

Universidad Internacional de La Rioja

Escuela Superior de Ingeniería y Tecnología

Grado en Ingeniería Informática

Título del Trabajo Fin de Estudios

|  |  |
| --- | --- |
| Trabajo fin de estudio presentado por: | Raúl García Blanco |
| Línea de investigación: |  |
| Director/a: | Jesús Pérez Melero |
| Fecha: | 11/06/2023 |
| Repositorio del código fuente: | https://github.com/rgb2021/TFG/ |

Resumen

Debido a la necesidad de la agricultura de mejorar la eficiencia y la productividad de los cultivos surge el concepto de agricultura de precisión o agricultura inteligente.

En este trabajo se ha construido un sistema de supervisión para cultivos inteligentes. Para ello se han usado sensores y otros elementos de bajo costo como son placas Arduino y Raspberry pi para la recolección de la información del cultivo.

Por otro lado se ha desarrollado una arquitectura de servicios en la plataforma de AWS para gestionar los datos recibidos y generar alarmas. Aprovechando así las características del entorno en la nube de Amazon escalabilidad, flexibilidad, reducción de costos, alta disponibilidad ,seguridad y velocidad de desarrollo.

Finalmente los mensajes de alarma son enviados al usuario vía Telegram, a través de un Bot que puede interactuar con el usuario intentando que sea de la forma más cercana posible.

**Palabras clave:** Internet de las cosas (IoT), agricultura inteligente, Arduino, Telegram Bot , AWS

Abstract

Due to the need of agriculture to improve the efficiency and productivity of crops, the concept of precision agriculture or smart agriculture arises.

In this work, a monitoring system for smart crops has been built. For this purpose, sensors and other low-cost elements such as Arduino and Raspberry Pi boards have been used to collect crop information.

On the other hand, a service architecture has been developed on the AWS platform to manage the data received and generate alarms. Thus taking advantage of the features of Amazon's cloud environment scalability, flexibility, cost reduction, high availability, security and speed of development.

Finally the alarm messages are sent to the user via Telegram, through a Bot that can interact with the user trying to be as close as possible.

Keywords: Internet of Things (IoT), smart agriculture, Arduino, Telegram Bot , AWS

Índice de contenidos

[1. Introducción 10](#_Toc137406623)

[1.1. Motivación 10](#_Toc137406624)

[1.2. Planteamiento del trabajo 11](#_Toc137406625)

[1.3. Estructura del trabajo 12](#_Toc137406626)

[2. Contexto y Estado del Arte 13](#_Toc137406627)

[2.1. Contexto 13](#_Toc137406628)

[2.1.1. Internet de las cosas (IoT) 13](#_Toc137406629)

[2.1.2. IoT en la Agricultura 14](#_Toc137406630)

[2.1.3. Sensores 14](#_Toc137406631)

[2.1.4. Hardware para la conexión de los sensores 15](#_Toc137406632)

[2.1.4.1. Arduino 15](#_Toc137406633)

[2.1.4.2. Raspberry Pi 17](#_Toc137406634)

[2.1.5. Plataformas IoT en la nube 18](#_Toc137406635)

[2.2. Estado del arte 20](#_Toc137406636)

[2.2.1. Sensores en la agricultura 20](#_Toc137406637)

[2.2.2. Modelos en la agricultura 22](#_Toc137406638)

[2.2.2.1. DSSAT 22](#_Toc137406639)

[2.2.2.2. Modelo APSIM 23](#_Toc137406640)

[2.2.2.3. Modelo STICS 24](#_Toc137406641)

[2.2.3. Productos en el mercado 24](#_Toc137406642)

[2.2.3.1. Arduino plant watering kit 24](#_Toc137406643)

[2.2.3.1. Netro Whisperer 25](#_Toc137406644)

[3. Objetivos y metodología de trabajo 26](#_Toc137406645)

[3.1. Objetivo general 26](#_Toc137406646)

[3.2. Objetivos específicos 26](#_Toc137406647)

[3.3. Metodología de trabajo 26](#_Toc137406648)

[4. SisGAC 27](#_Toc137406649)

[4.1. Análisis 27](#_Toc137406650)

[4.1.1. Caso de uso general 27](#_Toc137406651)

[4.2. Diseño 30](#_Toc137406652)

[4.2.1. Sensores 30](#_Toc137406653)

[4.2.1.1. Sensor DHT22 de temperatura y humedad relativa 31](#_Toc137406654)

[4.2.1.2. Sensor capacitivo de humedad del suelo v1.2 32](#_Toc137406655)

[4.2.1.3. Sensor BH1750FVI de Luz Ambiental 32](#_Toc137406656)

[4.2.2. Arduino 32](#_Toc137406657)

[4.2.3. Raspberry pi 33](#_Toc137406658)

[4.2.4. Protocolo MQTT 33](#_Toc137406659)

[4.2.5. Plataforma Amazon Web Services 33](#_Toc137406660)

[4.2.5.1. AWS IoT SiteWise 34](#_Toc137406661)

[4.2.5.2. AWS DynamoDB 34](#_Toc137406662)

[4.2.5.1. AWS Gateway API 36](#_Toc137406663)

[4.2.5.1. AWS Simple Queue Service 36](#_Toc137406664)

[4.2.5.2. AWS Lambda 36](#_Toc137406665)

[4.2.5.3. Proceso recepción y envió al Bot de Telegram 38](#_Toc137406666)

[4.2.5.1. AWS IoT Core 41](#_Toc137406667)

[4.2.5.2. AWS IoT Events 41](#_Toc137406668)

[4.2.1. Python 42](#_Toc137406669)

[4.2.2. GitHub 42](#_Toc137406670)

[4.2.3. Telegram Bot API 43](#_Toc137406671)

[4.3. Implementación 43](#_Toc137406672)

[4.3.1. Arduino 43](#_Toc137406673)

[4.3.2. Raspberry pi 44](#_Toc137406674)

[4.3.3. Plataforma Amazon Web Services 46](#_Toc137406675)

[4.3.3.1. AWS DynamoDB 46](#_Toc137406676)

[4.3.3.2. Configuración reglas de reenvío en AWS IoT Core 46](#_Toc137406677)

[4.3.3.3. AWS Lambdas 47](#_Toc137406678)

[4.3.3.4. Configuración estados AWS IoT Events 57](#_Toc137406679)

[4.3.4. Configuración Bot Telegram 60](#_Toc137406680)

[4.4. Pruebas 62](#_Toc137406681)

[5. Conclusiones y trabajo futuro 64](#_Toc137406682)

[5.1. Conclusiones del trabajo 64](#_Toc137406683)

[5.2. Líneas de trabajo futuro 64](#_Toc137406684)

[Referencias bibliográficas 65](#_Toc137406685)

[Anexo A. Título del anexo. 71](#_Toc137406686)

[Índice de acrónimos 72](#_Toc137406687)

[JSON 72](#_Toc137406688)

Índice de figuras

[Ilustración 1. Arquitectura general de IoT. (Kumar et al., 2019) 13](#_Toc137402978)

[Ilustración 2. Arquitectura de microservicios.(Gesteira et al., 2020) 20](#_Toc137402979)

[Ilustración 3. Modelo grados-días(Nestor Monroy et al., 2001) 22](#_Toc137402980)

[Ilustración 4. Software DSSAT versión 4.7 23](#_Toc137402981)

[Ilustración 5.Datos climáticos mensuales durante el ciclo del cultivo(Dreyli Maygualida Hidalgo Ramos, 2017) 23](#_Toc137402982)

[Ilustración 6. Dashboard de Arduino plant watering kit(Arduino Official Store, 2023) 25](#_Toc137402983)

[Ilustración 7. Caso de uso general 27](#_Toc137402984)

[Ilustración 8.Esquema arquitectura 30](#_Toc137402985)

[Ilustración 9. Diagrama Conexiones Arduino, Raspberry y sensores 30](#_Toc137402986)

[Ilustración 10. composición sensor DHT22 31](#_Toc137402987)

[Ilustración 11. Componentes del sensor de humedad 31](#_Toc137402988)

[Ilustración 12.Componentes del sensor de temperatura 31](#_Toc137402989)

[Ilustración 13. Sensor Capacitivo de humedad del suelo 32](#_Toc137402990)

[Ilustración 14. IoT SiteWise 34](#_Toc137402991)

[Ilustración 15. Ejemplo definición horario 35](#_Toc137402992)

[Ilustración 16. Secuencia de actualización limites horarios 38](#_Toc137402993)

[Ilustración 17. Proceso de recepción de peticiones y envió de respuestas 38](#_Toc137402994)

[Ilustración 18. Inicio petición Bot Telegram 38](#_Toc137402995)

[Ilustración 19.Mensaje respuesta Bot Telegram 38](#_Toc137402996)

[Ilustración 20. AWS IoT Architecture.(Amazon Web Services, 2023a) 41](#_Toc137402997)

[Ilustración 21. Ventana definición Lambda 53](#_Toc137402998)

[Ilustración 22.Formulario definición desencadenador 53](#_Toc137402999)

[Ilustración 23. API Gateway conectada con función Lambda 54](#_Toc137403000)

[Ilustración 24.AWS IoT Events. Modelos de detector 57](#_Toc137403001)

[Ilustración 25.Definición máquina de estados detectores 57](#_Toc137403002)

[Ilustración 26. Definición máquina de estados temporizadores 59](#_Toc137403003)

[Ilustración 27. Creación Bot con BotFather 60](#_Toc137403004)

[Ilustración 28. Confirmación estado Telegram Webhook 61](#_Toc137403005)

[Ilustración 29. Mensaje alerta 62](#_Toc137403006)

[Ilustración 30. Cambio estado temperatura baja 62](#_Toc137403007)

[Ilustración 31. Cambio estado lúmenes 62](#_Toc137403008)

[Ilustración 32. Resumen Bot Telegram con estados de alarma 63](#_Toc137403009)

[Ilustración 33.Mensajes cambio estado Telegram Bot 63](#_Toc137403010)

[Ilustración 34.Grafico SiteWise lúmenes cambio estado 63](#_Toc137403011)

[Ilustración 35. Gráfico SiteWise temperatura cambio estado 63](#_Toc137403012)

Índice de tablas

[Tabla 1. Especificaciones Arduino 15](#_Toc137403013)

[Tabla 2. Especificaciones Raspberry Pi 4 17](#_Toc137403014)

[Tabla 3. Comparativas plataformas IoT 19](#_Toc137403015)

[Tabla 4. Ejemplo documento tabla estados 35](#_Toc137403016)

[Tabla 5. Ejemplo documento tabla mediciones 36](#_Toc137403017)

[Tabla 6. Ejemplo de llamada a función Lambda actualizarLimites 36](#_Toc137403018)

[Tabla 7. Ejemplo de llamada a función Lambda actualizarValor 37](#_Toc137403019)

[Tabla 8. Ejemplo de llamada a función Lambda actualizarEstado 37](#_Toc137403020)

[Tabla 9. Ejemplo de llamada a función Lambda encolarMsgTelegram 37](#_Toc137403021)

[Tabla 10. Ejemplo JSON recibido por la función Lambda TelegramBot 40](#_Toc137403022)

[Tabla 11. Ejemplo JSON recibido en la función Lambda enviarMensajeTelegram 41](#_Toc137403023)

[Tabla 12. Código ejecutado en placa Arduino 44](#_Toc137403024)

[Tabla 13. Instrucciones instalación SDK de AWS para IoT 44](#_Toc137403025)

[Tabla 14. Código lectura datos del puerto serie y envió a AWS 45](#_Toc137403026)

[Tabla 15. Fichero definición servicio lectura 45](#_Toc137403027)

[Tabla 16. Comandos configuración servicio lectura 46](#_Toc137403028)

[Tabla 17. Ejemplo tabla estados 46](#_Toc137403029)

[Tabla 18. Codificación función Lambda actualizarLimites 48](#_Toc137403030)

[Tabla 19. Codificación función Lambda actualizarValor 48](#_Toc137403031)

[Tabla 20.. Codificación función Lambda actualizarEstado 49](#_Toc137403032)

[Tabla 21. Codificación función Lambda encloarMsgTelegram 50](#_Toc137403033)

[Tabla 22. Codificación función Lambda telegramBot 52](#_Toc137403034)

[Tabla 23. Codificación función Lambda enviarMensajeTelegramBot 54](#_Toc137403035)

# Introducción

## Motivación

El objetivo principal de este TFG es el de desarrollar una herramienta que ayude en el cuidado y monitorización de cultivos utilizando para ello sensores y hardware de bajo coste, servicios en la nube de AWS y dando feedback al usuario de forma sencilla.

De esta forma el usuario puede definir fácilmente un sistema de alarmas que le permitan hacer correcciones en los distintos parámetros que influyen en el cultivo para intentar tener el mayor rendimiento posible, minimizando el uso de agua y pesticidas y permitiendo conocer el estado del cultivo de forma remota de forma que se eviten desplazamientos.

## Planteamiento del trabajo

El sistema propuesto constaría de tres partes:

* Recolección y envió de la información de los sensores: Los sensores leerían ciertos valores del entorno del cultivo. Y los enviarían a un entorno en la nube.
* Procesado de los datos y creación de reglas para las alarmas. En un entorno cloud se recibirían los datos de los sensores, se procesarían con reglas y se enviarían las alarmas a los usuarios
* Envió de alarmas al usuario de forma sencilla. Para hacer fácil la visualización de las alarmas se propone usar un sistema con el que los usuarios están familiarizados como es el sistema de mensajería Telegram

## Estructura del trabajo

Primero se expondrá el contexto y el estado del arte de los distintos elementos del sistema, se comentará el uso del IoT en la agricultura, los distintos sensores y hardware que podemos usar así como las distintas plataformas de IoT existentes.

En un segundo bloque se definirán los objetivos del proyecto y la metodología de trabajo.

En el siguiente bloque analizaremos, diseñaremos, implementaremos y probaremos el Sistema SisGAC (Sistema de Gestión de Alarmas para Cultivos).

Por último expondremos las conclusiones y comentaremos posibles líneas de trabajo futuro.

# Contexto y Estado del Arte

## Contexto

### Internet de las cosas (IoT)

Este concepto surgió en 1999 (Auto-ID Center–Instituto tecnológico de Massachusetts MIT).

La internet de las cosas (IoT) es un paradigma emergente que permite la comunicación entre dispositivos electrónicos y sensores a través de internet como forma de facilitarnos la vida.(Kumar et al., 2019)

Generalmente son objetos de dos tipos, los sensores que permiten extraer datos del entorno físico y los actuadores que permiten realizar alguna opción por control remoto, por ejemplo un relé que enciende o apaga una bombilla.

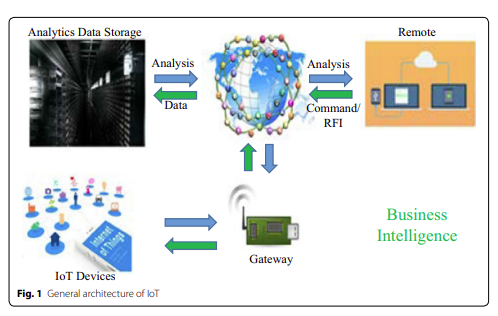


Ilustración . Arquitectura general de IoT. (Kumar et al., 2019)

Estos objetos se comunican entre si a través de distintos protocolos

Es aplicable en multitud de áreas como pueden ser en el ámbito medio ambiental, el industrial ,el médico , en transporte y en la agricultura.

El IoT consiste en una red de dispositivos físicos que contienen tecnología embebida para comunicarse con otros dispositivos .

La recolección de datos de estos dispositivos se puede usar para aprender del comportamiento de personas y procesos, reaccionar con acciones preventivas o correcciones o aumentar o transformar los procesos empresariales. (Espinosa et al., 2021)

### IoT en la Agricultura

Concretamente en el área de la agricultura podemos usar IoT para:

* **El monitoreo y la automatización**: La tecnología IoT permite monitorear y controlar aspectos del cultivo como la humedad, la luz solar, la temperatura, la calidad del suelo, el riego entre otros. La automatización de procesos para controlar estas variables puede ayudar a una mejor gestión de los recursos y para reducir costos.
* Para **ahorrar tiempo y energía**. Con los sensores y los dispositivos IoT se pueden monitorear los cultivos remotamente de forma que puedan ahorrar tiempo y esfuerzo.
* **Mejorar la calidad del producto**. El control constante de las condiciones de los cultivos ayuda a los agricultores a detectar problemas en etapas tempranas y corregir los posibles problemas
* **Optimizar el rendimiento**. Se puede detectar cuando es el momento óptimo para realizar la cosecha, cuando usar fertilizantes, realizar el riego y otros procesos.
* **Reducir el impacto ambiental**. Al optimizar el uso de fertilizantes y pesticidas reducimos es uso de productos químicos en la agricultura.

### Sensores

Para recolectar la información podemos usar una gran variedad de sensores aunque en el caso específico de la agricultura los más importantes podrían ser los siguientes:

* **Sensores de humedad del suelo**. Con ellos medimos la cantidad de agua en el suelo y nos permiten optimizar el riego.
* **Sensores de temperatura y humedad ambiental**. Con estos sensores podemos medir las condiciones ambientales y por ejemplo prevenir posibles enfermedades o situaciones de estrés en los cultivos.
* **Sensores de luz**. Con ellos medidos la intensidad de luz que están recibiendo los cultivos y podríamos ajustarla.
* **Sensores de PH y nutrientes**. Estos sensores miden el nivel del PH del suelo y la cantidad de nutrientes del suelo y permiten controlarlos.

### Hardware para la conexión de los sensores

Existen multitud de placas hardware de carácter general que se pueden usar para construir sistemas IoT. Las más importantes quizás sean las siguientes:

#### Arduino

Es una placa hardware opensource que incorpora un microcontrolador que se puede reprogramar usando el entorno de desarrollo Arduino IDE 2 y una serie de conectores que permiten establecer conexiones con sensores y actuadores de forma sencilla.(docs.arduino.cc, 2023b)

Las especificaciones son las siguientes:

|  |  |  |
| --- | --- | --- |
| **Board** | Name | Arduino UNO R3 |
| SKU | A000066 |
| **Microcontroller** | ATmega328P | |
| **USB connector** | USB-B | |
| **Pins** | Built-in LED Pin | 13 |
| Digital I/O Pins | 14 |
| Analog input pins | 6 |
| PWM pins | 6 |
| **Communication** | UART | Yes |
| I2C | Yes |
| SPI | Yes |
| **Power** | I/O Voltage | 5V |
| Input voltage (nominal) | 7-12V |
| DC Current per I/O Pin | 20 mA |
| Power Supply Connector | Barrel Plug |
| **Clock speed** | Main Processor | ATmega328P 16 MHz |
| USB-Serial Processor | ATmega16U2 16 MHz |
| **Memory** | ATmega328P | 2KB SRAM, 32KB FLASH, 1KB EEPROM |
| **Dimensions** | Weight | 25 g |
| Width | 53.4 mm |
| Length | 68.6 mm |

Tabla . Especificaciones Arduino(Arduino, 2023)

En la programación en la placa Arduino se usa un lenguaje basado en C++ llamado Sketch donde fundamentalmente se definen dos funciones principales, en la función setup se realiza la configuración de las conexiones, en loop se define el bucle que se ejecutara infinitamente.(docs.arduino.cc, 2023a)

#### Raspberry Pi

Es una computadora de placa única (SBC) de bajo costo y pequeño tamaño desarrollada por la Raspberry Pi Foundation en Reino Unido(Raspberry Pi Foundation, 2023). La primera versión fue lanzada en 2012 y desde entonces se han lanzado varias versiones.

En concreto las especificaciones de la versión 4 son las siguientes:

|  |  |
| --- | --- |
| **Processor** | Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz |
| **Memory** | 1GB LPDDR2 SDRAM |
| **Connectivity** | 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless  LAN, Bluetooth 4.2, BLE  Gigabit Ethernet over USB 2.0 (maximum throughput  300Mbps)  4 × USB 2.0 port |
| **Access** | Extended 40-pin GPIO header |
| **Video & sound** | 1 × full size HDMI MIPI DSI display port MIPI CSI camera port 4 pole stereo output and composite video port |
| **Multimedia** | H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics |
| **SD card support** | Micro SD format for loading operating system and data storage |
| **Input power** | 5V/2.5A DC via micro USB connector 5V DC via GPIO header Power over Ethernet (PoE)–enabled (requires separate PoE HAT) |
| **Environment** | Operating temperature, 0–50°C |

Tabla . Especificaciones Raspberry Pi 4 (Raspberry Pi Foundation, 2023)

En esta placa podemos instalar distintos sistemas operativos pero enfocado a IoT los principales podrían ser:

* **Raspbian**: es el sistema operativo oficial y está basado en Debian(Raspberry pi foundation, 2023)
* **Ubuntu MATE**: es una versión adaptada de Ubuntu para Raspberry Pi(Ubuntu, 2023)
* **Windows 10 IoT Core**: es una versión reducida de Windows 10 diseñada para dispositivos IoT.(Microsoft, 2023a)

### Plataformas IoT en la nube

Generalmente los datos recogidos en los sensores se envían a un backend donde se procesan, analizan, se generan informes, notificaciones, etc..

Actualmente existen varias plataformas como servicio (PaaS) que proporcionan un entorno de desarrollo e implantación completo en la nube. PaaS incluye infraestructura (servidores, almacenamiento y redes), pero también middleware, herramientas de desarrollo, servicios de inteligencia empresarial (BI), sistemas de administración de bases de datos, etc. PaaS está diseñado para sustentar el ciclo de vida completo de las aplicaciones web: compilación, pruebas, implementación, administración y actualización. (Microsoft, 2023c)

Las plataformas más importantes son las siguientes:

* **Amazon web services IoT**: Es la plataforma de IoT creada por Amazon y permite a los desarrolladores conectar dispositivos IoT, y controlarlos en la nube.(Amazon, 2023c)
* **Microsoft Azure IoT**: Es la alternativa de la empresa Microsoft, permite, como en el caso de Amazon conectar dispositivos a la nube y administrarlos. (Microsoft, 2023b)
* **IBM Watson IoT**: Esta plataforma IoT creada por IBM está enfocada en las aplicaciones en tiempo real.(IBM, 2023)
* **ThingsBoard**: Esta plataforma es de código abierto (ThingsBoard, 2023)
* **Kaa IoT**: Es otra plataforma de código abierto (Kaa, 2023)

Las características más importantes de estas plataformas pueden resumirse en la siguiente tabla:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Características** | **AWS IoT** | **Microsoft Azure IoT** | **IBM Watson IoT** | **ThingsBoard** | **Kaa IoT** |
| **Soporte para dispositivos y protocolos de IoT** | Sí | Sí | Sí | Sí | Sí |
| **Análisis de datos y visualización de datos** | Sí | Sí | Sí | Sí | Sí |
| **Integración con servicios de inteligencia artificial y machine learning** | Sí | Sí | Sí | No | Sí |
| **Seguridad de extremo a extremo** | Sí | Sí | Sí | Sí | Sí |
| **Monitoreo y administración de dispositivos** | Sí | Sí | Sí | Sí | Sí |
| **Escalabilidad** | Sí | Sí | Sí | Sí | Sí |
| **Compatibilidad con múltiples nubes** | Sí | Sí | Sí | No | Sí |
| **Precio** | Pago por uso | Pago por uso | Pago por uso | Código abierto | Código abierto |

Tabla . Comparativas plataformas IoT

## Estado del arte

Según el ministerio de agricultura, pesca y alimentación (Ministerio de Agricultura, 2023)los principales retos a los que se enfrenta el regadío español son:

* **Producir más con menos recursos** y con menor impacto ambiental;
* **El cambio climático**, causante de variaciones de temperatura y distribución de lluvias irregulares e impredecibles que suponen pérdidas cuantiosas para la agricultura;
* **El déficit hídrico** en algunas regiones de España donde la agricultura es el principal motor de la actividad económica (precipitaciones inferiores a 200 mm al año);
* El uso adecuado de los **recursos hídricos** y la **reducción de la contaminación** por lixiviados de los nutrientes aportados a los cultivos;
* **La mejora de las condiciones de vida y de trabajo** de los agricultores, para contribuir a revertir el fenómeno de la despoblación del medio rural y fomentar el relevo generacional.

### Sensores en la agricultura

Se puede encontrar mucha literatura sobre la utilización de sensores en la agricultura para intentar mejorar el rendimiento de los cultivos.

En el artículo (Sunehra, 2019) Se utiliza una placa Arduino para la lectura de los valores de los sensores y estos datos se envían a una raspberry pi a través de un transceptor wifi. La placa raspberry cuenta con un servidor web que permite al usuario visualizar los datos.

En el TFG (Gesteira et al., 2020) se desarrolla una arquitectura de microservicios para una red de sensores IoT sobre Arduino.

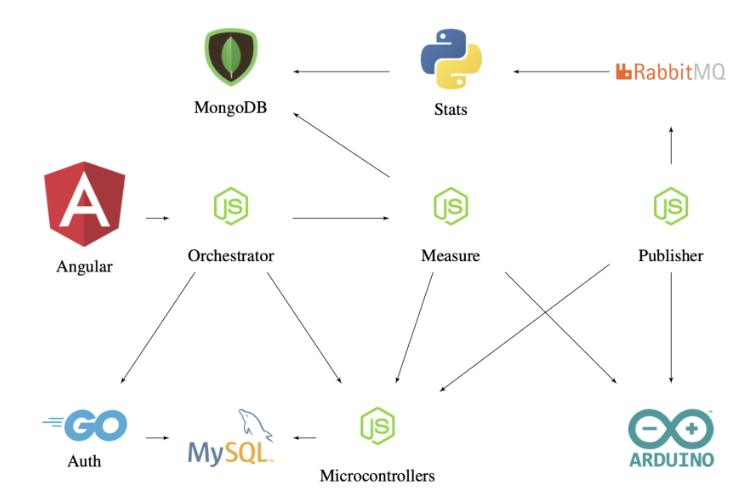


Ilustración . Arquitectura de microservicios.(Gesteira et al., 2020)

En el artículo (J. Cedeño et al., 2014) se explica como el uso de sensores puede ayudar a aprovechar el agua, a reducir el consumo de fertilizantes, al control de plagas, al monitoreo de animales y pastos y para el control de los invernaderos.

En (Juan Núñez et al., 2018)se propone un diseño de sistema de agricultura de precisión para tomateras llamado Smartnode. Creando una plataforma de hardware basada en la placa Arduino ATmega2560 para adquirir, enviar y recibir los datos de los sensores de temperatura, luminosidad y humedad del suelo.

En este caso se usa una topología de red Zigbee basado en un sistema WSN (Wireless Sensor Network)

Para la parte software se usa un servidor local para crear una aplicación en java y se usa un servidor de datos MySQL y por otra parte se crea una aplicación móvil para visualizar los datos en campo.

En (Santos-Olmo, 2022) se propone un sistema similar llamado Smartceta que conecta sensores de humedad de suelo capacitivo, un sensor de humedad y temperatura DHT11 y un reloj de tiempo real a una placa a una placa ESP32. Esta placa envía la información a la plataforma ThingsBoard donde se crean una serie de reglas de alarmas que se pueden usar para enviar mensajes a un Bot de Telegram.

En el artículo (Baggio, 2005) se describe un experimento donde se genera una red de sensores de temperatura, humedad, luminosidad, presión del aire, fuerza del viento y dirección y humedad del suelo en una plantación de patatas para detección temprana de posibles enfermedades en la planta y avisar al agricultor cuando es necesario el uso de fungicidas

En el TFM (Silva & Antonio, 2016) Se desarrolla un sistema IoT autónomo que permita horarios de riego y calcular las necesidades de agua para ahorrar en su consumo. Para ello primero se identifican los aspectos importantes para tener en cuenta durante el cultivo, como son la evapotranspiración, la Evaporación, la transpiración, las variables climáticas y los métodos para calcularos.

Además se hace una reflexión sobre cómo se podría programar el riego automático utilizando inteligencia artificial. En este caso se usa sensores de humedad , temperatura y humedad de suelo en varias configuraciones junto con un sistema de riego automático conectados a través de Servicios Web de tipo REST y un sistema de Machine Learning.

En (Dennys Francisco Salazar Domínguez, 2022) Se propone la utilización de una placa Arduino YUN para gestionar los actuadores de un invernadero. En este caso se usan sensores de humedad y temperatura DH11 , Sensores de CO2 y de humedad de suelo para medir los valores importantes del invernadero y ventiladores tiras de led y bombas de agua y un sistema de calefacción como actuadores para controlar el ambiente del invernadero.

### Modelos en la agricultura

Los modelos simulan los procesos de crecimiento de la planta, teniendo en cuenta las características del suelo, el clima y la propia planta para tener una descripción continua del desarrollo de la planta.

Por ejemplo en (José & Arboleda, 2009) se realiza un estudio para determinar el comportamiento en el crecimiento de algunas variedades de rosas y después se desarrolla varios modelos de proyecciones para pronosticar la producción de cada variante en función por ejemplo de la temperatura promedio de se registre en la temporada.

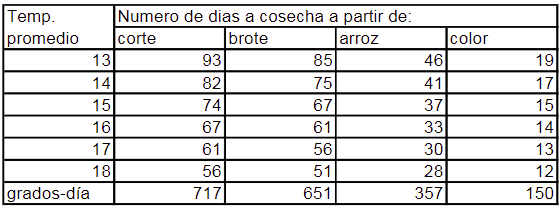


Ilustración . Modelo grados-días(Nestor Monroy et al., 2001)

#### DSSAT

Decision Support System for Agrotechnology Transfer (DSSAT) es una aplicación informática que comprende modelos de simulación para más de 42 cultivos, herramientas para el uso de modelos , bases de datos de suelo, clima, gestión de cultivos y datos experimentales. Los modelos simulan el desarrollo, crecimiento y rendimiento en función de la dinámica suelo-planta-atmosfera.(Hoogenboom et al., 2019)

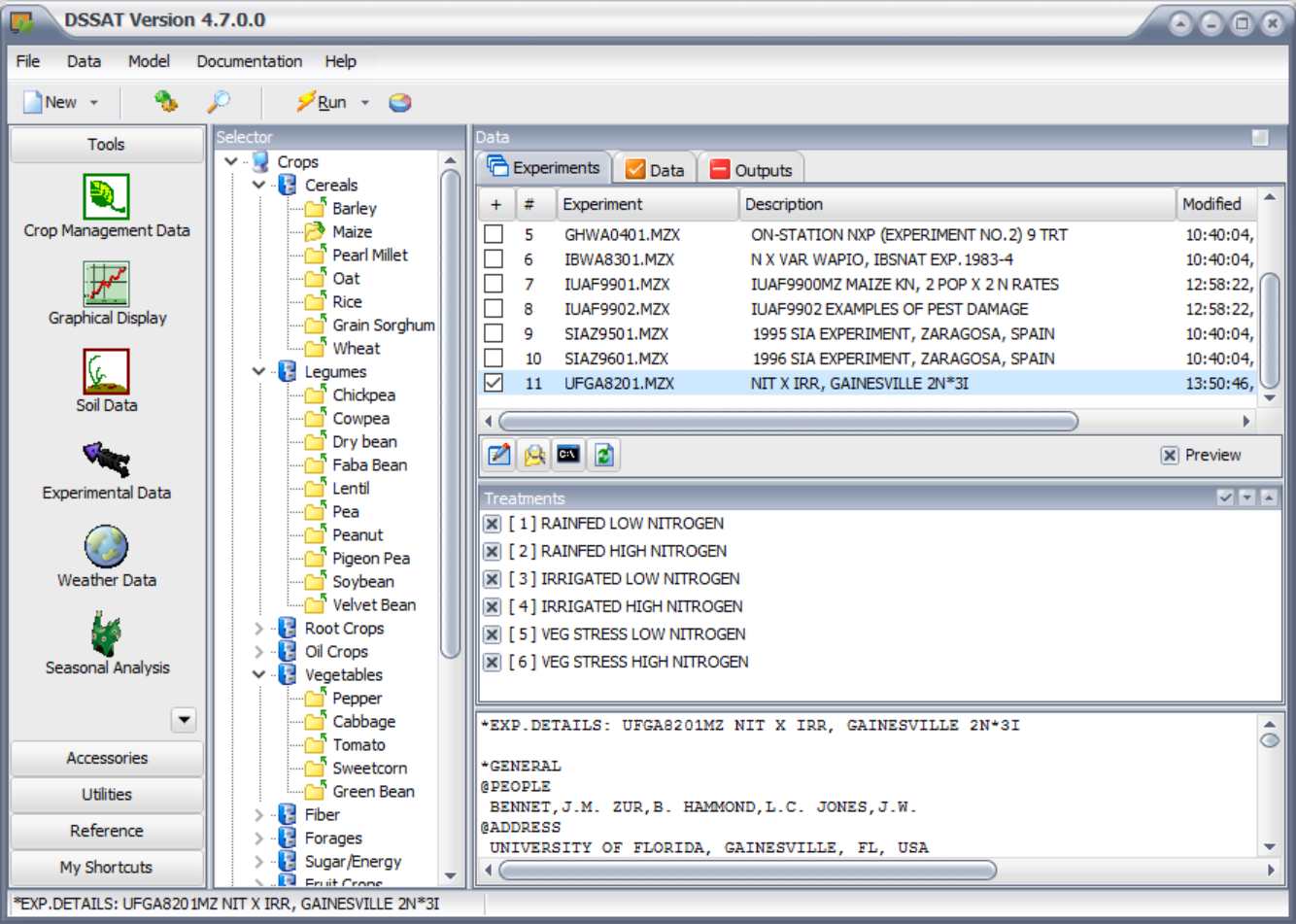


Ilustración . Software DSSAT versión 4.7

En (Dreyli Maygualida Hidalgo Ramos, 2017) hacen una validación del modelo CERES – Maize para la variedad de maíz ASGROW 7573 simulando el desarrollo y crecimiento del cultivo.

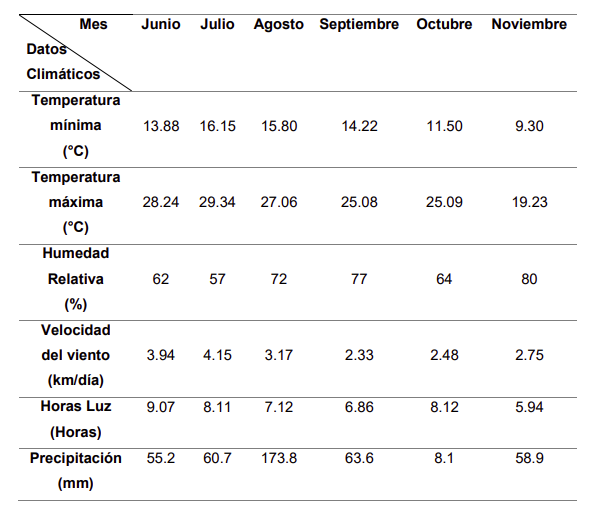


Ilustración .Datos climáticos mensuales durante el ciclo del cultivo(Dreyli Maygualida Hidalgo Ramos, 2017)

#### Modelo APSIM

The Agricultural Production Systems sIMulator (APSIM) es un modelo desarrollado para simular procesos biofísicos en sistemas agrícolas especialmente en lo que se refiere a los resultado económicos y ecológicos.

Esta desarrollada por una fundación cuyos socios son la Universidad de Queensland, la universidad de Southern Queensland y el departamento de agricultura y pesca de Nueva Zelanda, La Universidad de Iowa y Plant & Food Research. La iniciativa promueve el desarrollo y el uso de modelos científicos y la infraestructura del software APSIM fue creada en 2007.(APSIM, 2023)

Este modelo de simulación basado en procesos de centra en la interacción de varios factores que influyen en los cultivos como son el clima, el suelo, el agua, ect. Tiene un enfoque modular, cada proceso agrícola se representa como un módulo individual que se pueden combinar y reorganizar para adaptarse a los sistemas de cultivo y a las condiciones ambientales.

#### Modelo STICS

Es otro modelo creado por el INRA, el instituto francés para la investigación en la agricultura.(INRAE, 2023) Tiene modelos para unas 20 especies de plantas.

El modelo tiene un enfoque de equilibrio de masa, los flujos de nutrientes y agua dentro y fuera del sistema agrícola se equilibran en todo momento. Se usan ecuaciones matemáticas para simular el crecimiento y desarrollo de los cultivos y la respuesta a distintos factores ambientales.(Brisson et al., 2003)

### Productos en el mercado

También existen algunos productos actualmente en el mercado que sirven para monitorizar cultivos.

#### Arduino plant watering kit

La más básica podría ser el kit “Arduino Plant Watering Kit” (Arduino S.r.l, 2023) que está compuesto por una placa Arduino, un sensor de humedad del suelo y una pequeña bomba de agua que permite el riego. Además cuenta con [Arduino IoT Cloud](https://cloud.arduino.cc/?_gl=1*ab92w*_ga*NzE2MDcwODgxLjE2NzkxNjQ0NzQ.*_ga_NEXN8H46L5*MTY4MTU5NjI1NS4xMy4xLjE2ODE1OTYyNjkuMC4wLjA.) con la que se pueden crear horarios automáticos de riego, monitorizar el nivel del sensor y controlar remotamente el riego a través de una aplicación web.

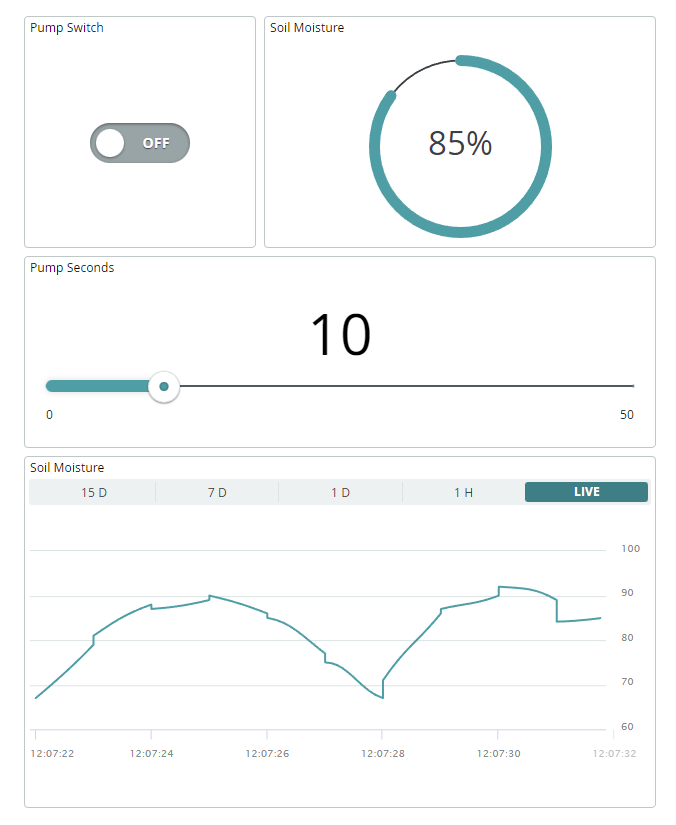


Ilustración . Dashboard de Arduino plant watering kit(Arduino Official Store, 2023)

#### Netro Whisperer

Es otro producto comercial de la marca Netro Inc. Cuenta con sensores de luz ambiental, temperatura y humedad del suelo. además es inalámbrico y se carga con energía solar.(Netro, 2023)

Incluye una aplicación para comprobar los datos de los sensores en el móvil.

# Objetivos y metodología de trabajo

## Objetivo general

El objetivo de este TFG es construir un sistema de gestión de alarmas para cultivos llamado SisGAC con las siguientes características:

* El usuario podrá interactuar con el sistema a través de un Bot de Telegram
* El sistema estará operativo siempre
* El usuario podrá construir reglas para generar alarmas que avisen de posibles problemas

## Objetivos específicos

* Modelado de las historias de usuario
* Creación de la infraestructura
  + Lectura de datos de los sensores y envió al entorno AWS
  + Desarrollo del sistema AWS
  + Desarrollo del Bot de Telegram
* Despliegue de la solución
* Puesta en marcha

## Metodología de trabajo

Para el desarrollo del sistema se usará un modelo en cascada ya que solo habrá un desarrollador y no se prevén muchos cambios durante el desarrollo. Constará de las siguientes fases:

* Análisis de requisitos: en esta fase definiremos los requisitos, crearemos los casos de uso y detallaremos las necesidades y objetivos del proyecto
* Diseño: En base a los requisitos definiremos la arquitectura, la estructura y lógica del software
* Implementación: Durante esta fase, se codificarán los distintos módulos software definidos en la etapa anterior.
* Pruebas: En esta fase se comprobará que el software cumple con los requisitos definidos durante el análisis

# SisGAC

## Análisis

### Caso de uso general

El siguiente diagrama describe el caso de uso general que pretende desarrollarse.

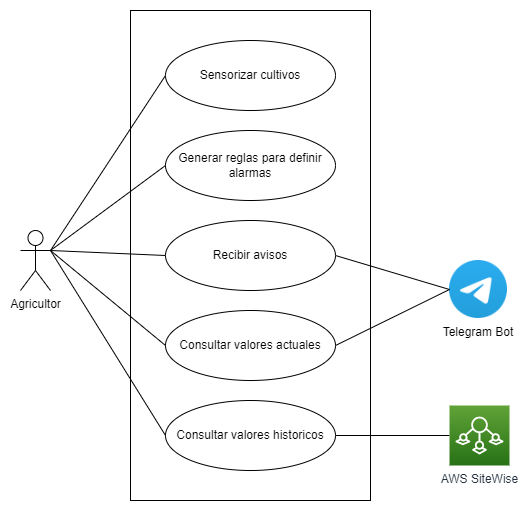


Ilustración . Caso de uso general

* Los cultivos tendrán que ser sensorizados con sensores de bajo coste.
* Debe ser posible configurar las reglas que generaran las alarmas
* El usuario debe recibir los avisos a través la aplicación de mensajería Telegram
* El usuario debe poder consultar los valores actuales a través de la aplicación de mensajería Telegram.
* El usuario debe poder consultar los datos históricos de los valores leídos.

La descripción de los casos de uso podría ser:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Caso de Uso** | Añadir sensores a los cultivos | | | | REQ001 | |
| **Actores** | Agricultor | | | | | |
| **Tipo** | Funcional | | | | | |
| **Autor** | Raúl | **Fecha** | 17/05/2023 | **Versión** | | 1.0 |
| **Resumen** | Leer los valores de humedad, temperatura, radiación y humedad del suelo | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Caso de Uso** | Generar reglas para definir alarmas | | | | REQ002 | |
| **Actores** | Agricultor | | | | | |
| **Tipo** | Funcional | | | | | |
| **Autor** | Raúl | **Fecha** | 17/05/2023 | **Versión** | | 1.0 |
| **Resumen** | Crear reglas para gestionar las alarmas | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Caso de Uso** | Recibir avisos | | | | REQ003 | |
| **Actores** | Agricultor | | | | | |
| **Tipo** | Funcional | | | | | |
| **Autor** | Raúl | **Fecha** | 17/05/2023 | **Versión** | | 1.0 |
| **Resumen** | Cuando se produzcan las alarmas se enviarán al usuario a través de un Bot de Telegram | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Caso de Uso** | Consultar valores históricos | | | | REQ004 | |
| **Actores** | Agricultor | | | | | |
| **Tipo** | Funcional | | | | | |
| **Autor** | Raúl | **Fecha** | 17/05/2023 | **Versión** | | 1.0 |
| **Resumen** | Se podrán consultar históricos de los valores registrados por los sensores | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Caso de Uso** | Consultar valores actuales | | | | 005 | |
| **Actores** | Agricultor | | | | | |
| **Tipo** | Funcional | | | | | |
| **Autor** | Raúl | **Fecha** | 17/05/2023 | **Versión** | | 1.0 |
| **Resumen** | Se podrán consultar los valores actuales a través del bot de Telegram | | | | | |

## Diseño

La arquitectura para la solución propuesta tiene los siguientes elementos:

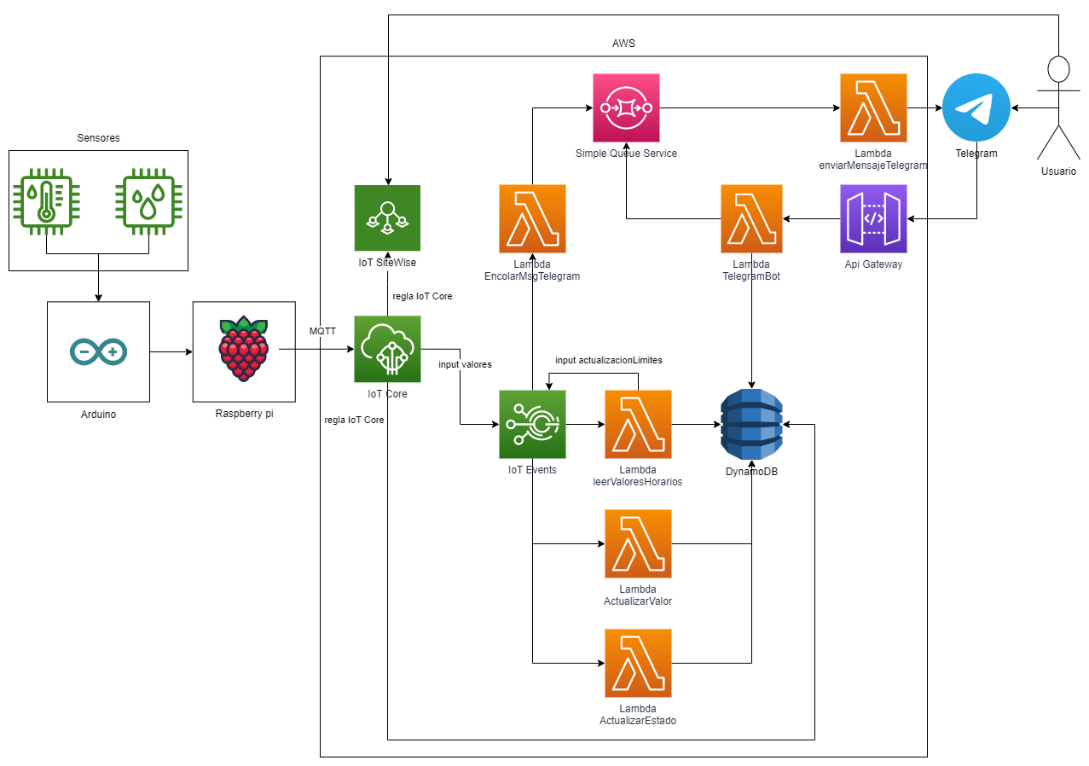


Ilustración .Esquema arquitectura

### Sensores

Los sensores se conectarán de la siguiente manera:

Diagrama

Descripción generada automáticamente

Ilustración . Diagrama Conexiones Arduino, Raspberry y sensores

#### Sensor DHT22 de temperatura y humedad relativa

Este sensor mide la temperatura y la humedad del ambiente y su precio es muy bajo. Está compuesto por dos elementos:

|  |  |
| --- | --- |
| Ilustración . composición sensor DHT22 | Ilustración . Componentes del sensor de humedad |
| Ilustración .Componentes del sensor de temperatura |

El sensor de humedad tiene un rango entre 0% y 100% con una precisión del 2-5%

El sensor de temperatura mide entre -40º y 125º con una precisión de +-0.5 ºC

Tiene una ratio de muestreo de 0.5Hz (cada 2 segundos).En el artículo (Bogdan Mihai, 2016) se explica cómo se usa este sensor para medir la temperatura y la humedad.

#### Sensor capacitivo de humedad del suelo v1.2



Ilustración . Sensor Capacitivo de humedad del suelo

Este sensor mide la humedad relativa gracias a que está compuesto por un capacitador que se ve afectado por el agua.(Placidi et al., 2020) su precio es muy bajo en comparación con otros sensores con mayor precisión pero su precisión es suficiente para la mayoría de las aplicaciones. (Persson et al., 2023)

Este sensor necesita calibrarse en un entorno seco y húmedo para calcular la humedad relativa ,además hay que tener en cuenta que puede verse afectado por el tipo de suelo (Kulmány et al., 2022)

#### Sensor BH1750FVI de Luz Ambiental

Un dato importante relativo al cultivo de plantas es la radiación.(Carrasco-Ríos, 2009) En concreto la radiación PAR que mide los espectros de radiación que afectan a las plantas. Como los sensores de radiación PAR son muy costosos para este TFG usaremos un sensor de luz ambiental para hacer una estimación de la cantidad de PAR que llega a la planta. (Beyaz & Gül, 2022)

Conociendo la fuente emisora, en este caso el sol, se puede estimar cual es la radiación PAR que nos llega en función de la luz que recibimos.

### Arduino

Todos los sensores se conectarán a la placa Arduino que cada minuto leerá los valores de cada sensor, generará un JSON con la información leída y enviará los datos a través del puerto serie.

### Raspberry pi

Debido a las limitaciones de la placa Arduino en cuanto a conectividad, capacidad de ejecución y almacenamiento usaremos esta placa para conectarnos a internet y ejecutar en ella un servicio que enviará los datos a la plataforma de AWS que procesará los datos.

Las dos placas se comunicarán a través del puerto serie donde Arduino escribirá cada minuto los valores de los sensores.

A estos datos que se reciben se le añade la hora actual en formato internacional (con ello evitamos tener que añadir a Arduino un CTR para añadir la hora de la lectura de los datos) , luego envía a través del Protocolo MQTT la información al servicio en la nube AWS IoT Core.

Además será necesario instalar el SDK de AWS IoT para tener las librerías necesarias para comunicarse con AWS.

### Protocolo MQTT

MQTT es un protocolo de mensajería estándar OASIS para el Internet de las Cosas (IoT). Está diseñado como un transporte de mensajería de publicación/suscripción extremadamente ligera que resulta ideal para conectar dispositivos remotos con una huella de código pequeña y un ancho de banda de red mínimo. En la actualidad, MQTT se utiliza en una amplia variedad de sectores, como automoción, fabricación, telecomunicaciones, petróleo y gas, etc.(mqtt.org, 2023)

### Plataforma Amazon Web Services

AWS es una plataforma creada por Amazon (Mathew, 2014)que permite crear servicios en la nube. Ofrece más de 200 servicios integrales de centros de datos a nivel global. Funcionalmente ofrece una gran cantidad de servicios y características desde almacenamiento y bases de datos hasta tecnologías emergentes como aprendizaje automático e inteligencia artificial.(Amazon Web Services, 2023c)

En nuestro caso nos centraremos en los servicios del internet de las cosas (IoT)(Amazon Web Services, 2023b), en concreto en los servicios IoT Device SDKs , IoT core, IoT SiteWise e IoT Events. Pero también utilizaremos los servicios AWS Lambda, AWS DynamoDB y AWS Simple Queue Service

#### AWS IoT SiteWise

Es un servicio administrado que permite la recopilación, la organización y el análisis de datos de equipos industriales.(Amazon, 2023d) .En nuestro caso lo usaremos para poder visualizar los datos históricos de las variables.



Ilustración . IoT SiteWise

#### AWS DynamoDB

Este servicio proporciona una base de datos NoSQL que admite documentos y clave valor. Diseñada para aplicaciones de alto rendimiento. En nuestro caso vamos a definir 3 tablas en él.

* Tabla estados: En esta tabla registraremos los datos de los sensores que nos interesan para que nuestro Bot pueda responder a las peticiones de información. Los documentos tienen el siguiente formato:

|  |
| --- |
| {  "id": "valoresActuales",  "humedad": {  "estado": "ok",  "maximo": **80**,  "minimo": **30**,  "valorActual": **35.4**  },  "humedadSuelo": {  "estado": "ok",  "maximo": **80**,  "minimo": **40**,  "valorActual": **49**  },  "lux": {  "estado": "porDebajo",  "maximo": **5000**,  "minimo": **1000**,  "valorActual": **1.666667**  },  "temperatura": {  "estado": "ok",  "maximo": **30**,  "minimo": **20**,  "valorActual": **24.4**  }  } |

Tabla . Ejemplo documento tabla estados

* Tabla horaria: En ella se almacenan los datos horarios esperados para cada sensor. Podríamos definir por ejemplo los valores esperados del sensor de iluminación cada hora.

|  |
| --- |
| {  "id": "lux",  "horario": {  "hora0": **0**,  "hora1": **0**,  "hora2": **0**,  "hora3": **0**,  "hora4": **0**,  "hora5": **0**,  "hora6": **0**,  "hora7": **0**,  "hora8": **1000**,  "hora9": **3000**  "hora10": **5000**,  "hora11": **6500**,  "hora12": **7500**,  "hora13": **8500**,  "hora14": **5500**,  "hora15": **4500**,  "hora16": **4000**,  "hora17": **4000**,  "hora18": **3500**,  "hora19": **3000**,  "hora20": **1000**,  "hora21": **500**,  "hora22": **0**,  "hora23": **0**  }  } |

Ilustración . Ejemplo definición horario

* Tabla mediciones: esta tabla almacena el histórico de datos recibido de los sensores. Los documentos almacenados tienen el siguiente formato:

|  |
| --- |
| {  "fecha": "2023-04-28T02:33:10.000Z",  "humedad": **40.7**,  "humedadSuelo": **72**,  "lux": **4270**,  "temperatura": **24.4**  } |

Tabla . Ejemplo documento tabla mediciones

#### AWS Gateway API

Con este servicio crearemos una API donde llegaran los mensajes del bot de Telegram, cada vez que llegue un mensaje este se reenviara a una función Lambda que la procesara, generará una respuesta y la enviará a la cola de mensajes para su posterior envió.

#### AWS Simple Queue Service

Este servicio permite crear colas(Amazon, 2023e), de tipo FIFO en nuestro caso donde se irán encolando los mensajes que se quieren enviar al Bot de Telegram desde las distintas Lambdas. De esta forma nos aseguramos de que los mensajes lleguen de forma ordenada.

#### AWS Lambda

Este servicio nos permite crear funciones dentro de la infraestructura de alta disponibilidad de Amazon sin preocuparse de administrar los recursos informáticos, el mantenimiento del servidor o el sistema operativo.(Amazon, 2023b)

Nos ofrece varios lenguajes de programación para crear las funciones lambda. Nosotros usaremos Python.

“Python es un lenguaje de programación potente y fácil de aprender. Tiene estructuras de datos de alto nivel eficientes y un simple pero efectivo sistema de programación orientado a objetos. La elegante sintaxis de Python y su tipado dinámico, junto a su naturaleza interpretada lo convierten en un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en muchas áreas, para la mayoría de las plataformas.”(Python Software Foundation, 2023)

Este lenguaje además tiene un gran número de librerías que usaremos para la programación de las funciones Lambda.

Se han creado las siguientes:

* actualizarLimites: esta función actualiza los valores mínimos y máximos de una variable de la tabla estados de DynamoDB. Recibe por parámetros un JSON con el formato:

|  |
| --- |
| {  "maximo": "100",  "minimo": "0",  "variable": "humedadSuelo"  } |

Tabla . Ejemplo de llamada a función Lambda actualizarLimites

* actualizarValor: esta función actualiza el valor actual (el ultimo leído) de una variable de la tabla estados de DynamoDB. Recibe por parámetros un JSON con el formato:

|  |
| --- |
| {  "valor": "26",  "variable": "temperatura"  } |

Tabla . Ejemplo de llamada a función Lambda actualizarValor

* actualizarEstado: esta función actualiza el estado de una variable de la tabla estados de DynamoDB. Recibe por parámetros un JSON con el formato

|  |
| --- |
| {  "variable": "humedadSuelo",  "estado": "porEncima"  } |

Tabla . Ejemplo de llamada a función Lambda actualizarEstado

* encolarMsgTelegram: esta función recibe un mensaje y lo encola en el servicio Simple Queue Service de Amazon, recibe por parámetro un JSON con el formato

|  |
| --- |
| {  "mensaje": "mensaje de respuesta"  } |

Tabla . Ejemplo de llamada a función Lambda encolarMsgTelegram

* procesarTelegramBot: Esta función recibe los mensajes del Bot de Telegram a través del AWS Gateway API, los procesa y encola las respuestas.
* enviarMensajeTelegramBot: recibe peticiones para procesar los elementos del servicio AWS Simple Queue Service
* leerValoresHorarios: Esta función lee los valores esperados para la hora actual de la tabla horario, calcula los máximos y mínimos con un valor margen que recibe por parámetro, llama a la lambda actualizarLimites para actualizar la información en la tabla de estados y genera un input para que IoT Events pueda actualizar la máquina de estados con los nuevos valores limite.

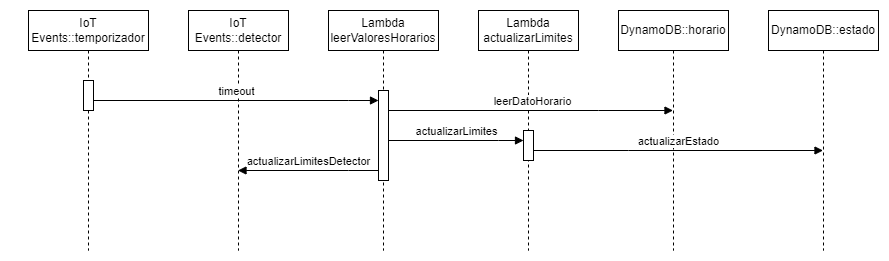


Ilustración . Secuencia de actualización limites horarios

#### Proceso recepción y envió al Bot de Telegram

|  |  |
| --- | --- |
| Diagrama  Descripción generada automáticamente  Ilustración . Proceso de recepción de peticiones y envió de respuestas | |
| Patrón de fondo  Descripción generada automáticamente  Ilustración . Inicio petición Bot Telegram | Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto  Descripción generada automáticamente  Ilustración .Mensaje respuesta Bot Telegram |

El proceso se inicia con la recepción de un mensaje del cliente de Telegram.

Al escribir el mensaje Telegram nos enviará la petición POST a nuestro Gateway API este lo capturará y reenviará a nuestra función Lambda TelegramBot que recibirá el siguiente mensaje JSON:

|  |
| --- |
| {  "version": "1.0",  "resource": "/telegramBot",  "path": "/default/telegramBot",  "httpMethod": "POST",  "headers": {  "Content-Length": "309",  "Content-Type": "application/json",  "Host": "amt44iczs3.execute-api.eu-central-1.amazonaws.com",  "X-Amzn-Trace-Id": "Root=1-645fcdf3-61d7eefa4f3e3af46b2e7090",  "X-Forwarded-For": "91.108.6.47",  "X-Forwarded-Port": "443",  "X-Forwarded-Proto": "https",  "accept-encoding": "gzip, deflate"  },  "multiValueHeaders": {  "Content-Length": [  "309"  ],  "Content-Type": [  "application/json"  ],  "Host": [  "amt44iczs3.execute-api.eu-central-1.amazonaws.com"  ],  "X-Amzn-Trace-Id": [  "Root=1-645fcdf3-61d7eefa4f3e3af46b2e7090"  ],  "X-Forwarded-For": [  "91.108.6.47"  ],  "X-Forwarded-Port": [  "443"  ],  "X-Forwarded-Proto": [  "https"  ],  "accept-encoding": [  "gzip, deflate"  ]  },  "queryStringParameters": **null**,  "multiValueQueryStringParameters": **null**,  "requestContext": {  "accountId": "329640145453",  "apiId": "amt44iczs3",  "domainName": "amt44iczs3.execute-api.eu-central-1.amazonaws.com",  "domainPrefix": "amt44iczs3",  "extendedRequestId": "E30eJh\_aFiAEMcg=",  "httpMethod": "POST",  "identity": {  "accessKey": **null**,  "accountId": **null**,  "caller": **null**,  "cognitoAmr": **null**,  "cognitoAuthenticationProvider": **null**,  "cognitoAuthenticationType": **null**,  "cognitoIdentityId": **null**,  "cognitoIdentityPoolId": **null**,  "principalOrgId": **null**,  "sourceIp": "91.108.6.47",  "user": **null**,  "userAgent": "",  "userArn": **null**  },  "path": "/default/telegramBot",  "protocol": "HTTP/1.1",  "requestId": "E30eJh\_aFiAEMcg=",  "requestTime": "13/May/2023:17:50:43 +0000",  "requestTimeEpoch": **1684000243849**,  "resourceId": "ANY /telegramBot",  "resourcePath": "/telegramBot",  "stage": "default"  },  "pathParameters": **null**,  "stageVariables": **null**,  **"body": "{\"update\_id\":473525447,\"message\":{\"message\_id\":119,\"from\":{\"id\":5129996758,\"is\_bot\":false,\"first\_name\":\"Raul\",\"last\_name\":\"Garcia\",\"username\":\"orgbo\",\"language\_code\":\"es\"},\"chat\":{\"id\":5129996758,\"first\_name\":\"Raul\",\"last\_name\":\"Garcia\",\"username\":\"orgbo\",\"type\":\"private\"},\"date\":1683994390,\"text\":\"resumen\"}}"**,  "isBase64Encoded": **false**  } |

Tabla . Ejemplo JSON recibido por la función Lambda TelegramBot

Procesara el campo ”text” de “body” y generara un mensaje de respuesta que será enviado al servicio SQS para ser encolado. Cada vez que el servicio de cola detecta elementos ejecuta la función Lambda “enviarMensajeTelegram” para ser procesados con el este JSON como parámetro:

|  |
| --- |
| {  "Records": [  {  "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",  "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",  **"body": "{\"mensaje\": \"El estado de la humedad está por debajo de los limites \\ud83d\\ude22 \\n el valor es 23 y debería ser superior a 30 \"}",**  "attributes": {  "ApproximateReceiveCount": "1",  "SentTimestamp": "1545082649183",  "SenderId": "AIDAIENQZJOLO23YVJ4VO",  "ApproximateFirstReceiveTimestamp": "1545082649185"  },  "messageAttributes": {},  "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",  "eventSource": "aws:sqs",  "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",  "awsRegion": "us-east-2"  }  ]  } |

Tabla . Ejemplo JSON recibido en la función Lambda enviarMensajeTelegram

La función Lambda lee el campo mensaje del elemento “body”, lo envía como respuesta al Bot de Telegram , después se elimina de la cola.

#### AWS IoT Core

Es un servicio en la nube que permite interactuar de forma segura los dispositivos IoT con otros dispositivos u otras aplicaciones, en nuestro caso los datos de input serán los valores de los sensores cada minuto que llegan en formato JSON.


                    AWS IoT architecture
                

Ilustración . AWS IoT Architecture.(Amazon Web Services, 2023a)

Se crearán 3 reglas en este servicio de forma que cuando lleguen los datos primero se envíen para ser almacenados en la tabla mediciones de DynamoDB, después se enviarán al servicio Events para ser procesados y ejecutar las reglas que se hayan definido. Por último se enviarán al servicio IoT SiteWise para poder visualizar los valores históricos.

#### AWS IoT Events

Este servicio detecta y responde a eventos de sensores y aplicaciones IoT. Los eventos son patrones que identifican ciertas situaciones. Como detectores de movimiento que activan cámaras de seguridad o luces. AWS IoT Events monitoriza continuamente los datos de sensores y aplicaciones de IoT, y se integra con otros servicios.

En nuestro caso tendremos dos entradas para este servicio, por un lado estarán los datos de las lecturas de los sensores con la siguiente información:

|  |
| --- |
| {  "fecha": "2023-04-28T02:33:10.000Z",  "humedad": **40.7**,  "humedadSuelo": **72**,  "lux": **4270**,  "temperatura": **24.4**  } |

Por otro lado tendremos las entradas de las actualizaciones de los valores límites con la información:

|  |
| --- |
| {  "lux": {  "maximo": 2000,  "minimo": 0  }  } |

Dentro de este servicio se definirán para cada propiedad al menos una máquina de estado que gestionara las entradas y ejecutara los eventos que se definan para mostrar las alarmas.

## Implementación

### Arduino

El código ejecutado en la placa Arduino es el siguiente:

|  |
| --- |
| #include <ArduinoJson.h>  #include <SoftwareSerial.h>  #include <SimpleDHT.h>  #include <BH1750.h>  BH1750 lightMeter;  **int** pinDHT22 = **2**;  SimpleDHT22 **dht22**(pinDHT22);  **int** pinHumedadSuelo = A0; // Puerto analógico A0 para la lectura del fotosensor  **unsigned** **long** tiempoPaso = **60000**; //milisegundos entre paso  **const** **int** seco = **800**;  **const** **int** humedo = **370**;  **void** **setup**() {  Serial.begin(**9600**);  Wire.begin();  lightMeter.begin();  }  **void** **loop**()  {  **int** humedadSuelo = map(analogRead(A0),humedo,seco,**100**,**0**);  **float** lux = lightMeter.readLightLevel();  **float** temperatura = **0**;  **float** humedad = **0**;  dht22.read2(&temperatura, &humedad, NULL);  StaticJsonDocument<**200**> doc;  doc["lux"] = lux;  doc["temperatura"] = temperatura;  doc["humedad"] = humedad;  doc["humedadSuelo"] = humedadSuelo;  String jsonStr;  serializeJson(doc, jsonStr);  Serial.println(jsonStr);  delay(tiempoPaso);  } |

Tabla . Código ejecutado en placa Arduino

En este caso en cada iteración se genera un JSON con la información de los sensores y se envía por el puerto serie.

Para la programación de este Sketch hemos usado las librerías:

* SimpleDHT, BH1750 para la lectura de los sensores
* ArduinoJson para construir los objetos Json que se enviaran
* SoftwareSerial para manejar las comunicaciones con Raspberry pi

Una vez codificado el programa en Arduino IDE 2 este se compila en el propio entorno de desarrollo y se envía a la placa para que comience su ejecución.

### Raspberry pi

En esta placa se ha instalado el Sistema Operativo Ubuntu 22.04.2 LTS junto con el SDK de AWS para IoT. Para ello ejecutamos los comandos siguientes:

|  |
| --- |
| cd ~  git clone https://github.com/aws/aws-iot-device-sdk-python.git  cd aws-iot-device-sdk-python  sudo python setup.py install |

Tabla . Instrucciones instalación SDK de AWS para IoT

Una vez instalado se crea el script Python que se ejecutará para recoger los datos del puerto serie, añadirá la fecha y enviará la información a AWS IoT Core.

|  |
| --- |
| **import** **serial**  **import** **json**  **import** **pymongo**  **from** **datetime** **import** datetime  **import** **paho.mqtt.publish** **as** **publish**  **import** **paho.mqtt.client** **as** **mqtt**  **import** **ssl**  **import** **json**  ser = serial.Serial('/dev/ttyACM0', **9600**) # Configurar el puerto serie USB  # funcion para la conexion a AWS IoT  **def** **on\_connect**(client, userdata, flags, rc):  **if** rc==**0**:  print("connected OK Returned code=",rc)  **else**:  print("Bad connection Returned code=",rc)  **def** **on\_log**(client, userdata, level, buf):  print("log: ",buf)  **def** **on\_disconnect**(client, userdata, rc):  print("Disconnected: " + mqtt.error\_string(rc))  **if** \_\_name\_\_ == '\_\_main\_\_':  client = mqtt.Client(client\_id="RaspiTFG")  client.on\_connect = on\_connect  client.on\_disconnect = on\_disconnect  client.on\_log=on\_log  client.tls\_set(ca\_certs='../certs/root-CA.crt', certfile='../certs/RaspiTFG.cert.pem', keyfile='../certs/RaspiTFG.private.key', tls\_version=ssl.PROTOCOL\_SSLv23)  client.tls\_insecure\_set(**True**)  client.connect("xxxxxxxxxxxxx.iot.eu-central-1.amazonaws.com", **8883**, **60**)  client.loop\_start()  **while** **True**:  **if** ser.in\_waiting > **0**:  data = ser.readline().decode('utf-8').rstrip() # Leer los datos entrantes  dt\_string = datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%S.000Z")  **try**:  json\_data = json.loads(data) # Decodificar los datos JSON  json\_data["fecha"] = dt\_string  print("dato recibido" , json\_data , sep="->")  json\_string = json.dumps(json\_data,default=str)  client.publish("mediciones/sensores", payload=json\_string, qos=**0**, retain=**False**)  **except** **Exception** **as** err:  print(f"Unexpected {err=}, {type(err)=}") |

Tabla . Código lectura datos del puerto serie y envió a AWS

Este script se ejecutará cada vez que se arranque el sistema así que creamos un servicio Linux que se iniciará en cada arranque.

Primero creamos el fichero /etc/systemd/system/lectura.service con el contenido:

|  |
| --- |
| [Unit]  Description=Script de lectura de datos del puerto serie  [Service]  User=pi  Type=simple  WorkingDirectory=/home/pi/TFG/  ExecStart=/usr/bin/python3 lectura.py  [Install]  WantedBy=multi-user.target |

Tabla . Fichero definición servicio lectura

Luego se ejecutan los comandos que arrancaran el servicio y lo ejecutaran en cada arranque

|  |
| --- |
| sudo systemctl start lectura  sudo systemctl enable lectura |

Tabla . Comandos configuración servicio lectura

### Plataforma Amazon Web Services

#### AWS DynamoDB

Se define la tabla estados con la siguiente información:

|  |
| --- |
| {  "id": "valoresActuales",  "humedad": {  "estado": "ok",  "maximo": **80**,  "minimo": **30**,  "valorActual": **35.4**  },  "humedadSuelo": {  "estado": "ok",  "maximo": **80**,  "minimo": **40**,  "valorActual": **49**  },  "lux": {  "estado": "porDebajo",  "maximo": **8000**,  "minimo": **1000**,  "valorActual": **1.666667**  },  "temperatura": {  "estado": "ok",  "maximo": **30**,  "minimo": **20**,  "valorActual": **24.4**  }  } |

Tabla . Ejemplo tabla estados

#### Configuración reglas de reenvío en AWS IoT Core

Este servicio transforma los mensajes MQTT y realiza acciones para interactuar con otros servicios.

Cuando se reciben los datos de Raspberry pi en el tópico “lectura/sensores” se ejecutan 3 reglas que reenvían información a los distintos servicios:

1. reglaEventos: esta regla reenvía la información a AWS IoT Events para ser procesada por la máquina de estados de cada variable.
2. reglaSiteWise: esta regla reenvía los datos al servicio SiteWise para poder consultar los datos en tiempo real a través del portal https://p-yd7ndfbl.app.iotsitewise.aws/
3. reglaHistorico: esta regla envía los datos a la tabla mediciones del servicio de base de datos noSQL DynamoDB

#### AWS Lambdas

Se codifican las distintas funciones Lambdas definidas durante el analisis

* actualizarLimites

|  |
| --- |
| **import** **boto3**  # Crear cliente de DynamoDB  dynamodb = boto3.client('dynamodb')  **def** **lambda\_handler**(event, context):  # Obtener el valor de 'minimo', 'maximo' y 'variable' desde el evento de Lambda  minimo = event['minimo']  maximo = event['maximo']  variable = event['variable']  # Definir clave principal del elemento a actualizar  key = {  'id': {'S': 'valoresActuales'}  }  # Definir expresión de actualización  update\_expression = 'SET #variable.maximo = :maximo, #variable.minimo = :minimo'  # Definir los nombres de atributos  attribute\_names = {  "#variable": variable  }  # Definir valores para los parámetros de la expresión de actualización  expression\_attribute\_values = {  ':maximo': {'N': str(maximo)},  ':minimo': {'N': str(minimo)}  }  # Ejecutar la operación UpdateItem para actualizar el elemento  response = dynamodb.update\_item(  TableName='estados',  Key=key,  UpdateExpression=update\_expression,  ExpressionAttributeNames=attribute\_names,  ExpressionAttributeValues=expression\_attribute\_values  )  #imprimir respuesta  **print**(response) |

Tabla . Codificación función Lambda actualizarLimites

* actualizarValor

|  |
| --- |
| **import** **boto3**  # Crear cliente de DynamoDB  dynamodb = boto3.client('dynamodb')  **def** **lambda\_handler**(event, context):  # Obtener el valor de 'valor' y 'variable' desde el evento de Lambda  valor = event['valor']  variable = event['variable']  # Definir clave principal del elemento a actualizar  key = {  'id': {'S': 'valoresActuales'}  }  # Definir expresión de actualización  update\_expression = 'SET #variable.valorActual = :valor'  # Definir los nombres de atributos que podrían ser palabras reservadas  attribute\_names = {  "#variable": variable  }  # Definir valores para los parámetros de la expresión de actualización  expression\_attribute\_values = {  ':valor': {'N': str(valor)}  }  # Ejecutar la operación UpdateItem para actualizar el elemento  response = dynamodb.update\_item(  TableName='estados',  Key=key,  UpdateExpression=update\_expression,  ExpressionAttributeNames=attribute\_names,  ExpressionAttributeValues=expression\_attribute\_values  )  #imprimir respuesta  **print**(response) |

Tabla . Codificación función Lambda actualizarValor

* actualizarEstado

|  |
| --- |
| **import** **boto3**  # Crear cliente de DynamoDB  dynamodb = boto3.client('dynamodb')  **def** **lambda\_handler**(event, context):  # Obtener el valor de 'estado' y 'variable' desde el evento de Lambda  estado = event['estado']  variable = event['variable']  # Definir clave principal del elemento a actualizar  key = {  'id': {'S': 'valoresActuales'}  }  # Definir expresión de actualización  update\_expression = 'SET #variable.estado = :estado'  # Definir los nombres de atributos que podrían ser palabras reservadas  attribute\_names = {  "#variable": variable  }  # Definir valores para los parámetros de la expresión de actualización  expression\_attribute\_values = {  ':estado': {'S': str(estado)}  }  # Ejecutar la operación UpdateItem para actualizar el elemento  response = dynamodb.update\_item(  TableName='estados',  Key=key,  UpdateExpression=update\_expression,  ExpressionAttributeNames=attribute\_names,  ExpressionAttributeValues=expression\_attribute\_values  )  #imprimir respuesta  **print**(response) |

Tabla .. Codificación función Lambda actualizarEstado

* encolarMsgTelegram

|  |
| --- |
| **import** **json**  **import** **boto3**  **import** **time**  **from** **datetime** **import** datetime  **def** **encolarMensaje** (mensaje) :  now = datetime.now()  current\_time = now.strftime("%H:%M:%S %p")    sqs = boto3.client('sqs') #client is required to interact with  sqs.send\_message(  QueueUrl="https://sqs.eu-central-1.amazonaws.com/329640145453/ColaMensajesBotTelegram.fifo",  MessageBody=json.dumps(mensaje),  MessageGroupId='TelegramBot',  MessageDeduplicationId= str(time.time())  )    **def** **lambda\_handler**(event, context): #required    encolarMensaje(event)    **return** {  'statusCode': **200**,  'body': 'encolado'  } |

Tabla . Codificación función Lambda encloarMsgTelegram

* telegramBot

|  |
| --- |
| **import** **boto3**  **import** **json**  **import** **os**  **import** **requests**    **def** **dameCadenaPropiedad** (titulo,estado,maximo,minimo,valor):  res=titulo  **if** (estado=='ok'):  res += ('**\n** 👍️ Estado: ')  res += (estado)  **else**:  res += ('**\n** 👎 Estado: ')  res += (estado)  res += ('**\n** 🔼️ Máximo: ')  res += (maximo)  res += ('**\n** 🔽️ Mínimo: ')  res += (minimo)  **if** (estado=='ok'):  res += ('**\n** ✅ Valor actual: ')  res += (valor)  **else**:  res += ('**\n** ❌Valor actual: ')  res += (valor)  res += ('**\n**')    **return** res  **def** **dameDatosEstados**():  # Nombre de la tabla DynamoDB  tabla = 'estados'  # Crear una instancia del cliente de DynamoDB  dynamodb = boto3.client('dynamodb')  **try**:  # Obtener el item de la tabla DynamoDB  response = dynamodb.get\_item(  TableName=tabla,  Key={  'id': {'S': 'valoresActuales'}  }  )  # Leer los valores de cada atributo  item = response['Item']  humedad = item.get('humedad', {}).get('M', {})  humedad\_suelo = item.get('humedadSuelo', {}).get('M', {})  lux = item.get('lux', {}).get('M', {})  temperatura = item.get('temperatura', {}).get('M', {})  # Leer los valores específicos de cada atributo  estado\_humedad = humedad.get('estado', {}).get('S', '')  maximo\_humedad = humedad.get('maximo', {}).get('N', '')  minimo\_humedad = humedad.get('minimo', {}).get('N', '')  valor\_actual\_humedad = humedad.get('valorActual', {}).get('N', '')  estado\_humedad\_suelo = humedad\_suelo.get('estado', {}).get('S', '')  maximo\_humedad\_suelo = humedad\_suelo.get('maximo', {}).get('N', '')  minimo\_humedad\_suelo = humedad\_suelo.get('minimo', {}).get('N', '')  valor\_actual\_humedad\_suelo = humedad\_suelo.get('valorActual', {}).get('N', '')  estado\_lux = lux.get('estado', {}).get('S', '')  maximo\_lux = lux.get('maximo', {}).get('N', '')  minimo\_lux = lux.get('minimo', {}).get('N', '')  valor\_actual\_lux = lux.get('valorActual', {}).get('N', '')  estado\_temperatura = temperatura.get('estado', {}).get('S', '')  maximo\_temperatura = temperatura.get('maximo', {}).get('N', '')  minimo\_temperatura = temperatura.get('minimo', {}).get('N', '')  valor\_actual\_temperatura = temperatura.get('valorActual', {}).get('N', '')  res = dameCadenaPropiedad('💧 Humedad ',estado\_humedad,maximo\_humedad,minimo\_humedad,valor\_actual\_humedad)  res += dameCadenaPropiedad('🪴 Humedad del suelo ',estado\_humedad\_suelo,maximo\_humedad\_suelo,minimo\_humedad\_suelo,valor\_actual\_humedad\_suelo)  res += dameCadenaPropiedad('☀️ Radiación ',estado\_lux,maximo\_lux,minimo\_lux,valor\_actual\_lux)  res += dameCadenaPropiedad('🌡 Temperatura ',estado\_temperatura,maximo\_temperatura,minimo\_temperatura,valor\_actual\_temperatura)  **return** res    **except** **Exception** **as** err:  **print**(f"Unexpected {err=}, {type(err)=}")    **def** **lambda\_handler**(event, context):  **try**:  request\_body = json.loads(event['body']) # EXtract the Body from the call  **except** **KeyError**:  request\_body = ""    **try**:  request\_msg = json.dumps(request\_body['message'])#['chat']['id'] # Extract the message object which contrains chat id and text  **except** **KeyError**:  request\_msg = ""    **try**:  chat\_id = json.dumps(request\_body['message']['chat']['id']) # Extract the chat id from message  **except** **KeyError** :  chat\_id = ""    **try** :  command = json.dumps(request\_body['message']['text']).strip('"') # Extract the text from the message  **except** **KeyError**:  command = ""  telegram\_msg = ""    **if** command == "resumen":  telegram\_msg = dameDatosEstados()  **else**:  telegram\_msg = "🤷️"    **print**(telegram\_msg)    chat\_id = os.environ['CHATID']  telegram\_token = os.environ['TOKEN']  api\_url = f"https://api.telegram.org/bot{telegram\_token}/"  params = {'chat\_id': chat\_id, 'text': telegram\_msg}  res = requests.post(f"{api\_url}sendMessage", data=params).json()  **return** {  'statusCode': **200**,  'msg': json.dumps('respuesta enviada!'),  'body': json.dumps('vamooos')  } |

Tabla . Codificación función Lambda telegramBot

En esta Lambda además de la codificación es necesario añadir el trigger para que se ejecute cada vez que llegue una petición a la API Gateway. Para ello pulsamos en opción Agregar desencadenador en la ventana de definición de la Lambda.

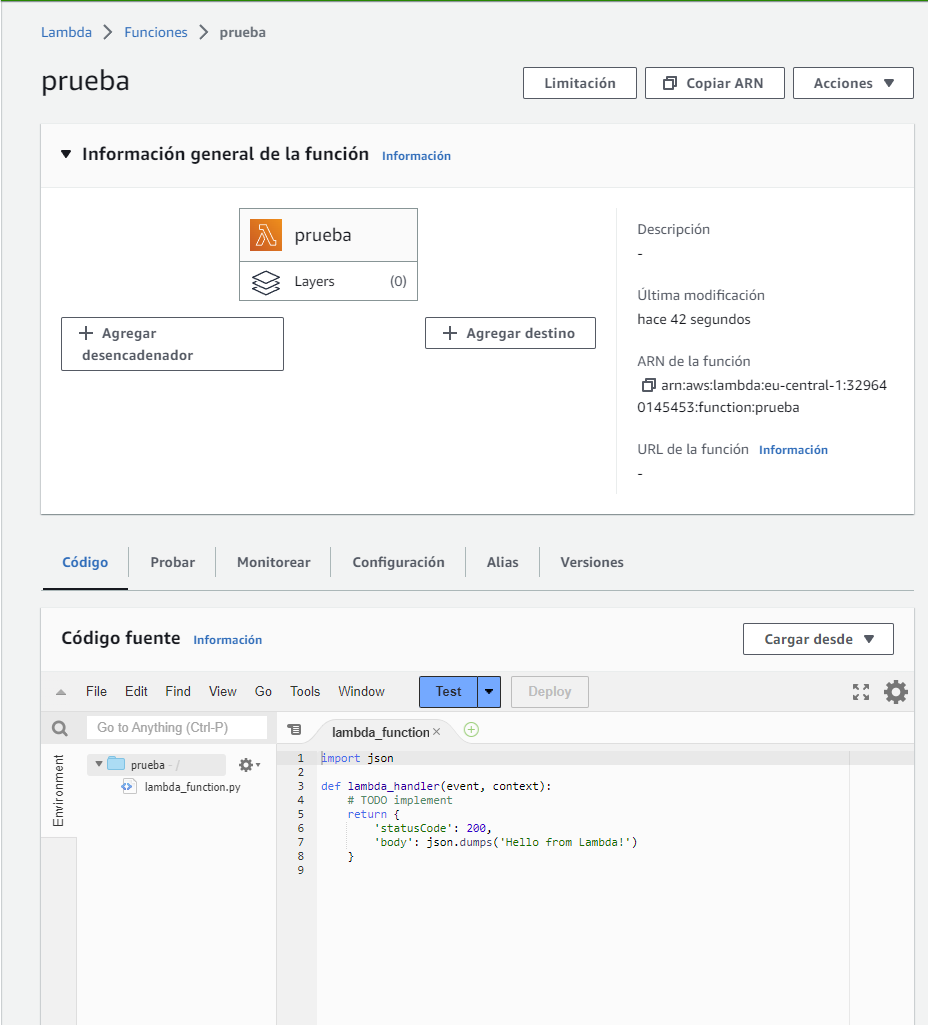


Ilustración . Ventana definición Lambda

Y seleccionar API Gateway en el siguiente formulario:



Ilustración .Formulario definición desencadenador

Al pulsar en “Agregar” se generará la API Gateway y se conectará con la función Lambda

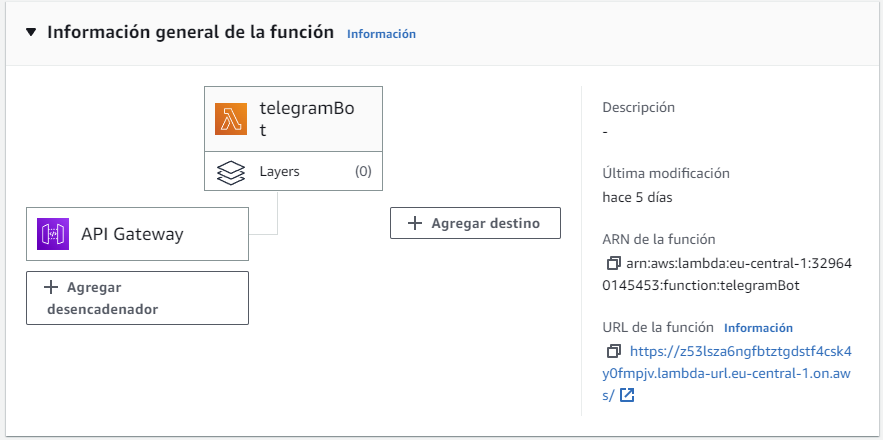


Ilustración . API Gateway conectada con función Lambda

* enviarMensajeTelegramBot

|  |
| --- |
| **import** **boto3**  **import** **json**  **import** **os**  **import** **requests**  **def** **lambda\_handler**(event, context):  **print**(json.dumps(event))  chat\_id = os.environ['CHATID']  telegram\_token = os.environ['TOKEN']  api\_url = f"https://api.telegram.org/bot{telegram\_token}/"  records = event['Records']  **for** record **in** records:  content = record['body']  **print**(content)    params = {'chat\_id': chat\_id, 'text': json.loads(content)['mensaje']}  res = requests.post(f"{api\_url}sendMessage", data=params).json()    **return** {  'statusCode': **200**,  'body': 'mensajeEnviado'  } |

Tabla . Codificación función Lambda enviarMensajeTelegramBot

* actualizarValoresHorarios

|  |
| --- |
| **import** **json**  **import** **boto3**  **import** **datetime**  **import** **dateutil.tz**  # Crear cliente de DynamoDB  dynamodb = boto3.client('dynamodb')  **def** **actualizarLimites** (variable,maximo,minimo) :  # Crear una instancia del cliente Lambda  lambda\_client = boto3.client('lambda')  # Parámetros para la invocación de la función Lambda objetivo  function\_name = 'actualizarLimites'  payload = {  'minimo': minimo,  'maximo': maximo,  'variable': variable  }  # Invocar a la función Lambda objetivo  response = lambda\_client.invoke(  FunctionName=function\_name,  InvocationType='RequestResponse', # Puedes cambiar el tipo de invocación según tus necesidades  Payload=json.dumps(payload)  )  # Obtener el resultado de la función Lambda objetivo  result = json.loads(response['Payload'].read())  print(result)    **def** **actualizarVariables** (maximo,minimo) :  # Configurar el cliente de AWS IoT Events  iot\_events\_client = boto3.client('iotevents-data')    # Nombre del detector de eventos  detector\_name = 'detectorLux'    # Definir los datos a enviar al detector de eventos  data = {  "lux": {  "maximo": maximo,  "minimo": minimo  }  }  # Convertir los datos a formato JSON  payload = json.dumps(data)    # Enviar los datos al detector de eventos  iot\_events\_client.batch\_put\_message(  messages=[  {  'inputName': 'actualizacionLimites', # Nombre del input en el detector de eventos  'messageId': '1', # ID único del mensaje  'payload': payload  }  ]  )  print(payload)  # Imprimir mensaje de éxito  print("Datos enviados al detector de eventos")  **def** **lambda\_handler**(event, context):  # Obtener el valor de hora pasado por parámetro y el id de la propiedad    zona\_horaria\_esp = dateutil.tz.gettz('Europe/Madrid');  fecha\_actual = datetime.datetime.now(zona\_horaria\_esp).time()  hora = fecha\_actual.hour  margen = event['margen']  idPropiedad = event['id']  # Crear una instancia del cliente DynamoDB  dynamodb = boto3.client('dynamodb')  # Consultar la tabla DynamoDB  response = dynamodb.get\_item(  TableName='horario',  Key={  'id': {'S': idPropiedad}  }  )  # Obtener el valor correspondiente a la hora solicitada  horario = response['Item']['horario']  valor = horario['M']['hora{}'.format(hora)]    maximo = int(valor['N']) + margen  minimo = int(valor['N']) - margen    actualizarLimites(idPropiedad,maximo,minimo)  actualizarVariables(maximo,minimo)    # Devolver el valor encontrado  **return** {  'hora': hora,  'valor': valor['N'],  'maximo': maximo,  'minimo': minimo,  'propiedad': idPropiedad  } |

#### Configuración estados AWS IoT Events

* Detectores: Los detectores son los encargados de procesar los datos de entrada de los sensores.

Se crean cuatro modelos de detector, una para cada variable, humedad, temperatura, lúmenes, y humedad de suelo.

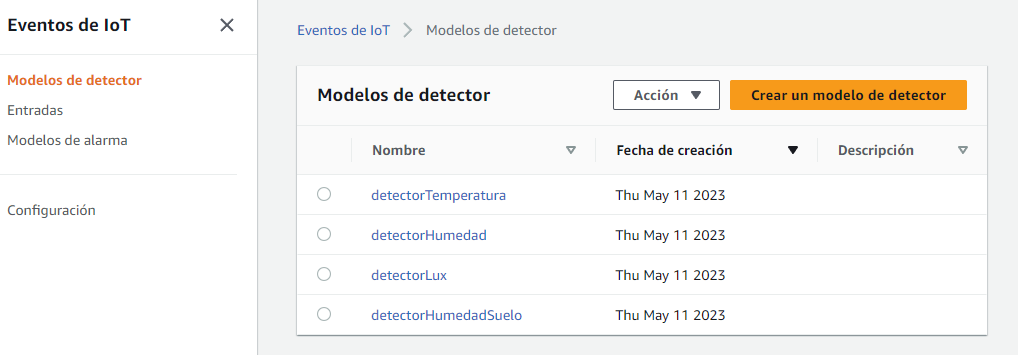


Ilustración .AWS IoT Events. Modelos de detector

Cada modelo de detector es una máquina de estados, podrían ser máquinas de estados distintas para cada propiedad pero para esta prueba todos tienen las mismas transiciones y estados.

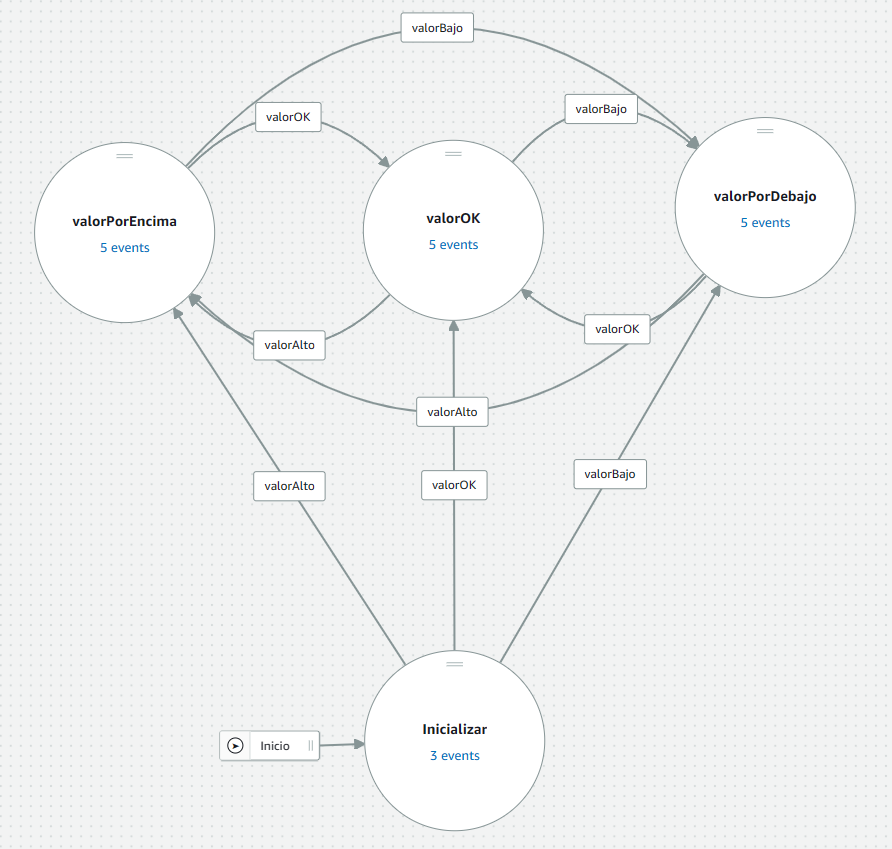


Ilustración .Definición máquina de estados detectores

Cada vez que se lee un valor se manda a esta máquina de estados. Los estados son los siguientes:

* Estado “Inicializar”: este estado es el inicial, cuando se entra en este estado se producen 3 eventos:
  + Inicio: en este evento se definen los valores mínimos y máximos por defecto que se usaran como condiciones para hacer las transiciones
  + actualizarEstado: en este evento se ejecuta la función lambda actualizarEstado pasando los parámetros en un JSON con el siguiente formato:

|  |
| --- |
| {  "variable": "temperatura",  "estado": "inicializar"  } |

Tabla .Ejemplo llamada a Lambda actualizarEstado

* + actualizarLimites: esta llamada actualizara el mínimo y máximo de la variable con los valores definidos en el evento “Inicio”, enviara el siguiente JSON a la función:

|  |
| --- |
| {  "variable": "temperatura",  "minimo": x,  "maximo": y  } |

Tabla . Ejemplo llamada a Lambda actualizarLimites

* En los Estados “TemperaturaPorEncima”, “TemperaturaOK” y “TemperaturaPorDebajo” al entrar en el estado se producen 3 eventos
  + actualizarValor: la llamada a esta función que actualizara el valorActual de la variable con el ultimo valor leído, el JSON del parámetro es el siguiente:

|  |
| --- |
| {  "valor": "${input.entrada.temperatura}",  "variable": "temperatura"  } |

Tabla . Ejemplo llamada a Lambda actualizarValor

* + actualizarEstado: actualiza el valor estado de la variable al estado en el que se encuentra, pueden ser “porDebajo”, “porEncima” u “ok”

|  |
| --- |
| {  "variable": "temperatura",  "estado": "porEncima"  } |

Tabla . Ejemplo llamada a Lambda actualizarEstado

* + notificarTelegram: hace una llamada a la función lambda para encolar el mensaje de respuesta enviando el JSON:

|  |
| --- |
| {  "mensaje": "El estado de la temperatura está por encima de los limites 😢 \n el valor es ${input.entrada.temperatura} y debería ser inferior a ${variable.valorMaximo}"  } |

Tabla . Ejemplo llamada a Lambda notificarTelegram

Al recibir un dato nuevo por la entrada de los datos de los sensores se produce el evento:

* + actualizarValor: esta llamada es igual a la que se produce cuando se entra en el estado. Actualiza el valor con el ultimo input llamando a la Lambda actualizarValor.

Al recibir un dato nuevo por la entrada de la actualización de limites se actualizan los valores máximos y mínimos que se usan para calcular las transiciones a los nuevos valores recibidos.

* + actualizarLimites: este evento asigna a la variable valorMaximo el valor de $input.actualizacionLimites.lux.maximo y a la variable valorMinimo el valor de $input.actualizacionLimites.lux.minimo

Además para variables en la que se quieran actualizar los valores limites horariamente hay que crear un nuevo modelo de detector con la siguiente máquina de estados:

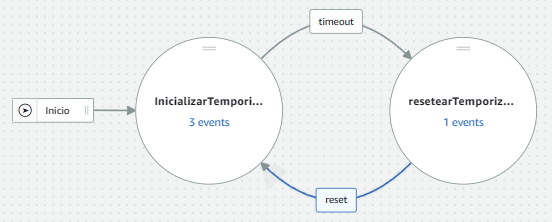


Ilustración . Definición máquina de estados temporizadores

Esta máquina tiene 2 estados:

* inicializarTemporizador: en este estado se producen 3 eventos:
  + inicioTemporizador: este evento crea el temporizador con una duración de 1 hora
  + actualizarValoresHorarios: este evento hace una llamada a la función Lambda actualizarValoresHorarios
  + leerFecha: guarda la fecha de la última actualización en una variable

cuando el tiempo se cumple se ejecuta la transición timeout.

* resetearTemporizador: En este estado se ejecutan 1 eventos
  + resetearTemporizador: que elimina el temporizador

la transición reset se ejecuta siempre que se entra en el estado resetearTemporizador con lo que se volvería al estado inicializarTemporizador.

### Configuración Bot Telegram

Para crear nuestro Bot tenemos que abrir una conversación desde la aplicación con un BotFather y ejecutar el comando:

|  |  |
| --- | --- |
| /newbot | Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto  Descripción generada automáticamente  Ilustración . Creación Bot con BotFather |
|  |

Para que las peticiones del Bot de Telegram lleguen a nuestro Gateway es necesario ejecutar el comando:

|  |
| --- |
| curl https://api.telegram.org/bot<TOKEN>/setWebhook?url=<URL\_GATEWAY> |

Donde <TOKEN> es el token que nos proporciona Telegram cuando registramos nuestro Bot y <URL\_GATEWAY> es la dirección de nuestro AWS Gateway API, <https://amt44iczs3.execute-api.eu-central-1.amazonaws.com/default/telegramBot>

De esta forma todos los mensajes serán reenviados a nuestra API en AWS.

Es posible comprobar el estado de nuestro Webhook ejecutando el comando:

|  |
| --- |
| curl https://api.telegram.org/bot<TOKEN>/getWebhookInfo |

Si todo funciono correctamente responderá con el mensaje:

|  |
| --- |
| StatusCode : 200  StatusDescription : OK  Content : {"ok":true,"result":{"url":"https://amt44iczs3.execute-api.eu-central-1.amazonaws.com/default/teleg  ramBot","has\_custom\_certificate":false,"pending\_update\_count":0,"max\_connections":40,"ip\_address":"  3....  RawContent : HTTP/1.1 200 OK  Connection: keep-alive  Strict-Transport-Security: max-age=31536000; includeSubDomains; preload  Access-Control-Allow-Origin: \*  Access-Control-Allow-Methods: GET, POST, OPTIONS  Acce...  Forms : {}  Headers : {[Connection, keep-alive], [Strict-Transport-Security, max-age=31536000; includeSubDomains;  preload], [Access-Control-Allow-Origin, \*], [Access-Control-Allow-Methods, GET, POST, OPTIONS]...}  Images : {}  InputFields : {}  Links : {}  ParsedHtml : mshtml.HTMLDocumentClass  RawContentLength : 214 |

Ilustración . Confirmación estado Telegram Webhook

## Pruebas

Para validar el sistema se ha ido probando con distintos valores límites y comprobando que se reciben las alertas correctamente por ejemplo:

Teniendo como valores límites de radiación [8000] y [1000] y los valores límite de temperatura entre [30] y [20]

cuando a las 19:16 se baja de ese valor el Bot nos envía mensajes de alerta y cuando a las 20:12 baja la temperatura vemos que también nos alerta del cambio de estado.

|  |  |
| --- | --- |
| Ilustración . Mensaje alerta | Ilustración 30. Cambio estado temperatura baja |
| Ilustración 31. Cambio estado lúmenes |

Al pedir el resumen podemos comprobar que esas variables están en un estado de alerta.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Ilustración . Resumen Bot Telegram con estados de alarma

Al día siguiente se recibe un mensaje a las 08:25 cuando la radiación vuelve a un estado correcto y a las 10:17 cuando la temperatura también se corrige, todas las variables están en estado ok

|  |  |
| --- | --- |
| Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto  Descripción generada automáticamente  Ilustración .Mensajes cambio estado Telegram Bot | Ilustración .Grafico SiteWise lúmenes cambio estado |
| Ilustración . Gráfico SiteWise temperatura cambio estado |

# Conclusiones y trabajo futuro

## Conclusiones del trabajo

Tras la ejecución de la prueba se puede concluir que todos los objetivos generales y específicos se han cumplido.

Se ha conseguido generar un sistema de alarmas para cultivos que lee los datos de sensores, los envía y procesa en un sistema cloud como AWS Events, donde se pueden crear reglas para ajustar las alarmas a nuestras necesidades y que interactúa con el usuario a través de un sistema sencillo de mensajería móvil como es Telegram.

## Líneas de trabajo futuro

Durante el desarrollo del TFG han surgido bastantes ideas para posibles trabajos futuros.

Por una parte el sistema podría ampliar su funcionalidad usando actuadores para que el sistema pudiese corregir los problemas automáticamente o incluso consultar al usuario a través de Telegram las posibles medidas correctoras. Por ejemplo si se detectase humedad en el suelo baja podría activar el riego o si en un invernadero hace mucho calor, podría preguntar si se quiere activar la ventilación.

Otras mejoras que se podrían desarrollar serian la conexión con otros servicios de AWS para ampliar la funcionalidad del Bot. Por ejemplo se podría conectar al servicio AWS Lex para ampliar las capacidades conversacionales del Bot y que sea capaz de responder de forma más natural.

También se podrían usar todos los datos de los sensores junto con los resultados de las cosechas para entrenar modelos de aprendizaje automático y poder hacer estimaciones más precisas de los futuros cultivos.

Por último todo el sistema podría modelarse como infraestructura como código (IaC) y ofrecerse a posibles clientes usando el servicio CloudFormation(Amazon, 2023a) para desplegar el sistema de forma automática.

Referencias bibliográficas

Amazon. (2023a). *Aprovisionamiento de infraestructura como código - AWS CloudFormation - AWS*. https://aws.amazon.com/es/cloudformation/

Amazon. (2023b). *Informática sin servidor – Características de AWS Lambda – Amazon Web Services*. https://aws.amazon.com/es/lambda/features/

Amazon. (2023c). *Internet de las cosas | Plataforma como servicio | AWS IoT*. https://aws.amazon.com/es/iot/

Amazon. (2023d). *Recopilar y procesar datos industriales – AWS IoT SiteWise – Amazon Web Services*. https://aws.amazon.com/es/iot-sitewise/

Amazon. (2023e). *What is Amazon Simple Queue Service? - Amazon Simple Queue Service*. https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html

Amazon Web Services. (2023a). *How AWS IoT works - AWS IoT Core*. https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html

Amazon Web Services, I. (2023b). *AWS IoT API Reference*.

Amazon Web Services, I. (2023c). *¿Qué es AWS?* https://aws.amazon.com/es/what-is-aws/?nc1=f\_cc

APSIM. (2023). *About Us – APSIM*. https://www.apsim.info/about-us/

Arduino Official Store. (2023). *Arduino Plant Watering Kit — Arduino Official Store*. https://store.arduino.cc/products/plant-watering-kit

Arduino S.r.l. (2023). *Arduino Plant Watering Kit — Arduino Official Store*. https://store.arduino.cc/pages/plant-watering-kit

Baggio, A. (2005). *Wireless sensor networks in precision agriculture*. http://www.lofar.org/p/Agriculture.htm

Beyaz, A., & Gül, V. (2022). Determination of Low-Cost Arduino Based Light Intensity Sensors Effectiveness for Agricultural Applications. *Brazilian Archives of Biology and Technology*, *65*. https://doi.org/10.1590/1678-4324-2022220172

Bogdan Mihai. (2016). *HOW TO USE THE DHT22 SENSOR FOR MEASURING TEMPERATURE AND HUMIDITY WITH THE ARDUINO BOARD*. *LXVIII*. https://doi.org/10.1515/aucts-2016-0005

Brisson, N., Gary, C., Justes, E., Roche, R., Mary, B., Ripoche, D., Zimmer, D., Sierra, J., Bertuzzi, P., Burger, P., Bussière, F., Cabidoche, Y. M., Cellier, P., Debaeke, P., Gaudillère, J. P., Hénault, C., Maraux, F., Seguin, B., & Sinoquet, H. (2003). An overview of the crop model stics. *European Journal of Agronomy*, *18*(3-4), 309-332. https://doi.org/10.1016/S1161-0301(02)00110-7

Carrasco-Ríos, L. (2009). EFECTO DE LA RADIACIÓN ULTRAVIOLETA-B EN PLANTAS. *Idesia (Arica)*, *27*(3), 59-76. https://doi.org/10.4067/S0718-34292009000300009

Dennys Francisco Salazar Domínguez. (2022). *ESCUELA POLITÉCNICA NACIONAL FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA SISTEMA DE RIEGO PARA INVERNADEROS CON INTERFAZ WEB UTILIZANDO UN ARDUINO YÚN MANEJO DE ACTUADORES CON ARDUINO YÚN TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN*.

docs.arduino.cc. (2023a). *Estructura programa arduino*. https://docs.arduino.cc/learn/starting-guide/getting-started-arduino#program-structure.

docs.arduino.cc. (2023b). *UNO R3 | Arduino Documentation | Arduino Documentation*. https://docs.arduino.cc/hardware/uno-rev3

Dreyli Maygualida Hidalgo Ramos. (2017). *UNIVERSIDAD AUTÓNOMA AGRARIA ANTONIO NARRO*.

Espinosa, A., Ponte, D., Gibeaux, S., & González, C. (2021). *Estudio de Sistemas IoT Aplicados a la Agricultura Inteligente*.

Gesteira, R., Director, M., Atilano, :, Fernández-Pacheco Sánchez, R., & Madrid, M. (2020). *Implementación de una arquitectura de microservicios para una red de sensores IoT sobre Arduino*. https://repositorio.comillas.edu/xmlui/handle/11531/43411

Hoogenboom, G., Porter, C. H., Boote, K. J., Shelia, V., Wilkens, P. W., Singh, U., White, J. W., Asseng, S., Lizaso, J. I., Moreno, L. P., Pavan, W., Ogoshi, R., Hunt, L. A., Tsuji, G. Y., & Jones, J. W. (2019). *The DSSAT crop modeling ecosystem*. 173-216. https://doi.org/10.19103/AS.2019.0061.10

https://www.json.org/. (2023). *JSON*. https://www.json.org/json-en.html

IBM. (2023). *Soluciones IoT | IBM*. https://www.ibm.com/es-es/cloud/internet-of-things?utm\_content=SRCWW&p1=Search&p4=43700068029441558&p5=p&gclid=Cj0KCQjwgLOiBhC7ARIsAIeetVDU49fAR-p0L2Ju9m4qUkew77jauI28jwsSvh-y0qCYd2MNKEbMs\_waAvC0EALw\_wcB&gclsrc=aw.ds

INRAE. (2023). *Agroclim STICS - Stics model overview*. https://www6.paca.inrae.fr/stics\_eng/About-us/Stics-model-overview

J. Cedeño, M. Zambrano, & C. Medina. (2014). *Vista de Redes inalámbricas de sensores eficientes para la agorindustria*. https://revistas.utp.ac.pa/index.php/prisma/article/view/518/513

José, J., & Arboleda, V. (2009). *Modelo de proyección para la producción de rosas, basado en las curvas de crecimiento de las plantas*. https://ciencia.lasalle.edu.co/administracion\_agronegocios

Juan Núñez, V. M., Faruk Fonthal, R., & Yasmín Quezada, L. M. (2018, diciembre 5). Design and Implementation of WSN and IoT for Precision Agriculture in Tomato Crops. *2018 IEEE ANDESCON, ANDESCON 2018 - Conference Proceedings*. https://doi.org/10.1109/ANDESCON.2018.8564674

Kaa. (2023). *➤ Enterprise IoT Platform with Free Plan | Kaa*. https://www.kaaiot.com/

Kulmány, I. M., Bede-Fazekas, Á., Beslin, A., Giczi, Z., Milics, G., Kovács, B., Kovács, M., Ambrus, B., Bede, L., & Vona, V. (2022). Calibration of an Arduino-based low-cost capacitive soil moisture sensor for smart agriculture. *Journal of Hydrology and Hydromechanics*, *70*(3), 330-340. https://doi.org/10.2478/JOHH-2022-0014

Kumar, S., Tiwari, P., & Zymbler, M. (2019). *Internet of Things is a revolutionary approach for future technology enhancement: a review*. https://doi.org/10.1186/s40537-019-0268-2

Mathew, S. (2014). *Overview of Amazon Web Services*.

Microsoft. (2023a). *Información general de Windows 10 IoT - Windows IoT | Microsoft Learn*. https://learn.microsoft.com/es-es/windows/iot-core/windows-iot

Microsoft. (2023b). *IoT Hub | Microsoft Azure*. https://azure.microsoft.com/es-es/products/iot-hub

Microsoft. (2023c). *¿Qué es PaaS? Plataforma como servicio | Microsoft Azure*. https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-paas

Ministerio de Agricultura, P. y A. (2023). *Principales retos del regadío Español*. https://www.mapa.gob.es/es/desarrollo-rural/temas/gestion-sostenible-regadios/regadio-espanya/default\_1.1.3.aspx

mqtt.org. (2023). *mqtt protocol*. https://mqtt.org/.

Nestor Monroy, Ignacio Pérez, & José Ricardo Cure. (2001). *Vista de Estudio de la variabilidad en el clima y la producción de rosas en la sabana de Bogotá | Revista de Ingeniería*. https://revistas.uniandes.edu.co/index.php/rdi/article/view/6943/7266

Netro. (2023). *Netro Whisperer | Smart Plant Sensor*. https://netrohome.com/en/shop/products/whisperer

Persson, M. ; ;, Berndtsson, R. ;, Bertotto, L. E. ;, Kobayashi, A. N. A. ;, Wendland, E. C., Schwamback, D., Persson, M., Berndtsson, R., Bertotto, L. E., Naoki, A., Kobayashi, A., & Wendland, E. C. (2023). Automated Low-Cost Soil Moisture Sensors: Trade-Off between Cost and Accuracy. *Sensors 2023, Vol. 23, Page 2451*, *23*(5), 2451. https://doi.org/10.3390/S23052451

Placidi, P., Gasperini, L., Grassi, A., Cecconi, M., & Scorzoni, A. (2020). Characterization of low-cost capacitive soil moisture sensors for IoT networks. *Sensors (Switzerland)*, *20*(12), 1-14. https://doi.org/10.3390/S20123585

Python Software Foundation. (2023). *El tutorial de Python — documentación de Python - 3.11.3*. https://docs.python.org/es/3/tutorial/

Raspberry Pi Foundation. (2023). *Raspberry Pi 3 Model B+*. www.raspberrypi.org/products/raspberry

Raspberry Pi Foundation. (2023). *Raspberry Pi Foundation – About us*. https://www.raspberrypi.org/about/

Raspberry pi foundation. (2023). *Raspberry Pi OS – Raspberry Pi*. https://www.raspberrypi.com/software/

Santos-Olmo, A. B. (2022). *SMARTCETA SISTEMA DE MEDICIÓN DE VALORES BÁSICOS DE UNA PLANTA UTILIZANDO IOT*.

Silva, C., & Antonio, J. (2016). *Trabajo Fin de Máster*.

Sunehra, D. (2019). Article ID: IJARET\_10\_02\_0 06 Cite this Article: Dr. Dhiraj Sunehra, Web Based Smart Irrigation System Using Raspberry PI. *International Journal of Advanced Research in Engineering and Technology (IJARET)*, *10*(2), 55-64. http://iaeme.com/Home/journal/IJARET55editor@iaeme.comhttp://iaeme.com/Home/issue/IJARET?Volume=10&Issue=2http://iaeme.com

ThingsBoard. (2023). *ThingsBoard - Open-source IoT Platform*. https://thingsboard.io/

Ubuntu. (2023). *Ubuntu MATE | For a retrospective future*. https://ubuntu-mate.org/

Arduino. (2023). *UNO R3 | Arduino Documentation*. https://docs.arduino.cc/hardware/uno-rev3

1. Título del anexo.

Índice de acrónimos

|  |  |
| --- | --- |
| JSON | Es el formato de texto sencillo para intercambio de datos (https://www.json.org/, 2023). |