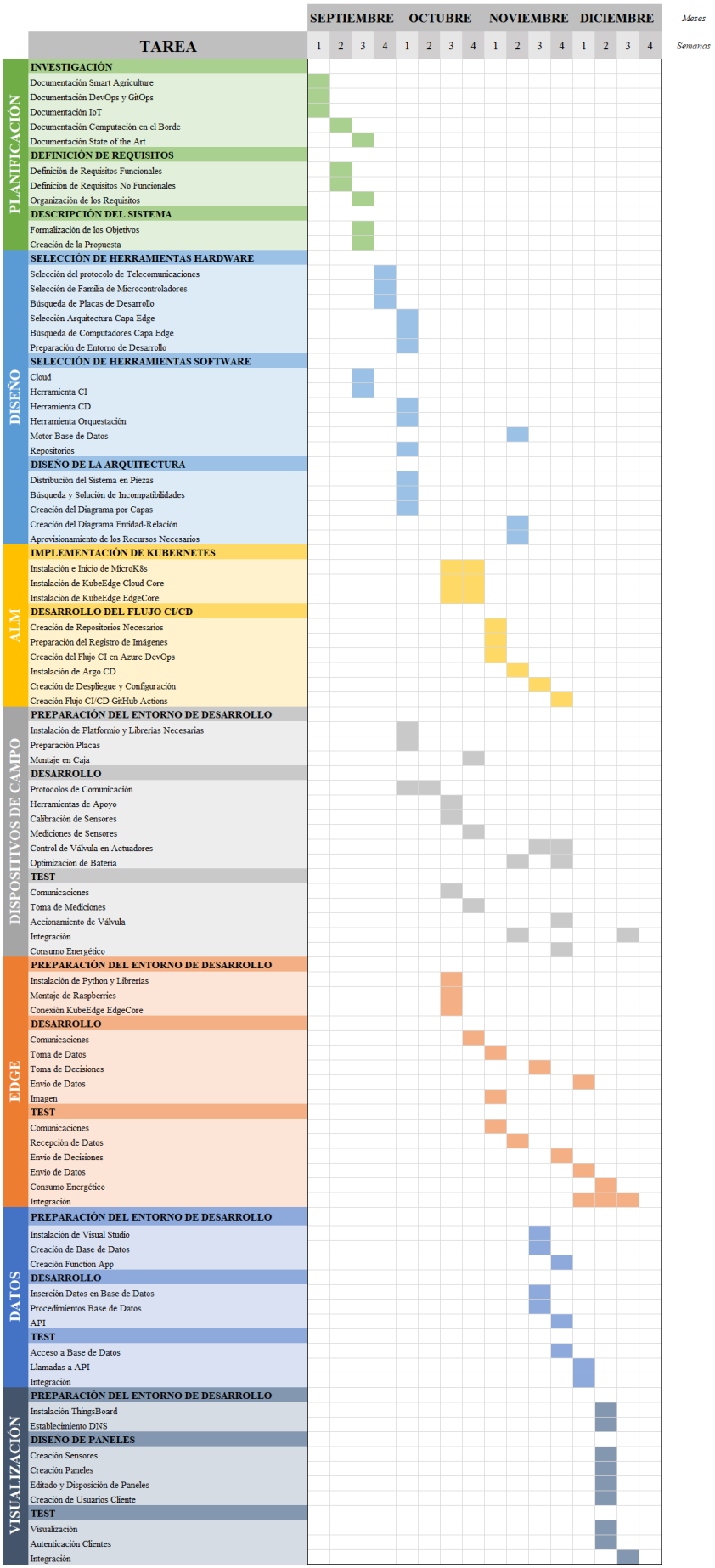



*Diagrama 2 Planificación del proyecto*

### 3.1.3. Desarrollo

El desarrollo de este proyecto ha seguido diversas metodologías, diferentes para cada una de las piezas que componen la totalidad. Siguiendo la metodología DevOps y GitOps se han realizado todas las piezas software, sin embargo, el hardware por su propia naturaleza no ha podido seguir las mismas metodologías.

#### 3.1.3.1. Hardware

El proceso para la creación de las placas es desconocido dado que han sido adquiridas y, por lo tanto, su desarrollo ha sido llevado a cabo por una tercera empresa. Su selección ha sido iterativa, buscando entre todas las alternativas y seleccionando la más cercana disponible a los requisitos del proyecto. De la misma forma se ha realizado con los sensores y las válvulas. El principal criterio de selección ha sido encontrar unos componentes por un precio asequible que cumpliesen con la función encomendada de la mejor forma posible, siempre teniendo en cuenta que este proyecto es una prueba o ensayo y que, por lo tanto, no tiene los mismos requerimientos a nivel de calidad hardware que un producto final. Para paliar los posibles inconvenientes de la probable falta de suministro se ha intentado desarrollar todo conforme a plataformas establecidas y no conforme a elementos concretos privativos que no permitan su sustitución. Precisamente la existencia de un esquema electrónico de ambos tipos de placa fue uno de los puntos clave para decantarme por esa plataforma y fabricante. Otro motivo por el que se han seleccionado es por la facilidad de sustituir una placa por otra dado que todos los periféricos del ESP32 que incorporan son estándar y, por lo tanto, están disponibles para su compra individual o son sustituibles por otros con la misma funcionalidad. Es importante mantener que este proyecto puede sustituir cualquiera de sus componentes y mantener o mejorar la funcionalidad con los menores cambios.

Para el proceso de ensamblaje de sensores y actuadores en sus respectivas cajas estancas se han diseñado piezas que, de forma iterativa, han ido evolucionando para poder adaptarse a nuevos sensores o diferentes puntos de anclaje. En definitiva, han ido mejorando conforme a los resultados de las pruebas de montaje que se han venido realizando. Es importante destacar que el diseño de las piezas ha permitido que estas puedan ser adaptadas conforme a la necesidad y a que la fabricación de las piezas sea sencilla mediante una impresora 3D u otros métodos de fabricación como inyección de plástico o mecanizado. Esta fabricación ha demostrado ser suficiente para el objetivo del proyecto y permite ser adaptada en un futuro si fuese necesario porque las necesidades cambiasen. Cabe destacar la gran ayuda que supone el diseño asistido por computador (CAD) para la realización de todas las piezas, y que hace que el modelado y probado de las pruebas sea mucho más simple. Además, añade el control de versiones al desarrollo de piezas físicas de forma digital, en el propio programa se consigue comparar versiones como es habitual en el desarrollo software. Gracias a esto, se puede decir que se ha seguido una metodología similar a la parte del software, pero marcando las diferencias.

#### 3.1.3.2. Software

El desarrollo del software sí que ha seguido metodologías típicas de este a las que ya se le han hecho referencia en puntos anteriores. El desarrollo de todo el software ha sido incremental, es decir, se han ido desarrollando partes del software una a una, que se han ido probando individualmente y en funcionamiento en conjunto. El software se compone de distintos componentes individuales que se encargan de cumplir una única función, esto

puede verse en la Planificación del proyecto en la que queda patente que el software se divide en secciones más pequeñas que fueron finalizándose en orden. Como se explica en el capítulo 3.1.2 Planificación, los pequeños ciclos de desarrollo marcan el desarrollo de cada pieza software.

El despliegue del software ha seguido distintas metodologías adaptándose a las distintas particularidades que supone un proyecto tan amplio como este. Como se explicará a lo largo del capítulo 5.2, el despliegue de la capa física se ha realizado de forma manual, pero el de la capa *edge* y *cloud* lo han hecho siguiendo los principios DevOps en los que el despliegue está automatizado. Para ello se han implementado dos ciclos CI/CD, el primero de ellos con Azure DevOps (CI) y Argo CD (CD), y el segundo con GitHub Actions (CI/CD). Esto presenta la ventaja de que todos los despliegues se realizan siempre de la misma forma y cuando los cambios son enviados a la rama principal del repositorio, por lo que se pueden hacer pruebas en otras ramas o separar los despliegues en distintos escenarios (prueba y producción, por ejemplo).

La metodología GitOps que se ha empleado a lo largo del desarrollo de todas las partes de este proyecto ha supuesto una gran forma de mejorar la calidad del desarrollo DevOps. La existencia de un repositorio Git para cada elemento mejora la legibilidad y comprensión del código y permite mejorar la calidad de los desarrollos y despliegues. El flujo implementado es simple y consistente, coherente con la finalidad de cada repositorio y fuente de la realidad del desarrollo y despliegue. Gracias a GitOps, este proyecto podría ser desarrollado por un mayor número de desarrolladores sin que estos se tengan que preocupar por cuál es la fuente de verdad o averiguar cuál es el punto de desarrollo en el que se encuentra el software.

## 3.2. Presupuesto

### 3.2.1. Hardware

En la Tabla 1 puede verse el presupuesto total del proyecto, los precios indicados en la tabla son sin IVA, pudiendo ver el IVA y el total incluyendo el IVA en las filas finales de totales.

| *#* | ***Nombre*** | ***Fabricante*** | ***Precio/Ud*** | ***Ud*** | ***Total*** |
|---|---|---|---|---|---|
| *1* | TTGO T-Beam ESP32 WiFi GPS NEO-6M LoRa 868MHz | LILYGO | 38,95 € | 2 | 77,90 € |
| *2* | Sensor de temperatura y humedad I2C AM2315C | Adafruit y Asair | 23,50 € | 2 | 47,00 € |
| *3* | Sensor de humedad del suelo anticorrosión | DFROBOT | 6,90 € | 2 | 13,80 € |
| *4* | Pack de 2 Baterías Li-ion 18650 MJ1 - 3500mAh, 3.7V | LG | 3,88 € | 1 | 3,88 € |
| *5* | Adafruit LoRa Radio Bonnet OLED - RFM95W (900MHz) | Adafruit | 36,95 € | 2 | 73,90 € |
| *6* | Válvula con solenoide 12V - ½” | Adafruit | 3,95 € | 2 | 7,90 € |
| *7* | MOSFET FQP30N06L | ONSEMI | 2,10 € | 2 | 4,20 € |
| *8* | Módulo sensor de lluvia y gotas de agua YL-83 | Genérico | 1,40 € | 2 | 2,80 € |
| *9* | Diodo HER203 | WTE | 0,10 € | 2 | 0,20 € |
| *10* | Tarjeta microSD 64GB | Amazon Basics | 8,74 € | 2 | 17,48 € |

| | | | | | |
|---|---|---|---|---|---|
| *11* | Antena flexible 868MHz | DollaTek | 2,00 € | 2 | 4,00 € |
| *12* | Cable SMA macho a SMA macho | RF JKM | 1,85 € | 2 | 3,70 € |
| *13* | LILYGO LoRa32 V1.0 | LILYGO | 13,00 € | 2 | 26,00 € |
| *14* | Materiales de montaje varios | | 20,00 € | 1 | 20,00 € |
| *15* | Ventiladores 80mm USB | upHere | 5,50 € | 2 | 11,00 € |
| *16* | Cable ethernet exterior CAT6 | Nanocable | 0,40 € | 8 | 3,20 € |
| *17* | Caja estanca 150x100 | Legrand | 7,40 € | 2 | 14,80 € |
| *18* | Caja estanca 100x100 | Legrand | 4,35 € | 2 | 8,70 € |
| *19* | Manguera 3p 1mm | Top Cable | 1,60 € | 7 | 11,20 € |
| *20* | Cable USB – MicroUSB | Tedi | 1,00 € | 2 | 2,00 € |
| *21* | Conversor DC – DC 12V – 5V | Tedi | 2,50 € | 2 | 5,00 € |
| *22* | Kit Raspberry Pi 4B - 2 GB | Raspberry | 93,78 € | 1 | 93,78 € |
| *23* | Kit Raspberry Pi 4B - 8 GB | Raspberry | 162,61 € | 1 | 162,61 € |
| *24* | Kit Raspberry Pi 3 - 1 GB | Raspberry | 50,00 € | 1 | 50,00 € |
| ***Total*** | | | | | *665,05 €* |
| ***IVA*** | | | | *21%* | *139,66 €* |
| ***Total, con IVA*** | | | | | *804,71 €* |

*Tabla 1 Presupuesto de Hardware*

Todos los productos a excepción de los números 12, 13 y algún elemento del 14 han sido comprados en España, por lo que no han requerido de ningún tipo de pagos adicionales. Las excepciones anteriormente mencionadas han sido importadas desde China por la imposibilidad de conseguirlos en España y no han requerido el pago de aranceles por su bajo costo. Los tiempos de entrega y la garantía han sido los principales motivos para seguir esta planificación.

### 3.2.2. Software

Para el desarrollo de este proyecto se ha utilizado software gratuito y de pago. Las licencias del software de pago empleado han sido obtenidas mediante el convenio existente entre la Universidad Politécnica de Madrid y estas empresas y cuyo uso para realizar tesis está previsto en sus licencias. Algunos ejemplos de esto son: Microsoft 365 (redacción y procesado de tablas) y Autodesk Fusion 360 (modelado de piezas). Existen alternativas gratuitas para ambos softwares, pero se ha decidido su uso por lo extendidos que están en el mercado y su buen funcionamiento. Ambos cuentan con versiones gratuitas, web en el primer caso y licencias para aficionados en el segundo.

En la Tabla 2 se muestran los dos únicos programas de pago empleados en el desarrollo de este proyecto con su precio para las versiones comerciales cuya licencia incluye los proyectos comerciales. La posibilidad de obtener licencias de forma gratuita gracias a los convenios de la universidad con las empresas ha supuesto un gran ahorro.

| *#* | ***Nombre*** | ***Desarrollador*** | ***Precio*** |
|---|---|---|---|
| *1* | Fusion 360 | Autodesk | 336,36 € |

| | | | |
|---|---|---|---|
| *2* | Microsoft 365 para Negocios | Microsoft | 105,60 € |
| ***Total*** | | | ***441,96 €*** |
| ***IVA*** | | *21%* | *92,81 €* |
| ***Total, con IVA*** | | | *534,78 €* |
| ***Total, con IVA durante 4 meses*** | | | *178,26 €* |

*Tabla 2 Presupuesto Software*

### 3.2.3. Recursos Cloud

La totalidad de los recursos cloud están alojados en la plataforma Azure. A continuación, se desglosarán los gastos realizados en esta plataforma y los costes operacionales del producto.

Como puede verse en la Tabla 3, el coste de los servicios cloud está muy descompensado entre los diferentes tipos de servicio, siendo la base de datos, al utilizar DTU como mecanismo de medida y costo, la opción más barata por el bajo consumo de recursos que se hace. Las máquinas virtuales, sin embargo, son el gasto mayor en un entorno de desarrollo real. Por las particularidades de este proyecto y la no necesidad de tener el flujo CD corriendo permanentemente, se han apagado siempre que no ha sido necesario su uso para ahorrar costes.

| ***Servicio*** | ***COSTE*** |
|---|---|
| Almacenamiento | 51,62 € |
| Máquinas virtuales | 21,07 € |
| Redes virtuales | 26,42 € |
| Registro de contenedores | 29,31 € |
| Base de datos | 9,33 € |
| ***Total*** | *137,75 €* |
| ***IVA (21%)*** | *28,93 €* |
| ***Total, con IVA*** | *166,68 €* |

*Tabla 3 Presupuesto servicios Cloud*

### 3.2.4. Mano de obra

Para el cálculo del coste de la mano de obra se han previsto tres conceptos con tres precios diferentes conforme al trabajo realizado y su dificultad, considerando 30 €/hora para un ingeniero junior.

En la Tabla 4 puede verse el coste en mano de obra del proyecto, las horas están desglosadas por tarea siguiendo la clasificación ya realizada en el punto 3.1.1. El total de horas de realización de este proyecto asciende a 511.

| ***Concepto*** | ***PRECIO/HORA*** | ***HORAS*** | ***TOTAL*** |
|---|---|---|---|
| **PLANIFICACIÓN** | 30 € | 25 | 750,00 € |

| | | | |
|---|---|---|---|
| Diseño | 30 € | 30 | 900,00 € |
| ALM | 30 € | 110 | 3.300,00 € |
| Dispositivos de campo | 30 € | 150 | 4.500,00 € |
| Edge | 30 € | 130 | 3.900,00 € |
| Datos | 30 € | 20 | 600,00 € |
| Visualización | 30 € | 30 | 900,00 € |
| Montaje y ensamblado | 30 € | 16 | 480,00 € |
| ***Total*** | | | *15.330,00 €* |
| ***IVA*** | | *21%* | *3.219,30 €* |
| ***Total, con IVA*** | | | *18.549,30 €* |

*Tabla 4 Presupuesto mano de obra*

### 3.2.5. Coste total del proyecto

El coste total del proyecto es el equivalente a la suma de los costes de los tres capítulos anteriores, para ello se va a suponer el coste real del software para indicar el precio real del servicio que este presta.

Como puede observarse en la Tabla 5, la mayor parte del gasto va destinada a pagar la mano de obra. Esto es debido a que se trata del desarrollo del producto y no a la comercialización. Si la fabricación de dispositivos hubiese sido mayor, el concepto de hardware hubiese sido mucho más alto. Hay que tener en cuenta que esto es un proyecto de prueba. De igual forma, en un entorno de desarrollo completo, el costo de la parte cloud sería mucho más alto por el mayor uso de la base de datos y la necesidad de tener al menos una máquina virtual siempre en funcionamiento.

| ***Concepto*** | ***COSTE*** |
|---|---|
| Hardware | 665,05 € |
| Software | 178,26 € |
| Cloud | 137,75 € |
| Mano de obra | 18.549,30 € |
| ***Total*** | *19.530,36 €* |
| ***IVA (21%)*** | *4.101,38 €* |
| ***Total, con IVA*** | *23.631,74 €* |

*Tabla 5 Presupuesto total del proyecto*

# 4. Descripción del Sistema

Como una definición más formal y acertada de la solución propuesta en el apartado 1.3 y para dar respuesta a los objetivos expuestos en el 1.4; se definen los siguientes requisitos de este proyecto. Para ello se seguirá la clasificación de Koelsch [19]. Se trata de una clasificación binaria entre requisitos funcionales y no funcionales.

## 4.1. Requisitos funcionales

- **RF1:** El sistema debe medir la temperatura y la humedad ambiente.
- **RF2:** El sistema debe medir la cantidad relativa de agua en el suelo.
- **RF3:** El sistema debe medir la presencia de humedad en las hojas.
- **RF4:** El sistema debe activar y desactivar el riego automáticamente dependiendo de las condiciones ambientales.
- **RF5:** El sistema debe utilizar una tecnología de comunicación a larga distancia y bajo consumo para comunicar sensores y actuadores con el nodo edge.
- **RF6:** Los componentes que se sitúen en la plantación deben ser estancos y aguantar las inclemencias climatológicas.
- **RF7:** El consumo energético de los sensores debe estar contenido y estos deben estar alimentados por una batería.
- **RF8:** El sistema debe ser fácilmente instalable por el usuario final tras una pequeña formación.
- **RF9:** El sistema debe ofrecer una interfaz web en la que el usuario final (agricultor) sea capaz de visualizar las mediciones y el funcionamiento de su plantación.
- **RF10:** La interfaz de visualización debe ser personalizable.
- **RF11:** Los actuadores deben funcionar con un suministro de 12V que activará de igual forma la válvula que estos controlan.
- **RF12:** Por cada plantación deben poder existir varios sensores y actuadores.

## 4.2. Requisitos no funcionales

- **RNF1:** La iniciación de nuevos dispositivos por parte del usuario debe ser automática.
- **RNF2:** El sistema debe ser capaz de incorporar un número alto de sensores y actuadores por cada nodo instalado.
- **RNF3:** El sistema debe ser capaz de iniciarse automáticamente tras el encendido del nodo.
- **RNF4:** En caso de fallo, la imagen del nodo se debe reiniciar automáticamente para evitar el tiempo sin servicio.
- **RNF5:** El sistema debe ser capaz de incorporar un número de nodos suficiente para el desarrollo de la actividad.
- **RNF6:** El sistema debe ser escalable.
- **RNF7:** El sistema no debe depender exclusivamente de los productos de un único vendedor y debe poder ser relocalizado si fuese necesario.
- **RNF8:** Las imágenes que se ejecutan en los nodos deben ser automáticamente desplegables a partir del código que se encuentre en un repositorio.

- **RNF9:** Los clientes deben contar con las últimas actualizaciones automáticamente sin necesidad de interactuar y, si sucede, con el mínimo tiempo sin prestar servicio.
- **RNF10:** La capacidad de procesamiento de los nodos debe ser suficiente para trabajar en las plantaciones en las que sean instalados.
- **RNF11:** Los contenidos de los mensajes enviados desde los sensores hacia los nodos y de los nodos hacia los actuadores deben estar cifrados para evitar ser leídos por terceros.
- **RNF12:** Cada sensor y actuador debe tener un identificador único guiado por su hardware (dirección MAC).
- **RNF13:** Los sensores solo deben tomar la posición GPS cada 12 h para ahorrar batería.
- **RNF14:** El panel web debe ser capaz de mostrar todos los datos necesarios para el usuario.
- **RNF15:** El usuario debe tener su propia cuenta dentro del panel web para poder ver únicamente el estado de sus dispositivos.
- **RNF16:** El administrador debe poder crear tantos usuarios como clientes tenga y debe asignar a cada uno de ellos las plantaciones que les pertenezcan.
- **RNF17:** La API debe ser capaz de atender todas las peticiones y debe escalar para ello.
- **RNF18:** Los nodos deben acceder cada hora a la API para así reducir el tráfico web.
- **RNF19:** El sistema debe ser calibrable para cada plantación.

# 5. Diseño del Sistema: Medios y materiales

Para la realización de este proyecto se han utilizado un gran número de tecnologías. En esta sección se exponen todas ellas clasificadas por su categoría. El uso de cada tecnología será justificado y su funcionamiento será definido con mayor profundidad en el siguiente capítulo. En esta sección también se describe el entorno de trabajo y toda la preparación previa.

El Diagrama 3, la arquitectura del sistema junto con las herramientas de soporte del ciclo de vida (ALM). La arquitectura muestra una estructura de capas. Cada una de estas capas cumplen con una función concreta para el sistema general. En la columna de la izquierda se pueden ver, de abajo hacia arriba, las capas desde más proximidad a la plantación hasta la más lejana y próxima al usuario. En la columna de la derecha se encuentra la capa ALM, encargada de servir de soporte para el desarrollo y despliegue de todo el sistema, siendo, por tanto, la capa más cercana al desarrollador, que es transversal, a todas las demás capas de la arquitectura del sistema.

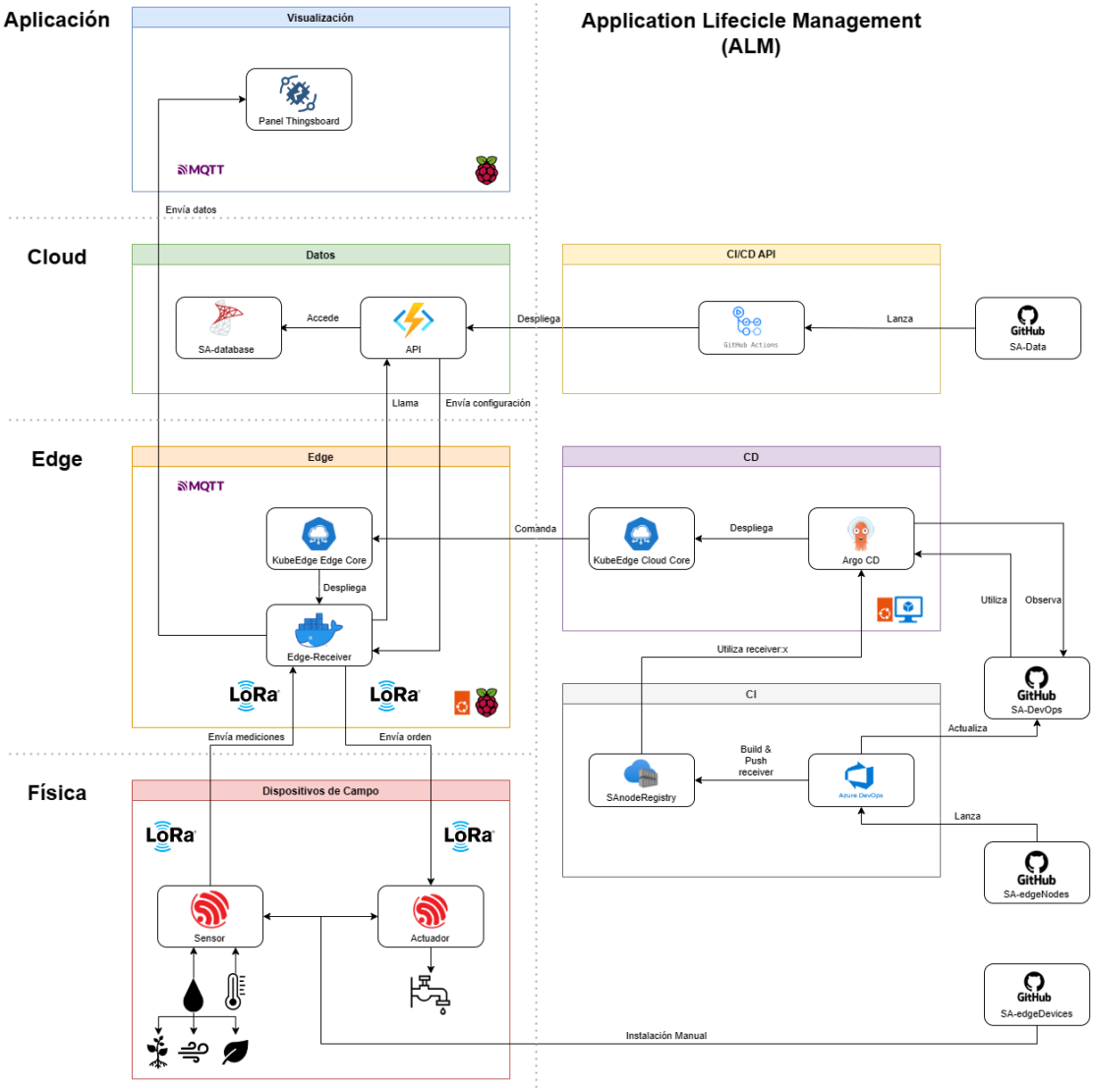



*Diagrama 3 Estructura por capas del proyecto*

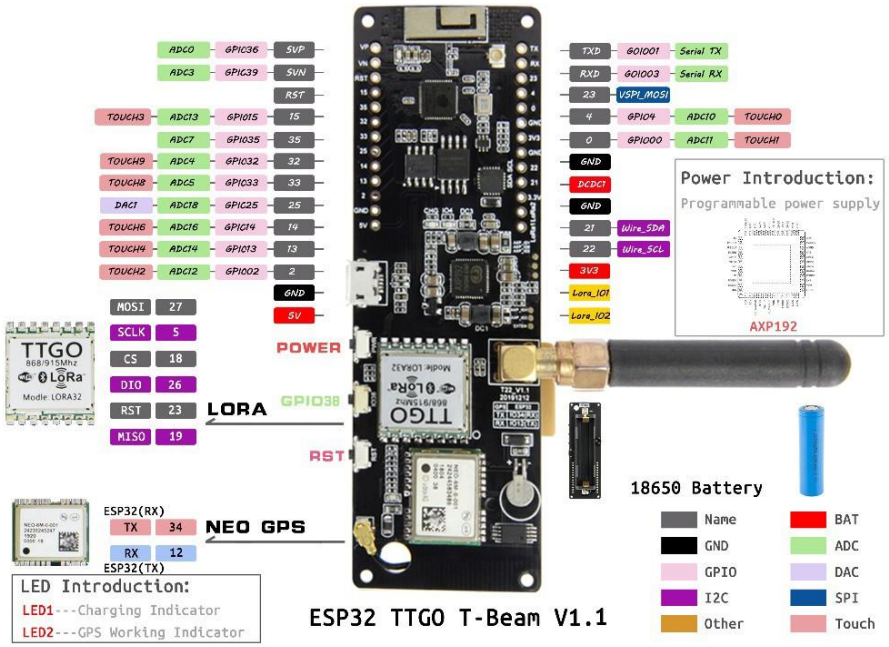
A continuación, se realiza una breve introducción de cada una de ellas:

- CI: comprende la definición y gestión del proceso o flujo de integración continua, la cual es soportada por una serie de herramientas, como por ejemplo Azure DevOps, así como otros servicios *cloud*.
- CD: comprende la definición y gestión del proceso o flujo de despliegue continuo, así como la gestión de la infraestructura de despliegue, en particular, la orquestación de contenedores. Ambas funciones están soportadas por una serie de herramientas, como Argo CD y KubeEdge, que han sido instaladas y se ejecutan en una máquina virtual en la nube de Azure.
- Dispositivos de campo: comprende los sensores y actuadores instalados en las plantaciones de los agricultores, es decir, sobre el terreno.
- *Edge*: comprende los nodos que forman el clúster que compone la capa *edge*. Ejecutan un contenedor llamado "receiver" en el que se procesa la información recibida de los sensores y toma la decisión del riego, además, prepara la información para ser enviada a visualización. Se ejecutan sobre una Raspberry Pi 4B.
- Datos: comprende aquellos componentes software para la gestión de los datos necesarios para el funcionamiento del proyecto, implementados en este caso mediante servicios *cloud*.
- Visualización: comprende un componente software que implementa un panel de visualización de los datos. La plataforma de soporte junto con el código que implementa el panel se ejecuta sobre una Raspberry Pi 3B.

## 5.1. Hardware

### 5.1.1. Sensores

Para los sensores se han empleado placas LilyGO T-Beam v1.1 [20], las cuales incorporan un microprocesador ESP32, un chip GPS NEO 6M y conectividad LoRa a 868 MHz, además de la conectividad WiFi y BLE que no se han utilizado en este proyecto. Se ha seleccionado este modelo de placa por estas características y la presencia de un chip AXP192 que permite configurar su gestión energética. Como sistema de alimentación, se ha elegido una batería LiPo 18650 MJ1 de 3500 mAh de capacidad y de 3.7 V que puede ser conectada directamente en la parte trasera de la placa en un soporte que así lo permite. Se encuentran instalados en la capa física. Pueden verse sus componentes en el Esquema 1, donde, además se muestra la distribución de su GPIO.




*Esquema 1 TTGO T-Beam V1.1. Fuente* [20]

A este módulo de sensorización se le han conectado tres sensores:

- AM2315C: que permite la medida de temperatura y humedad ambiente al incorporar un módulo AHT20.
- Capacitive Soil Moisture Sensor V1.0 de DFrobot: permite la medición de la humedad en el suelo.
- YL-83: posibilita la medición de la cantidad de agua depositada en las hojas de la plantación.

El primer sensor se conecta utilizando los puertos correspondientes al protocolo *Inter-Integrated Circuit* (I²C) tal y como puede verse en el Esquema 3. Este protocolo permite comunicar dos dispositivos punto a punto de forma directa, sin la necesidad de una red. Utiliza dos líneas de comunicaciones SDA (*Serial Data*) y SCL (*Serial Clock*), a los que en este caso se suma la alimentación del sensor ya que esta la provee la propia placa. Los dispositivos pueden estar identificados dentro del sistema con identificadores únicos, aunque en el caso de este sensor en concreto siempre está identificado por la dirección `0x05C`. De esta forma, se puede identificar y comunicar al sensor utilizando las líneas de comunicación previstas y la longitud del cable no altera las mediciones como en los sensores analógicos. Otra ventaja que tiene el uso de este protocolo es la conexión en cadena de los diferentes sensores que pueden componer la red, por la imposibilidad de encontrar sensores que utilizasen este protocolo, son analógicos y deben ser calibrados individualmente.

El segundo sensor utiliza el principio de la capacitancia para la medición de la humedad relativa en el sustrato en el que se encuentra instalado, o lo que es lo mismo, la proporción de agua que hay en este suelo. En el Anexo 10.6 se puede ver cómo se ha realizado el proceso de calibración de estos sensores.

El tercer sensor se compone de dos partes: una placa medidora que simula ser una hoja y un preamplificador que incorpora un comparador y se sitúa en el interior de la caja estanca. Emplea un comparador para señalar cuándo hay gotas de lluvia sobre el sensor y cuándo no. Esto se logra comparando la resistencia de la placa medidora expuesta (siendo

infinita cuando no hay agua y, gradualmente menor, cuando sí que lo hay) y la que ofrece un potenciómetro previamente configurado. Esta calibración se hace previamente y para cada pareja de sensor y preamplificador. Este último indica con un LED cuando su salida digital (la utilizada) está activa y cuándo no, es decir, cuando detecta agua y cuando no, de esta forma podemos ajustar el nivel de activación con el potenciómetro para que se active en el momento en que nos parezca conveniente. En este caso se ha seleccionado un nivel de activación medio, obtenido con dos pulverizaciones de agua. El nivel de agua en la hoja no es tan importante, conocer si hay agua, sí que lo es. Esta agua no es necesario que se deba a la lluvia, puede corresponder también a la condensación de agua por el rocío u otros fenómenos meteorológicos como la niebla.  Se trata de una medida muy útil en conjunto, junto con la de temperatura y humedad ambiente.

#### 5.1.1.1. Aspecto e instalación

Como una prioridad y requisito es la impermeabilidad del sensor para su posible instalación en el exterior, como se ve en la Ilustración 3, ha sido instalado en una caja estanca que cumple con la certificación IP65. Los diferentes sensores salen de la caja a través de unos prensaestopas que cumplen la misma norma y la antena a través de un agujero y asegurada con una goma. Como ninguno de los sensores cuenta con un certificado IP oficial, se ha hecho todo lo posible por hacer que lo sean.

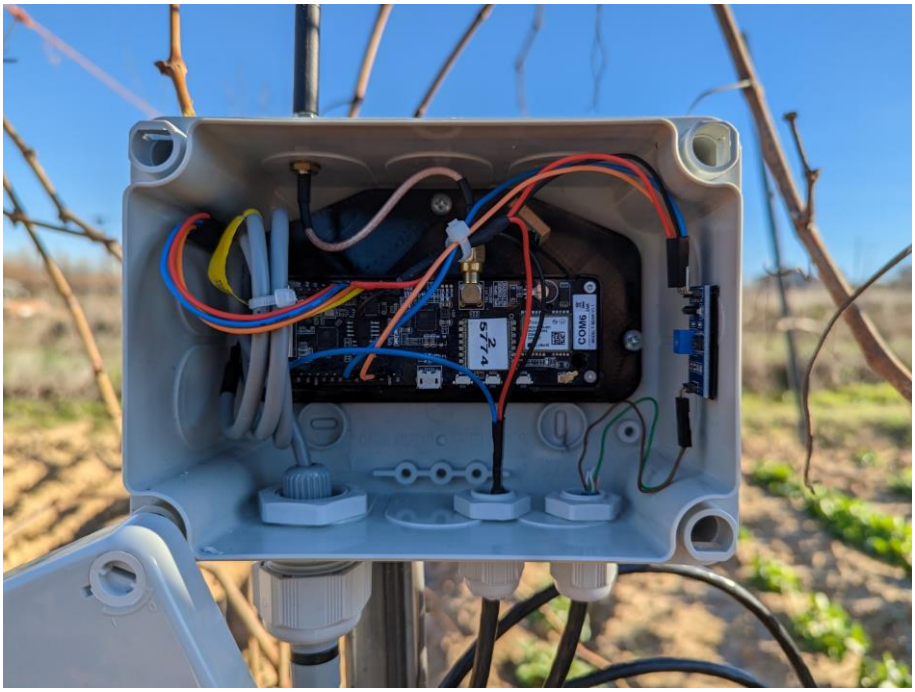


*Ilustración 3 Disposición interior del Sensor*

El sensor AM2315C se ha instalado en la parte inferior de la caja para evitar que caiga agua en su interior. No se ha observado ningún problema en su funcionamiento al estar instalado en esta posición. Únicamente asoma la parte medidora estando el resto del sensor y su cableado en el exterior.

El sensor de humedad en el suelo ha sido instalado y protegido por una regleta eléctrica de similar ancho y aislado usando silicona. De esta forma sus circuitos expuestos han sido protegidos del agua y polvo que hay en la tierra en la que se encuentra instalado. Está completamente enterrado a 40 cm de la superficie. Para su instalación se ha soldado un cable ethernet CAT6 UTP rígido de sección 24 awg indicado para su uso en el exterior.

De esta forma se ha podido instalar a una mayor distancia y permite que el cable aguante las inclemencias meteorológicas.

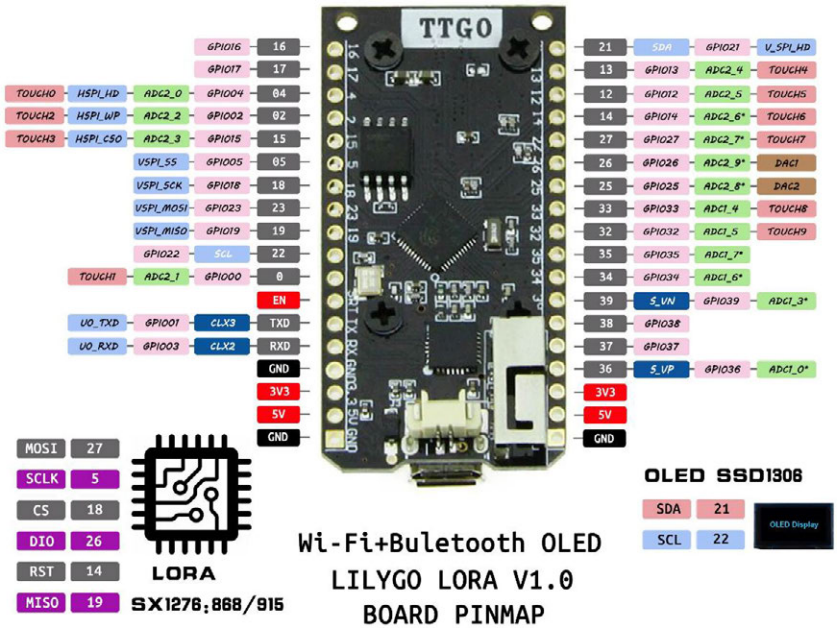
El sensor YL-83 ha sido aislado igualmente utilizando silicona para que no caiga agua entre los dos terminales y desvirtúe su funcionamiento. Este sensor es el que más problemas tendrá asociados dado que su funcionamiento se basa en mojar las láminas metálicas que conforman el sensor que simula ser una hoja. El preamplificador se sitúa en el interior de la caja estanca, por lo que no presenta ningún problema de estanqueidad. Se ha utilizado un cable como el del caso anterior para poder instalarlo sobre un árbol.

Todo el cableado de maniobra situado en el interior de la caja es de sección 25 AWG, suficiente para permitir un correcto funcionamiento y no suponer un problema. Las conexiones se han realizado utilizando conectores de tipo DuPont. Para el anclaje y protección de la placa a la caja estanca, se ha creado un soporte cerrado, Esquema 5, al que se atornilla la placa empleando los orificios disponibles para ello. De esta forma se protege y fija la placa de una forma segura y se aprovechan orificios de sujeción ya presentes en la caja estanca para anclar este conjunto a la caja. Esta pieza ha sido impresa utilizando una impresora 3D y utilizando filamento PLA. El preamplificador del sensor de gotas ha sido pegado en un lateral de la caja.

En el Anexo 10.1 pueden verse más detalles de la instalación con sus correspondientes fotografías.

### 5.1.2. Actuadores

Los actuadores están basados en la placa LilyGO LoRa V1.0 ESP32 [20]. Esta placa ha sido seleccionada porque comparte una gran parte de la arquitectura de la utilizada para los sensores, a excepción del chip GPS y AXP, que en este caso no será necesario ya que estarán situadas en una posición fija y alimentada por una fuente externa. Esto se justifica porque van alimentadas y situadas junto con las válvulas que permiten el riego a las que acompañan. Debido a su bajo costo y simplicidad, permiten la creación de válvulas de riego inteligentes y eliminan la necesidad de sistemas de control generales. Se encuentran instalados en la capa física. En el Esquema 2 pueden verse los componentes y pines de conexión de la placa utilizada, como puede observarse, es más simple que la anterior.




*Esquema 2 LilyGO LoRa V1.0 ESP32. Fuente* [20]

A este módulo, se le ha incorporado el control de una válvula accionada por solenoide que requiere de 12 V. Esta válvula controla el suministro de agua a un circuito de riego por goteo, permitiendo con su apertura que se inicie el riego. Su funcionamiento es normalmente cerrado para que, en caso de fallo en el suministro eléctrico, cese el riego ya que esta es la opción más lógica y segura.

#### 5.1.2.1. Aspecto e instalación

Como puede verse en el Esquema 4 la alimentación de la placa es de 12 V que, con un conversor se adapta a los 5 V necesarios (sería más eficiente con 3,3 V, pero se ha decidido esta alimentación por ser más económico el conversor). Esta alimentación proviene de una batería alimentada con un panel solar ya disponible en la plantación y utilizada para la extracción de agua de un pozo. Esta decisión se ha tomado porque es necesario contar con esa alimentación para el accionamiento de la válvula y, de esta forma, se elimina la necesidad de contar con una batería adicional cuya existencia no tendría ningún sentido en caso de fallo del suministro de energía a la válvula. La placa debe funcionar cuando la válvula puede hacerlo. Como mejora, de esta forma se cuenta con una masa o GND común a ambos elementos que es la que se abre o cierra utilizando un MOSFET cuando la placa así lo pide.

Al igual que en el caso de los sensores y como puede verse en la Ilustración 4, la impermeabilidad es un factor imprescindible. Por ello todos los componentes principales han sido instalados en el interior de una caja estanca. Al igual que en los sensores, la entrada y salida de cables se ha realizado utilizando prensaestopas que permiten la estanqueidad del conjunto. La antena, al igual que en el caso anterior, ha sido extraída a través de un agujero y asegurada con una goma para evitar la entrada de líquido. Tanto el cable exterior como el de maniobra del interior de la caja es de 1 mm² de sección de cobre, en una manguera adecuada para el exterior para llevar la alimentación a la caja y de la caja a la válvula. Los conectores empleados en la válvula son del tipo lengüeta o *faston* aislados y, para conectar el MOSFET a la placa, de tipo DuPont.

*Ilustración 4 Disposición interior del Actuador*

Como puede verse en el Esquema 6, para proteger y anclar la placa a la caja se ha diseñado un soporte que permite ser atornillado a la caja estanca utilizando un único tornillo que se atornilla a un inserto roscado instalado en un orificio a tal efecto de la caja estanca. Este soporte estabiliza y protege a la placa, que se ancla a ella por medio de dos tornillos. De igual forma, el MOSFET también se ha anclado a la caja utilizando el mismo método en otro orificio disponible. Esta pieza ha sido impresa utilizando una impresora 3D y con filamento PLA que es idóneo para esta aplicación.

Pueden verse más imágenes de cómo ha sido la instalación sobre el terreno de los actuadores en el Anexo 10.2.

### 5.1.3. Nodos edge

La capa edge desarrollada está compuesta por una Raspberry Pi 4B [21] acoplada a un "sombrero" Adafruit LoRa Radio Bonnet OLED - RFM95W [22] que permite que esta reciba datos de los sensores y envíe instrucciones a los actuadores. Este nodo está instalado en una planta dependiente del agricultor y desde la que tiene acceso a alimentación eléctrica e Internet.

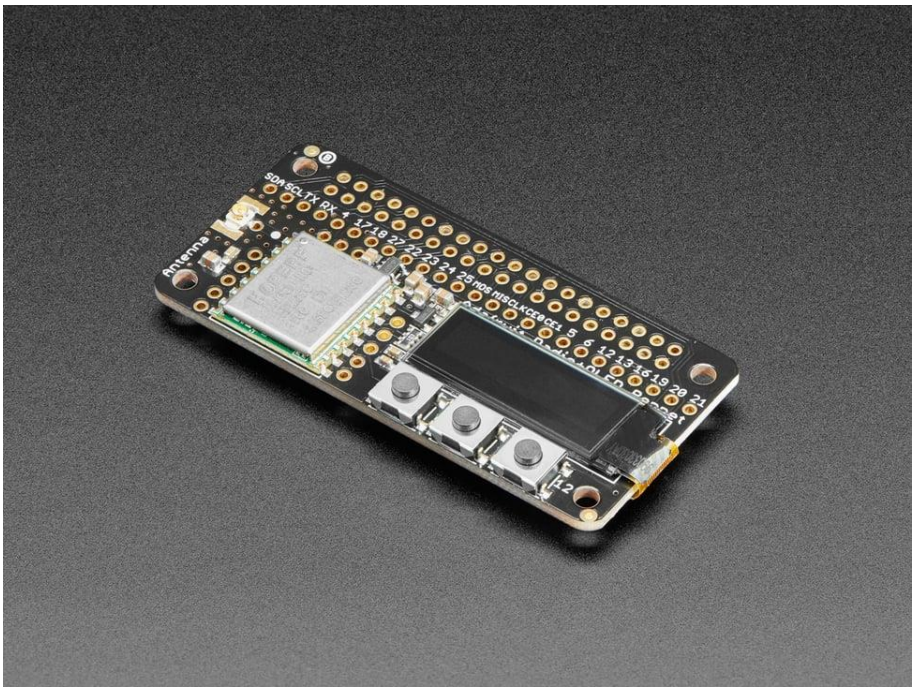



*Ilustración 5 Adafruit LoRa Radio Bonnet OLED - RFM95W. Fuente* [22]

La decisión de este hardware está justificada por la buena relación entre rendimiento y consumo eléctrico que presenta. El consumo eléctrico máximo se sitúa en 6 W, aunque el consumo más habitual se sitúa en torno a los 4 W. Su rendimiento es suficiente para esta tarea, manteniendo un uso de CPU inferior al 20 % y de RAM de unos 512 Mb. Estas métricas son justificables por la arquitectura ARM que emplea. Esto ha hecho necesario adaptar todos los despliegues para la arquitectura Linux/ARM64 y no la tradicional x64.

Actualmente, la disponibilidad de estos microcomputadores es muy limitada, siendo muy difícil poder adquirir una, aunque esto se prevé que, con el paso del tiempo, se facilite. Por lo tanto, la opción más correcta, tanto por la relación entre rendimiento y consumo y el sombrero LoRa seleccionado, es el modelo de Raspberry Pi seleccionado. No se han podido encontrar alternativas funcionales con mejores características sin sacrificar ninguno de estos dos puntos clave.

#### 5.1.3.1. Aspecto e instalación

La instalación se ha realizado en una caja de PVC ya que este material no bloquea las conexiones electromagnéticas (ver Ilustración 6). Para la alimentación se han utilizado módulos oficiales de Raspberry Pi que permiten una alimentación ininterrumpida a 5 V @ 3 A, que es lo que necesita una Raspberry Pi 4B para su funcionamiento. La propia Raspberry Pi se encarga de alimentar tanto el sombrero como el ventilador de 80 mm que funciona conectado a un USB y que refrigera a todo el conjunto. Para mejorar la disipación térmica, se han instalado disipadores de aluminio que, junto con el ventilador, contienen sobradamente la temperatura de la Raspberry Pi a unos 35 ºC con una temperatura ambiente de 25 ºC. Con esta instalación se garantiza que no bajará el rendimiento del equipo por un exceso de temperatura y siempre funcionará a su máximo rendimiento.

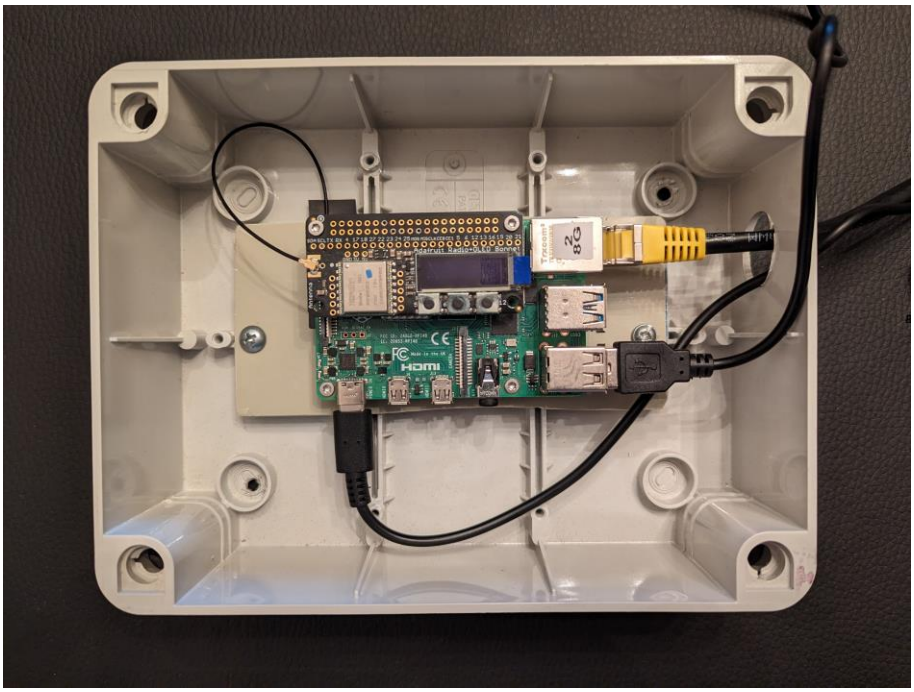


*Ilustración 6 Disposición interior del nodo edge*

Para el anclaje de la Raspberry Pi a la caja se han empleado separadores de bronce. De igual forma y para mejorar y asegurar la conexión del sombrero con los puertos GPIO, también se han utilizado estos mismos separadores.

### 5.1.4. Servidor de visualización

El panel de visualización se encuentra instalado en una Raspberry Pi 3B. Ofrece 1 GB de memoria y un procesador Broadcom BCM2837 con cuatro núcleos a 1,2 GHz [23]. Presenta algunos problemas como su deficiente capacidad de comunicación, con capacidad para redes de hasta 100 Mb, pero es suficiente para esta prueba de concepto. Sería necesario adaptarlo en el caso de realizar un despliegue de mayor tamaño a un hardware más potente o un servidor cloud dedicado. Se ha tomado la decisión de utilizar este hardware por estar disponible, su bajo consumo eléctrico (similar al de la Raspberry Pi 4 B del punto 5.1.3) y el ahorro económico que supone el no desplegar un servidor cloud dedicado.

#### 5.1.4.1. Aspecto e instalación

Para su correcto funcionamiento, se ha instalado en una carcasa que protege a la placa y que cuenta con un pequeño ventilador para su refrigeración. Para la alimentación se ha utilizado una fuente que permite una alimentación a 5 V @ 2 A que permite un funcionamiento correcto. De esta forma, puede ser instalada donde se quiera, no requiere estar emplazada en la propia plantación, sino que puede estar establecida en otro lugar dado que la latencia no es un factor limitante para su funcionamiento.

### 5.1.5. Máquina virtual para despliegue continuo

Para ejecutar las herramientas correspondientes al despliegue continuo, se ha utilizado una máquina virtual Azure “standard B2s” con 2 núcleos virtuales y 4GiB de memoria [24]. Para su configuración se ha utilizado la interfaz gráfica de la que Azure dispone.

Sus especificaciones han sido seleccionadas siguiendo las herramientas software que en ella se van a ejecutar.

#### 5.1.5.1. Configuración e instalación

La máquina virtual utiliza el sistema operativo Ubuntu Server 22.04 LTS, ideal para la instalación de las herramientas necesarias. Ha sido configurada también una red virtual y se han redirigido todos los puertos que los servicios software han requerido. Como añadido, se ha utilizado una dirección IP pública estática a la que se le ha asignado un servicio DNS con su correspondiente URL identificativa. Gracias a esta configuración, la máquina es accesible desde el exterior utilizando una dirección legible y personalizable.

## 5.2. Comunicaciones

Para las comunicaciones se ha empleado LoRa, en cada dispositivo se ha desarrollado de una forma diferente por sus particularidades, pero tienen puntos en común que se mostrarán primero.

### 5.2.1. Mensajería LoRa

LoRa (*Long Range*) [25] es una tecnología de comunicación inalámbrica de largo alcance (unos 20km, pero depende de los emisores y los receptores, y de la orografía) que se utiliza para la transmisión de pequeños paquetes a una gran distancia. Para ello utiliza bandas sin licencia, en el caso de Europa es la banda EU868 (863-873 MHz) que son de uso libre. Utiliza la tecnología de modulación *Chirp Spread Spectrum* (CSS) para transmitir la señal a través del aire. Su principal ventaja es el bajo consumo de energía, lo que hace que sea ideal para el escenario de este proyecto. Por eso se ha decidido su uso para este proyecto.

### 5.2.2. MQTT

MQTT (*Message Queue Telemetry Transport*) [26] es un protocolo de comunicación de mensajería de código abierto especialmente indicado para dispositivos conectados a redes con bajo ancho de banda. Es un protocolo de comunicación ligero y seguro que, por las características anteriores, es muy utilizado en IoT. Para su funcionamiento necesita un servidor central o bróker que actúa como un intermediario entre los clientes. Este bróker distribuye los mensajes que recibe desde los dispositivos de la red y los publica en tópicos específicos. Para recibir los mensajes, los dispositivos deben suscribirse a los tópicos de su interés y, de esta forma, reciben solo los datos que necesitan y no todos, ahorrando recursos de esta forma.

### 5.2.3. Identificación de los dispositivos

Para la identificación de los dispositivos físicos, se ha utilizado la dirección MAC de su tarjeta de red WiFi. Esta información está contenida en ellos mismos de fábrica (E_FUSE) [27] y hace que sea muy fácil su identificación. Para la obtención de dicha información se ha desarrollado una herramienta llamada “ESP_Identifier” que simplemente imprime por la salida serial la dirección MAC almacenada en cada dispositivo. Posteriormente esta información es almacenada en el campo “serial” de la tabla “sensor” dentro de la base de datos. De esta forma, sin la necesidad de programar cada placa para almacenar un código único, se consigue la misma función.

La dirección MAC se divide en 6 octetos, de los cuales los 3 primeros identifican al fabricante (`24:0A:C4` en el caso de Espressif) y los 3 últimos al dispositivo en sí. Por las características de este mecanismo pueden existir colisiones, aunque estas son extremadamente raras ya que Espressif tiene reservadas direcciones MAC para 400 millones de dispositivos. Por las características de la comunicación LoRa, se hace realmente difícil que se produzcan estas colisiones en el radio de alcance de un nodo y solo sería necesarios comprobar que no existan colisiones en los dispositivos que se encuentren en este radio.

Por lo tanto, los tres últimos octetos de la dirección MAC del WiFi de los ESP32 son los elegidos como identificadores únicos de cada dispositivo.

### 5.2.4. Estructura de mensajes y su seguridad

El protocolo de mensajes es muy simple, consta de una cabecera y un cuerpo para ambos casos de mensajes: del sensor al nodo y del nodo al actuador. El Diagrama 4 muestra un ejemplo de mensaje formateado. Se divide en tres partes:

- Cabecera: muestra la dirección MAC del sensor.
- Delimitador cabecera/cuerpo: es el carácter (“#”) encargado de separar ambas partes para el posterior procesado del mensaje.
- Cuerpo: muestra las mediciones, que son las siguientes:
    - Batería conectada: booleano
    - Voltaje de la batería: coma flotante con 2 decimales
    - Latitud: coma flotante con 5 decimales
    - Longitud: coma flotante con 5 decimales
    - Altitud: coma flotante con 5 decimales
    - Número de satélites: entero
    - Temperatura del aire: coma flotante con 2 decimales
    - Humedad del aire: coma flotante con dos decimales
    - Humedad del suelo: entero
    - Presencia de gotas de agua en las hojas: booleano

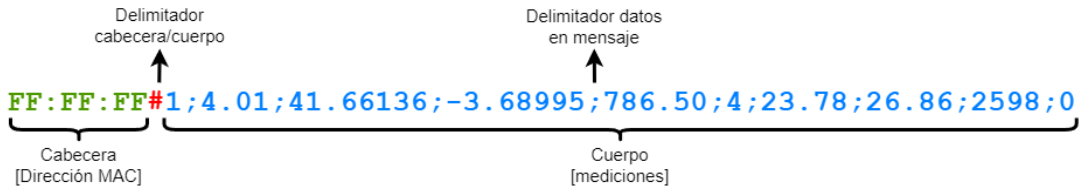



*Diagrama 4 Estructura mensaje Sensor a Nodo*

El Diagrama 5 muestra un ejemplo del mensaje formateado que el nodo envía al emisor. Se divide en 4 partes:

- Delimitador de inicio del mensaje: carácter (“!”) indica el comienzo del mensaje para el receptor.
- Cabecera: muestra la dirección MAC del sensor.
- Delimitador cabecera/cuerpo: es el carácter (“#”) encargado de separar ambas partes para el posterior procesado del mensaje.
- Cuerpo: JSON que indica la necesidad o no de regar.

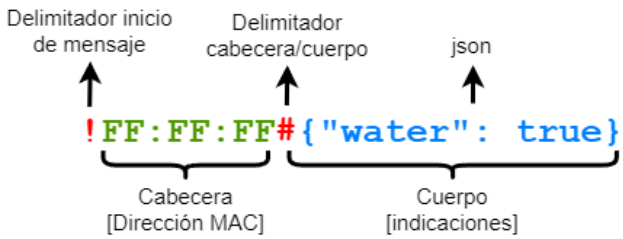



*Diagrama 5 Estructura mensaje Nodo a Actuador*

Como puede verse en ambos casos, la estructura es muy similar y surge para facilitar y optimizar las comunicaciones entre los dispositivos físicos y el nodo. Este protocolo creado se ha implementado en todos los dispositivos, en los puntos posteriores se explicará cómo para cada caso.

La seguridad en las comunicaciones se ha logrado implementando el algoritmo de cifrado XXTEA [28], que está especialmente indicado para ser rápido y requerir pocos recursos. Su uso es muy habitual en dispositivos IoT por estos motivos. Utiliza claves de 128 bits y se basa en el cifrado por bloques, es decir, divide la información a cifrar en bloques y, luego, cifra estos bloques. Tiene la particularidad de utilizar una técnica de desplazamiento circular para conseguir bloques de un tamaño específico y, por lo tanto, no utiliza una técnica de relleno. El cifrado es simétrico, por lo que emisor y receptor deben contar con la misma clave para poder cifrar y descifrar, respectivamente, los mensajes. Se considera que es un algoritmo de cifrado seguro, aunque que sea simétrico hace que sea más vulnerable. Para este uso se ha considerado que es un algoritmo lo suficientemente seguro por la naturaleza de los datos que son enviados. También, se ha seleccionado este algoritmo por ser de fácil implementación, tanto en los dispositivos físicos como en los nodos. Utilizando para los primeros la biblioteca (*library*) "xxtea-iot-crypt" de Abhijit Bose [29] y para los segundos "xxtea" de ifduyue [30].

## 5.3. Software

A continuación, se expondrán todas las piezas software que componen este proyecto y cómo interactúan entre ellas.

### 5.3.1. Datos de configuración

Para el almacenamiento de todos los datos necesarios para la ejecución del código, es necesario que estos estén estructurados y sean fiables. Para ello, se necesita una base de datos relacional en la que almacenar toda esta información.

#### 5.3.1.1. La base de datos: SQL Server

Esta base de datos será un servicio SQL Server de Microsoft [31] alojado en Azure. Esta decisión se ha tomado por la facilidad de trabajar con las herramientas de Microsoft en Azure. Otras bases de datos relacionales como MySQL o PostgreSQL no ofrecen tantos beneficios en Azure ya que no permiten su ejecución por *Database Transaction Unit* (DTU) [32] y requieren de una máquina dedicada con lo que el coste es mayor. La ventaja principal del modelo por DTU es que su coste está regido por unidades de procesamiento y operaciones de entrada/salida, lo que permite que para una aplicación básica como esta sea utilizada su opción más básica de bajo coste y no sea necesaria una máquina dedicada más potente cuyo rendimiento va a estar parado la mayor parte del tiempo de ejecución del sistema. En el caso de que con la capa básica no fuese suficiente por la existencia de

un gran número de nodos conectados, ampliar el número de DTU es muy fácil y su aplicación es casi instantánea. Esta facilidad no existe para el caso de máquinas dedicadas.

Para acceder e interactuar con la base de datos, se utiliza el programa SQL Server Management Studio (SSMS) [33], que permite ejecutar cualquier mandato sobre la base de datos con una interfaz gráfica y sin la necesidad de utilizar Transact-SQL [34] (la versión de Microsoft del lenguaje SQL) para realizar muchas acciones. Esto reduce los errores y mejora la eficacia a la hora de trabajar con esta base de datos.

Como puede verse en el Diagrama 6, la estructura entidad-relación de la base de datos empleada es completa y permite almacenar todos los datos concernientes al sistema. En verde se encuentran marcado las entidades y las relaciones que generan tabla (cardinalidad n:m) con los nombres de las tablas que, finalmente, generaron.

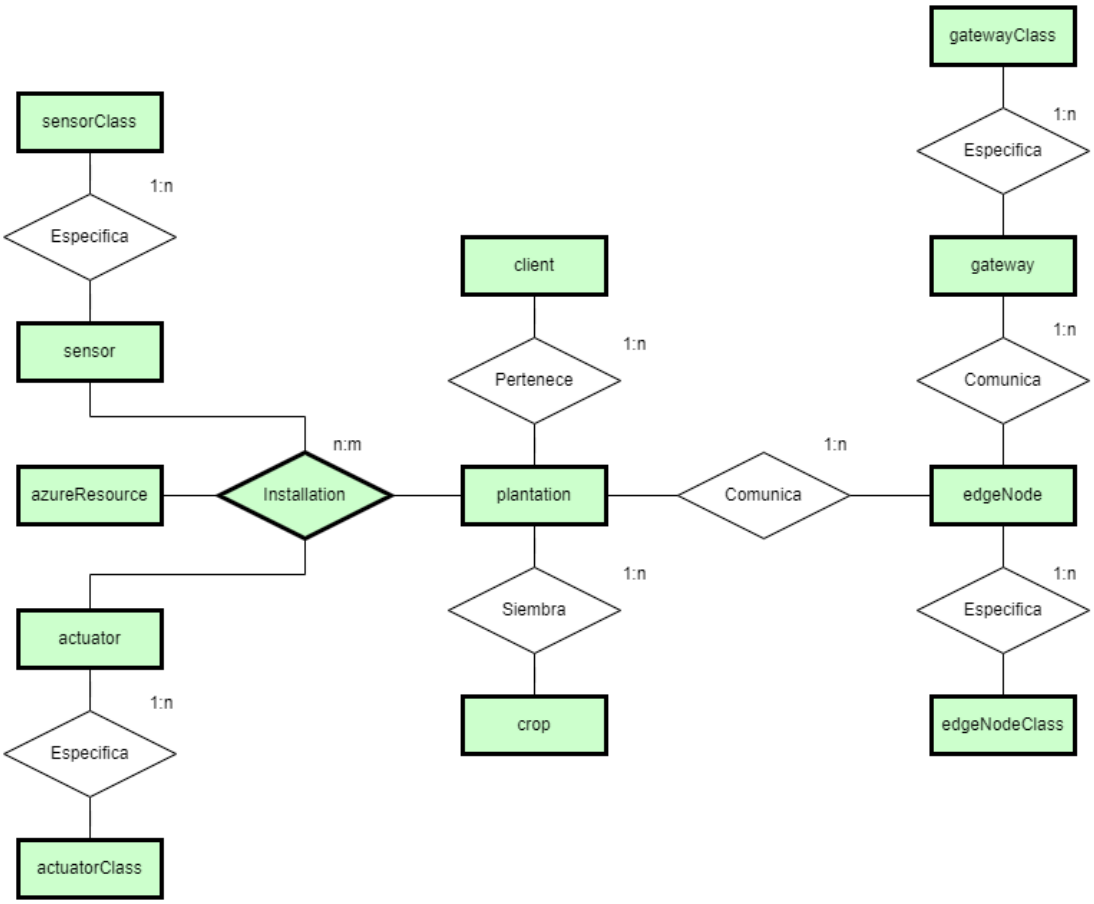



*Diagrama 6 Modelo Entidad-Relación*

Como puede verse en el Diagrama 7, se han creado todas las tablas necesarias para el correcto funcionamiento de la base de datos. Las funciones de cada tabla son las siguientes:

- actutorClass y sensorClass: almacena los datos de cada tipo de actuador o sensor respectivamente, existiendo una posible variedad de ellos.
- actuator y sensor: almacena cada instancia específica de un actuador o un sensor, cada uno con sus peculiaridades.
- azureResource: almacena los datos de los distintos recursos de Azure disponibles.

- edgeNodeClass: guarda los datos de las distintas variedades de nodos que están preparados para su despliegue en la capa *edge*.
- edgeNode: guarda cada instancia específica de un nodo edge con sus detalles particulares.
- gatewayClass: almacena los distintos tipos de *gateways* disponibles.
- gateway: almacena cada registro de un Gateway.
- crop: almacena cada tipo de plantación y sus peculiaridades.
- client: almacena la información de los clientes.
- plantation: guarda los datos específicos de cada plantación y qué recursos tiene asignados.
- installation: acumula la información de cada una de las instalaciones realizadas de sensores y actuadores en las plantaciones.

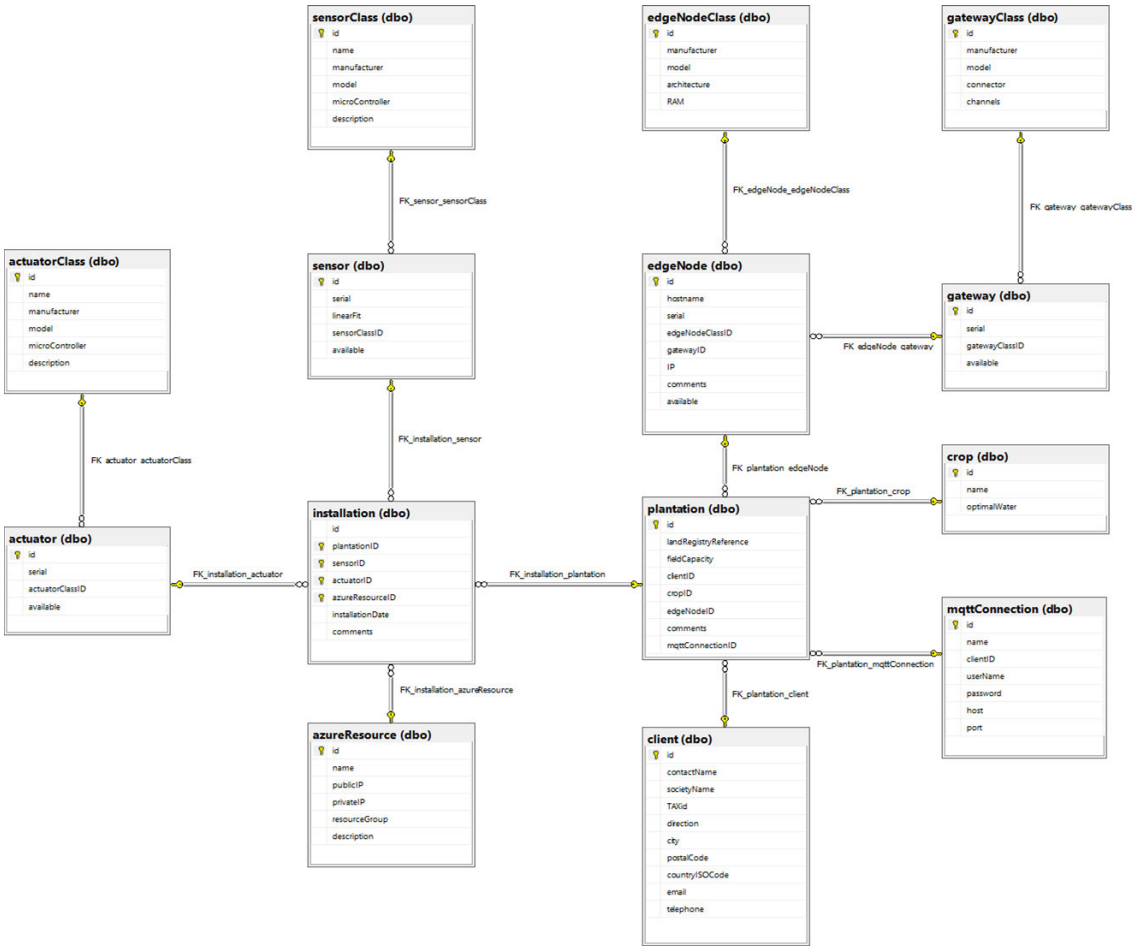



*Diagrama 7 Tablas y estructura de la base de datos*

Gracias a estas tablas y sus relaciones, se facilita la gestión de todos los activos que componen la base de datos. Está diseñada para posibilitar una futura evolución del sistema y su adaptación a los distintos escenarios que pueden presentarse. Con una simple edición se pueden añadir nuevos dispositivos y estos ya pueden ser instalados por cualquier persona cualificada cumpliendo así con el requisito *plug-and-play*.

#### 5.3.1.2. API: Azure Function

Para acceder a estos datos desde un nodo, se ha implementado una aplicación que, para mejor funcionamiento y sencillez de despliegue se ha hecho utilizando Azure Functions

[35]. Este programa es una Interfaz de Programación de Aplicaciones (API o *Application Programming Interface*), que permite la integración de los dos sistemas: base de datos y programa que recibe y trata la información en el *edge* con una simple conexión. Esta API está programada con C# utilizando para ello el IDE Visual Studio [36] e implementa tres funciones:

- GetEdgeNodes: esta función devuelve una lista de todos los nodos que están registrados en el sistema y sus datos fundamentales. Con el parámetro "hostname" devuelve el ID correspondiente a ese nodo. La única información con la que cuentan los nodos de sí mismos es su *hostname*.
- GetEdgeNodesConfig: esta función devuelve todos los parámetros que un nodo necesita para su funcionamiento inicial:
    - ID de los sensores que tiene asignados.
    - Seriales de los sensores que tiene asignados.
    - Ajuste de los sensores de humedad en el suelo para ese sensor.
    - ID de las plantaciones en la que se encuentran esos sensores.
    - Capacidad de campo de dichas plantaciones.
    - Cantidad óptima de agua para dicha plantación.
- GetActuatorsFromPlantationID: devuelve la información de los posibles actuadores que lleva aparejada esa plantación en base al ID de la plantación.
- GetMqttConnectionFromPlantationID: devuelve la información para la conexión MQTT de cada plantación.

El despliegue de esta aplicación se produce automáticamente (siguiendo la cultura DevOps de este proyecto) pero utilizando GitHub Actions [37]. Este *pipeline* automáticamente despliega en Azure Functions el código, sin la intervención de ningún agente externo.

#### 5.3.1.3. Procedimientos guardados

Una de las funciones ofrecidas por SQL Server es el almacenamiento de procedimientos (*stored procedures*) [38]. Estos procedimientos son consultas SQL que se almacenan en una sección de la propia base de datos y que permiten realizar una consulta simple que sea mucho más compleja en realidad. Para el funcionamiento de la API, se han desarrollado tres diferentes, uno para cada una de las funciones más complejas. De esta forma, el código de la API es más limpio y permite ser más eficiente con el uso de la red ya que no es necesario enviar tanta información en la petición, únicamente el procedimiento al que hace referencia y el parámetro de entrada que esta función necesita. Además, añade una capa de seguridad al no mostrarse en el mensaje el código exacto, nombre de tablas, parámetros, etc. Pese a que el envío de datos es seguro y cifrado, de esta forma se añade una capa adicional de seguridad.

### 5.3.2. Microk8s

Microk8s [39] es una distribución ligera de Kubernetes, se ejecuta directamente en el host y es fácil de instalar y utilizar por diseño. Es ideal para proyectos como este ya que, por la propia arquitectura del sistema, únicamente existirá un nodo Kubernetes, la sencillez del sistema a implementar no justifica una instalación completa de Kubernetes de Kubernetes. Otro motivo por el que una instalación completa de Kubernetes no está indicada para este uso son los requerimientos mínimos, requiere al menos 2 GB de memoria y una CPU de al menos 2 núcleos [40] frente a los 540 MB y una CPU de 1

núcleo [39]. Estos requisitos adicionales añaden costes operativos que no se ven compensados por las prestaciones añadidas, ya que no son necesarias. Por lo tanto, Microk8s es una de las mejores soluciones a esta problemática. Microk8s incorpora su propio kubelet y otros *plug-in* de los cuales se han activado: dns, dashboard y storage [41]. De esta forma se ha logrado realizar una instalación de Kubernetes fácilmente e incorporando una solución ligera suficiente para las necesidades del proyecto.

Se ha instalado en la capa ALM dentro de la máquina virtual encargada del despliegue continuo (ver 5.1.5).

### 5.3.3. KubeEdge

Para la implementación de los núcleos IoT Edge y permitir su control y despliegue continuos se ha utilizado KubeEdge [42], que no es más que una plataforma de código abierto que permite la orquestación de aplicaciones basadas en contenedores que se ejecutan sobre nodos con restricciones computaciones, en este caso, la orquestación de aplicaciones que se ejecutan en la capa *edge*. KubeEdge está basada en Kubernetes y cuenta con varias de las ventajas de su plataforma primogénita, así como compatibilidad con los desarrollos basados en Kubernetes. KubeEdge extiende lo que nos ofrece Kubernetes al *edge*. La capa *edge* presenta varias dificultades como puede ser la replicación del servicio o la actualización de las aplicaciones utilizando herramientas CD, gracias a KubeEdge se resuelven en gran medida.

La instalación de KubeEdge es sencilla, pero debe ser seguida con exactitud para no hallar futuros fallos. Previamente a iniciar la instalación, es necesario conocer la estructura básica y sus funciones.

Como puede verse en el Diagrama 8, la arquitectura de KubeEdge se divide en dos partes: *cloud* y *edge*. El CloudCore debe ser instalado en al menos una máquina cloud y el EdgeCore debe ser instalado en cada uno de los nodos *edge* que conforman el sistema.

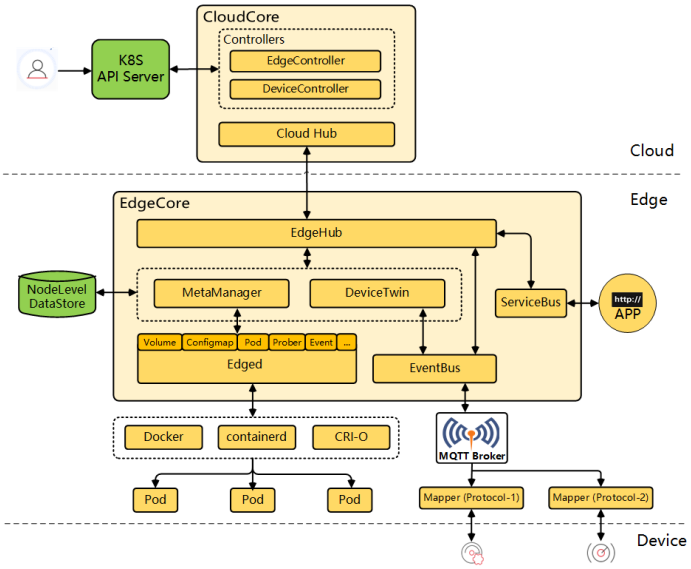



*Diagrama 8 Esquema de KubeEdge Fuente:* [43]

CloudCore debe ser instalado sobre un clúster de Kubernetes ya desplegado, en este caso se ha utilizado el clúster Microk8s del punto anterior. Para ello solo es necesario seguir

esta guía [44]. Una vez instalado, se generará un *namespace* llamado "kubeedge" en el que se podrán encontrar todos los *pods*.

EdgeCore se instala en cada nodo IoT Edge que compone el sistema, para ello simplemente es necesario utilizar una herramienta proporcionada por KubeEdge (keadm) que permite, junto con un código que se extrae del CloudCore, relacionar ese nodo con el CloudCore correspondiente [45]. Lo único que necesita el nodo es tener configurado correctamente tanto Docker como su servicio DNS. Para lo primero solo es necesario seguir la guía oficial [46] y para lo segundo solo es necesario activar el servicio `systemd-resolved` en la Raspberry Pi en la que se está instalando el nodo y enlazar mediante un enlace simbólico el fichero que este servicio genera con el estándar de Ubuntu del que kubeadm tomará la configuración para poder conectar los *pod* a Internet (toda la información se encuentra en el Anexo 10.7). Una vez realizados estos dos pasos ya podremos utilizar la herramienta keadm para conectar el nodo al CloudCore e instalar EdgeCore en este.

### 5.3.4. Registro de contenedores

Para el almacenamiento de los contenedores que utilizarán los nodos, se ha utilizado un Registro de Contenedores (*Container Registry*) [47] el cual se utilizará para almacenar dichos contenedores. Esto se ha hecho utilizando la herramienta de Azure que ofrece dicho servicio. Para posibilitar el acceso desde el exterior es necesario habilitar una cuenta que tenga permiso para hacerlo [48].

Con esto, se podrán subir y bajar las imágenes de los contenedores desde los flujos CI y CD respectivamente.

### 5.3.5. Receiver

La herramienta receiver es la pieza clave de este proyecto, supone el centro del procesamiento en el *edge* y realiza todas las funciones que se requieren para el correcto funcionamiento de esta parte.

En el Diagrama 9 se muestra todo el flujo de funcionamiento de la aplicación receiver y las demás herramientas con las que interactúa para su funcionamiento. Además, muestra el flujo del resto de aplicaciones con las que interactúa, de forma que, de un solo vistazo se pueda ver el funcionamiento general de todo el software que se expondrá en este punto y en los siguientes.

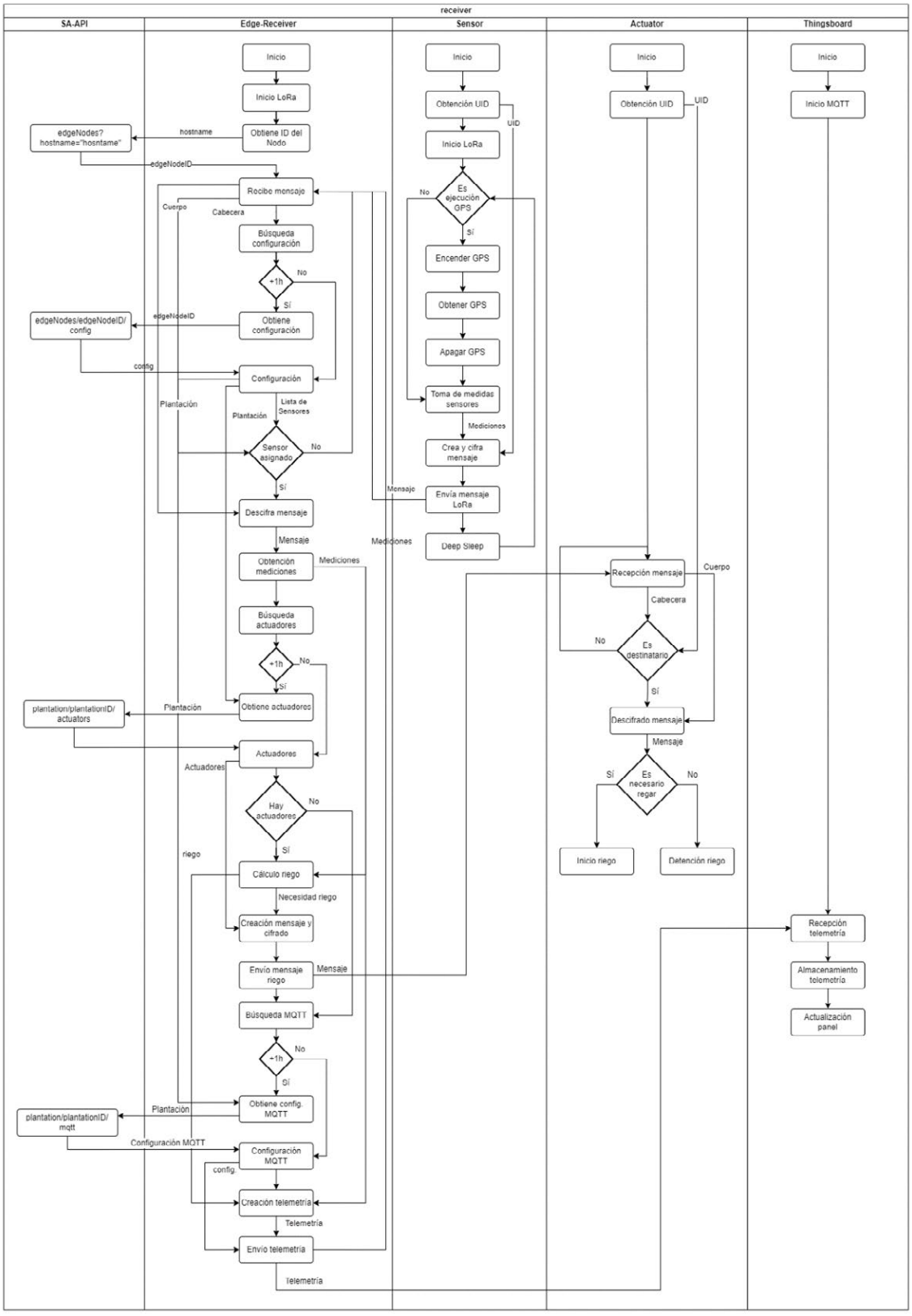



*Diagrama 9 Flujo de la aplicación receiver*

La función de este software es: recibir los datos de los sensores utilizando LoRa, tratar esa información, decidir si debe regar y enviar todas estas mediciones al panel utilizando para ello MQTT. Este software está diseñado para ejecutarse en un *pod* en un nodo Kubernetes.

Para esto se han creado dos imágenes: “sa-python-linux.arm64” y “receiver”, la primera se trata de una imagen base creada para reducir el tiempo de construcción de la segunda.

Como puede verse en el Código 1, esta imagen utiliza una versión de la imagen de Python basada en Alpine ya preparada para la arquitectura arm64v8. Esto es un requisito para el correcto funcionamiento de esta en una Raspberry Pi u otro computador que utilice dicha arquitectura. Instala posteriormente todos los programas en Alpine que el programa receiver necesitará, tras esto instala los requerimientos de Python que receiver también precisará. Es decir, se crea una imagen estática que servirá para basar en ella todo el desarrollo posterior de receiver ya que estos requisitos no variarán. Con este método se ahorra tiempo cada vez que se hace un *build* de la mencionada imagen receiver.

```
1. FROM arm64v8/python:3.9-alpine3.16
2.
3. WORKDIR /usr/bin/app
4.
5. RUN apk add—update—no-cache—virtual .tmp-build-deps \
6.     gcc libc-dev linux-headers postgresql-dev i2c-tools g++\
7.     && apk add libffi-dev
8. COPY requirements.txt ./
9. # RUN pip3 install—no-cache-dir—upgrade pip
10. RUN pip3 install—no-cache-dir -r requirements.txt
```

*Código 1 sa-python-linux.arm64 Dockerfile*

Tal y como se ve en el Código 2, esta imagen se basa en la presentada anteriormente y lo único que hace es copiar todos los archivos necesarios y ejecutar el código utilizando para ello Python con la opción `-u` que permite al desarrollador ver la salida del programa en los logs de la imagen.

```
1. FROM sanoderegistry.azurecr.io/sa-python-linux.arm64:2
2.
3. WORKDIR /usr/bin/app
4.
5. COPY app app
6.
7. WORKDIR /usr/bin/app/app/
8. CMD [”python”,”-u”, “receiver.py”]
```

*Código 2 Receiver Dockerfile*

Una vez el programa inicie su ejecución, el *pod* tomará su *hostname* como identificador único y con esa información solicitará su id a la API (`https://sa-api.azurewebsites.net/api/edgeNodes?hostname=hostname`). Cuando se reciba un mensaje, el nodo (con el id obtenido anteriormente) solicitará su configuración a la API (`https://sa-api.azurewebsites.net/api/edgeNodes/{edgeNodeID}/config`). Solo se llamará a la API una vez cada hora cuando se reciba un mensaje, esto permite ahorrar tráfico de red y no demorar mucho las posibles actualizaciones de la base de datos. Es decir, con la recepción del mensaje se comprueba la configuración almacenada y, si esta no existe o ha pasado más de una hora de la anterior llamada, se vuelve a llamar a la API y se actualizan los datos. Esto se hace para todas las llamadas a los distintos métodos de la API que veremos a continuación.

Una vez el programa cuente con su configuración, podrá comprobar si el mensaje que acaba de recibir está dirigido a él porque es un sensor relacionado con una plantación que le corresponde a este nodo. Para ello, comprueba la cabecera del mensaje (ver Diagrama 4) y, si está asignado a este nodo, descarga su información. Si no está asignado, se ignora. Una vez se ha comprobado si ese mensaje estaba dirigido a este nodo, este se descifra

utilizando para ello el algoritmo XXTEA. Con los datos en claro, se procesa la información obtenida y se almacena. Tras esto, se busca si la plantación a la que pertenece el sensor del que se ha recibido el mensaje tiene asignado algún actuador, en caso afirmativo se realiza la descarga de su información utilizando para ello la API (`https://sa-api.azurewebsites.net/api/plantation/{plantationID}/actuators`), de la misma forma que en el caso de la configuración: solo una vez cada hora. En caso afirmativo se calcula con la función "watering" la necesidad de riego, en caso contrario, se pasa al siguiente paso. Posteriormente, se crea un mensaje para cada actuador cuya cabecera será la dirección MAC de cada uno de ellos, en su cuerpo se incluirá la necesidad de riego de dicho actuador. Finalmente, y para terminar el ciclo de ejecución, se envían las mediciones mediante MQTT al panel, la información, al igual que en los casos anteriores, se solicita a la API (`https://sa-api.azurewebsites.net/api/plantation/{mqttConfigID}/mqtt`) una vez cada hora.

Los parámetros de riego se encuentran almacenados en la clase "edgeProcessor" y lo calcula el método "isWateringNecessary". Ahí pueden ser modificados para adaptarse a los nuevos requisitos del agricultor o la plantación.

### 5.3.6. Sensor

El software incorporado en los sensores cumple la función de tomar los datos de los sensores instalados y enviarlos utilizando LoRa al nodo. Su funcionamiento debe ser sencillo y eficaz, para que, de esta forma, se pueda confiar en ellos. El algoritmo que se ha implementado es muy sencillo (puede verse en el Diagrama 9).

En el método `setup()` se realizan todas las funciones que desarrolla este microcontrolador:

- Se inicializan las comunicaciones Serial: se utiliza para el desarrollo y depuración del código.
- Se configuran los pines de entrada y salida: su uso se puede ver en el Esquema 3.
- Se obtiene la dirección MAC: se utilizará como identificador único de la unidad.
- Se establece la clave que se utilizará para el cifrado de los mensajes.
- Se llaman a las distintas funciones que inician: AXP192, LoRa, GPS y los sensores.
- Si estamos en el momento de activación GPS (dos veces al día, cada 12h y siguiendo la constante `PERIOD_ACTIVATION_GPS`), se inicia el GPS y se busca la señal hasta 3 veces si no se localiza la ubicación de la unidad. Tras esto, se apaga el chip GPS.
- Se toman las mediciones:
    - Se inicia el sensor de humedad y temperatura y se toma una medida.
    - Se toma la medición de tensión del sensor de humedad del suelo.
    - Se suministra energía al sensor de gotas de agua y toma la medida booleana.
- Se crea un mensaje cifrado (utilizando XXTEA) cuya cabecera es la dirección MAC y el cuerpo las mediciones (ver Diagrama 4).
- Se incrementa el contador de arranques.
- Se apaga el chip LoRa.
- El ESP32 entra en modo *deep sleep* que durará 10 minutos.

En la Ilustración 7 puede verse la estimación que hace el compilador del consumo de recursos que provoca este código. Como puede verse, el código está optimizado y el rendimiento es correcto, dejando margen para futuras funcionalidades.

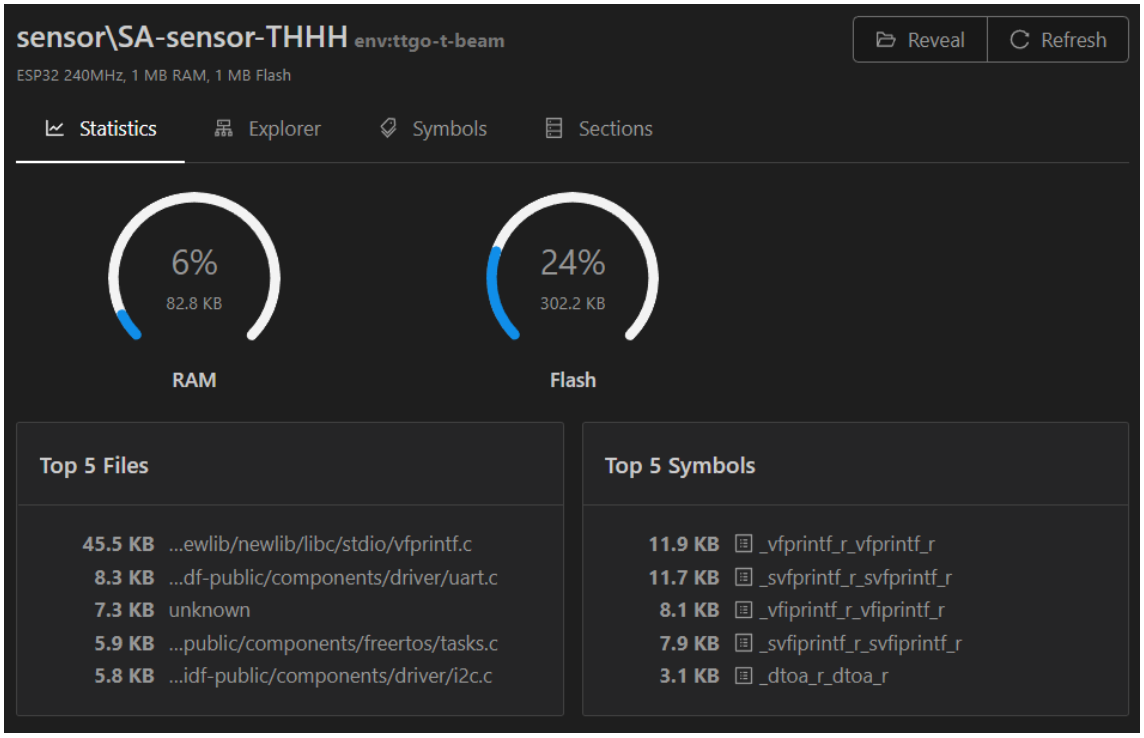


*Ilustración 7 Resultado de la inspección del Sensor*

El consumo energético es una prioridad en el desarrollo de los sensores ya que estos tienen que funcionar por sí mismos situados en las plantaciones sin el aporte externo de energía. Para ello se ha optimizado su consumo con tres métodos: haciendo que el ESP32 entre en modo *deep sleep* [49] y despertando solo para realizar su función; apagando los periféricos cuando estos no se utilizan y utilizando el GPS lo mínimo posible que son 2 veces al día. De esta forma el consumo se reduce drásticamente y la batería puede alcanzar entre 15 y 20 días de uso, esta cifra varía en gran medida por la entrada en funcionamiento del GPS, que, si requieren los 3 intentos de conexión, consume mucha más batería.

El apagado del sensor que mide la presencia de gotas en las hojas cobra una gran importancia dado que su consumo es elevado y su degradación mucho más alta cuando está alimentado. Por estos motivos, solo es alimentado utilizando un GPIO cuando es necesario: medio segundo antes de realizar la medición. De esta forma, se reducen en gran medida los gastos excesivos de alimentación y se minoriza su degradación. Dado que es un único sensor, puede ser alimentado correctamente por un pin de salida del ESP32 sin que exista ningún problema causado por un consumo excesivo que el microcontrolador no pueda procesar.

### 5.3.7. Actuator

El software incorporado en los actuadores cumple la función de recibir las peticiones que realiza el nodo de riego o no riego y ordenar regar o dejar de regar en consecuencia. El algoritmo que se ha implementado es aún más sencillo que el del anterior (puede verse en el Diagrama 9).

El funcionamiento básico es muy simple, el actuador recibe mensajes vía LoRa compuestos por una cabecera y un cuerpo (ver Diagrama 5). Si la cabecera coincide con su dirección MAC, descifra el mensaje utilizando para ello el algoritmo XXTEA. Este mensaje se da en formato JSON y se extrae la clave "watering". Si esta es 1, se inicia el riego y si es 0 se detiene. Por razones de seguridad y medioambientales el riego (si no se reciben más mensajes) se detendrá en una hora, si se siguen recibiendo mensajes, estos controlarán el comportamiento del riego.

En la Ilustración 8 puede verse la estimación que hace el compilador del consumo de recursos que provoca este código. Queda patente que el rendimiento es correcto, pero, a diferencia del punto anterior, el porcentaje de uso de recursos es superior. Esto se debe a que el procesador ESP32 instalado en esta placa es de inferiores prestaciones que el presente en la del Sensor. Esto no influye en el rendimiento pues los porcentajes siguen siendo bajos.

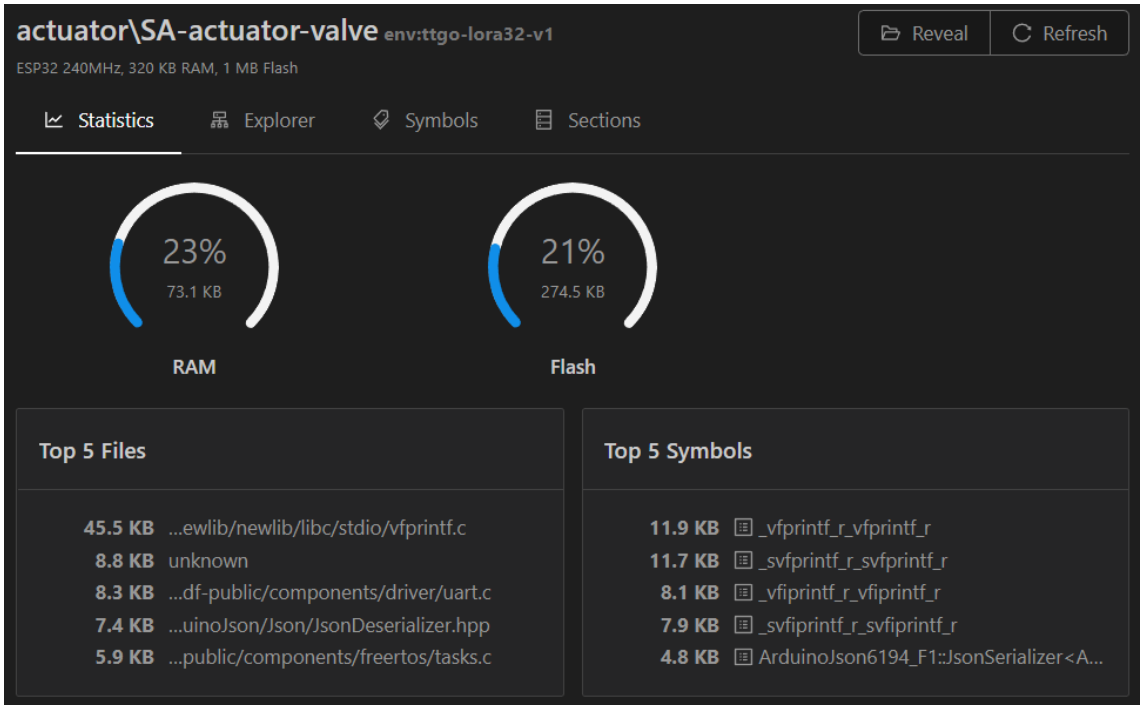


*Ilustración 8 Resultado de la inspección del Actuador*

Como principal diferencia con los sensores, la economía energética no es tan importante, la prioridad es recibir paquetes y actuar. Por esto, no se han implementado ningún mecanismo de ahorro energético ya que esto reduciría la eficacia de funcionamiento del conjunto. Como la alimentación que tiene la placa que, ahora sí, depende del exterior como puede verse en el capítulo 5.1.2.1, el suministro está garantizado. En este caso prima el tiempo de funcionamiento sobre la economía energética.

### 5.3.8. Visualización

Para la visualización de los datos se ha implementado un panel ThingsBoard [50]. Este permite la gestión integral de la visualización de los datos, de clientes y de los sensores. Se ha favorecido una implementación sencilla, que permita ver los datos de forma fácil y desde cualquier lugar. ThingsBoard es la plataforma ideal porque reúne todas las herramientas necesarias en un paquete fácil de instalar y configurar.

Su instalación se ha basado en el sistema operativo oficial Raspberry Pi OS basado en Debian Bullseye. Para su instalación se ha seguido la guía oficial [51] ya que es la solución óptima para este escenario. Para ello es necesario instalar Java ("openjdk 11"), descargar el paquete oficial de instalación y, tras esto, instalar postgresql. Hay que destacar que es necesario limitar la memoria RAM que utilizará ThingsBoard a 256 MB por la reducida memoria de la que dispone la Raspberry Pi 3 (1 GB). Una vez se hayan realizado estos pasos, es necesario crear una base de datos nueva llamada "thingsboard" y editar con toda la configuración relativa a la base de datos en el fichero que utiliza ThingsBoard. Dado que las necesidades de esta prueba son pocas, se ha decidido utilizar el servicio de colas proporcionado por el propio programa en vez de otros más sofisticados (pero complejos de desplegar), como pueden ser AWS SQS, Azure Service Bus o Kafka. Tras esto, se puede instalar ThingsBoard con el script oficial, que instala todo lo necesario incluyendo una demo. Finalmente, se inicia el servicio.

El rendimiento obtenido es justo, pero suficiente para los dos nodos que componen este sistema. Los tiempos de carga son algo lentos, pero las interfaces se muestran correctamente y no se producen fallos inesperados.

Un paso adicional que se ha realizado es establecer un DNS (*Domain Name System*) dinámico para poder asignar un nombre de dominio estático a la dirección IP dinámica de la Raspberry Pi. Al no contar con un servicio de IP estática contratada, no se puede contar con que la IP de este elemento sea constante a lo largo del tiempo. Por esto, se ha seleccionado la herramienta FreeDNS [52], que permite, de forma gratuita y libre, seleccionar un subdominio web de tipo A (redirige a una única IP). La Raspberry Pi ejecuta una tarea utilizando cron cada 15 minutos que actualiza la IP en el servicio ofrecido por FreeDNS, por lo que siempre el dominio seleccionado referencia a la IP pública de la red donde la Raspberry Pi está conectada. Para permitir que los nodos se conecten al servidor MQTT implementado, se ha tenido que abrir un puerto en el rúter y dirigirlo al puerto 1883 correspondiente al servicio MQTT de la Raspberry Pi. Gracias a esto, esta Raspberry Pi se puede iniciar en cualquier localización y, automáticamente, volverá a ofrecer el servicio MQTT. Esto mismo se podría haber hecho con una Red Privada Virtual (VPN), pero añade una complejidad en el despliegue innecesaria por la seguridad que ya implementa MQTT para el acceso y autenticación.

Tras toda la configuración anterior, la plataforma ThingsBoard está lista para ser utilizada, para ello es necesario configurar el sistema. Primero se crea un cliente con un usuario para simular la vista que tendrá el usuario final del sistema. Tras esto, es necesario crear un perfil sobre el que se basarán los sensores a implementar. En este perfil se introduce el tema MQTT a utilizar (`v1/devices/me/telemetry`) y el formato de la carga útil de los mensajes (JSON). Se crean dos sensores y se les identifica, posteriormente sus parámetros para la conexión MQTT son almacenados en la tabla "mqttConnection" de la base de datos. Cada sensor tendrá un único identificador MQTT para realizar la conexión que se compone de un identificador único, nombre de usuario y contraseña ya que se utiliza la autenticación básica de MQTT por la facilidad de gestión que supone.

Una vez creados los sensores se puede establecer el panel que verá el usuario, para ello se pueden utilizar *widgets* que ya ofrece el propio software y personalizarlos para lograr la vista que busquemos o crear widgets nuevos. En el caso del panel desplegado, que se muestra en la Ilustración 9, se han utilizado varios *widgets* ya generados y se han personalizado para cumplir con los requisitos. En ellos se muestra de izquierda a derecha y de arriba a abajo:

- Nombre del dispositivo
- Estado de actividad
- Temperatura de forma visual
- Voltaje de la batería
- Ubicación del sensor
- Gráfica con el histórico de la batería
- Gráfica con temperatura y humedad ambiente
- Gráfica con el histórico de cantidad de agua en el suelo
- Cantidad de agua en el suelo actualmente

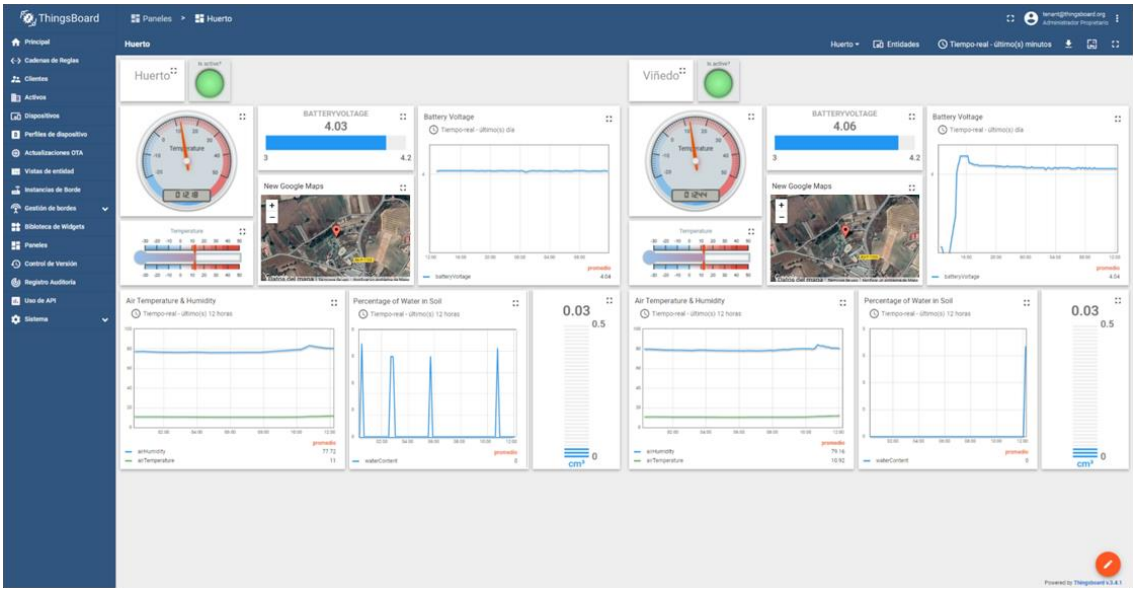


*Ilustración 9 Panel de visualización de datos*

Con estos datos básicos el agricultor puede ver cómo se encuentra la plantación en estos momentos y visualizar los datos históricos almacenados en la propia plataforma.

Por lo tanto, la solución de visualización de cara al cliente se compone de un acceso personalizado y un panel a través del que puede acceder a todos los dispositivos que tiene asignado.

## 5.4. Tecnologías y Herramientas de Soporte

### 5.4.1. Repositorios

Siguiendo la cultura GitOps, el uso de repositorios es fundamental. Se ha elegido GitHub [53] como repositorio por su facilidad de uso, familiaridad y su buena integración con el resto de las herramientas seleccionadas. Otros repositorios habituales como GitLab también suponían una buena elección, pero se han descartado por su inferior interacción con el resto de las herramientas empleadas en este proyecto.

Los repositorios principales de este proyecto son:

- SA-DevOps: donde se almacenan los *manifest* de Kubernetes empleados y el *pipeline* de Azure DevOps. Enlace: juanked/SA-DevOps (github.com).

- SA-edgeNodes: donde se almacena todo el código empleado por la capa *edge* y la información relevante para su despliegue. En él están los dos módulos básicos que generan, respectivamente, la imagen básica sobre la que se trabaja y la imagen receiver que sirve de núcleo al nodo edge. Enlace: juanked/SA-edgeNodes (github.com).
- SA-edgeDevices: en él se almacena todo el código relativo a los dispositivos de la capa física. Por lo tanto, cuenta con el código de sensores y actuadores, pero además los de las demás herramientas empleadas como el identificador MAC, calibrador de sensores y el procesador de la anterior calibración. Enlace: juanked/SA-edgeDevices (github.com).
- SA-data: en este repositorio se almacena todo lo relacionado con la capa cloud o de datos. En él están los ficheros con los que replicar la creación tanto de la base de datos general como de los procedimientos generados. En este repositorio también está almacenado el código de la API que accede a la base de datos mencionada. Enlace: juanked/SA-data (github.com).
- SA-dashboard: en este repositorio se han almacenado todos los parámetros de configuración del panel ThingsBoard una vez han sido desarrollados. Es el control de versiones de dicho panel. Enlace: juanked/SA-dashboard (github.com).

### 5.4.2. CI

Para la integración continua se ha utilizado la herramienta Azure DevOps [54] que se ejecuta en la propia nube de Microsoft, Azure. Es una herramienta muy completa que, frente a alternativas como Jenkins [55], no requiere de una instalación por parte del usuario y cuenta con ajustes y herramientas que hacen su integración con Azure mucho más fiable y sencilla. Es importante que la ejecución del ciclo CI se produzca siempre acorde a los requisitos impuestos, por ello Azure DevOps, en su sección "*pipelines*" permite realizar procesos automatizados de integración continua.

Previamente es importante a contar cómo se ha hecho, recalcar la necesidad de que las imágenes generadas tienen que estar preparadas para la arquitectura de los nodos (Linux/ARM64), por eso ha sido la mayor prioridad y necesidad de este flujo de integración continua. Es necesario recalcar que la arquitectura de este procedimiento CI está centrada en obtener un resultado satisfactorio sin requerir otro agente en otra arquitectura que aumentase la dificultad del flujo ya que, hoy en día, Azure no ofrece ningún agente nativo ARM64.

Como se puede ver en el Código 3 SmartAgriculture-CI.yaml, la tarea cuenta con varias tareas o *task* que, en su conjunto, forman el proceso de integración continua y preparan todo para la siguiente etapa. Son las siguientes:

- Install Qemu: esta tarea instala Qemu [56] para poder compilar imágenes Docker siguiendo la arquitectura ARM64, diferente de la del agente que ejecuta el *pipeline* (x64), lo cual es necesario para que el código pueda ejecutarse en Raspberry Pis. Se trata de una tarea que ejecuta un mandato en una terminal CMD.
- Build: esta tarea ejecuta un comando Docker build a partir del Dockerfile contenido en el repositorio SA-EdgeNodes donde se detalla cómo será la imagen que se ejecutará en los nodos de la capa *edge*. En la tarea se detalla que el contenedor debe ser compilado para la arquitectura ARM64.

- Push: esta tarea ejecuta un comando Docker push y envía esta imagen al Container Registry donde será almacenada para su posterior uso.
- Update Kubernetes manifest: esta tarea actualiza el manifiesto Kubernetes que luego utilizará el flujo de CD incluyendo la modificación en la imagen que se acaba de crear. Un ejemplo se puede encontrar en Código 4 deployment.yaml.
- Copy files y publish: esta tarea copia y publica las imágenes para que pueda ser descargada toda la información accediendo al *pipeline*.

Este *pipeline* o flujo de integración continua prepara la imagen Docker que, a continuación, se despliega mediante un pipeline o flujo de despliegue continuo, tal y como se describe a continuación.

### 5.4.3. CD

Como herramienta para el despliegue continuo (*continuous deployment* o CD), se ha utilizado Argo CD [57]. Esta herramienta se instala dentro de su propio *namespace* dentro del clúster Microk8s. Su instalación es muy simple utilizando la guía oficial para ello [58]. Argo CD toma ficheros de configuración y despliegue o *manifests*. En estos ficheros se detalla cómo debe producirse el despliegue del *pod* y se encuentran en el repositorio "SA-DevOps".

Tras la instalación, es necesario configurar el servicio para hacer el despliegue continuo según nuestras necesidades. Para ello se le debe indicar cuál es el repositorio y directorio al cual tiene que "vigilar". Del directorio correspondiente del repositorio anterior, tomará los ficheros de despliegue y, con ellos, ejecutará los despliegues. Argo CD automatiza el proceso de despliegue continuo totalmente y permite, junto con la herramienta anterior, el desarrollo DevOps de la aplicación. También, ofrece una interfaz gráfica web muy útil y funcional, desde la que se puede consultar el estado del despliegue. Argo CD se encarga de, cuando haya una actualización en la base de código, realizar el despliegue y sustituir el *pod* por uno nuevo, no quedándose el servicio sin cubrir en ningún momento. Es decir, no se desactiva el *pod* actual hasta que el nuevo no está lanzado completamente. Tras esto, se desactiva el anterior, pero queda listo por si fuese necesario. Puede suceder que el nuevo *pod*, por su configuración u otra causa, presente un fallo que impida su correcto funcionamiento. En este caso Argo CD recuperaría el anterior para, así, evitar el fallo del sistema.

Para poder visualizar la aplicación web y dado que está alojado en el clúster Microk8s que está alojado en una máquina virtual que está corriendo en la nube, es necesario habilitar la redirección de puertos de dicho clúster con el siguiente comando:

```
1. kubectl port-forward svc/argocd-server -n argocd 8090:443 --address='0.0.0.0'
```

Con este comando, se configura la redirección de puertos del clúster Kubernetes. El servicio de Argo CD utiliza el puerto HTTPS (443) para su conexión. Por lo tanto, se establece una redirección desde el puerto exterior 8090, que ha de ser abierto y redirigido desde la tarjeta de red de la máquina virtual para permitir la entrada y salida desde este puerto de la máquina virtual.

Para que Argo CD pueda acceder al registro de imágenes, es necesario que utilice la clave generada en el paso 5.3.4. Para facilitar esto, es necesario crear un secreto en el clúster con los datos señalados [59]. Para ello se ha empleado este comando:

```
1. kubectl create secret docker-registry azsecret \
2.     --namespace kubeedge \
3.     --docker-server=sanoderegistry.azurecr.io \
4.     --docker-username=SAnodeRegistry \
5.     --docker-password=XXXXXXX[1]
```

Una vez esta configuración esté realizada, se podrá iniciar el flujo CD y comenzar los despliegues de la imagen "edge-receiver".

### 5.4.4. GitHub Actions CI/CD

Para la integración y despliegue de la API se ha empleado la herramienta GitHub Actions [37] por las facilidades que Microsoft pone al respecto y el poder probar otra herramienta que, al igual que Azure DevOps, es gratuita. De esta forma, se ha logrado explorar otras alternativas que pueden ser útiles para despliegues como este. En el caso anterior no se ha realizado así por la facilidad que pone Argo CD al despliegue dentro de un Kubernetes y la posibilidad de configurar de una mejor forma la arquitectura de las imágenes a desplegar (ARM64).

En el Código 5 se puede ver el flujo CI/CD de esta aplicación. Este flujo ha sido autogenerado por Azure desde la sección de la Function App: centro de despliegue. Esta facilidad y predeterminación son el mayor valedor de este sistema, es extremadamente sencillo automatizar el despliegue de este tipo de aplicaciones. Es una alternativa menos potente pero interesante para este tipo de proyectos más sencillos frente al anteriormente mencionado Azure DevOps.

### 5.4.5. Lenguajes de programación

En este proyecto se han utilizado una gran variedad de lenguajes de programación para intentar aprovechar las ventajas de estos o por obligación impuesta por el fabricante. Son los siguientes:

- Arduino [60]: se ha empleado para la programación de los sensores y actuadores y de las herramientas para su configuración. Es un lenguaje de alto nivel basado en C++ y está muy indicado para programar microcontroladores como lo son los ESP32. Se ha utilizado por su buena disponibilidad de librerías y tutoriales.
- C# [61]: se ha empleado en la programación de la API. Es un lenguaje de alto nivel y orientado a objetos. Está soportado por Microsoft y está muy bien optimizado para sus servicios. Por eso se ha utilizado en este proyecto para la API que es una Function App y llama a la base de datos SQL Server.
- Python [62]: se ha empleado en el módulo edge-receiver y en la herramienta "dataProcessor". Es un lenguaje de alto nivel, interpretado y de fácil uso que ha sido utilizado por la clara orientación que tiene Raspberry Pi hacia este lenguaje y sus derivados (CircuitPython y MicroPython).
- TSQL [34]: se ha empleado para la configuración de la base de datos y en las llamadas de la API. Es un lenguaje de programación diseñado para gestionar y manipular bases de datos de Microsoft SQL Server.

[1] En este campo iría la clave generada anteriormente mencionada. Se ha sustituido por privacidad.

Como puede verse, la variedad de lenguajes empleados es muy grande, esto ha supuesto que la metodología GitOps sea de gran ayuda para la gestión de estos múltiples entornos y repositorios.

### 5.4.6. Programas utilizados durante el desarrollo

Para realizar este proyecto se han utilizado una gran cantidad de programas cada uno de ellos orientado a un uso o varios usos.

#### 5.4.6.1. Capa Física

La programación ha utilizado PlatformIO [63], que es una plataforma de desarrollo, compilación, depuración y prueba de código gratuita. Está enfocada a microcontroladores de una gran variedad de plataformas. Ha sido instalado en forma de extensión de Visual Studio Code [64] para mayor comodidad.

El diseño electrónico ha sido realizado utilizando CircuitMaker [65], es una plataforma de diseño electrónico y de simulación gratuita. Ha sido utilizada para el diseño de todos los circuitos, para los que se ha creado componentes personalizados.

El diseño de los componentes físicos se ha realizado con Autodesk Fusion 360 [66], es una plataforma de diseño y fabricación de productos y piezas mecánicas. Con ella se han modelado todas las piezas que posteriormente han sido procesadas utilizando Ultimaker Cura [67]. Esta se trata de un software que se emplea para preparar modelos 3D para, posteriormente, ser imprimidos por una impresora 3D, convirtiendo figuras sólidas en instrucciones G-CODE que la impresora 3D pueda comprender. Con estos dos programas se ha realizado todo el flujo de diseño y prueba de los componentes impresos en 3D.

#### 5.4.6.2. Capa Edge

La programación se ha realizado utilizando el editor de código gratuito Visual Studio Code (VS Code), que proporciona herramientas de edición depuración y desarrollo para la práctica totalidad de los lenguajes de programación gracias a la gran cantidad de extensiones cuyos creadores ponen a la disposición del público. Ofrece varias herramientas para programar en Python que han resultado de gran utilidad. Como añadido, permite la conexión remota a otros *hosts* con lo que ha permitido desarrollar directamente en el hardware del nodo (ver 5.1.3) y de esta forma desarrollar directamente con herramientas compatibles y poder realizar las pruebas de forma rápida y sencilla.

El compilado de la imagen base (ver 5.3.5) se ha realizado utilizando Docker Desktop [68]. Esta es una aplicación de escritorio que permite a los desarrolladores utilizar contenedores Docker, además de construir, probar y subir a un repositorio de imágenes dichas imágenes creadas. Ha sido de gran utilidad para probar en local los cambios realizados en las imágenes.

#### 5.4.6.3. Capa Cloud

Para el desarrollo de la base de datos se ha empleado el programa SQL Server Management Studio (SSMS) que es una herramienta de gestión y desarrollo de bases de datos SQL Server y Azure SQL Database. Además, incorpora herramientas muy útiles que también han sido utilizadas como la del diseño de procedimientos o la creación automática de diagramas. Se ha empleado para el diseño y depuración de la base de datos.

El desarrollo de la API se ha realizado utilizando Visual Studio, es el IDE referencia de Microsoft y está especialmente enfocado al desarrollo de software basado en C# y otras herramientas de Microsoft. Para el desarrollo de una API que se ejecuta en una Function App es la mejor opción pues permite un desarrollo más fluido y un entorno de pruebas de mucha calidad.

## 5.5. Medios bibliográficos

Para la realización de este trabajo se han utilizado diversos buscadores de referencias de gran utilidad. Los principales han sido:

- Ingenio UPM [69]
- ScienceDirect [70]
- Google Académico [71]
- ResearchGate [72]
- IEEE Xplore [73]

Con estos buscadores se han obtenido todos los *papers* que han sido utilizados en este proyecto.

Para la gestión bibliográfica del proyecto se ha utilizado Mendeley Cite [74] y Mendeley Reference Manager [75]. Son unas herramientas de creación y gestión de citas y referencias bibliográficas. Se han clasificado las fuentes dentro de su propia carpeta para este proyecto y, posteriormente, se ha utilizado la extensión Mendeley Cite.

Para las citas en este documento se ha empleado el estilo de citación IEEE (*Institute of Electrical and Electronics Engineers*) [76], que es el más habitual en el campo de la ingeniería y la tecnología. Este incluye reglas específicas que se deben seguir a la hora de citar distintos tipos de documentos (revistas, informes, libros, etc.). Es la opción más coherente debido a la temática y objetivo de este proyecto.

# 6. Integración y validación

Para la comprobación de los sistemas y su correcta integración se han desarrollado multitud de pruebas. A continuación, se enunciarán alguna de ellas y cuáles eran sus objetivos.

## 6.1. Toma de mediciones

Con anterioridad a la integración del sensor con el resto de los sistemas, se comprobaron todas las mediciones utilizando para ello la salida serial que ofrece el microcontrolador.

El primero fue el GPS que incorpora la placa. Para su comprobación se programó un código que simplemente imprimiera las coordenadas del dispositivo. Se pudo comprobar con esto que se conseguía una precisión de un radio de 5 m, no muy alta, pero suficiente para esta aplicación. Se podría mejorar con la instalación de una antena cerámica de mejor calidad, pero no merece la pena para la finalidad del proyecto.

El segundo fue el de humedad y temperatura ambiente, para ello se utilizó otro sensor (posiblemente de inferior calidad) como herramienta de comprobación. Las mediciones se recogían con una exactitud del ± 0,5 ºC y ± 5 % HR, pero entre los dos sensores se encuentra una precisión de ± 0,2 ºC y ± 2 % lo cual demuestra que, al menos, son coherentes entre ellos. El no tener disponibles otros sensores calibrados ni las condiciones de laboratorio necesarias para calibrar estos sensores ha hecho imposible que se mejoren estas cifras. Se considera que los sensores elegidos tienen una precisión alta situándose teóricamente en ± 0,1 ºC y ± 2 % HR. Para el propósito de este trabajo se considera que son suficientes.

El tercero fue el sensor de humedad del suelo, cuya comprobación y calibrado se puede encontrar en el Anexo 10.6. Tras crear el modelo para cada sensor, estos se probaron con una precisión muy positiva a otras cantidades de agua, con una desviación de tan solo entre un 4 y un 6 %.

Finalmente, se midió y calibró el funcionamiento del sensor de gotas de agua utilizando para ello un sencillo pulverizador de agua intentando que la cantidad de agua que cayese en cada sensor fuese similar. Su funcionamiento es correcto, aunque por la calidad del calibrado y el propio funcionamiento del dispositivo puede variar su medición.

## 6.2. Envío de datos

Primero se probó el envío de datos de un sensor a otro que actuaba de receptor. Para ello se implementaron dos clases sencillas: una que enviaba los datos y otra que los recibía y mostraba por la salida serie. Tras esto se probó la creación de los mensajes utilizados finalmente (ver 5.2.4), para ello se probó la obtención de las direcciones MAC de cada uno de los dispositivos, tanto Sensores como Receptores.

Tras conseguir una comunicación de calidad, se probó esto mismo, pero utilizando como receptor el sombrero LoRa para la Raspberry Pi. Con esto se vio la dificultad incorporada al integrar dos sistemas diferentes. Tras solucionar los problemas de compatibilidad se probó el envío de mensajes desde el nodo al actuador y también se hizo necesario solucionar los problemas que surgieron. Todos estos problemas se han producido por la

inclusión de bytes de información antes y después del mensaje debido a las posibles distintas interpretaciones del protocolo LoRa causadas por las diferentes librerías y arquitecturas empleadas durante el desarrollo. Para prevenir estos problemas en el actuador se tuvo que implementar un carácter de inicio de mensaje (ver 5.2.4).

## 6.3. Procesamiento de los datos

Una vez el nodo recibe los datos del sensor, este debe procesarlos y tomar una decisión sobre si regar o no. Esto se probó introduciendo datos artificiales para ver la respuesta. Posteriormente, también se comprobó el envío de los datos al panel de visualización. Esto se realiza en formato JSON y la dirección MQTT correspondiente a cada dispositivo la obtiene de la anterior mencionada API.

## 6.4. Accionamiento de la válvula

Primero se comprobó el accionamiento de la válvula con un código simple que la abría y cerraba a intervalos alternos. La intención era comprobar si la válvula funcionaba correctamente con el circuito electrónico implementado y si su apertura era total. Tras comprobar que esto era así, se probó el accionamiento remoto mediante el envío de mensajes personalizados vía LoRa desde el nodo.

Una vez esto funciona, se prueba la integración y recepción de los datos generados por el nodo.

## 6.5. Cifrado de datos

Para mayor seguridad el cuerpo del mensaje se cifra utilizando el algoritmo XXTEA. Las pruebas que se han desarrollado se componen por ensayos de cada componente individual y, posteriormente, de cifrado y descifrado entre los distintos componentes. Cuando el cifrado y el descifrado se realizan en el mismo componente o en otro con su misma arquitectura y librerías, no existe ningún tipo de problema. La integración entre librerías sí que ha causado un gran número de problemas que han tenido que ser solucionados posteriormente. La librería para Python es la que más problemas ha supuesto dada la peculiaridad del tratamiento de tipos en este lenguaje. Todos han sido solucionados con forzados y adaptaciones al tipo de variable, con ello, se ha conseguido la total integración de ambos sistemas y su perfecta interoperabilidad.

## 6.6. Base de Datos y API

La base de datos se probó utilizando para ello consultas TSQL desde el cliente SSMS de Microsoft. Mediante este tipo de consultas también se probaron los procedimientos a los que hace referencia la API para comprobar, desde la propia base de datos su funcionamiento.

La API se probó al inicio haciendo consultas manuales utilizando para ello Thunder, una extensión para VS Code similar a Postman. Con esta extensión se realizaron consultas GET y analizando las respuestas JSON aportadas por la API. Esto al principio se hizo en local y luego, una vez estuvo desplegada, accediendo a la dirección de la Function App. De esta forma se comprobó la integración entre la API y la base de datos.

Tras comprobar el correcto funcionamiento de la API, se probó la integración con el programa receiver que, primero, llama a la API, segundo procesa el mensaje de que recibe y tercero lo almacena. Todas estas pruebas se llevaron a cabo utilizando para ello datos reales de configuración y de prueba generados a tal efecto (para simular más dispositivos de los que realmente hay). Tras esto se probaron las cachés implementadas para comprobar su correcto funcionamiento y la ausencia de llamadas innecesarias a la API.

## 6.7. Ahorro de energía

Con los Sensores ya funcionando, se midió la autonomía del sensor sin realizar ningún tipo de economización y esta se estimó en unos 4 días. Al considerarse esto insuficiente, se probó a reducir el consumo implementando ciclos de *deep sleep* y se probó su funcionamiento. Con esta nueva implementación la autonomía se incrementó en gran medida y con la reducción de las mediciones GPS a 2 por jornada, la autonomía aumentó hasta los 10 días. Se realizaron test de apagado de periféricos resultando en un error la prueba de apagado del circuito dedicado a la pantalla. Tras consultar los esquemas eléctricos del fabricante, se vio que este circuito en concreto es el que alimenta al ESP32 y, por lo tanto, no puede apagarse. Simplemente se han podido apagar los circuitos responsables de alimentar los chips GPS y LoRa. Estos se encienden únicamente cuando van a ser utilizados y, tras esto, se apagan. De esta forma se ha logrado aumentar la batería hasta los 14 días aproximadamente. Por la gran complejidad del circuito y la inclusión de un chip de administración de la energía, la eficiencia energética de este controlador no es muy buena, a lo que es necesario sumar el elevado consumo de GPS (unos 80 mW) y LoRa (unos 20 mW), es difícil estimarlo mejor por la falta de equipamiento dedicado a ello.

Tras esto se comprobó que las nuevas introducciones no causaban fallos en el resto de los dispositivos y que todo seguía funcionando con normalidad.

Seguidamente se realizó la medición del consumo del resto de dispositivos, viendo su mejora realmente difícil y no teniendo una elevada importancia por el suministro empleado en actuadores, nodos y panel.

## 6.8. Conexión con el panel de visualización

Tras la implementación del panel ThingsBoard, se realizaron pruebas de conexión MQTT ya mencionadas anteriormente y de actualización de la IP por el servicio FreeDNS cuando la IP pública ofrecida por el ISP cambiaba. Esto se probó cambiando la red a la que se conecta la Raspberry Pi para comprobar el cambio automático de dicha IP. Esto fue satisfactorio, por lo que la ubicación (e IP) de la Raspberry Pi empleada por el panel no afecta al funcionamiento del sistema.

## 6.9. Visualización de datos en el panel

Tras la creación del panel, se realizaron pruebas de envío de datos artificiales para poder comprobar la representación de datos en los distintos *widgets* que se muestran en el panel. Este resultado fue satisfactorio y tras esto se inició el envío de datos al panel desde los nodos, comprobando que esto también funcionaba correctamente.

## 6.10. Instalación en la plantación

Con la instalación del sistema en la plantación se comprobaron las facilidades que este pone para su instalación. Se comprobó que el sistema solo requería su conexión y ensamblado para estar operativo y no surgió ninguna complicación más allá de la complejidad de la propia instalación del sistema de riego. Su funcionamiento fue tal y como estaba esperado y solo se produjeron unos pocos fallos en las comunicaciones.

# 7. Resultados y discusión

Los resultados del proyecto han sido satisfactorios, se han cumplido la totalidad de los requisitos y la funcionalidad ha demostrado ser completa, aunque hay varios puntos de mejora que se expondrán a continuación.

La totalidad de los requisitos se cumplen, aunque con algunos matices. A continuación, se justificará cada uno de ellos:

- **RF1:** se utiliza el sensor AM2315C que cumple con esta función. Se puede ver en el capítulo 5.1.1.
- **RF2:** se utiliza el sensor Capacitive Soil Moisture Sensor V1.0 que ha sido calibrado para lograr esta medición. Se puede ver en el capítulo 5.1.1.
- **RF3:** se utiliza el sensor YL-83 que ha sido calibrado para detectar la humedad en la hoja. Se puede ver en el capítulo 5.1.1.
- **RF4:** la activación y desactivación del riego es controlada por el nodo *edge* siguiendo las mediciones de los sensores. Como se muestra en el capítulo 5.3.5.
- **RF5:** La comunicación entre la capa física y la capa *edge* se realiza utilizando LoRa, que cumple con ambos requisitos como se ha visto en el capítulo 5.2.1.
- **RF6:** Se ha hecho todo lo posible para que los sensores y actuadores sean estancos, aunque es algo que no se ha podido comprobar en condiciones de homologación. En los capítulos 5.1.1.1 y 5.1.2.1 se puede ver el resultado.
- **RF7:** Los sensores consumen la mínima energía posible, se estima su duración en alrededor de 15 días con una batería utilizada en el 5.1.1.
- **RF8:** Como se ha expuesto en el capítulo 6.10, la instalación ha resultado sencilla y podría realizarla cualquier persona con unos conocimientos técnicos mínimos. Pero se hace necesaria la configuración de esos dispositivos introduciendo todas sus especificaciones en la base de datos antes de su instalación.
- **RF9:** Como se ha narrado en el capítulo 5.3.8, el sistema ofrece la capa de aplicación, en la que un panel ThingsBoard permite al agricultor visualizar mediciones y comprobar el correcto funcionamiento de su plantación.
- **RF10:** El panel de visualización es personalizable para cada cliente con sus sensores y los *widgets* que se estimen necesarios. Los utilizados en este proyecto constituyen un ejemplo, pero puede personalizarse al gusto de cada cliente.
- **RF11:** Tal y como muestra el circuito Esquema 4, la alimentación de todo el circuito de cada actuador es de +12 V DC proveniente de una batería instalada en cada plantación. Esta es la misma fuente empleada para la apertura de las válvulas que controlan dichos actuadores.
- **RF12:** Pese a que en este proyecto únicamente existían una pareja sensor/actuador por cada plantación, la implementación permite que existan más de un sensor y actuador por cada plantación. Por motivos económicos no se ha podido comprobar el máximo número de sensores y actuadores que cada nodo puede soportar concurrentemente sin presentar fallos.
- **RNF1:** Los usuarios únicamente necesitan encender los dispositivos para su conexión. Los nodos se comunicarán automáticamente con el CloudCore al que estén asignados e iniciarán la ejecución de *pods* conforme vaya indicando Argo CD. Los sensores únicamente necesitarán ser encendidos utilizando el botón que tienen con ese fin una vez la batería se encuentre instalada. Los actuadores se iniciarán cuando se conecte la entrada de batería a su correspondiente regleta. El

cliente no tendrá que intervenir en nada más para su inicio. Previamente, el desarrollador deberá dar de alta todos los sistemas en KubeEdge y la base de datos.

- **RNF2:** No se ha podido comprobar el número máximo por cuestiones económicas, aunque no debería ser bajo, con 2 sensores y 2 actuadores funciona correctamente.
- **RNF3:** El sistema empieza a procesar y enviar datos y decisiones tras el encendido del nodo.
- **RNF4:** El sistema incorpora todos los nodos (2) de los que se tiene disponibilidad para este proyecto.
- **RNF5:** El número de nodos es escalable; el panel puede ser instalado en una máquina virtual que así lo sea (la Raspberry Pi 3B actual no es suficiente si aumentase mucho el número de nodos y clientes); el número de sensores y actuadores puede aumentar todo lo que se considere necesario y, tanto la Function App como la base de datos, son escalables por el propio diseño de Azure. La máquina virtual encargada de la gestión del CloudCore y Argo CD puede aumentar su capacidad o replicarse para satisfacer la demanda, aunque esto no está desarrollado por la falta de necesidad para este proyecto. Las herramientas CI y CD no necesitan ningún tipo de escalado.
- **RNF6:** Se ha tratado de cumplir este requisito en su totalidad, pero por razones presupuestarias la capa Cloud depende directamente de Azure. Se podría recrear en cualquier otro proveedor cloud la Function App (todos tienen una alternativa) o se podría dedicar un servidor o entorno Docker a esta tarea con un mínimo esfuerzo. La base de datos se podría traspasar a otro formato como MySQL y alojarse en un servidor dedicado. La cuestión económica ha sido muy fuerte en este aspecto y ha impedido crear una solución que no dependa directamente de Azure. El CI implementado en Azure DevOps podría ser relocalizado en una instancia Jenkins si se estimase necesario, aunque Azure DevOps tiene un nivel gratuito más que suficiente. El repositorio de imágenes se ha instalado en Azure por comodidad, aunque podría utilizarse uno ofrecido por cualquier otro vendedor sin ningún tipo de problemas. En definitiva, depende de un único vendedor (en este caso Microsoft Azure), aunque podría ser relocalizado y adaptado a otro vendedor ya que no se aprovecha de herramientas únicas.
- **RNF7:** Como se ha visto en el flujo CI/CD creado, las imágenes cuyo Dockerfile se almacena en el repositorio "SA-EdgeNodes" es desplegado en los nodos siguiendo el despliegue almacenado en el repositorio "SA-DevOps".
- **RNF8:** Como consecuencia del punto anterior, los usuarios siempre tendrán la última versión del software en sus nodos. El despliegue será automático tras la actualización del despliegue y de la imagen en los repositorios indicados.
- **RNF9:** Este punto no ha podido ser comprobado con más sensores y actuadores de los disponibles (2 de cada), pero sería necesario hacer comprobaciones adicionales con más recursos. Aun así, ante la necesidad, podrían instalarse más nodos y dividir las plantaciones en plantaciones con un menor número de sensores. Para plantaciones de excesivo tamaño o de complicada orografía esto es obligatorio por el límite físico de alcance que tiene la comunicación LoRa.
- **RNF10:** Como ya se muestra en el capítulo 5.2.4, el cuerpo de los mensajes está cifrado utilizando para ello el algoritmo XXTEA que se ha considerado lo suficientemente seguro para esta implantación.
- **RNF11:** Como se muestra en el capítulo 5.2.3, cada dispositivo se identifica con los últimos tres octetos de su dirección MAC y este identificador lo sitúan en la

cabecera del mensaje que envían o se sitúa por el nodo para que el actuador sepa a quién va dirigido.

- **RNF12:** Los sensores actualizan su ubicación GPS cada aproximadamente 12 h.
- **RNF13:** El panel web muestra todos los datos que se han estimado necesarios, aunque estos pueden ser personalizados.
- **RNF14:** El cliente tiene su propia cuenta personal que es creada por un administrador en la plataforma de ThingsBoard utilizada.
- **RNF15:** El administrador puede crear tantos clientes como sea necesario y les podrá asignar a cada uno los sensores y plantaciones que posea.
- **RNF16:** La API puede escalar por el diseño de las Function Apps de Azure y del SQL Server utilizado, aunque para ello sería necesario aumentar el nivel del segundo.
- **RNF17:** Como se puede ver en el Diagrama 9 Flujo de la aplicación receiver, solo se llama a la API si ha pasado más de una hora desde la última llamada.
- **RNF19:** La existencia de una base de datos donde se almacena la configuración de cada grupo sensor, incluyendo la calibración del sensor de agua en el suelo, permiten la calibración para cada plantación. Esta información se almacena en la tabla “sensor” en la columna “linearFit” (ver Diagrama 7).

Los resultados obtenidos en este proyecto han sido, en su mayor parte, muy satisfactorios, aunque dejan margen para la mejora del proyecto en su totalidad. Eso sí, se han cumplido en la medida de lo posible los objetivos marcados para el desarrollo de este proyecto.

Esto no quiere decir que no haya habido fallos en el diseño y en los componentes que no se han mostrado hasta la parte final del desarrollo de este proyecto y cuya solución propuesta se hará en la sección 8.2 Líneas futuras. Resaltan dos fallos de la planificación inicial que son los siguientes:

Un sensor de humedad en el suelo ha fallado tras tan solo un mes de uso y, sencillamente, ha dejado de funcionar. El otro sensor adquirido no ha fallado, por lo que, o bien lo hará antes de lo previsto (su vida útil debería situarse entre los 2 y 3 años) o no lo hará y esto supondría que estamos ante una instalación o una unidad defectuosas. El hecho de que no haya fallado desde el primer momento y haya sido de repente a las dos semanas de estar instalado indica más bien la primera posibilidad, aunque no ha podido ser investigada la causa.

Los MOSFET han fallado, muy probablemente por no estar refrigerados. En un primer momento no estimé esa posibilidad y es un punto que mejorar y cambiar en el futuro. También, por estas mismas causas es necesario revisar si el modelo adquirido es el más indicado para el uso que se le ha dado.

# 8. Conclusiones y trabajo futuro

Este TFM en su conjunto se puede considerar un éxito, tanto por el sistema desarrollado como por los resultados obtenidos y el conocimiento adquirido. Ha sido una tarea ardua y de largo desarrollo, pero, viendo los resultados, ha merecido totalmente la pena.

Desde el punto de vista del sistema y el dominio de este su funcionamiento ha sido lo esperado y su implementación, aunque compleja, ha sido satisfactoria. Las mediciones de los sensores han demostrado ser lo suficientemente precisas como para considerar que son datos positivos y usables. Es otra muestra de la importancia de las tecnologías IoT y la automatización en los entornos más tradicionales que, afortunadamente, reciben con los brazos abiertos todo tipo de mejoras que hagan más fácil este duro trabajo. El aumento de la eficacia del suelo es un asunto de absoluta importancia para los agricultores, los consumidores y el medio ambiente. Con finalidades tan importantes como estas se hace necesaria la existencia de iniciativas que conlleven una mejora en ámbitos tan automatizables y mejorables.

Desde el punto de vista de la ingeniería del software: La automatización del ciclo de vida de las aplicaciones es un aspecto fundamental, y sobre el cual, en los últimos años se ha incidido notablemente. Sin embargo, aún ciertas automatizaciones presentan retos en entornos IoT Edge, más cercanas a lo físico. Como ya ha quedado demostrado, la automatización de despliegues facilita los procesos y reduce los errores, crea un procedimiento auditable que permite una mejora continuada. Estos procedimientos, tradicionalmente, no se han empleado en aplicaciones IoT "industriales" (IIoT) en las que las actualizaciones, si existían, eran manuales. En gran parte por el estado de la técnica y la antigüedad de los equipos, era y sigue siendo la tendencia más habitual. Aun en estos años nos sorprende ver cómo se actualiza un dispositivo como puede ser un termostato vía OTA (*over-the-air*) y, de esta forma, pueden corregir fallos o implementar nuevas funcionalidades en dispositivos, que, de otra forma, se hubiesen quedado sin estas mejoras. Este es el principal objetivo de este trabajo y es el tema central del proyecto. Gracias a esto se cumple el OBJ1.

Las tecnologías de gestión del ciclo de vida de las aplicaciones y de la infraestructura en la que estas aplicaciones se despliegan que he aplicado en este TFM bajo el paraguas de los principios DevOps y GitOps (como son CI/CD, Git o Kubernetes, en su variante KubeEdge), han permitido la creación de un sistema robusto y sencillo de administrar. Las capacidades de resistencia a fallos y auditoría de los despliegues han permitido que el desarrollo se lleve a cabo con mayor facilidad y rapidez. Sin estas herramientas el sistema no sería tan completo y, como añadido, ha sido mucho más fácil de implementar una vez estas herramientas han estado configuradas. Ha merecido la pena el esfuerzo de implementación de estas herramientas de desarrollo. Con esto se cumple el OBJ2.

La entrega de valor al usuario ha demostrado ser rápida y positiva por la fácil instalación y el inicio automático de todos los componentes de este sistema. Realmente, este objetivo era difícil de conseguir por la conjunción hardware/software que se requiere para que esto suceda. El hardware tiene que ser lo suficientemente sencillo de instalar como para que cualquiera pueda llegar a hacerlo, pero esto no debe restar funcionalidad al sistema. Por su parte, el software, gracias a las herramientas expuestas en el párrafo anterior, debe desplegarse y replicarse de forma automática, para permitir que cualquier usuario pueda

instalar el software sin que requiera ninguna acción de configuración por su parte. Esta parte cumple el OBJ3.

La aplicación de culturas orientadas a soportar la construcción de software, al hardware, es realmente difícil, dado que los tiempos y ciclos de desarrollo no son los mismos. Con este proyecto se ha demostrado que es beneficioso para el desarrollo aplicar este tipo de técnicas ya que, al menos, permiten la mejora en la revisión del código y del estado general del proyecto. Aunque, queda claro, que no se ha desarrollado ningún tipo de circuito o placa compleja, ya que este no era el objetivo de este proyecto.

Otro de los objetivos de este proyecto es hacer que sea libre y totalmente reproducible por cualquiera: la intención es que el dispositivo le pertenezca realmente al usuario, al contrario que otras muchas soluciones en las que el hardware es del usuario, pero este no tiene ningún tipo de control sobre él. Si el fabricante decidiese dejar de prestar servicio, el usuario, en consecuencia, se quedaría con un hardware que además de caro, es inútil. Por esto mismo, la inclusión de dispositivos fácilmente replicables, físicos y que el usuario realmente puede poseer marca la diferencia frente a otros proyectos comerciales. La nube, realmente puede ser un servidor más o menos sencillo en el que se ejecutarían las capas: nube y visualización, y, por supuesto, el usuario tendría completo control sobre su propia plataforma.

Como herramienta de aprendizaje, este proyecto ha conseguido generar en mí una visión total de lo que sería un despliegue completo al menos desde la perspectiva de un Ingeniero Informático. He tocado muchas, por no decir la mayor parte de las materias estudiadas en este Máster y, en extensión, una cantidad importante de las asignaturas de Grado, que, en parte, tenía olvidadas. Ha supuesto una reflexión en cómo las distintas ramas de conocimiento que componen a la Ingeniería Informática se unen e interactúan entre ellas, cómo se configura como una Ingeniería. Este conjunto de técnicas engranándose en perfecta sincronía ha sido justo lo que estaba buscando, algo que tenía muy claro antes del comienzo de este proyecto: quería ver cómo se relacionaba todo, cómo es un desarrollo completo y no una sola pieza de este puzle. Es importante destacar que, al no haberme podido centrar en una pieza en concreto, ninguna de las que componen este sistema es excelente, pero todas trabajan en sincronía y conforman un conjunto funcional del que estoy muy orgulloso. Mis conocimientos en cuanto a programación y entendimiento de microcontroladores son mucho más altos ahora que cuando empecé, que quizás era la parte que llevaba más floja. Los conocimientos que he adquirido sobre desarrollo y despliegue con este proyecto tampoco se quedan cortos, he disfrutado viendo como todo encajaba y funcionaba y, cuando no lo hacía, he sentido esa frustración que me ha hecho continuar y, finalmente, lograr que sucediese.

## 8.1. Impacto de este proyecto

Finalmente, y como fin de estas conclusiones, me gustaría referenciar de forma breve cuáles creo que son las áreas dentro de los Objetivos de Desarrollo Sostenible (ODS) [77] en las que este proyecto tendría un impacto real.

### 8.1.1. ODS 2: Hambre cero

El aumento en la eficiencia de la agricultura está íntimamente ligado con la cantidad de alimentos que se producen con la misma inversión y esfuerzo. Una mejora en este ámbito equivale al abaratamiento de los alimentos y que, con ello, la desnutrición se reduzca. El

objetivo dicta la necesidad de poner fin al hambre y asegurar el acceso de todas las personas a los alimentos necesarios.

Sistemas de agricultura inteligente como este refuerzan las medidas de aumento de la productividad y la producción con respeto al medio ambiente por la reducción de consumo de recursos hídricos reduciendo el desperdicio de estos.

### 8.1.2. ODS 12: Producción y consumo responsables

Dentro de este objetivo se encuentra la producción ecológica y sostenible de alimentos, reduciendo los desperdicios que se provocan al desechar los no consumidos. Aumentando la eficacia de las plantaciones y sus cosechas se consigue mejorar este tipo de producciones con un menor coste para el planeta.

## 8.2. Líneas futuras

Como ya se ha comentado en el apartado anterior, ninguno de los componentes de este sistema ha sido desarrollado intensivamente, aquí surgen las líneas futuras de este proyecto: mejorar lo ya desarrollado y resolver los problemas que han surgido durante y al finalizar el desarrollo.

En la capa física, la mejora fundamental y la que considero que es un tema muy importante es la inclusión de los dispositivos que componen esta capa dentro del flujo CI/CD. Actualmente, por cómo está desarrollado el sistema es realmente difícil y poco probable. La única herramienta de comunicación de la que se dispone es LoRa y, por el ancho de banda de esta y las pérdidas que se pueden producir, se hace harto complicado realizar una actualización vía OTA de estos dispositivos utilizando esta red. La solución sería la incorporación de otra vía de comunicación, como por ejemplo WiFi o Bluetooth que sí que tienen el ancho de banda suficiente para realizar esta tarea. Esto viene también con el perjuicio de la necesidad de que un trabajador se desplazase dispositivo a dispositivo (el alcance de ambas tecnologías es muy bajo) para conectarse a ellos y permitir dicha actualización. Precisamente por esto las ventajas frente al tradicional *flasheo* utilizando el puerto USB que incorporan estas placas no son tan altas. Aunque sería un gran logro hacer que esto fuese posible sin la intervención humana.

La segunda mejora importante para la capa física es la mejora de la calidad de sensores y actuadores. Por cuestiones económicas y la finalidad de este proyecto estaba fuera del alcance la inclusión de mejores sensores y actuadores, aunque es una mejora totalmente necesaria para hacer de este un producto realmente funcional. En especial el sensor de humedad del suelo que, como ya se ha contado, ha fallado. La dificultad radica en la falta de sensores libres que hay en el mercado, casi todos los fabricantes ponen trabas a su compra para poder suministrarlos únicamente con sus sistemas. Este es un gran problema para la libertad del consumidor.

La mejora más importante para los actuadores es evitar que el MOSFET que utilizan quede inservible por el calor que disipan. El siguiente paso es encontrar una solución como puede ser un disipador para este problema y reducir así la problemática. También, es necesario estudiar otras posibles soluciones que no eviten este problema y que puedan mejorar el funcionamiento general de los actuadores.

Otra mejora para la capa física es la inclusión de otro sistema de riego como pueden ser los aspersores, tener dos sistemas de riego es más eficaz para determinados cultivos como son los frutales porque reducen los riesgos de heladas y otras inclemencias y mejoran la capacidad de riego. Más sensores como pluviómetros y un mayor número de sensores de humedad en el suelo a distintas profundidades siempre mejoran la calidad de los datos y la toma de decisiones.

La principal mejora para la capa edge sería la inclusión de un Gateway LoRa que permitiese una mayor capacidad de comunicación con los dispositivos y un mayor alcance. En decremento de la facilidad y economización de la solución, pero en incremento de la calidad de servicio ofrecida por el sistema. Esto se planteó al comienzo del proyecto, pero se desechó por su alto coste y porque, tal y como se ha hecho, considero que he podido aprender más creando mis propios métodos de identificación de dispositivos dentro de una red y el cifrado de los mensajes utilizando algoritmos habituales en el IoT.

Para la capa de datos, la verdad es que hay pocas mejoras disponibles, quizás mejorar la calidad de la API, aumentando su seguridad y resolviendo el problema de alojar certificados en los propios nodos y actualizarlos.

La capa de visualización, sin embargo, tiene un amplio margen de mejora, ha sido desde su planteamiento un simple panel en el que mostrar los datos y esto tiene mucho potencial por delante. La configuración debería ser realizable desde esta propia plataforma y actualizable por cada agricultor para cada una de sus plantaciones. El análisis de los datos es muy deficiente en comparación con otras soluciones disponibles en el mercado y con cualquiera de los sistemas de análisis agrícolas. Es un ámbito en el que mejorar y mucho, pero se escapaba del objetivo de este proyecto. Es cierto que estos datos deberían aprovecharse mucho mejor e, incluso, mejorar el comportamiento del sistema incorporando aprendizaje automático a las reacciones de riego o no riego. Es una buena línea de investigación que tiene mucho potencial en el largo plazo.

La capa ALM también tiene un amplio rango de mejora, el principal es la unificación de todos los procesos en un único ciclo para mejorar la auditabilidad y la calidad del análisis que se le pueda hacer al software. La inclusión de distintos escenarios: desarrollo, preproducción y producción, por ejemplo, es algo imprescindible en un sistema completo y que, por falta de tiempo y recursos, no se ha llevado a cabo. En un sistema más evolucionado es algo imprescindible en lo que seguir trabajando. Otro punto importante es la medición del rendimiento de los nodos dentro de KubeEdge y el mayor aprovechamiento de los recursos que esta solución ofrece. En este proyecto, en parte por la inclusión del sombrero LoRa en las Raspberry Pis no se ha podido aprovechar mejor todas estas ventajas como el gemelo virtual o DeviceTwin y los importantes avances y facilidades que esto traería de la mano. El sistema de mapeo de diferentes protocolos también es algo muy para tener en cuenta por la facilidad que supone su uso junto con MQTT.

A modo de conclusión, este es un proyecto prometedor que necesita mejoras en todas sus áreas, mejoras que harían de este proyecto una solución integral para el control de riego dentro del marco de la agricultura inteligente. Utilizando siempre recursos económicos que estén disponibles para todos los usuarios y no cerrados para unos pocos y de los que el usuario tenga el control total.

# 9. Referencias


[1] L. Atzori, A. Iera, y G. Morabito, «Understanding the Internet of Things: definition, potentials, and societal role of a fast evolving paradigm», *Ad Hoc Networks*, vol. 56, pp. 122-140, mar. 2017, doi: 10.1016/J.ADHOC.2016.12.004.

[2] N. Ahmed, D. De, y I. Hussain, «Internet of Things (IoT) for Smart Precision Agriculture and Farming in Rural Areas», *IEEE Internet Things J*, vol. 5, n.º 6, pp. 4890-4899, dic. 2018, doi: 10.1109/JIOT.2018.2879579.

[3] Docker, «Docker Overview ». https://docs.docker.com/get-started/ (accedido ene. 13, 2023).

[4] J. Castro *et al.*, «Don't Panic: Kubernetes and Docker», *Kubernetes Blog*, dic. 02, 2020. https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/ (accedido ene. 13, 2023).

[5] A. TORETI *et al.*, «Drought in Europe July 2022», *JRC Global Drought Observatory Analtical Report - July 2022*, p. 17, 2022, doi: 10.2760/014884.

[6] R. Abbasi, P. Martinez, y R. Ahmad, «The digitization of agricultural industry – a systematic literature review on agriculture 4.0», *Smart Agricultural Technology*, vol. 2, p. 100042, dic. 2022, doi: 10.1016/J.ATECH.2022.100042.

[7] Join Research Centre (JRC) of the European Commision y Monitoring Agriculture ResourceS (MARS) Unit H04, «PRECISION AGRICULTURE: AN OPPORTUNITY FOR EU FARMERS - POTENTIAL SUPPORT WITH THE CAP 2014-2020», jun. 2014. doi: 10.2861/58758.

[8] A. Whitmore, A. Agarwal, y L. Xu, «The Internet of Things—A survey of topics and trends», *Information Systems Frontiers*, vol. 17, pp. 261-274, 2015.

[9] L. Atzori, A. Iera, y G. Morabito, «The Internet of Things: A survey», *Computer Networks*, vol. 54, n.º 15, pp. 2787-2805, oct. 2010, doi: 10.1016/J.COMNET.2010.05.010.

[10] R. Roman, J. Zhou, y J. Lopez, «On the features and challenges of security and privacy in distributed internet of things», *Computer Networks*, vol. 57, n.º 10, pp. 2266-2279, jul. 2013, doi: 10.1016/J.COMNET.2012.12.018.

[11] LF Edge y K. Rinehart, «Sharpening the Edge: Overview of the LF Edge Taxonomy and Framework», *LFedge*, 2020, Accedido: ene. 11, 2023. [En línea]. Available: https://www.lfedge.org/wp-content/uploads/2020/07/LFedge_Whitepaper.pdf

[12] S. Wang, «Edge Computing: Applications, State-of-the-Art and Challenges», *http://www.sciencepublishinggroup.com*, vol. 7, n.º 1, p. 15, 2019, doi: 10.11648/J.NET.20190701.12.

[13] C. Ebert, G. Gallardo, J. Hernantes, y N. Serrano, «DevOps», *IEEE Softw*, vol. 33, n.º 3, pp. 94-100, may 2016, doi: 10.1109/MS.2016.68.

[14] D. Spinellis, «Don’t install software by hand», *IEEE Softw*, vol. 29, n.º 4, pp. 86-87, 2012, doi: 10.1109/MS.2012.85.

[15] RedHat, «What is GitOps?», may 10, 2022. https://www.redhat.com/en/topics/devops/what-is-gitops (accedido ene. 09, 2023).

[16] R. Lopez-Viana, J. Diaz, V. H. Diaz, y J.-F. Martinez, «Continuous Delivery of Customized SaaS Edge Applications in Highly Distributed IoT Systems», *IEEE Internet Things J*, vol. 7, n.º 10, pp. 10189-10199, 2020, doi: 10.1109/JIOT.2020.3009633.

[17] Eclipse Foundation, «2020 IoT Developer Survey Key Findings», 2020, Accedido: ene. 12, 2023. [En línea]. Available: https://f.hubspotusercontent10.net/hubfs/5413615/2020%20IoT%C2%A0Developer%20Survey%20Report.pdf

[18] Eclipse Foundation, «IoT & Edge Developer Survey Report», sep. 2022, Accedido: ene. 12, 2023. [En línea]. Available: https://5413615.fs1.hubspotusercontent-na1.net/hubfs/5413615/2022%20IoT%20&%20Edge%20Developer%20Survey%20Report.pdf

[19] G. Koelsch, *Requirements Writing for System Engineering*. Herdon, Virginia, USA: Apress, 2016. doi: 10.1007/978-1-4842-2099-3.

[20] LilyGo, «Xinyuan-LilyGO/LilyGo-LoRa-Series: LilyGo LoRa Series examples». https://github.com/Xinyuan-LilyGO/LilyGo-LoRa-Series (accedido ene. 09, 2023).

[21] Raspberry Pi Foundation, «Raspberry Pi 4 Computer Model B», ene. 2021, Accedido: ene. 09, 2023. [En línea]. Available: www.raspberrypi.org

[22] Adafruit, «Adafruit LoRa Radio Bonnet with OLED - RFM95W ». https://www.adafruit.com/product/4074#description (accedido ene. 09, 2023).

[23] Raspberry Pi Foundation, «Raspberry Pi 3 Model B+», Accedido: ene. 09, 2023. [En línea]. Available: www.raspberrypi.org/products/raspberry

[24] Microsoft Learn, «Serie B ampliable: Azure Virtual Machines - Azure Virtual Machines», sep. 27, 2022. https://learn.microsoft.com/es-es/azure/virtual-machines/sizes-b-series-burstable (accedido ene. 09, 2023).

[25] Semtech, «LoRa Modulation Basics», may 2015. Accedido: ene. 01, 2023. [En línea]. Available: https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R0000001OJu/xvKUc5w9yjG1q5Pb2IIkpolW54YYqGb.frOZ7HQBcRc

[26] A. Banks, E. Briggs, K. Borgendale, R. Gupta, y OASIS, «MQTT Version 5.0», *OASIS Standard*, mar. 2019, Accedido: ene. 12, 2023. [En línea]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-

os.htmlhttps://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.pdfhttp://docs.oasis-open.org/mqtt/mqtt/v5.0/cos01/mqtt-v5.0-cos01.docx

[27] Espressif, «eFuse Manager - ESP32 - — ESP-IDF Programming Guide latest documentation». https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/efuse.html (accedido ene. 09, 2023).

[28] E. Yarrkov, «Cryptanalysis of XXTEA», *Cryptology ePrint Archive*, 2010, Accedido: ene. 09, 2023. [En línea]. Available: https://eprint.iacr.org/2010/254

[29] A. Bose, «boseji/xxtea-iot-crypt: XXTEA Cryptography Library for use in IoT…», ene. 09, 2020. https://registry.platformio.org/libraries/boseji/xxtea-iot-crypt (accedido ene. 09, 2023).

[30] idfduyue, «xxtea · PyPI», oct. 25, 2022. https://pypi.org/project/xxtea/ (accedido ene. 09, 2023).

[31] Microsoft Learn, «Documentación técnica de SQL Server - SQL Server | Microsoft Learn». https://learn.microsoft.com/es-es/sql/sql-server/?view=sql-server-ver16 (accedido ene. 09, 2023).

[32] Microsoft Learn, «Modelo de compra basado en DTU - Azure SQL Database | Microsoft Learn», sep. 26, 2022. https://learn.microsoft.com/es-es/azure/azure-sql/database/service-tiers-dtu?view=azuresql (accedido ene. 09, 2023).

[33] Microsoft Learn, «SQL Server Management Studio (SSMS) - SQL Server Management Studio (SSMS) | Microsoft Learn», ene. 04, 2023. https://learn.microsoft.com/es-es/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16 (accedido ene. 09, 2023).

[34] Microsoft Learn, «Referencia de Transact-SQL (Motor de base de datos) - SQL Server | Microsoft Learn», dic. 20, 2022. https://learn.microsoft.com/es-es/sql/t-sql/language-reference?view=sql-server-ver16 (accedido ene. 09, 2023).

[35] Microsoft Learn, «Información general sobre Azure Functions | Microsoft Learn», dic. 05, 2022. https://learn.microsoft.com/es-es/azure/azure-functions/functions-overview (accedido ene. 09, 2023).

[36] Microsoft Learn, «Información general sobre Visual Studio | Microsoft Learn», oct. 28, 2022. https://learn.microsoft.com/es-es/visualstudio/get-started/visual-studio-ide?view=vs-2022 (accedido ene. 09, 2023).

[37] GitHub, «Understanding GitHub Actions - GitHub Docs», oct. 17, 2022. https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions (accedido ene. 09, 2023).

[38] Microsoft Learn, «Crear un procedimiento almacenado - SQL Server | Microsoft Learn», dic. 20, 2022. https://learn.microsoft.com/es-es/sql/relational-databases/stored-procedures/create-a-stored-procedure?view=sql-server-ver16 (accedido ene. 09, 2023).

[39] Canonical, «MicroK8s - MicroK8s documentation», dic. 22, 2022. https://microk8s.io/docs (accedido ene. 09, 2023).

[40] Kubernetes, «Installing kubeadm», 2022. https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/ (accedido ene. 13, 2023).

[41] Canonical, «MicroK8s - Use, edit or create addons», abr. 22, 2022. https://microk8s.io/docs/howto-addons (accedido ene. 09, 2023).

[42] KubeEdge, «KubeEdge». https://kubeedge.io/en/ (accedido ene. 09, 2023).

[43] KubeEdge, «Why KubeEdge | KubeEdge», jun. 05, 2022. https://kubeedge.io/en/docs/kubeedge/ (accedido ene. 09, 2023).

[44] R. López Viana, «rlopezv/gitops: Repository for GitOps Implementation with Kubeedge», dic. 06, 2022. https://github.com/rlopezv/gitops/ (accedido ene. 09, 2023).

[45] KubeEdge, «Deploying using Keadm», oct. 11, 2022. https://kubeedge.io/en/docs/setup/keadm/ (accedido ene. 13, 2023).

[46] Docker, «Install Docker Engine on Ubuntu | Docker Documentation». https://docs.docker.com/engine/install/ubuntu/ (accedido ene. 09, 2023).

[47] Microsoft Learn, «Registros de contenedor administrados - Azure Container Registry | Microsoft Learn», dic. 05, 2022. https://learn.microsoft.com/es-es/azure/container-registry/container-registry-intro (accedido ene. 09, 2023).

[48] Microsoft Learn, «Habilitación de una clave administrada por el cliente - Azure Container Registry | Microsoft Learn», nov. 29, 2022. https://learn.microsoft.com/es-es/azure/container-registry/tutorial-enable-customer-managed-keys (accedido ene. 09, 2023).

[49] Espressif, «Sleep Modes - ESP32 - — ESP-IDF Programming Guide latest documentation». https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html (accedido ene. 09, 2023).

[50] ThingsBoard, «What is ThingsBoard? | ThingsBoard Community Edition». https://thingsboard.io/docs/getting-started-guides/what-is-thingsboard/ (accedido ene. 09, 2023).

[51] ThingsBoard, «Installing ThingsBoard on Raspberry Pi 3 Model B | ThingsBoard Community Edition». https://thingsboard.io/docs/user-guide/install/rpi/ (accedido ene. 09, 2023).

[52] J. Anderson, «FreeDNS». https://freedns.afraid.org/ (accedido ene. 09, 2023).

[53] GitHub, «GitHub». https://github.com/ (accedido ene. 09, 2023).

[54] Microsoft Learn, «¿Qué es Azure DevOps? - Azure DevOps | Microsoft Learn», nov. 18, 2022. https://learn.microsoft.com/es-es/azure/devops/user-guide/what-is-

azure-devops?toc=%2Fazure%2Fdevops%2Fget-started%2Ftoc.json&view=azure-devops (accedido ene. 09, 2023).

[55] Jenkins, «Jenkins User Documentation». https://www.jenkins.io/doc/ (accedido ene. 09, 2023).

[56] QEMU Project, «About QEMU — QEMU documentation», 2022. https://www.qemu.org/docs/master/about/index.html (accedido ene. 09, 2023).

[57] Argo CD, «Argo CD - Declarative GitOps CD for Kubernetes». https://argo-cd.readthedocs.io/en/stable/ (accedido ene. 09, 2023).

[58] Argo CD, «Getting Started - Argo CD». https://argo-cd.readthedocs.io/en/stable/getting_started/ (accedido ene. 09, 2023).

[59] Kubernetes, «Secrets | Kubernetes», nov. 15, 2022. https://kubernetes.io/docs/concepts/configuration/secret/ (accedido ene. 09, 2023).

[60] K. Söderby, «Getting Started with Arduino | Arduino Documentation | Arduino Documentation», dic. 15, 2022. https://docs.arduino.cc/learn/starting-guide/getting-started-arduino (accedido ene. 09, 2023).

[61] Microsoft Learn, «Un paseo por C#: información general | Microsoft Learn», sep. 22, 2022. https://learn.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/ (accedido ene. 09, 2023).

[62] Python Software Foundation, «El tutorial de Python — documentación de Python - 3.11.1». https://docs.python.org/es/3/tutorial/ (accedido ene. 09, 2023).

[63] PlatformIO, «Professional collaborative platform for embedded development — PlatformIO latest documentation». https://docs.platformio.org/en/latest/ (accedido ene. 09, 2023).

[64] Microsoft, «Documentation for Visual Studio Code». https://code.visualstudio.com/docs (accedido ene. 09, 2023).

[65] Altium, «CircuitMaker Documentation | User Manual | Documentation», feb. 10, 2022. https://www.altium.com/documentation/altium-circuitmaker (accedido ene. 09, 2023).

[66] Autodesk, «Fusion 360 | Software de CAD 3D, CAM, CAE y PCB basado en la nube | Autodesk». https://www.autodesk.es/products/fusion-360/overview?term=1-YEAR&tab=subscription (accedido ene. 09, 2023).

[67] Ultimaker, «Ultimaker Cura: software de impresión 3D potente y fácil de usar | Ultimaker». https://ultimaker.com/es/software/ultimaker-cura (accedido ene. 09, 2023).

[68] Docker, «Docker Desktop | Docker Documentation». https://docs.docker.com/desktop/ (accedido ene. 09, 2023).

[69] UPM, «Biblioteca UPM - Ingenios». https://ingenio.upm.es/primo-explore/search?vid=34UPM_VU1&lang=es_ES (accedido ene. 09, 2023).

[70] Elsevier, «ScienceDirect.com | Science, health and medical journals, full text articles and books.» https://www.sciencedirect.com/ (accedido ene. 09, 2023).

[71] Google, «Google Académico». https://scholar.google.es/schhp?hl=es (accedido ene. 09, 2023).

[72] ResearchGate, «ResearchGate». https://www.researchgate.net/ (accedido ene. 09, 2023).

[73] IEEE, «IEEE Xplore». https://ieeexplore.ieee.org/Xplore/home.jsp (accedido ene. 13, 2023).

[74] Mendeley, «Mendeley Cite». https://www.mendeley.com/reference-management/mendeley-cite (accedido ene. 09, 2023).

[75] Mendeley, «Mendeley Reference Manager». https://www.mendeley.com/reference-management/reference-manager (accedido ene. 09, 2023).

[76] I. Periodicals, «IEEE REFERENCE GUIDE», 2018, Accedido: ene. 09, 2023. [En línea]. Available: https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf

[77] Instituto Nacional de Estadística, «Indicadores Agenda 2030 Desarrollo Sostenible de la para el Instituto Nacional de Estadística», 2021, Accedido: ene. 13, 2023. [En línea]. Available: https://www.ine.es/ods/publicacion_ods.pdf

[78] J. Hrisko, «Capacitive Soil Moisture Sensor Theory, Calibration, and Testing», 2020, doi: 10.13140/RG.2.2.36214.83522.

[79] S. van Steenis, «Change default DNS nameserver used by Kubernetes pods», jul. 18, 2018. https://gist.github.com/superseb/f6894ddbf23af8e804ed3fe44dd48457 (accedido ene. 09, 2023).

# 10. Anexos o apéndices

## 10.1. Instalación de los Sensores

Como puede verse en: Ilustración 10 e Ilustración 13, la instalación se ha realizado en un peral. En la Ilustración 10 pueden verse todos los componentes y su disposición, de igual forma, puede verse el peral monitorizado. Se puede ver también la instalación de riego por goteo. El sensor de humedad del suelo ha sido situado próximo al árbol y a una profundidad suficiente. El sensor de humedad en hoja ha sido situado en una rama que se encuentra en una posición alcanzable desde el suelo y a una altura media.

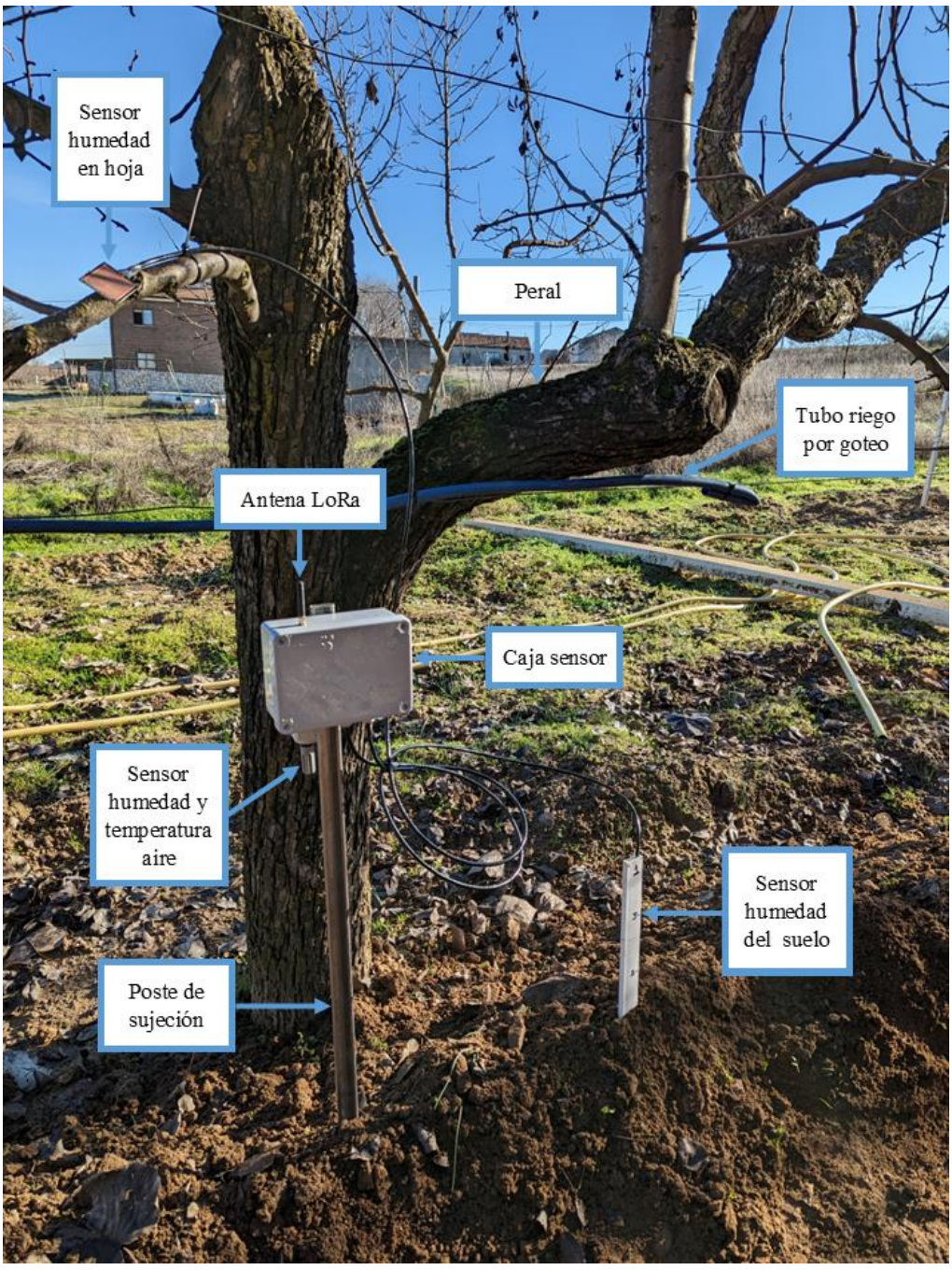



*Ilustración 10 Vista detallada instalación en peral*

*Ilustración 11 Vista general instalación en peral*

En Ilustración 11 e Ilustración 12 puede verse la instalación del sensor en una vid. De forma similar a la del peral, se pueden ver los componentes que forman este sensor. Se puede apreciar que la instalación es algo diferente, pero fundamentalmente es igual.

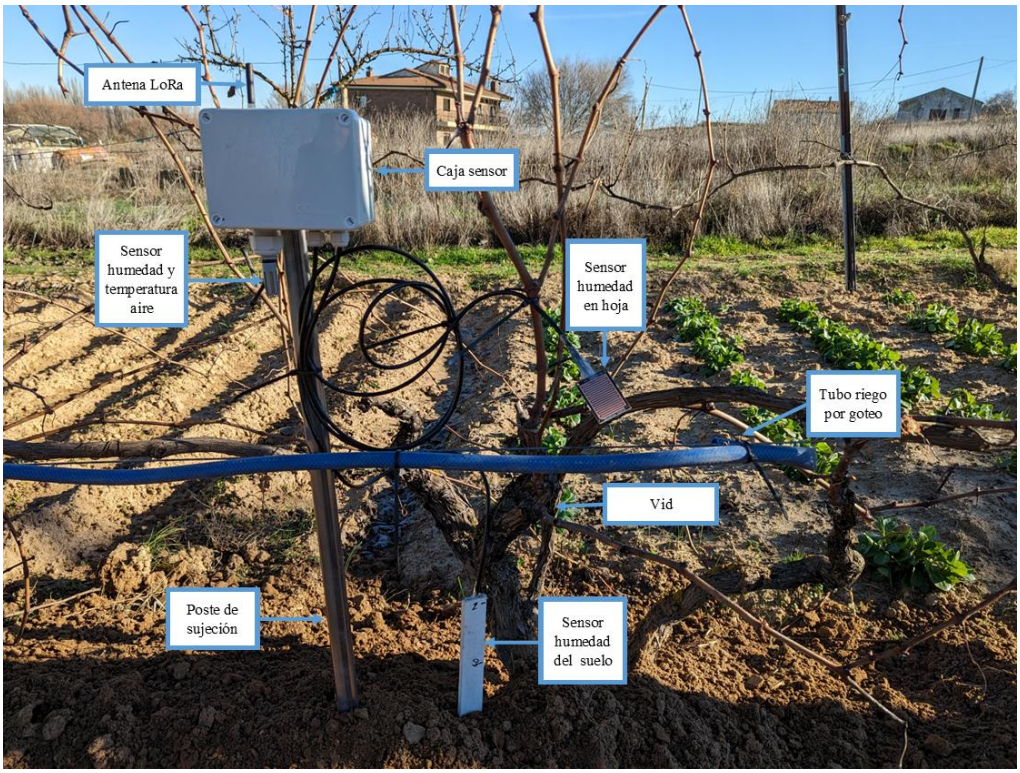



*Ilustración 12 Vista detallada instalación en vid*

*Ilustración 13 Vista general instalación en vid*

Como puede apreciarse, la instalación es simple y rápida una vez se tiene claro el lugar y el cómo hacerlo. Los componentes son sencillos de instalar y permiten su retirada si fuese necesario.

## 10.2. Instalación de los Actuadores

Como puede verse en: Ilustración 14 e Ilustración 15, así ha resultado la instalación final del sistema de actuación. En la Ilustración 14 se descubre la instalación general del sistema de actuación, detallando cada componente en la misma. Como puede observarse es muy simple, contando con únicamente las cajas donde están instalados los actuadores y las válvulas a accionar. El suministro eléctrico se obtiene de la caja marcada, en su interior se encuentra la batería utilizada para la alimentación de los actuadores e instalada anteriormente en la plantación para la captación de agua de un pozo subterráneo y su almacenamiento en depósitos. Esta batería es recargada por un panel solar situado en la misma plantación.

En ellas puede percibirse cuál es la distribución de ambos actuadores y sus respectivas válvulas. Como puede distinguirse, han sido instalados en la zona en la que hay una toma de agua del sistema de riego y también está la batería del sistema de extracción de agua y, por lo tanto, la fuente de alimentación del sistema de actuadores. Este es el lugar óptimo para la instalación por las circunstancias anunciadas anteriormente. Es importante recalcar que las válvulas tras la toma de las fotos fueron tapadas para que no sufriesen problemas por las inclemencias climatológicas.

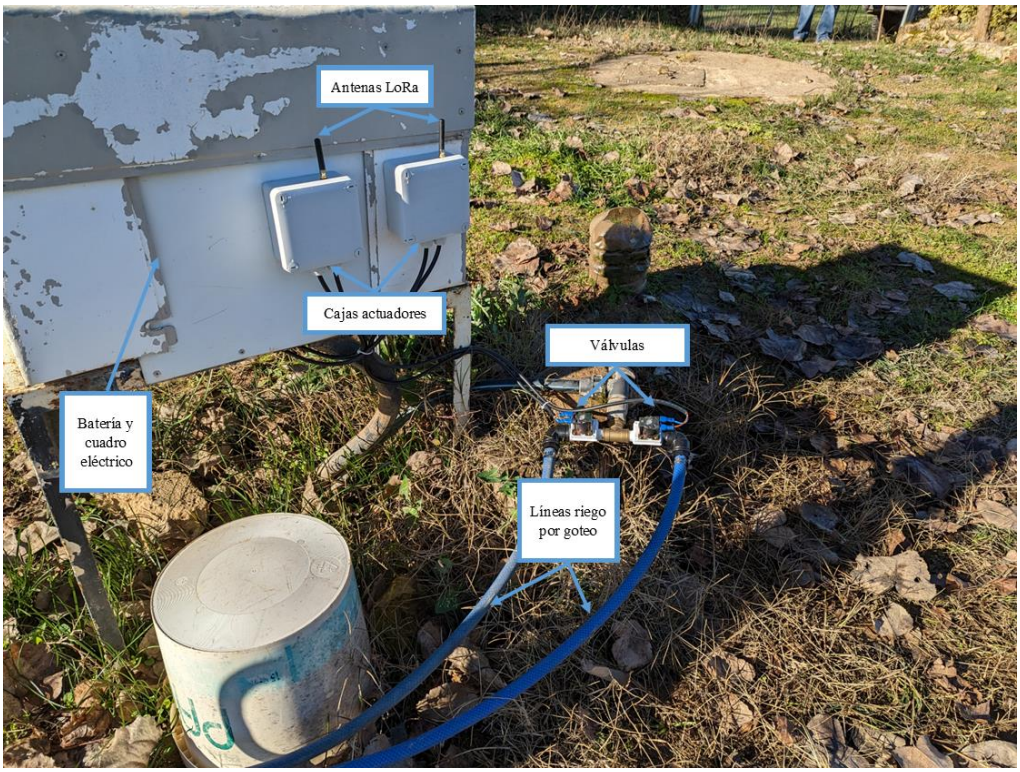



*Ilustración 14 Vista detallada de la instalación de actuadores*

En la Ilustración 15 se puede contemplar el resultado de la conexión de las válvulas. El suministro de agua proviene de los depósitos presentes en la plantación. Las dos líneas de riego por goteo son las que luego son utilizadas en el peral y la vid del capítulo anterior.

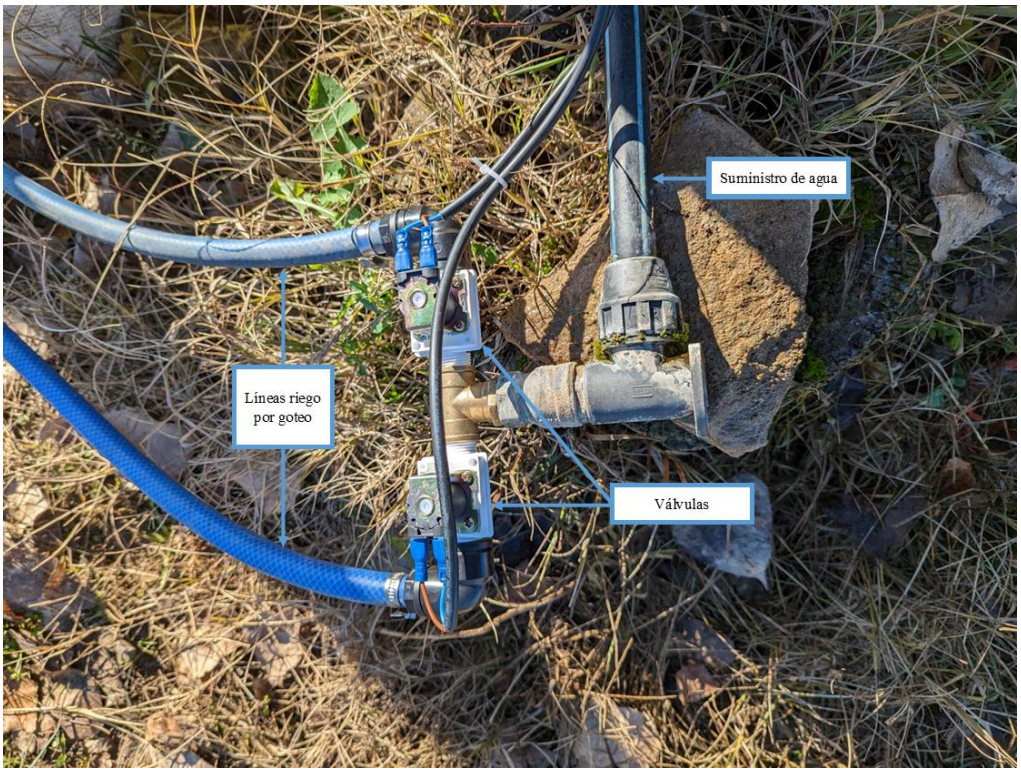


*Ilustración 15 Vista detallada de instalación de válvulas*

Como puede verse, el sistema de actuación no dista mucho de los tradicionales sistemas automatizados de riego, aunque a una escala menor por las necesidades del proyecto. Se han utilizado válvulas a 12 V porque esta era la alimentación presente en la plantación, aunque lo más habitual es el uso de válvulas de 24 V ya que es el estándar.

## 10.3. Pipeline CI

```
# Variable ‘githubkey’ was defined in the Variables tab
trigger:
  branches:
    include:
    - main
resources:
  repositories:
  - repository: self
    type: git
    ref: refs/heads/main
jobs:
- job: Job_1
  displayName: Agent job 1
  pool:
    vmImage: ubuntu-latest
  steps:
  - checkout: self
    clean: true
    fetchTags: false
  - task: CmdLine@2
    displayName: Install Qemu
    inputs:
      script: >
        docker run–rm–privileged multiarch/qemu-user-static–reset -p yes
  - task: Docker@2
    displayName: build
    inputs:
      containerRegistry: 58a7b671-f9eb-4d0d-bebb-deeded693942
      repository: receiver
      command: build
      Dockerfile: modules/receiver/Dockerfile
      arguments: --platform linux/arm64
  - task: Docker@2
    displayName: push
    inputs:
      containerRegistry: 58a7b671-f9eb-4d0d-bebb-deeded693942
      repository: receiver
      command: push
  - task: Bash@3
    displayName: Update kubernetes manifest
    inputs:
      filePath: DevOps/updateImage.sh
  - task: CopyFiles@2
    displayName: ‘Copy Files to: $(Build.ArtifactStagingDirectory)’
    inputs:
      Contents: >-
        **

        ${{bake.manifestsBundle}}
      TargetFolder: $(Build.ArtifactStagingDirectory)
  - task: PublishBuildArtifacts@1
    displayName: ‘Publish Artifact: drop’
...
```

*Código 3 SmartAgriculture-CI.yaml*

## 10.4. *Manifest* Kubernetes

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: receiver
  name: edge-receiver
spec:
  replicas: 1
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: receiver
  template:
    metadata:
      labels:
        app: receiver
    spec:
      containers:
      - name: edge-receiver
        image: sanoderegistry.azurecr.io/receiver:37
        imagePullPolicy: IfNotPresent
        env:
        - name: NODE_NAME
          value: "rasp1"
        ports:
        - name: mqtt
          containerPort: 1883
          protocol: TCP
        - name: mqtt-2
          containerPort: 11883
          protocol: TCP
        - name: https
          containerPort: 443
          protocol: TCP
        - name: websocket
          containerPort: 80
          protocol: TCP
        securityContext:
          privileged: true
      imagePullSecrets:
      -  name: azsecret
```

*Código 4 deployment.yaml*

## 10.5. Pipeline GitHub Actions

```yaml
# Docs for the Azure Web Apps Deploy action: https://github.com/azure/functions-action
# More GitHub Actions for Azure: https://github.com/Azure/actions

name: Build and deploy dotnet core app to Azure Function App - SA-API

on:
  push:
    branches:
      - main
  workflow_dispatch:

env:
  AZURE_FUNCTIONAPP_PACKAGE_PATH: 'SmartFunction/apiSmartAgriculture' # set this to the path to your web app project, defaults to the repository root
  DOTNET_VERSION: '6.0.x' # set this to the dotnet version to use

jobs:
  build-and-deploy:
    runs-on: windows-latest
    steps:
      - name: 'Checkout GitHub Action'
        uses: actions/checkout@v2

      - name: Setup DotNet ${{ env.DOTNET_VERSION }} Environment
        uses: actions/setup-dotnet@v1
        with:
          dotnet-version: ${{ env.DOTNET_VERSION }}

      - name: 'Resolve Project Dependencies Using Dotnet'
        shell: pwsh
        env:
          CONNECTIONSTRING: ${{ secrets.CONNECTIONSTRING }}
        run: |
          pushd './${{ env.AZURE_FUNCTIONAPP_PACKAGE_PATH }}'
          dotnet build --configuration Release --output ./output
          popd

      - name: 'Run Azure Functions Action'
        uses: Azure/functions-action@v1
        id: fa
        with:
          app-name: 'SA-API'
          slot-name: 'Production'
          package: '${{ env.AZURE_FUNCTIONAPP_PACKAGE_PATH }}/output'
          publish-profile: ${{ secrets.AZUREAPPSERVICE_PUBLISHPROFILE_EE8568197A114E42AA5227981DB3AD97 }}
```

*Código 5 main_sa-api.yml*

## 10.6. Calibración de los sensores de humedad del suelo

Por la naturaleza de los sensores capacitivos estos tienen que ser calibrados específicamente para cada suelo [78]. La introducción de factores ajenos al sensor y que le afectan a este como pueden ser: la compactación del suelo, el contenido de sales en este o la propia capacidad de campo, imponen la necesidad de una calibración dedicada a cada sensor y suelo. Esto provoca que la instalación de estos sensores no sea tan simple como pueden serlo otros y que requieran de una dedicación mayor. Para la calibración se ha desarrollado una herramienta específica llamada "ESP32_moistureSensorsCalibrator" que puede encontrarse en el repositorio SA-edgeDevices.

El primer paso para la calibración es la preparación de la muestra. Para ello, es necesario tomar una muestra del suelo en el que se va a instalar, para lo que es evidentemente necesario tener acceso a la plantación. En este caso se tomaron alrededor de 500ml de tierra. Esta tierra debe ser tratada para poder trabajar con ella, para ello deben eliminarse todas las impurezas que esta tenga como raíces u otros elementos vegetales y posteriormente debe ser secada completamente. En este caso el método de secado elegido es el horneado, para ello se calienta en un horno la muestra a 150ºC durante unas 2h, cada media hora se remueve para lograr un secado completo. Una vez se ha terminado este proceso de secado, es necesario separar los grandes bloques y tomar solo la arena suelta.

El segundo paso consiste en la toma de las mediciones necesarias para la calibración. Para ello y gracias a la herramienta expuesta en el punto anterior, se sitúa la sonda en una muestra del suelo a la que se le va añadiendo agua, aumentando así la proporción de volumen de agua por volumen de tierra. Esto se ha hecho con una muestra de 600 $cm^3$ y añadiendo agua 50 $cm^3$ cada paso hasta llegar al punto se saturación (200 $cm^3$). Tras añadir agua, se mezcla la muestra para obtener una muestra homogénea y se vuelve a tomar otra medida. Estas medidas son la media de 10 mediciones del voltaje retornado por la sonda y, por lo tanto, se asigna un voltaje a cada proporción de agua en tierra. De esta forma se reduce el error que introduce el sensor en cada medición y se obtienen datos más realistas. Los resultados obtenidos para cada par voltaje-cantidad de agua son almacenados. Este proceso se debe hacer para cada conjunto de placa, sensor y suelo ya que la medición de cada placa puede variar ya que la calibración no es exacta.

En la Tabla 6 se puede observar un ejemplo de las mediciones obtenidas y cómo se han realizado. Lo ideal es hacer cuantas más medidas, mejor, pero con 5 ya se puede extrapolar una función coherente y precisa.

| *Mediciones* | *2879* | *2165* | *1508* | *1362* | *1316* |
|---|---|---|---|---|---|
| *Contenido agua* | 0 | 50 | 100 | 150 | 200 |

*Tabla 6 Resultados calibración del Sensor 1*

Con estas medidas se puede observar que la capacidad de campo se sitúa alrededor de la medición de 150$cm^3$, siendo esta próxima a lo esperado para el tipo de suelo del terreno.

$$CC = \frac{VolumenDeAguaCC}{VolumenSueloSeco} = \frac{150cm^3}{600cm^3} = 0{,}25$$

El tercer paso es el tratamiento de estos datos. Previamente, los datos obtenidos en el paso anterior han sido formalizados en un CSV en el que la primera fila indica el volumen total de tierra seca, la segunda las mediciones realizadas y, en la tercera, el contenido de agua introducido en cada medición. Para este tratamiento se ha desarrollado otra herramienta, esta vez un cuaderno Jupyter utilizando Python para el procesamiento de estos datos: "dataProcessor". En este se ajusta un polinomio al conjunto de datos introducidos, para ello se utiliza la librería NumPy. Para el ejemplo anterior se obtienen los siguientes resultados:

$$y = 1{,}21331970 * 10^{-7} * x^2 - 6.86297454 * 10^{-4} * x + 9{,}76469830 * 10^{-1}$$

Para lo que da un coeficiente de determinación:

$$R^2 = 0.9140799861147443$$

Lo que implica que el método empleado es correcto y, por lo tanto, el polinomio generado es utilizable. Los coeficientes *a*, *b* y *c* de este polinomio son incorporados al sensor específico en la base de datos para que, posteriormente, se puedan interpolar los valores y hallar la cantidad de agua en la capa *edge* utilizando para ello el valor enviado por el sensor. El error que se obtiene aquí debería sumarse a los posibles errores que se han añadido por las mediciones manuales en el primer y segundo paso ya que para ello no se ha seguido un proceso tan estricto como el que se hubiese podido seguir en un laboratorio.

Este proceso debe repetirse para cada uno de los sensores que se incluyan en el sistema y el suelo de la plantación en la que se vayan a instalar. Es cierto que se trata de un proceso laborioso, pero es el único disponible para lograr una precisión razonable del sensor y poder considerarlo como medida para el funcionamiento del sistema. Cabe destacar que estas calibraciones se deben hacer a todos los sensores sean de la gama que sean dado que no hay otro método de medir el contenido de agua en el terreno sin requerir de una calibración específica.

En la Ilustración 16 se puede ver el sensor instalado ya en el campo con su envoltorio y la regla que refleja su profundidad.

*Ilustración 16 Instalación final del sensor de humedad*

## 10.7. Configuración DNS en Raspberry Pi

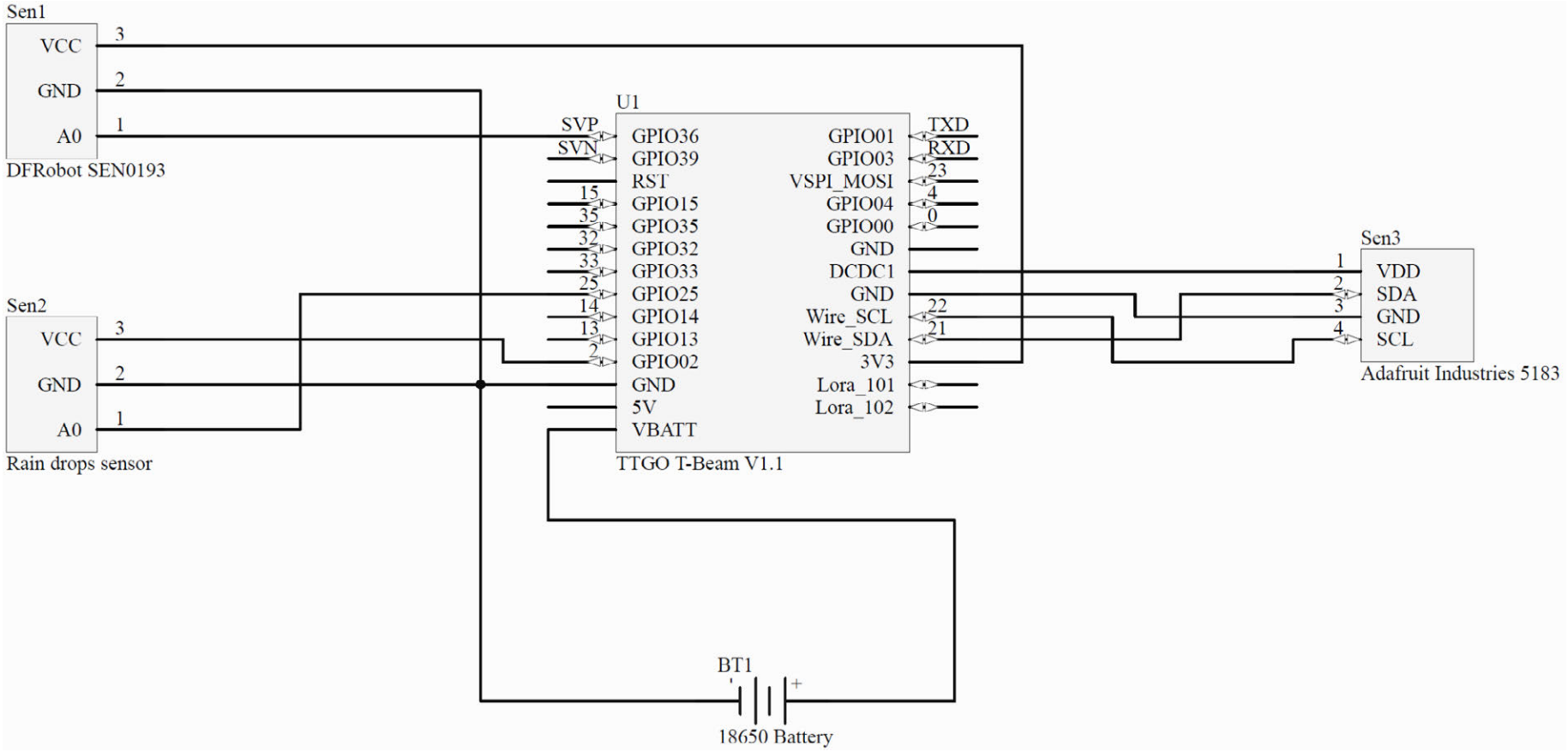
Para poder acceder a Internet desde dentro de un pod, keadm acude al fichero de configuración de resolución de nombres del sistema operativo donde se encuentra instalado, en el caso de Ubuntu es: `/etc/resolv.conf`. El proceso para permitir crear siempre esta configuración es el siguiente:

```
1. sudo systemctl start systemd-resolved
2. sudo systemctl enable systemd-resolved
3. rm -f /etc/resolv.conf
4. sudo ln -s /run/systemd/resolve/resolv.conf /etc/resolv.conf
```

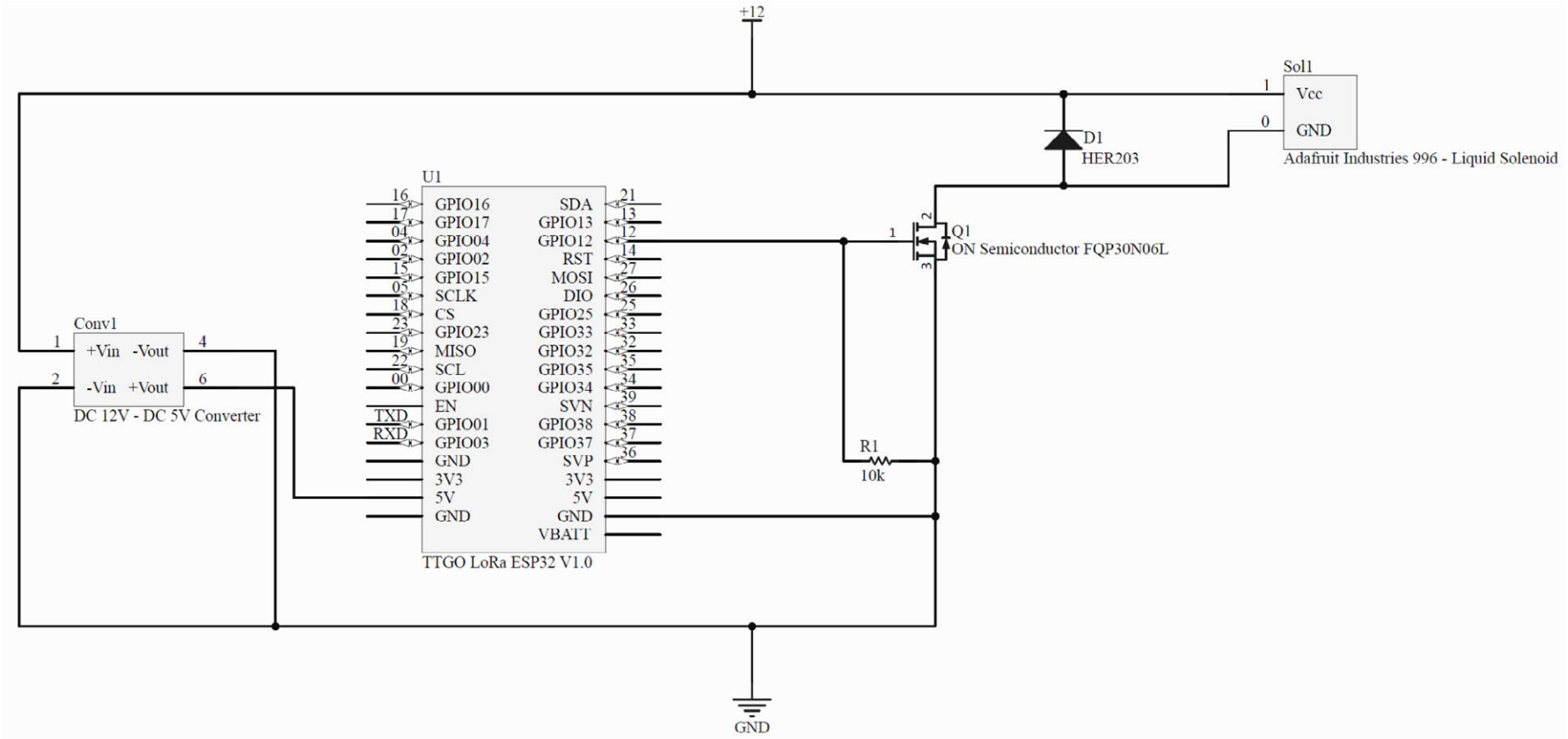
*Código 6 Configuración /etc/resolv.conf* [79]

Como puede verse en el Código 6, se inicia y habilita el servicio de systemd que permite ver la resolución de nombres, esta información systemd la obtiene del rúter al que está conectado el dispositivo (en este caso la Raspberry). Tras esto, se elimina el fichero original y se sustituye por un enlace simbólico al fichero generado por el servicio anterior. De esta forma, se consigue que keadm pueda hacer las resoluciones DNS necesarias para el correcto funcionamiento de los *pods*.

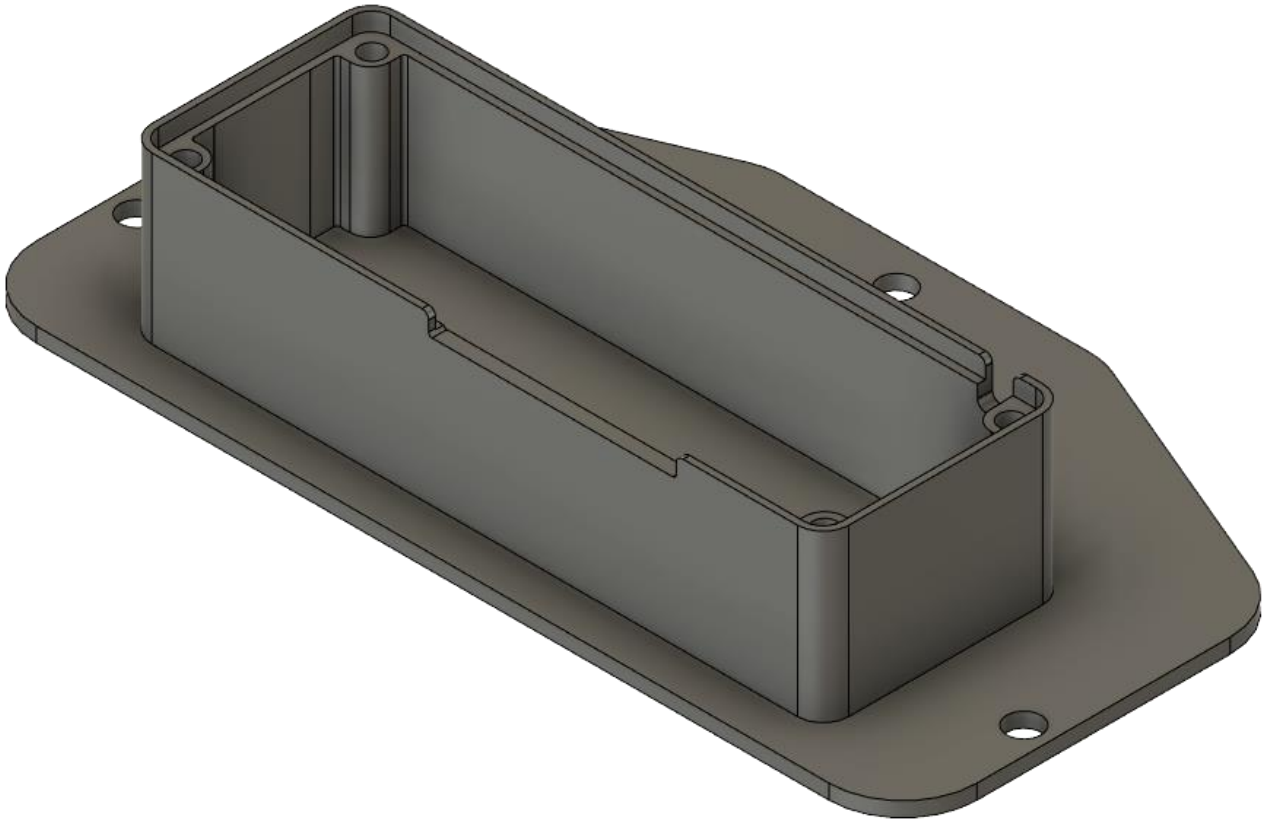
# 11. Esquemáticos electrónicos y planos mecánicos



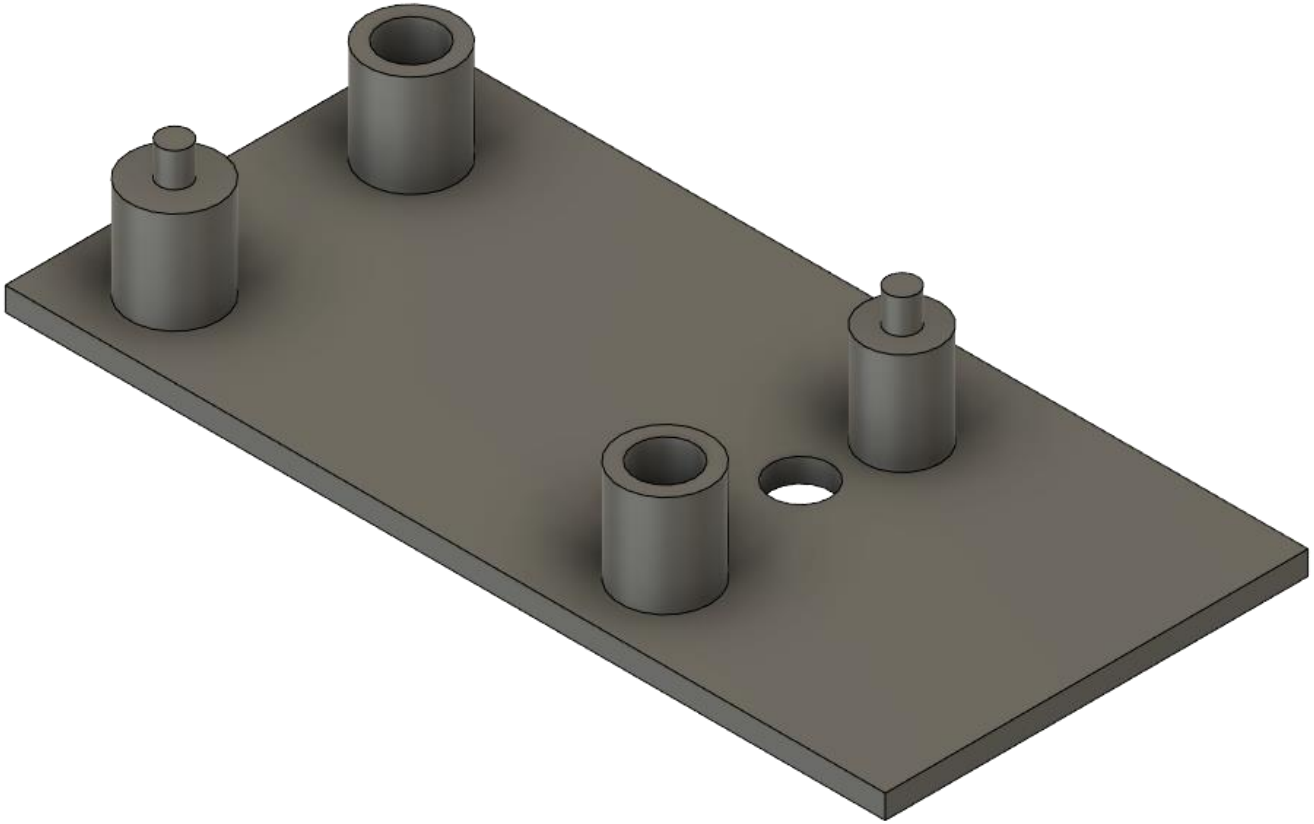

*Esquema 3 Circuito T-Beam*




*Esquema 4 Circuito LoRa ESP32*



*Esquema 5 Soporte para caja estanca T-Beam*



*Esquema 6 Soporte LilyGO LoRa V1.0 ESP32*