UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Extendiendo las capacidades de las plataformas IoT. Una aplicación práctica

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

AUTOR/A: Albert Casañ, Alejandro Manuel

Tutor/a: Fons Cors, Joan Josep

CURSO ACADÉMICO: 2023/2024

Resumen

La Internet de las Cosas (IoT) es una realidad en constante evolución que está permitiendo construir un mundo en el que lo físico se entremezcla con los procesos digitales, aportando una combinación que está generando muchos beneficios en diferentes sectores, como la industria, las ciudades inteligentes o la agricultura.

En este ámbito han aparecido un buen número de plataformas y *frameworks* de desarrollo que ofrecen servicios típicos y comunes, como la funcionalidad para registrar y provisionar dispositivos y servicios, mecanismos para securizar el acceso a la infraestructura, APIs y Gateways para la recolección y consulta de información heterogénea y multidispositivo, sistemas de soporte al mantenimiento y consulta de datos, o la posibilidad de definir paneles/dashboards de visualización.

Sin embargo, existe un conjunto de funciones, también relacionadas con la gestión de la infraestructura, que este tipo de herramientas y soluciones no suele proveer. Estas meta-operaciones de gestión permiten tener mayor control sobre operaciones típicas, como detener/iniciar dispositivos (*shutdown*/*reboot*), actualizarlos (*update*), ejecutar scripts, activar/desactivar servicios, actualizar certificados o políticas de seguridad, entre otras. Este tipo de funciones son de gran ayuda a la hora de gestionar y mantener una infraestructura de computación distribuida, ofreciendo incluso ciertos servicios de orquestación funcional, similar a como ofrecen soluciones como Kubernetes o Docker Swarm en el mundo de los microservicios.

El objetivo de este proyecto es extender una plataforma IoT existente, como ThingsBoard, para ofrecer tanto mecanismos "de plataforma" (prediseñados y preimplementados) como otros que se puedan adaptar y personalizar a soluciones concretas. De esta manera, un dispositivo o servicio que forme parte de una solución IoT, además de conectarse y usar los servicios que ya ofrezca la plataforma IoT, se enriquecerá con estas capacidades de gestión y mantenimiento, habilitando cierto grado de autonomía. En concreto, en este proyecto, se extenderá la plataforma IoT elegida y se aplicará para rediseñar el SmartCity Lab (grupo TaTami, centro PROS en el instituto VRAIN). Este laboratorio cuenta en la actualidad con diferentes servicios en el ámbito de las ciudades inteligentes (relacionados con el tráfico, la contaminación ambiental o servicios en edificios inteligentes), además de un conjunto de dispositivos IoT (maquetas). Se extenderán, pues, estos servicios para prototipar estas nuevas capacidades de gestión de infraestructuras IoT, permitiendo la integración de los dispositivos en una solución más amplia y ofreciendo una plataforma para el desarrollo y la gestión de soluciones IoT personalizadas.

Palabras clave: IoT, plataforma, gestión, operaciones, ciudades inteligentes.

Abstract

The internet of Things (IoT) is a constant evolving reality that is allowing to build a world where the physical mixes with digital procedures bringing a combination that is generating many benefits in different sectors like industry, smart cities, or agriculture.

A lot of platforms and development frameworks have appeared in this matter offering common services like registering and provisioning devices and services, infrastructure security mechanisms, APIs and Gateways for collecting heterogeneous and multidevice information, maintenance support and data consulting, or the possibility of defining dashboards for visualization.

However, a group of functions related to the management of the infrastructure exist that this kind of tools and solutions do not usually offer. These meta-operations of management allow having more control of typical operations like stopping/booting devices (shutdown/reboot), updating them (update), executing scripts, activating/deactivating services, updating certificates or security policies, among others. These types of functions are of significant help when managing and maintaining a distributed computing infrastructure, even offering certain functional orchestration services, like how solutions such as Kubernetes or Docker Swarm offer in the world of microservices.

The goal of this project is to extend an existing IoT platform, such as ThingsBoard, to offer both "platform" mechanisms (pre-designed and pre-implemented) and others that can be adapted and customized to specific solutions. In this way, a device or service that is part of an IoT solution, in addition to connecting and using the services already offered by the IoT platform, will be enriched with these management and maintenance capabilities, enabling a certain degree of autonomy. Specifically, in this project, the chosen IoT platform will be extended and applied to redesign the SmartCity Lab (TaTami group, PROS center at the VRAIN institute). This laboratory currently has different services in the field of smart cities (related to traffic, environmental pollution, or services in smart buildings), in addition to a set of IoT devices (models). These services will therefore be extended to prototype these new IoT infrastructure management capabilities, allowing the integration of devices into a broader solution and offering a platform for the development and management of customized IoT solutions.

Keywords: IoT, platform, management, operations, smart cities.

Tabla de contenidos

Tabla de ilustraciones

1. Introducción

Motivación

Hoy en día estamos viendo cómo se ponen en marcha soluciones inteligentes para mejorar procesos o resolver problemas que ocurren en multitud de ámbitos como las Smart City, la agricultura o la automatización industrial gracias a la IoT. Estas soluciones revolucionan la forma que tenemos de interactuar con nuestro entorno cambiando el paradigma ya que gracias a ellas podemos optimizar el uso de recursos o la eficiencia de un sistema e incluso vemos como en soluciones de inteligencia ambiental eliminamos completamente la intervención humana para tomar decisiones.

Sin embargo, conforme estas soluciones se introducen en mundo real, la complejidad a la hora de gestionar y mantener infraestructuras complejas o grandes flotas de dispositivos crece de forma exponencial. Las plataformas IoT actuales han avanzado de forma significativa en la gestión de dispositivos, pero todavía ofrecen funcionalidades básicas o las funciones avanzadas están restringidas a dispositivos concretos. En el momento en el que queremos crear un sistema con cierta complejidad introduciendo operaciones más avanzadas o es necesaria una personalización profunda y recurrimos a estas plataformas nos damos cuenta de que no satisfacen esas necesidades. Uno de los casos reales donde notamos esta falta es en las ciudades inteligentes.

En las Smart City tenemos requisitos particulares a la hora de ofrecer operaciones como reiniciar o actualizar dispositivos, gestionar certificados de seguridad y activar o desactivar servicios de manera eficiente por lo que no podemos asentar nuestra solución únicamente en una plataforma IoT. Además, nos encontramos que no hay un procedimiento estándar para planificar u orquestar estas operaciones dentro de las propias plataformas. Podemos pensar en un escenario donde los semáforos de una calle se tengan que apagar porque la calle está cortada. Sin mecanismos o protocolos que nos permitan definir como se llama a la ejecución de esta operación o como el dispositivo la va a recibir acabamos recurriendo a que un operario se conecte de forma remota o vaya físicamente a desconectar cada uno de los semáforos. Este procedimiento tradicional puede dar lugar a fallos, agujeros de seguridad e incluso peligros para el propio operario.

Objetivos

En este contexto, surge la motivación y el objetivo principal a conseguir con el desarrollo del proyecto: crear un sistema para permitir la implementación de operaciones personalizables en infraestructuras IoT para extender una plataforma de este ámbito. Este objetivo general de desglosa en los siguientes objetivos secundarios:

1. Sistema integrable: el sistema resultante del desarrollo del proyecto ofrecerá una interfaz que facilite la integración en soluciones actuales.
2. Orquestación: el sistema capaz de orquestar operaciones, permitiendo automatizar su lanzamiento.

3. Personalización: el resultado del proyecto permitirá un grado de personalización que pueda dar respuesta a las necesidades de los múltiples ámbitos de uso de la IoT. Entre ellos el SmartCity Lab del grupo Tatami.

Impacto esperado

El resultado del proyecto tiene el potencial de convertirse en un producto complementario a cualquier plataforma IoT e incluso otros productos que necesiten una forma de orquestar el envío y control de operaciones a todo tipo de dispositivos desde sensores a robots. Se espera que las funcionalidades proporcionen:

- Flexibilidad y adaptabilidad: los diferentes entornos IoT podrán utilizar la solución para automatizar tareas según las necesidades de la infraestructura.
- Reducción de costes de tiempos y costes de mantenimiento: la automatización de las tareas más repetitivas reducirá los costes y tiempos del mantenimiento de las infraestructuras IoT.
- Mejora en la resiliencia y seguridad del sistema: al centralizar, automatizar tareas y delegar en nuestro sistema la orquestación de estas podemos reducir los riesgos asociados a la seguridad del entorno.
- Mayor autonomía: los propios dispositivos que configuran las infraestructuras IoT podrán utilizar esta solución para tener cierta autonomía operativa a la hora de decidir la respuesta a cada operación en cada caso particular.

Metodología

Para el desarrollo del software de este proyecto se seguirá una metodología iterativa y centrada en el desarrollo ágil:

En primer lugar, se revisará el estado del arte para ver en profundidad las plataformas IoT actuales para conocer sus limitaciones y evaluar la mejor forma de extenderlas para conseguir los objetivos planteados.

A continuación, se diseñarán varias alternativas que podemos considerar para la arquitectura de nuestra solución y de estas se escogerá la más idónea para nuestro caso de uso especificando detalladamente el diseño. Más tarde, se detallará el proceso de implementación de la solución donde veremos la mayor parte del trabajo de este proyecto. Posteriormente, analizaremos un escenario común en el Smart City Lab del grupo Tatami en el que la solución podría aportar la automatización de tareas y crearemos una propuesta de plan de implantación de nuestro proyecto en ese entorno.

Estructura de la memoria

El presente trabajo está estructurado de la siguiente manera:

- Introducción: se explica la motivación del proyecto, los objetivos de este y su impacto esperado. Además, se presenta la metodología del trabajo y la estructura de este.

- Estado del arte: se realiza un análisis exhaustivo del campo de estudio y de las plataformas IoT que existen actualmente.
- Análisis del problema: se describe la problemática encontrada después de hacer el análisis del estado de arte a la que nuestro proyecto dará respuesta para encontrar la justificación de este proyecto. Se tendrán en cuenta los riesgos que pueden afectar al proyecto.
- Diseño de la solución: se explica el proceso de diseño de la solución del problema expuesto con las diferentes alternativas consideradas.
- Implementación: se detalla el proceso de la implementación de la solución escogida. Se explicarán la arquitectura técnica y la implementación de los componentes más relevantes de la aplicación. Posteriormente, se explicarán las pruebas realizadas en la solución.
- Propuesta de implantación de la solución: se analizan las necesidades del Smart City Lab para crear un escenario de implantación de la solución.
- Conclusiones: se concluye el proyecto revisando si se cumplen los objetivos, los conocimientos adquiridos y los trabajos futuros derivados del resultado explorando posibles mejoras.

2. Estado del arte

En este apartado se va a dar un contexto tecnológico al presente trabajo. Primero, se empezará con lo básico, conocer qué es la IoT.

La internet de las cosas

La Internet de las cosas o IoT (por sus siglas en inglés, Internet of Things) es una gran red de dispositivos conectados entre sí. Los dispositivos conectados recopilan información de sus sensores para monitorizar circunstancias del mundo real, realizar acciones o tomar decisiones en base a ellos.

El IoT se puede ilustrar con nuestra vida diaria. Podríamos pensar en una televisión donde podemos cambiar los canales sin tocar el televisor con un mando a distancia. El concepto de Internet de las cosas es idéntico: podemos controlar casi todos dispositivos electrónicos de nuestro entorno de forma remota.

Desde electrodomésticos como un frigorífico y dispositivos con control remoto como una televisión o la puerta de un garaje hasta automóviles que nos brindan la ruta más corta y segura o climatizadores que mantienen la temperatura constante en un hogar.

Un ecosistema de IoT consiste en dispositivos inteligentes que ensamblan, envían y actúan sobre los datos que recopilan. En ocasiones, estos dispositivos se comunican con otros dispositivos conectados y actúan según la información que comparten. Los dispositivos realizan la mayoría de sus funciones sin la intervención humana. Sin embargo, las personas todavía pueden interactuar con ellos de varias maneras, como configurarlos, dar instrucciones o acceder a los datos.

Muchas soluciones de IoT también se basan en que este sistema puede aprender con el uso de inteligencia artificial, ayudando a que los procesos de datos sean más fáciles de usar y mucho más dinámicos. Es fundamental para los proyectos de análisis de Big Data, ya que puede generar muchos datos en tiempo real. [1]

Ahora que ya conocemos que es la IoT se puede comenzar a explicar qué son las plataformas que se asientan sobre este concepto.

Las Plataformas IoT

Definición y Funcionalidad de las Plataformas IoT

Una plataforma IoT es un conjunto de tecnologías, herramientas o servicios que facilita la gestión, monitorización y análisis de la información recopilada por dispositivos IoT de todas las clases. Las plataformas se encargan de almacenar y procesar los datos recibidos por parte de una multitud de dispositivos interconectados de forma que sea capaz de presentar información con un valor alto para las empresas a partir de los datos recibidos.

Estas plataformas nos permiten construir una base para un ecosistema digital en el que podemos crear soluciones optimizadas o productos basados en infraestructuras IoT sin afrontar el riesgo y

coste que supone el desarrollo de un software que interconecte los dispositivos, gestione los datos y presente la información siendo robusto, seguro y escalable.

Ahora que ya conocemos qué son las plataformas IoT vamos a ver los componentes que podemos encontrar en ellas

Componentes de las Plataformas IoT

Dependiendo de la plataforma y su objetivo pueden variar los componentes que la componen y sus funcionalidades, pero a grandes rasgos se podrían agrupar en los siguientes bloques principales:

- Bloque de Identificación: se identifica a cada dispositivo de la infraestructura de forma única. Este bloque es crucial para la gestión de la flota y su monitorización
- Bloque de Sensado: se encarga de recopilar los datos del entorno provenientes de los dispositivos IoT que son requeridos para la presentación y toma de decisiones en tiempo real
- Bloque de Comunicación: es la representación de todas las vías de comunicación que ofrece una plataforma IoT en la forma de protocolos. Estos protocolos son los que hacen posible el intercambio de información y la interacción entre los dispositivos.
- Bloque Computacional: se encarga del procesamiento de los datos. Es clave que tenga una baja latencia.
- Bloque de Servicios: se encarga de proporcionar las funcionalidades que ofrece la plataforma como herramientas de visualización, integraciones, creación de flujos o almacenamiento de datos.
- Bloque Semántico: es el que permite a las aplicaciones IoT “conocer el contexto” para tomar decisiones. Es esencial en soluciones que involucren la inteligencia artificial ya que facilita la interpretación de los datos.

Una plataforma IoT debe cumplir adicionalmente estos requisitos:

- Amplia conectividad: la capacidad de integración de una plataforma es esencial ya que facilita el uso de cualquier dispositivo que se esté o se vaya a utilizar y su interacción con el resto dentro de la infraestructura.
- Sistema de Gestión Centralizado: se debe poder gestionar, configurar y monitorizar los dispositivos a lo largo de su ciclo de vida.
- Ciberseguridad: las plataformas deben ser seguras y robustas frente a ataques ya que pueden manejar información que requiera protección por lo que las comunicaciones deben ir cifradas y el acceso a la plataforma debe ser con autenticación.

Tipos y Beneficios de las Plataformas IoT

Dentro de estas soluciones también podemos distinguir estos tipos de plataformas IoT:

- Plataformas en la Nube: permiten que los procesos operen en una infraestructura centralizada y escalable. Ideal para las empresas que se vayan a expandir junto con su sistema IoT
- Plataformas de Gestión de Conectividad: se enfocan en la red IoT, proporcionando la capa software e incluso hardware que hace posible la transmisión de datos por parte de los dispositivos de manera eficiente.

- Plataformas de Análisis Avanzado: se han diseñado para procesar grandes cantidades de datos. Son ideales para aplicaciones que utilizan inteligencia artificial o aprendizaje automático para realizar tareas particulares.
- Plataformas de Gestión de Dispositivos: se han confeccionado especialmente para manejar los dispositivos de forma que estén conectados, actualizados y monitorizados.
- Plataformas de Habilitación de Aplicaciones: brindan soluciones completas para el desarrollo, despliegue y gestión de sistemas IoT de forma que se ahorre la complejidad de una implementación habitual.

Las plataformas IoT han supuesto una revolución industrial como lo demuestra un estudio del Observatorio Nacional de Tecnología y Sociedad (ONTSI) que destaca un crecimiento del 17% al 28% en el uso de estas plataformas entre 2020 y 2022 en España. [2] [3] [4]

Plataformas IoT actuales

Ahora que ya conocemos qué son las plataformas IoT se van a explorar cuáles son las soluciones más populares en el momento del desarrollo del proyecto. Actualmente, podemos destacar los siguientes productos:

AWS IoT

Amazon Web Services IoT es una de las soluciones más populares de la industria a la hora de crear aplicaciones IoT ya que ofrece una amplia gama de servicios de gestión de dispositivos y de sus datos para tareas como análisis o monitorización. AWS IoT Core permite concectar una gran variedad de dispositivos a través de una multitud de protocolos, pero el punto fuerte de esta plataforma es su integración con otros servicios en la nube como Lambda. [5]

Thingsboard

ThingsBoard es una plataforma *open-source* IoT para la recolección de datos, procesamiento, visualización y gestión de dispositivos. Permite que estos se conecten a mediante protocolos estándar para IoT como MQTT, CoAP y HTTP y permite un despliegue tanto en el *cloud* como *on-premise*. ThingsBoard combina escalabilidad, tolerancia a fallos, rendimiento. Su principal atractivo es su forma sencilla de crear *dashboards* en los que representar los datos que se obtienen de la telemetría y de su motor de reglas que permite crear flujos de actuación mediante una interfaz amigable basada en bloques. [6]

Azure IoT Hub

Azure IoT Hub es la alternativa que ofrece Microsoft para la gestión de dispositivos IoT. Esta plataforma destaca por su enfoque en la seguridad y por su integración en otros servicios de Azure como Power BI lo cual nos permite recopilar los datos y presentarlos en reportes con una integración sencilla o utilizar Azure Machine Learning para alimentar de datos un modelo de inteligencia artificial. [7]

3. Análisis del problema

Ahora que el contexto tecnológico se ha explicado podemos pasar a analizar la información recopilada para observar si las plataformas actuales ya cubren las necesidades de operaciones de gestión o en cambio necesitan una extensión.

Observaciones

Se ha visto durante la investigación para el estado del arte que las plataformas IoT han puesto sobre la mesa una gran cantidad de funcionalidades que han sido adoptadas por la industria. En concreto, podemos destacar la conectividad de los dispositivos y el procesamiento de los datos recopilados gracias a integraciones con servicios existentes de procesamiento de datos. Sin embargo, también se identifican carencias a la hora de requerir un alto grado de personalización y control de los dispositivos IoT:

- Falta de capacidades de meta-operaciones: pese a que algunas plataformas como Azure IoT Hub tienen funcionalidades de actualización de firmware se siguen quedando sin abarcar otras operaciones de plataforma necesarias en infraestructuras complejas. Por ejemplo, no podemos mandar a nuestros dispositivos la orden de que ejecuten un script en el que por ejemplo se descargan el nuevo certificado de seguridad.
- Necesidad de planificación de las meta-operaciones: además de la falta de meta-operaciones vemos como no tenemos módulos para planificarlas como podrían ser los CronJobs de Kubernetes. Infraestructuras complejas pueden requerir que la plataforma lance operaciones de actualización o reinicio de forma planificada.
- Limitaciones a la integración: las soluciones más comerciales ofrecen las funcionalidades más avanzadas solo en el contexto de sus propios productos. Por ejemplo, podemos pensar en usar Lambda de AWS para automatizar tareas, pero incurriríamos en un aumento de costes, dificultad de migración y complejidad que no se tendrían si se dispusiera de una solución abierta.
- Riesgos de seguridad: la seguridad es muy importante en las infraestructuras IoT ya que a menudo involucran dispositivos con una capacidad limitada de cómputo por lo que son más vulnerables a ataques externos. No poder automatizar operaciones como actualizar software o renovar certificados sin intervención humana supone un riesgo de seguridad a cuenta.

Justificación del proyecto

Dado el análisis anterior, se justifica la necesidad de desarrollar un sistema o protocolo con el cuál las plataformas IoT puedan abordar sus limitaciones y ofrecer a los usuarios funcionalidades más avanzadas. La solución propuesta basará su desarrollo en los siguientes aspectos:

1. Implementación de una interfaz para el lanzamiento de meta-operaciones personalizables: la extensión incluirá el desarrollo de una funcionalidad que ofrezca un marco en el que los usuarios puedan lanzar este tipo de operaciones de forma sencilla y posteriormente en sus dispositivos puedan implementar la respuesta conforme a sus necesidades.

2. Planificación: la solución podrá planificar las operaciones para que se ejecuten conforme a una planificación. Por ejemplo, queremos que todos los días uno del mes se mande la operación de actualizar el software.
3. Monitorización de las operaciones: la solución ofrecerá la capacidad de monitorizar los resultados de las operaciones.
4. Mantenible: la arquitectura propuesta debe estar diseñada de forma que permita añadir nuevas funcionalidades sin comprometer el rendimiento.

Además, para favorecer la integración se elige la primera plataforma integrable con nuestra solución a Thingsboard ya que su motor de reglas puede hacer la integración con nuestra solución muy sencilla.

Riesgos y consideraciones del proyecto

En el desarrollo del proyecto se deben tener en cuenta los siguientes riesgos de la solución propuesta:

- Complejidad técnica: la integración de nuevas funcionalidades en plataformas existentes puede complicar el desarrollo del proyecto ya que podemos introducir errores o vulnerabilidades en la infraestructura.
- Seguridad: al incorporar este grado de personalización podemos aumentar las capacidades de los atacantes de atacar la infraestructura. Por ejemplo, interceptando los mensajes y modificándolos de forma maliciosa.
- Escalabilidad: si el número de dispositivos es muy grande podríamos no ser capaces de gestionar toda esa carga de mensajes por parte de los dispositivos.
- Costes de implementación: debido a la complejidad técnica del proyecto puede pasar que el tiempo y coste en recursos sea mayor al esperado.
- Aceptación por parte de los usuarios: dado que si la solución propuesta no está dentro de la propia plataforma los usuarios no la van a encontrar familiar en primera instancia. Podría pasar que los usuarios no la utilicen y sigan recurriendo a sus soluciones anteriores.
- Dependencia de la infraestructura existente: aunque en el desarrollo del proyecto se intente que la solución sea adaptable a otros entornos, dependemos inicialmente de que haya una instancia de una plataforma IoT que aporte valor sobre la información que se recopila de los dispositivos o las acciones que son capaces de hacer.

En resumen, la solución propuesta deberá abordar dar una solución viable a las limitaciones observadas en las plataformas IoT actuales. Por otra parte, se deberán considerar los distintos riesgos presentes en el proyecto para cumplir con los objetivos.

4. Diseño de la solución

La solución propuesta como resultado del desarrollo de este proyecto aborda las limitaciones que se han observado en las plataformas IoT actuales. En concreto, se implementará una solución que permita a los usuarios crear operaciones y enviarlas a los dispositivos para que ellos mismos puedan dar una respuesta individualizada satisfaciendo las necesidades de cada uso particular. Por ejemplo, no todos los dispositivos se actualizan de la misma forma, pero cualquier dispositivo que tenga una rutina de actualización implementada podría adaptarlo a nuestro sistema.

Soluciones planteadas

Durante el transcurso del proyecto surgen tres ideas para resolver el problema en Thingsboard:

- Desarrollo de módulo personalizado para Thingsboard: ya que se cuenta con un sistema de código abierto se valora implementar un módulo que incorpore las capacidades de gestión avanzada necesarias en las plataformas IoT. El módulo ofrecería una interfaz a los usuarios para usar widgets con los que lanzar las operaciones desde cualquier *dashboard* de la organización que cumpla con las políticas de seguridad.
- Desarrollo se soluciones basadas en microservicios:
    - Desarrollar un microservicio API REST en el que los dispositivos se den alta para recibir las operaciones y cuando se crea una operación se envía a todos los dispositivos registrados y luego cada dispositivo enviaría información sobre el ciclo de vida de la operación.
    - Desarrollar un microservicio API REST que publique las operaciones creadas en colas de mensajería. Los dispositivos consultarían las operaciones en las colas y luego publicarían los resultados con el mismo mecanismo.
- Adaptar la funcionalidad de comandos RPC de Thingsboard: la plataforma nos ofrece un módulo con el cual podemos enviar procedimientos de forma remota a un dispositivo concreto. En el proyecto se ampliarían estas capacidades para poder enviar los procedimientos remotos a flotas enteras de dispositivos y así conseguir la automatización que se busca.

Entre estas alternativas se deberá escoger la más adecuada para satisfacer la problemática encontrada y que minimice los riesgos asociados.

Comparativa de las alternativas y solución elegida

A continuación, vamos a reflejar en una tabla los puntos fuertes y puntos débiles de cada alternativa para poder hacer una valoración de cuál es la adecuada para el proyecto:

Alternativa de solución	Factores a favor	Factores en contra
Módulo personalizado en Thingsboard	• No necesitaríamos desplegar más componentes. • La solución sería integrada nativamente en Thingsboard. • Interfaz unificada • Gestión centralizada	• Difícil de mantener. Futuras actualizaciones de la plataforma podrían comprometer el funcionamiento del módulo. • Coste técnico de comprender y modificar el código de Thingsboard. • Limitaciones del framework. • La cantidad a pruebas a realizar sería demasiado grande ya que podríamos afectar a muchísimas partes de la aplicación.
Modificación del módulo RPC de Thingsboard	• No necesitaríamos desplegar más componentes. • La solución sería integrada nativamente en Thingsboard. • RPC es un protocolo bastante extendido e integrable.	• Difícil de mantener. Futuras actualizaciones de la plataforma podrían comprometer el funcionamiento del módulo. • Coste técnico de comprender y modificar el código de Thingsboard.
Microservicio gestor de operaciones	• Altamente escalable • Control sobre los dispositivos. • Altamente adaptable a otras infraestructuras. • Mantenible gracias a la independencia tecnológica de la plataforma	• Debemos desplegar al menos un componente adicional. • Complejidad en la sincronización. • El dispositivo debe implementar una lógica costosa para estar siempre listo.
Microservicio publicador de operaciones	• Altamente escalable • Tolerancia a fallos	• Debemos desplegar al menos un

	• Separación de responsabilidades • El uso de colas de mensajería libera carga tanto en el microservicio como en los dispositivos que reciben las operaciones. • Las colas de mensajería nos proporcionan mecanismos de filtrado que pueden ser muy útiles. • Altamente adaptable a otras infraestructuras. • Permite asincronía. • Mantenible gracias a la independencia tecnológica de la plataforma.	componente adicional. • Sin control de los dispositivos. • Latencia.

Contemplando todos los puntos y pensando en todos los riesgos que puede tener el proyecto se cree que la mejor opción es la de desarrollar una API REST que ofrezca la posibilidad de crear operaciones para que se publiquen inmediatamente o planificarlas para que se publiquen con una frecuencia específica mediante colas de mensajería. Además, el mismo microservicio recibirá por parte de los dispositivos (los que lo deseen implementar) los resultados de las operaciones. Esta forma de afrontar el problema nos da mucha flexibilidad y en el momento actual donde los despliegues de contenedores son un estándar podríamos hacer que nuestra solución sea fácilmente integrable en otras plataformas que permitan enviar peticiones HTTP.

Diseño de la solución propuesta

Con la alternativa escogida se puede proceder al diseño de la solución. En primer lugar, vamos a describir la arquitectura de la solución.

Arquitectura

"La arquitectura describe los patrones y las técnicas que se utilizan para diseñar y desarrollar aplicaciones". [8]

En nuestro caso la arquitectura de nuestra aplicación debe ser sencilla ya que no se puede sobrecargar una red que puede estar repleta de dispositivos. Por esa misma razón, el sistema debe ser escalable ya que debe dar servicio a un número de agentes que pueden variar en el tiempo. Por otra parte, debemos tener presente que esta solución deberá existir en un entorno donde ya hay una plataforma IoT y otros servicios como aplicaciones de inteligencia artificial, procesamiento de datos o ciberseguridad.

Para empezar, se deben describir los componentes con sus funciones y las tecnologías que se plantean para el desarrollo (se describirán más adelante en el proceso de implementación):

- Base de datos:
  - Función: almacenar y servir la información a la API REST.
  - Tecnologías utilizadas:
    - MySQL
- API REST gestora de las operaciones:
  - Funciones: recibir la información de las operaciones en formato JSON mediante una interfaz REST, publicarlas mediante mensajes o planificar su publicación si requieren planificación.
  - Tecnologías utilizadas:
    - Java 17
    - Spring Boot
    - Spring data JPA
    - Mapstruct
    - Lombok
    - MySql-connector
    - Paho MQTT
- Bróker de mensajería:
  - Funciones: intermediario que se encarga de coordinar los mensajes entre el publicador y los dispositivos.
  - Tecnologías utilizadas:
    - MQTT
    - Thingsboard/Mosquitto. Si la solución existente ya incluye un Broker MQTT no tiene sentido añadir otro con nuevas políticas de seguridad, identificación...por lo que lo recomendable sería utilizar el que ya está funcionando. En el caso de Thingsboard ya incluye un Broker pero se deja la alternativa de Mosquitto en el caso de que se desee utilizar uno independiente para transmitir las operaciones.
- Plataforma IoT:
  - Funciones: en el ámbito del presente proyecto es desde donde se hacen las peticiones a la API REST con la información de las operaciones.
  - Tecnologías utilizadas:
    - Thingsboard.
    - JavaScript
- Dispositivos:
  - Funciones: en este proyecto reciben las operaciones con el protocolo MQTT y responden con los estados de la operación si es necesario con RECIBIDA, RECHAZADA y COMPLETADA.

A continuación de muestra un diagrama donde podemos ver los componentes representados y como se relacionan entre sí.

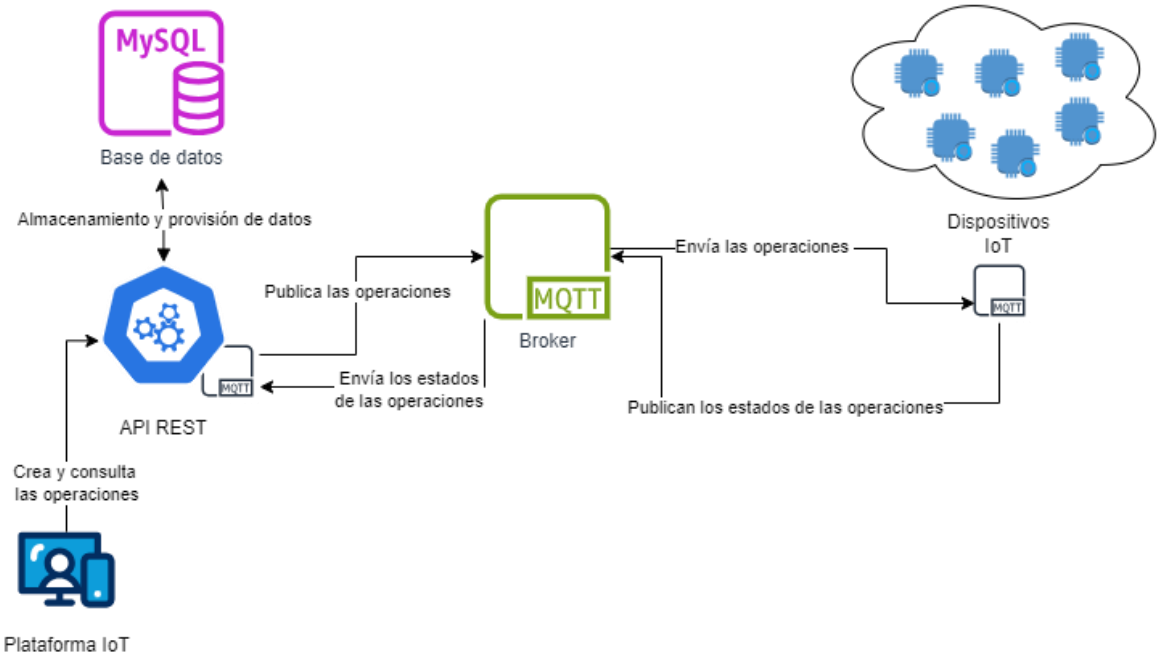

Ilustración 1-Arquitectura del proyecto

Ahora que conocemos a grandes rasgos como se va a ver conceptualmente nuestra aplicación debemos detallar los flujos que se deben implementar

Procesos

Antes de desarrollar técnicamente el modelo de datos se debe conocer el caso de uso principal. Con el caso de uso podemos definir las entidades sobre las que se basa la aplicación. El sistema pretende proporcionar a los usuarios una interfaz para crear operaciones, así que partiremos desde la base de que el cliente de la API crea una operación. Se van a preimplementar tres tipos de operaciones (SCRIPT, UPDATE, CUSTOM) pero no tienen particularidades en el flujo del proceso de gestión de las operaciones.

En primer lugar, cuando se recibe una petición REST con una operación esta se guarda en base de datos. Posteriormente, se envía de forma inmediata a todos los dispositivos si no es planificada. Si la operación requiere una planificación, se crea una planificación para que el servicio encargado la añada a las programadas para que cuando se cumpla la planificación la operación sea enviada. En el momento en el que se envía la operación se publica en un tópico MQTT (este tópico se confecciona para que permita a los dispositivos filtrar por tipos de operaciones, grupo...) la información de la operación:

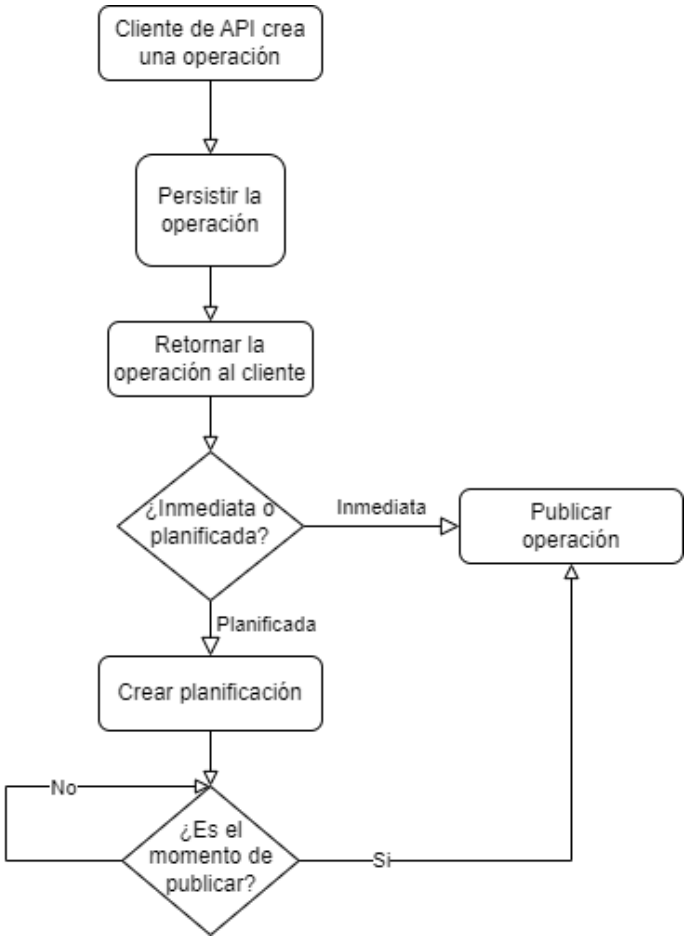

Ilustración 2- Proceso de creación de una operación

Cuando esta operación es recibida por los dispositivos la aplicación está preparada para recibir mensajes de recibido, completada y rechazada mediante mensajes MQTT. Cuando llega un mensaje se guarda en base de datos para consultar:

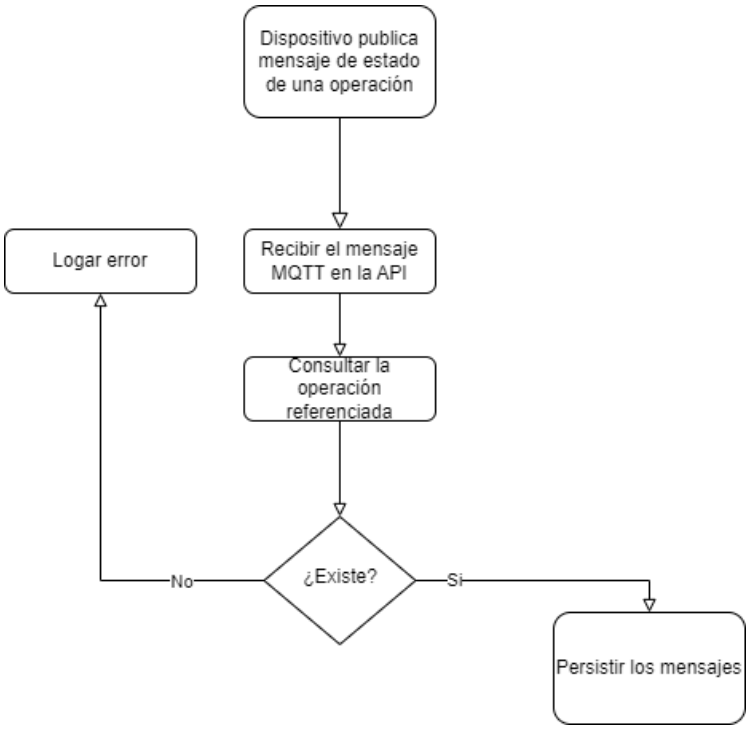

Ilustración 3 - Proceso de recepción de mensajes

A continuación, un usuario puede querer consultar todos los mensajes de una operación para ver si todos los dispositivos la han recibido o completado mediante una petición REST:

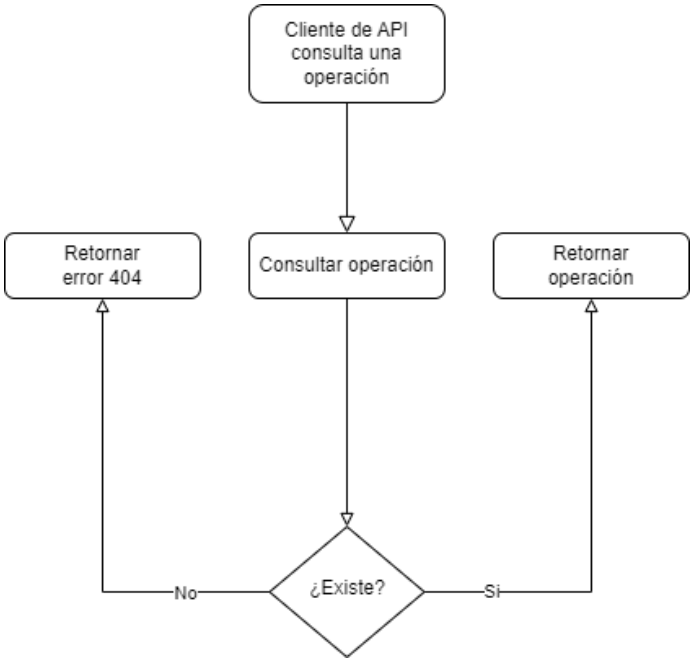

Ilustración 4 - Proceso de consulta de una operación

Por otra parte, un usuario podría querer actualizar o cancelar una operación planificada porque detectó un error, y, al recibir la petición REST pertinente, modificaremos la entidad asociada con la nueva información o eliminaremos su planificación manteniendo los datos de la operación por si requieren consultarse después:

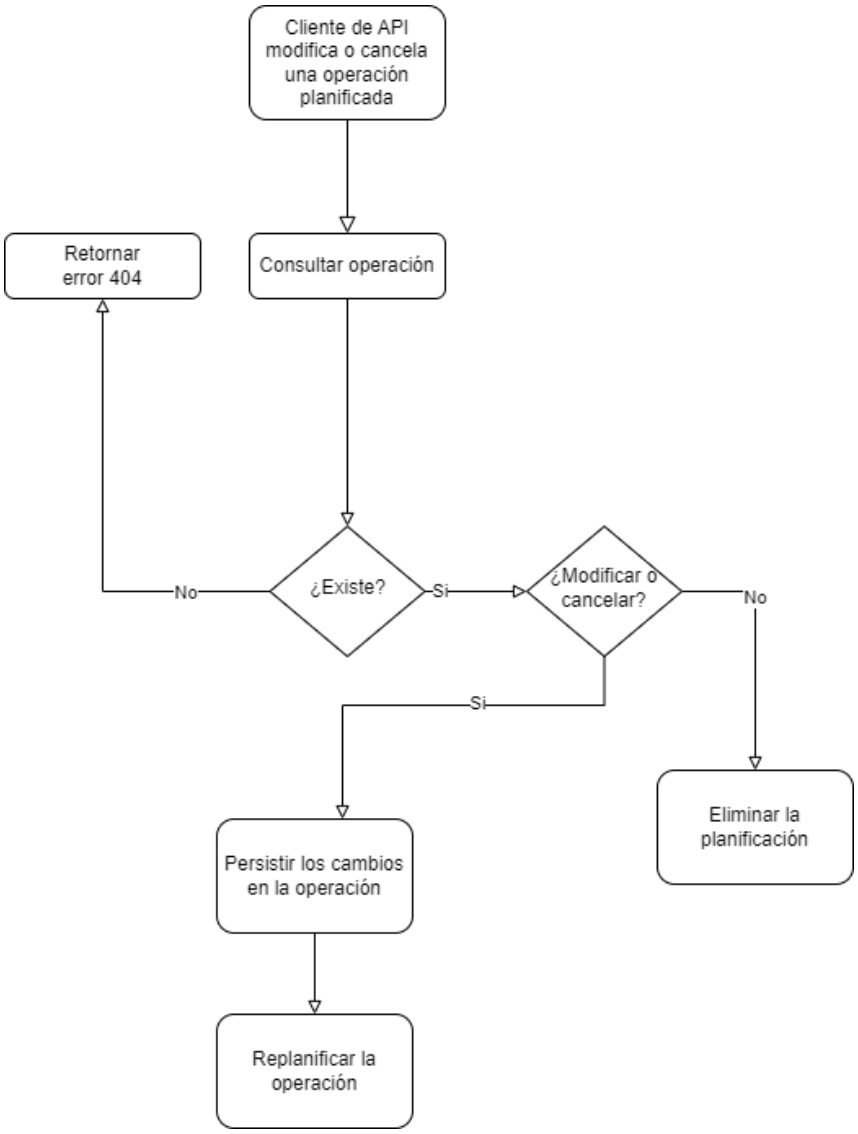

Ilustración 5 - Proceso de modificación o cancelación de una operación

Ahora que se ha expuesto la arquitectura del sistema y conocemos los procesos que se implementarán en la aplicación se puede describir el modelo de datos sobre el que se trabajará.

Modelo de datos

Según la información que podemos extraer y las posibles personalizaciones que un usuario pueda necesitar en sus operaciones necesitaremos las siguientes entidades:

- Operación: representa y estructura la información asociada a una operación que se crea y envía a los dispositivos.
    - Id.
    - Fecha de creación.
    - Fecha de modificación.
    - Frecuencia.
    - Iteraciones.
    - Fecha de planificación.

- Estado.
- Tipo de dispositivo.
- Grupo de dispositivos.
- Tipo.
- Mensajes.

En el caso de que la operación sea del tipo SCRIPT deberá incluir la siguiente información adicional:

- Fecha.
- Autor.
- Sistema operativo.
- Lenguaje.
- Versión de firmware máxima.
- Versión de firmware mínima.
- Nombre.
- Tipo de script.
- Versión.
- Datos

En el caso de que la operación sea del tipo UPDATE deberá incluir la siguiente información adicional:

- Paquetes.

En el caso de que la operación sea del tipo CUSTOM deberá incluir la siguiente información adicional:

- Parámetros.

- Mensaje: representa y estructura los mensajes que se reciben por parte de los dispositivos
  - Id.
  - Contenido.
  - Fecha.
  - Tipo de dispositivo.
  - Grupo de dispositivo.
  - Id de dispositivo.
  - Tipo de mensaje.
  - Operación.
- Paquete: representa y estructura los paquetes que se deben actualizar en una operación de tipo UPDATE
  - Operación.
  - Nombre.
  - Versión.
- Planificación: representa y estructura la planificación de una operación planificada:
  - Cron Expression.
  - Iteraciones completadas.
  - Operación.

Para comprender mejor las relaciones entre las entidades se va a ilustrar un diagrama de clases en el que se podrán apreciar de forma visual:

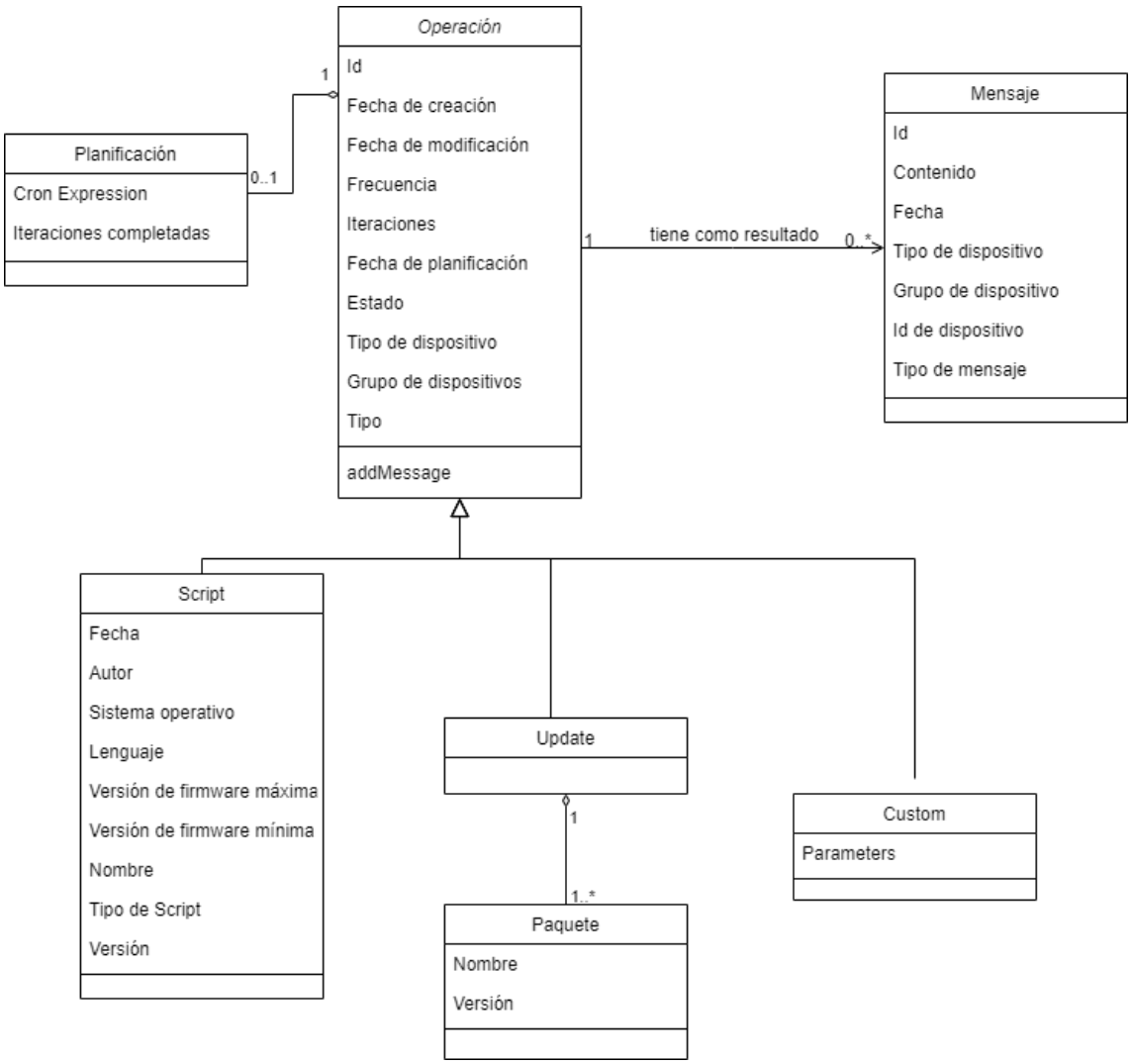

Ilustración 6 - Diagrama de clases

5. Implementación

Introducción

En el siguiente apartado se va a explicar el proceso de implementación de la solución detallada. Partimos de un diseño de una API Rest dedicada a la gestión de operaciones destinadas a dispositivos IoT. Esta gestión incluye el envío, almacenamiento y consulta de la información relacionada con las operaciones. Durante la explicación se verá cómo se ha confeccionado el código para cumplir los requisitos del proyecto y a la vez cumplir con patrones comunes en el desarrollo de microservicios. Para empezar, se hablará de las herramientas y el entorno de desarrollo en el que se ha realizado el proceso más en detalle que en la fase de diseño.

Herramientas y entorno de desarrollo

Lenguaje de programación y frameworks

El desarrollo del microservicio se va a realizar en Java 17 por la experiencia del desarrollador en este lenguaje. Java es uno de los lenguajes más populares en el desarrollo de microservicios gracias al framework Spring Boot que es el que se adoptará en nuestra aplicación por lo que se sabe de antemano que se podrá contar con numerosa documentación técnica. Este framework aporta mucha modularidad y flexibilidad al proyecto ya que facilita la labor de inyectar dependencias y la creación de un código legible, mantenible y probable.

Base de datos

La base de datos que será la base sobre la que operará el microservicio será una instancia de MySQL. Se ha elegido una base de datos relacional porque permite estructurar los datos de manera uniforme y obtener de ella información significativa. En el caso de las operaciones que gestionará nuestra API, los datos deben vivir en un entorno que acepte nuevos tipos de operaciones y mensajes de los dispositivos complejos, por lo que contar con las funcionalidades de tal base de datos es crucial. Aunque el modelo de datos no relacional nos pueda dar más flexibilidad en el corto plazo, en las nuevas versiones de MySQL contamos con tipos de datos más flexibles como JSON que nos puede proporcionar esa flexibilidad.

[9]

Mensajería

La tecnología que utilizaremos para la mensajería entre la aplicación y los dispositivos es MQTT. En primer lugar, utilizamos esta tecnología porque nos permite implementar el patrón publicador suscriptor. Este patrón es clave en nuestro diseño ya que nunca se va a conocer la cantidad ni la ubicación de los dispositivos de la red IoT por lo que proporcionar una interfaz unificada en la que todos los agentes se pueden comunicar mediante mensajes hace posible este desarrollo. Se utiliza el protocolo por los siguientes beneficios:

- Es ligero y eficiente: nos permite integrarlo en un gran abanico de capacidades de cómputo de los dispositivos.
- Escalable: podemos conectar una gran cantidad de dispositivos con una sobrecarga mínima gracias a su diseño eficiente.

- Fiable: MQTT está diseñado para aportar distintos grados de fiabilidad mediante su QoS (calidad de servicio) que van del 0 al 2:
    - 0: Máximo una vez.
    - 1: Al menos una vez.
    - 2: Una vez.
- Seguro: ofrece muchas opciones para cifrar mensajes y autenticar los dispositivos, aunque debemos evitar quedarnos solamente con la autenticación por usuario y contraseña que si pudiera ser vulnerable
- Soporte: es un protocolo que está altamente integrado en lenguajes de programación.

Nuestra aplicación utilizará la librería Paho MQTT de Java para integrar el protocolo en el código. Se publicarán los mensajes en topics que permitan a los dispositivos filtrar las operaciones deseadas en el broker MQTT (para el que utilizaremos Mosquitto en el desarrollo, pero si ya existe uno en la infraestructura lo conveniente sería utilizar el ya presente) y los dispositivos las recibirán mediante el concepto de suscripción. Por otra parte, este protoclo nos ofrece los *Retained Messages* que nos pueden ser útiles para que dispositivos que hayan sufrido desconexiones reciban operaciones pasadas.[10][11]

Contenedores

Para el despliegue de los diferentes componentes en el entorno de desarrollo se optará por la contenedorización ya que permite desplegar instancias del software que se necesiten prácticamente sin realizar instalaciones. Por ejemplo, en un minuto se puede tener un contenedor de una base de datos MySQL o un bróker MQTT Mosquitto con el que se puede trabajar para la integración del microservicio. Además, la propia aplicación debe poderse desplegar contenerizada en soluciones como Docker compose, Docker Swarm o Kubernetes que son el estándar en el despliegue de microservicios actualmente. En concreto, vamos a utilizar Docker y Docker compose para el despliegue de nuestros componentes.

Control de versiones

El control de versiones es una de las herramientas esenciales hoy en día ya que da una trazabilidad completa de los cambios que ha habido en un repositorio de archivos. GIT es la herramienta elegida para este cometido y se alojará el repositorio en la plataforma GitHub.

Herramientas de pruebas y validación

En relación con el lenguaje de programación escogido utilizaremos sus principales librerías para construir tests: Junit y Mockito. Por otra parte, se hará uso de SwaggerUi para hacer las peticiones REST en las pruebas de validación de la API.

Arquitectura de la aplicación

Visión general

En el capítulo anterior se ha detallado el diseño del sistema a nivel de componentes y, pero no se ha indagado en el patrón que se va a seguir para estructurar los componentes de la aplicación que en este caso ser el hexagonal. Antes de ver cómo se utilizará en el proyecto, se explicará en qué consiste.

La Arquitectura Hexagonal o arquitectura de puertos y adaptadores es un patrón de diseño de aplicaciones que se basa en separarlas en diferentes capas donde la lógica de negocio de la aplicación se encuentra en el centro o núcleo. Este núcleo queda accesible solamente a través de puertos y adaptadores:

- Puertos: interfaces que definen como se interactúa con el núcleo desde el exterior.
- Adaptadores: implementan los puertos para conectar los sistemas externos con el núcleo.

[12]

Los beneficios que proporciona la elección de esta arquitectura al proyecto son los siguientes:

- Independencia de la infraestructura: permite que la lógica de negocio quede separada de funcionalidades de comunicación con sistemas externos lo que facilita que podamos cambiarlos o actualizarlos sin afectar a esta,
- Facilidad para las pruebas: podemos probar la lógica de negocio de manera aislada aprovechando las librerías de testing para crear mocks de los adaptadores.
- Flexibilidad y evolución: facilita la integración con nuevos módulos o sistemas que puedan ser necesarios en el futuro.
- Reutilización: se puede reutilizar la lógica de negocio en cualquier parte de la aplicación ya que no está ligada a ningún interfaz externo.

Teniendo ya claro el concepto de arquitectura hexagonal se puede ver su reflejo en el desarrollo de este proyecto.

Adaptadores

Uno de los componentes de una arquitectura hexagonal son los adaptadores y en el caso de la aplicación necesitaremos utilizarlos para la comunicación con sistemas externos:

- Controladores REST: serán las clases que se utilicen como interfaz para recibir peticiones REST. Aquí residirá la especificación de cómo serán las llamadas.
- Gateway MQTT: gracias a la librería PahoMQTT se creará una interfaz capaz de conectarse al bróker MQTT y ofrecer la funcionalidad de publicar o suscribirse a un topic.
- Repositorios: mediante la tecnología de Spring Data JPA y el adaptador de MySQL se crearán repositorios para cada entidad necesaria que encapsulen todas las operaciones CRUD que se necesiten en la capa de servicios.

Estos adaptadores serán utilizados por el núcleo para integrarse con otros sistemas.

Núcleo de lógica de negocio

Para cumplir con todos los procesos que se deben implementar crearemos servicios que encapsulen la lógica de negocio mediante servicios de Spring Boot para que el framework gestione su ciclo de vida y dependencias. Los servicios quedarán distribuidos de la siguiente manera:

- Servicio de gestión de mensajes: servicio al cual llamaremos para guardar o obtener la información de los mensajes resultantes del lanzamiento de operaciones.

- Servicio de gestión de operaciones: servicio al cual llamaremos para guardar o obtener la información de las operaciones.
- Servicio de planificación: servicio al cual llamaremos para planificar o cancelar una planificación de una operación y que se encargará de preparar todas las tareas para ejecutar las planificaciones.
- Servicio de envío/recepción de mensajes: servicio que se encarga de gestionar la preparación de envío de los mensajes y especificará las rutinas que seguir a la hora de recibir los mensajes de los dispositivos.

Componentes

Por último, tenemos los componentes. Los componentes son los que sin tener lógica de negocio ejecutan operaciones en el dominio de la aplicación. En la aplicación se construirá un componente que mapeará los DTO con las entidades para separar y conectar los datos de sistemas externos con los del dominio de la aplicación. De esta forma podemos incorporar o retirar información que resulte útil o no de cualquiera de las dos partes.

Para tener una visión más completa de la composición de la aplicación se proporciona el siguiente diagrama:

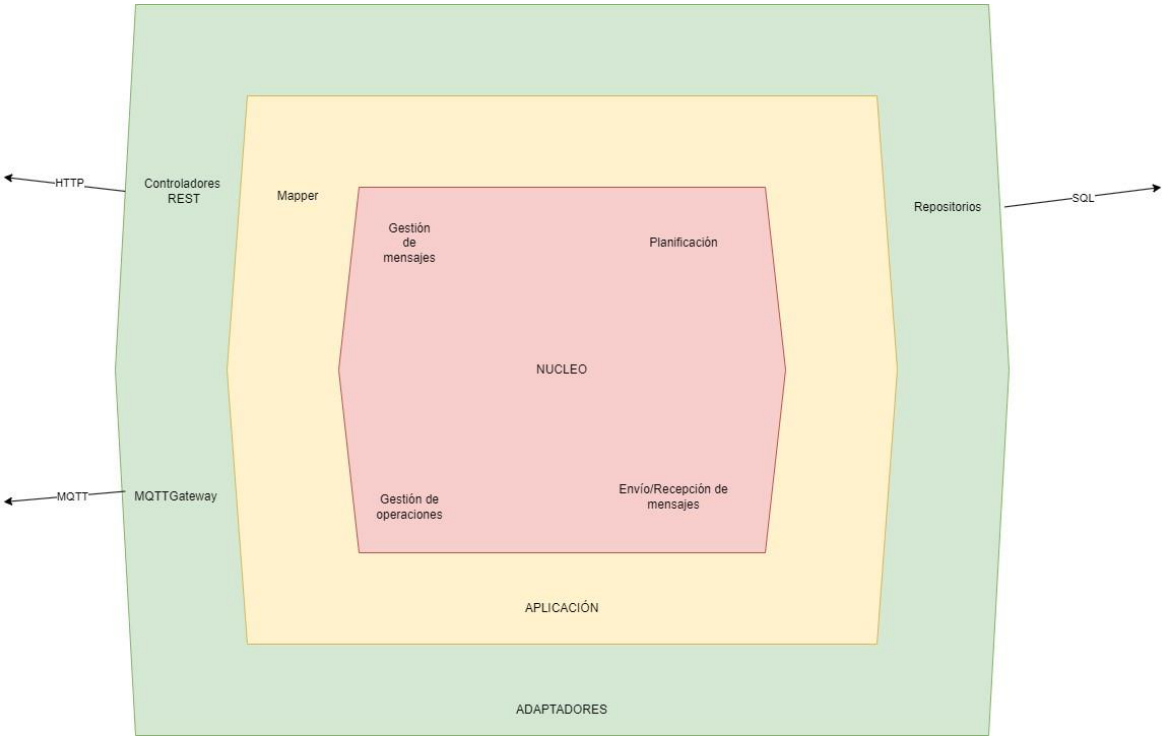

Ilustración 7 - Diagrama de la arquitectura de la aplicación

Desarrollo de la solución

Ahora que ya se ha explicado la arquitectura de la aplicación ya se puede detallar el proceso de implementación de los componentes. Durante este apartado se explicará cómo se desarrollan las distintas partes del software a nivel de código e iremos analizando fragmentos de este o configuración relevante para entender el desarrollo:

Modelo de datos

Para representar el modelo de datos en una aplicación que utiliza spring boot debemos crear una clase entidad. Esta clase deberá importar las anotacions necesarias del paquete jakarta.persistence comenzando por @Entity que es la que indica que es la que indica que se trata de una clase de este tipo. En esta clase definimos los atributos como si de una clase habitual se tratase, pero en el momento que tenemos que referenciar otra entidad deberemos utilizar una anotacion para especificar la relación entre ellas (One to One entre Operación y Planificación, por ejemplo). En el siguiente fragmento de código se puede apreciar la clase Operation que representa una operación:

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@EntityListeners(AuditingEntityListener.class)
@Data
@NoArgsConstructor
public class Operation {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"operation_seq")
    @SequenceGenerator(name = "operation_seq", sequenceName =
"operation_seq", allocationSize = 50)
    private Long id;

    @Enumerated(EnumType.STRING)
    private OperationType type;

    @Enumerated(EnumType.ORDINAL)
    private OperationStatus status;
    @Embedded
    private OperationTargetDeviceData targetDevice;
    @Embedded
    private OperationSchedulingData schedulingData;

    @OneToMany(mappedBy = "operation", fetch = FetchType.LAZY, cascade =
CascadeType.ALL)
    private List<Message> messages;
    @OneToOne(mappedBy = "operation", optional = true, orphanRemoval =
true, cascade = CascadeType.ALL)
    private OperationSchedule schedule;
    @CreatedDate
    private Instant creationDate;
    @LastModifiedDate
    private Instant modificationDate;

    public void addMessage(Message message) {
    if (this.messages == null) {
        this.messages = new ArrayList<>();
    }
    this.messages.add(message);
    }
}
```

Aquí hemos podido ver cómo se establecen las relaciones que tiene una operación con el resto de las entidades y además hemos visto como se implementa la herencia en las entidades lo cual es muy importante para seguir implementando operaciones en el futuro (actualmente tenemos SCRIPT, UPDATE y CUSTOM) [13]. Otro detalle para destacar son los campos con la anotación @Embedded que nos permiten agrupar atributos en otras clases para así no sobrecargar la clase

principal y hacer el código mucho más legible y mantenible. También vemos otras anotaciones que nos proporcionan funcionalidades adicionales como establecer el valor de la fecha de creación y modificación de forma automática o la forma de generación de la secuencia para la clave primaria.

Por último, también hemos visto un par de anotaciones de la librería Lombock para que genere de forma automática un constructor sin argumentos (esencial para Spring Data JPA) con @NoArgsConstructor y que genere los *getters* y *setters* de los atributos de la clase.

Clases DTO

De forma paralela al modelo de datos también se necesita su representación de cara a los sistemas externos. En este caso solamente debemos crear una clase con los mismos atributos o los que queramos utilizar de la clase entidad, pero sin las anotaciones. En particular se quiere mostrar como quedaría la clase DTO de operación ya que se utilizan anotaciones de la librería Jackson (incluida en Spring Boot) para especificar sus subtipos:

```
@Data

@NoArgsConstructor

@JsonTypeInfo(use = Id.NAME, include = As.EXISTING_PROPERTY, property =
"type", visible = true)
@JsonSubTypes({ @JsonSubTypes.Type(value = CustomOperationDTO.class, name =
"CUSTOM"),
    @JsonSubTypes.Type(value = ScriptOperationDTO.class, name = "SCRIPT"),
    @JsonSubTypes.Type(value = UpdateOperationDTO.class, name = "UPDATE")})
public class OperationDTO {
    private Long id;
    private OperationType type;
    private OperationStatus status;
    private OperationTargetDeviceDataDTO targetDevice;
    private OperationSchedulingDataDTO schedulingData;
    private Instant creationDate;
    private Instant modificationDate;
}
```

Repositorios

Los repositorios son los adaptadores que permiten ejecutar consultas a la base de datos respecto a una entidad definida. En el caso de estudio se ha visto que será necesario como mínimo contar con un repositorio de mensajes y uno de operaciones.

Para definir un repositorio deberemos crear una interfaz que extienda la clase JpaRepository del paquete org.springframework.data.jpa.repository.JpaRepository pasando como parámetros la clase correspondiente a la entidad que queremos gestionar con el repositorio y el tipo de la clave primaria de esa entidad (la que lleva la anotación @Id). La clase padre ya nos da métodos muy útiles como getById o sabe, pero nosotros podemos definir consultas a nuestro gusto especificando los métodos e incluyendo la consulta en el lenguaje JPQL dentro de la anotación @Query. En el siguiente fragmento de código podemos ver cómo se pueden implementar algunas consultas en este lenguaje:

```
public interface MessageRepository extends JpaRepository<Message, Long> {
    @Query("SELECT m FROM Message m WHERE m.operation.id = :operationId AND
m.type = :type")
    List<Message> findByOperationIdAndType(@Param("operationId") Long
operationId, @Param("type") MessageType type);

    @Query("SELECT m FROM Message m WHERE m.operation.id = :operationId
ORDER BY m.date DESC")
    List<Message> findByOperationId(@Param("operationId") Long operationId);
}
```

Mappers

Los *mappers* son las interfaces que nos generan el código que nos hace posible convertir entre DTOs y entidades. Estas clases son posibles gracias al paquete org.mapstruct. Aquí podemos ver la base de un *mapper* de una operación y definición para uno de sus subtipos:

```
public interface BaseOperationMapper<E extends Operation, D extends
OperationDTO> {
    D toDto(E entity);

    List<D> toDtos(Collection<E> entities);

    @InheritInverseConfiguration
    E toEntity(D dto);

    List<E> toEntities(Collection<D> dtos);

    E merge(E source, @MappingTarget E target);

    D merge(D source, @MappingTarget D target);
}

@Mapper(componentModel = "spring")
public interface ScriptOperationMapper extends
BaseOperationMapper<ScriptOperation, ScriptOperationDTO> {
}
```

[14]

Componentes

En el proyecto se tienen como componentes dos clases que nos ayudan de formas distintas:

- MqttGateway: funciona como adaptador para operar sobre el bróker MQTT proporcionando la posibilidad de publicar mensajes o suscribirse a ellos. Cuando el componente se crea debe conectarse al bróker de forma inmediata. Esta clase también utiliza una clase del patrón *factory* cuya función es crear el cliente MQTT de forma que ayude a las pruebas y la lógica esté desacoplada. En ambas se utiliza la librería Paho MQTT
- Operation Mapper: unifica todos los *mappers* creados proporcionando un *mapper* centralizado para hacer más mantenible ir añadiendo operaciones en el futuro. Se implementa añadiendo los mismos métodos que se han visto que tiene una clase *mapper* pero filtrando internamente por el tipo para escoger el método con el mismo nombre del *mapper* adecuado.

En todos ellos se debe comenzar la cabecera de la clase de implementación con la anotación @Component para que se inicialicen al comenzar la ejecución y se puedan utilizar como

dependencias en los servicios. Podemos ver el ejemplo de MqttGateway en el siguiente fragmento para destacar como podemos publicar mensajes y suscribirnos a topics con un callback concreto a través de esta interfaz:

```
@Component
public class MqttGatewayImpl implements MqttGateway {
    private static final Logger LOGGER =
LoggerFactory.getLogger(MqttGatewayImpl.class);

    private IMqttClient client;

    private MqttClientFactory mqttClientFactory;

    private final MqttProperties mqttProperties;

    @Autowired
    public MqttGatewayImpl(MqttProperties mqttProperties, MqttClientFactory
mqttClientFactory) {
    this.mqttProperties = mqttProperties;
    this.mqttClientFactory = mqttClientFactory;
    this.client = mqttClientFactory.create(mqttProperties);

    }

    @PostConstruct
    protected void connect() {
    this.mqttClientFactory.connect(this.client, this.mqttProperties);
    }

    @Override
    public void subscribe(String[] topicFilters, MqttCallback callback) {
    this.client.setCallback(callback);
    int[] qos = new int[topicFilters.length];
    Arrays.fill(qos, this.mqttProperties.getQos());
    try {
        this.client.subscribe(topicFilters, qos);
        LOGGER.info("Subscribed to MQTT topics [{}] with QOS {}",
topicFilters, this.mqttProperties.getQos());
    } catch (MqttException e) {
        LOGGER.error("Error subscribing to MQTT Broker", e);
    }
    }

    @Override
    public void publish(String topic, String payload) {
    MqttMessage message = new MqttMessage(payload.getBytes());
    message.setQos(this.mqttProperties.getQos());
    message.setRetained(true);
    try {
        this.client.publish(topic, message);
        LOGGER.debug("Message published in MQTT topic [{}]: {}", topic,
payload);
    } catch (MqttException ignored) {
        LOGGER.error("Error publishing message in MQTT topic [{}]", topic);
    }
    }
}
```

En el anterior fragmento se puede ver también como se declaran las dependencias lo cual servirá como ejemplo para otro tipo de componentes de Spring Boot.

Servicios

Como se ha visto en la descripción de la arquitectura de la aplicación se necesitan cuatro servicios para encapsular la lógica de negocio. Para crear un servicio en Spring Boot se necesita añadir la anotación @Service antes de la cabecera y configurar sus dependencias de forma correcta (similar a los componentes). Los servicios de gestión de operaciones y mensajes obtienen del repositorio correspondiente las entidades para ofrecérsela a la capa de controlador REST o las insertan en el orden inverso.

En el caso del servicio de envío y recepción se inicializa como dependencia el MqttGateway para la comunicación con el bróker y se aplica la lógica de negocio para suscribirse a los tópicos pertinentes. Para recibir los mensajes se utilizan los siguientes (luego veremos que podemos configurarlos):

- Recibida: op/messages/+/ack/
- Completada: op/messages/+/completetion/
- Rechazada: op/messages/+/rejection/

En este momento es cuando se ve el gran potencial de MQTT ya que como vemos en el topic estamos utilizando una wildcard (símbolo +). Este mecanismo permite suscribirse a todos los tópicos que cumplan con un filtro concreto y en este caso sería que en el sitio del símbolo + vaya cualquier id de operación lo cual nos evita lógica innecesaria de estar suscribiéndonos y cancelando la suscripción constantemente de operaciones en ejecución y finalizadas. Además, debemos definir un callback para que se guarden los mensajes en base de datos. Gracias al filtrado del tópico podemos saber de qué tipo es el mensaje por lo que resulta sencillo invocando al servicio de gestión de mensajes.

Por otra parte, todavía quedaría el servicio de planificación que sería el más complejo porque necesitamos entender las *cron expresión*. Estas expresiones son descripciones de una planificación provenientes de Unix con el siguiente formato: <segundos> <minutos> <horas> <día del mes> <mes> <día de la semana>. Se generarán estas expresiones a partir de la fecha de planificación y la frecuencia que especifique el usuario gracias a la clase CronExpressionFactory y su método fromInstantAndFrequency:

```
public static String fromInstantAndFrequency(Instant instant, Frequency
frequency) {
        LocalDateTime dateTime = LocalDateTime.ofInstant(instant,
ZoneId.systemDefault());

        String seconds = Integer.toString(dateTime.getSecond());
        String minute = Integer.toString(dateTime.getMinute());
        String hour = Integer.toString(dateTime.getHour());
        String dayOfMonth = Integer.toString(dateTime.getDayOfMonth());
        String month = Integer.toString(dateTime.getMonthValue());
        String dayOfWeek = "?";

        switch (frequency) {
        case DAILY:
            // Reset dayOfMonth, month and dayOfWeek to "*"
            dayOfMonth = "*";
            month = "*";
            dayOfWeek = "*";
            break;
        case WEEKLY:
            // Reset dayOfMonth and month to "*"
            dayOfMonth = "*";
            month = "*";
            // Set dayOfWeek to the day of the week
            dayOfWeek =
Integer.toString(dateTime.getDayOfWeek().getValue());
            break;
        case MONTHLY:
            // Reset month and dayOfWeek to "*"
            month = "*";
            dayOfWeek = "*";
            break;
        case YEARLY:
            // Reset dayOfWeek to "*"
            dayOfWeek = "*";
            break;
        }

        return String.format("%s %s %s %s %s %s", seconds, minute, hour,
dayOfMonth, month, dayOfWeek);
    }
```

Con la expresión generada podemos planificar la ejecución de la tarea dentro del servicio gracias a la clase TaskScheduler del paquete org.springframework.scheduling que nos permite planificar tareas. Para que el planificador tenga la tarea se debe crear un CronTrigger del paquete org.springframework.scheduling.support a partir de la expresión y la tarea que hará que la operación se publique contando la iteración. Para verlo de forma más clara se proporciona el siguiente fragmento de código:

```
@Override
    public <E extends Operation> E scheduleOperation(E operation) {
    OperationSchedulingData schedulingData = operation.getSchedulingData();
    Frequency frequency = schedulingData.getFrequency();
    Instant scheduledDate = schedulingData.getScheduledDate();
    OperationSchedule schedule = new OperationSchedule(operation,
        CronExpressionFactory.fromInstantAndFrequency(scheduledDate,
frequency));
    operation.setSchedule(schedule);
    schedule.setId(operation.getId());
    operation = this.repository.save(operation);
    this.scheduleOperationTask(operation.getSchedule());
    return operation;
    }

    private void scheduleOperationTask(OperationSchedule schedule) {
    Runnable task = () -> {
        LOGGER.info("Launching operation ...");
        this.publishOperation(schedule.getOperation());
        this.handleIterations(schedule);
        LOGGER.info("Operation launched");
    };
    CronTrigger cronTrigger = new
CronTrigger(schedule.getCronExpression());
    ScheduledFuture<?> scheduledTask = this.taskScheduler.schedule(task,
cronTrigger);
    this.scheduledTasks.put(schedule.getId(), scheduledTask);
    }
```

[15]

Controlador REST

Un controlador REST es la clase que proporciona un marco de desarrollo para la capa de controlador de nuestra API de forma que se puedan especificar los *endpoints* que se van a ofrecer a los clientes tanto a nivel de petición como de respuesta. En el caso de este desarrollo se implementan las siguientes rutas correspondientes a los procesos que el microservicio debe implementar:

Proceso para adaptar	Ruta	Lógica de negocio (Servicio – Llamada)
Publicar	/op/publish	Operación - create
Cancelar planificación de operación	/op/{id}/cancel	Operación - cancel
Obtener información sobre una operación	/op/{id}	Operación - get
Obtener los mensajes recibidos como resultado de una operación	/op/{id}/messages	Mensajes - getByOperationId

Cada proceso tiene un punto de entrada en una ruta específica y el controlador REST encamina la información recibida en la petición a los métodos que ofrecen los servicios para aplicar la lógica de negocio correspondiente en cada caso (habiéndolos agregado como dependencias previamente).

Para añadir estos *endpoints* tenemos que crear una clase cuya cabecera está precedida por @RestController para que Spring Boot la detecte como un controlador REST. Posteriormente,

después de haber agregado las dependencias (servicios en este caso), añadimos los métodos que implementarán las rutinas a seguir en cada petición. Estos métodos deberán ir acompañados de una anotación que indique el método HTTP que se va a mapear como @PostMapping en el caso de POST con la ruta que debe seguir entrecomillada ("/op/publish" en el caso de la ruta para publicar operaciones). Además, para poder obtener la información de la petición correctamente se deben definir los parámetros del método con anotaciones que indiquen en qué parte de la petición se encuentran. En el caso de la publicación de una operación se adjunta la información de la operación en el *body* de la petición HTTP por lo que anotaremos la variable que contendrá la operación con la anotación @RequestBody. También podemos recibir variables de la ruta como se ha visto en las que tienen una operación objetivo sobre la que realizar una acción. En ese caso se especificaría cuando se define la ruta su posicionamiento en ella y en los parámetros del método con la anotación @PathVariable y el mismo nombre que en la ruta para la variable. Podemos verlo más claramente en el siguiente fragmento del código de OperationController:

```
@PostMapping("/op/publish") public <D extends OperationDTO>
ResponseEntity<D> publishOperation(@RequestBody D operation) { return
ResponseEntity.ok(this.operationService.create(operation));}

@GetMapping("/op/{id}/messages") public <D extends OperationDTO>
ResponseEntity<List<MessageDTO>> getMessages(@PathVariable Long id) { return
ResponseEntity.ok(this.messageService.getMessagesByOperationId(id));}
```

Aplicación

Por último, para crear una aplicación Spring Boot es necesario crear la clase principal muy sencilla que arrancará el microservicio gestionando el ciclo de vida de todos sus componentes gracias a la anotación @SpringBootApplication y a invocar el método run de SpringApplication. En el caso de la solución desarrollada también se incluyen las siguientes anotaciones:

- @EnableJpaRepositories
  - Habilita el uso de repositorios JPA y activa el escaneo de los repositorios en el código para configurarlos.
- @EnableJpaAuditing
  - Habilita la auditoría de JPA para proporcionar funcionalidades extra. En el caso del modelo de datos de nuestra solución queremos que se audite la fecha de creación de una operación y su fecha de modificación.
- @EnableScheduling
  - Habilita la capacidad de Spring para ejecutar tareas planificadas lo cual es clave para que se puedan planificar las operaciones como hemos visto anteriormente.

Configuración

Con el objetivo de hacer de la solución lo más personalizable posible se decide utilizar clases de configuración para la definición de parámetros constantes en la ejecución de la aplicación como la información de conexión para los adaptadores. Esta configuración se especifica mediante el fichero application.yaml presente en el paquete src.main.resources con la sintaxis YAML y se importa en las clases anotadas con @ConfigurationProperties. También, se recomienda añadir la anotación @Configuration para que Spring la detecte como una clase de ese tipo. La clase seguirá la misma estructura jerárquica de las variables siempre utilizando el mismo nombre o indicando la propiedad que representan con la etiqueta correspondiente. Posteriormente, estas clases se pueden agregar como dependencias en los componentes donde sean necesarias. La función que

cumplen en la aplicación es la de recoger las propiedades de conexión con el bróker MQTT y parámetros como la calidad del servicio, si se debe reconectar automáticamente o si debe indicar que los mensajes sean retenidos. Estas propiedades nos permiten aprovechar al máximo las capacidades del protocolo MQTT.

Testing y validación

Para garantizar que la solución es fiable es necesaria una labor de *testing* que ayude a localizar posibles problemas o defectos en la implementación. Las pruebas que se han realizado en el proyecto son las siguientes:

Pruebas unitarias

Las pruebas unitarias son las que se realizan a unidades funcionales identificables del código para ver si siguen el comportamiento esperado. Estas pruebas ayudan a localizar de forma más precisa errores en el desarrollo o verificar que un cambio en un evolutivo no ha supuesto un cambio inesperado en otra parte. Gracias a la elección de la arquitectura ya se cuenta con la separación necesaria entre las regiones del código de la aplicación por lo que se puede utilizar las librerías para elaborar los tests:

- JUnit – permite automatizar la ejecución de *tests* de forma controlada y nos da un marco de desarrollo de clases para pruebas.
- Mockito – permite configurar simulaciones del comportamiento de los distintos componentes de los que una unidad funcional puede utilizar.

En el caso de la solución propuesta se pueden plantear los siguientes casos de pruebas por componente (los repositorios y *mappers* estarían excluidos, por lo tanto) desarrollado:

Componente	Casos de prueba
Mapper de operaciones.	• Transformación de DTO a Entidad. • Transformación de Entidad a DTO.
MqttGateway / Servicio de gestión de envío y recepción de mensajes.	• Publicar mensaje MQTT. • Suscripción MQTT. • Simular recepción de mensaje MQTT.
Controlador REST.	• Recibir petición de publicar una operación • Recibir petición de cancelar una operación. • Recibir solicitud de la información de una operación. • Recibir la solicitud de la información de los mensajes de una operación.
Servicio de planificación.	• Ejecutar la tarea programada de publicar una operación. • Crear una planificación. • Cancelar una planificación.
Servicio de gestión de mensajes	• Guardar mensaje. • Obtener mensajes de una operación.
Servicio de gestión de operaciones	• Crear una operación. • Obtener la información de una operación.

Para crear *tests* unitarios con estas librerías, por ejemplo, para métodos de un servicio, se crea una clase nueva en un paquete de src.test a elegir (recomendable nombrarla con el nombre del servicio seguido de Test) utilizando la anotación @RunWith(MockitoJUnitRunner.class) en la cabecera. En esta clase se declararán las dependencias necesarias anotadas con @Mock y el servicio a probar con @InjectMocks para que el servicio tenga sus dependencias simuladas. A continuación, crearíamos métodos con la anotación @Test en el que verificaríamos el comportamiento de las unidades funcionales. Dentro de estos métodos podemos utilizar Mockito para definir los comportamientos de las dependencias en cada escenario y comparadores de JUnit para asegurarnos de que el resultado es correcto (assertEquals, assertNotNull...).

Pruebas de validación

Las pruebas de validación son las que se realizaran con la aplicación en funcionamiento y en las que se utilizará todo el flujo para ver si las diferentes unidades funcionales consiguen el objetivo común. Para las pruebas utilizaremos la librería Swagger con Swagger-UI ya que con ella es posible generar una interfaz gráfica donde es posible hacer llamadas a la API para verificar el funcionamiento de la parte REST.

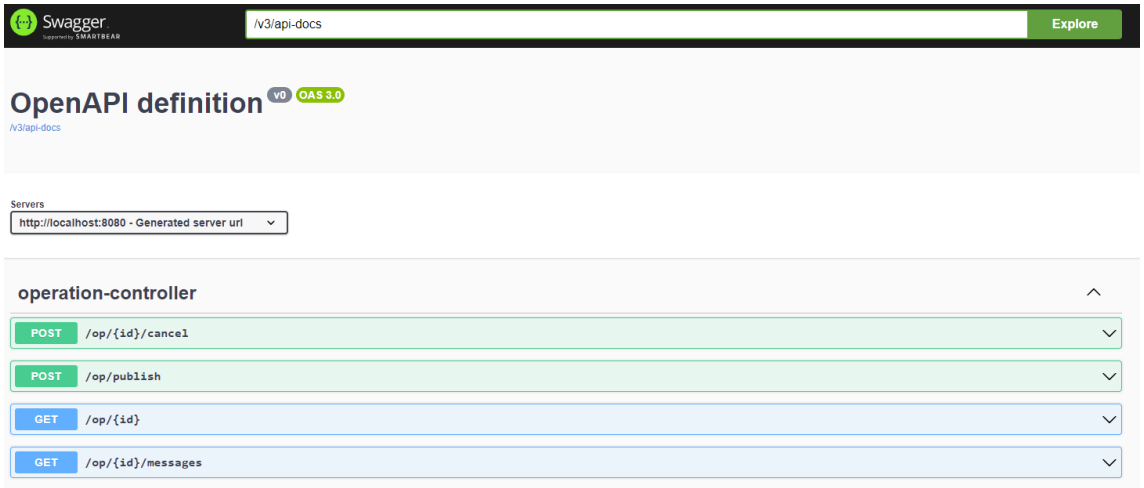

Ilustración 8 - Interfaz Swagger-Ui

Y se utilizará MQTT Explorer para ver los mensajes MQTT generados y simular la publicación de mensajes por parte de los dispositivos. Como ejemplo de una prueba de validación se mostrará el proceso de publicación de una operación de tipo CUSTOM.

Para empezar, se lanza una petición con Swagger-Ui al endpoint /op/publish con el siguiente JSON en el cuerpo de la petición:

```
{"type":"CUSTOM", "targetDevice":
{"group":"sensores", "deviceType":"sensor-humedad"},
"parameters": {"param":"example param"}}
```

A continuación, verificamos que la operación se ha publicado en el bróker MQTT:

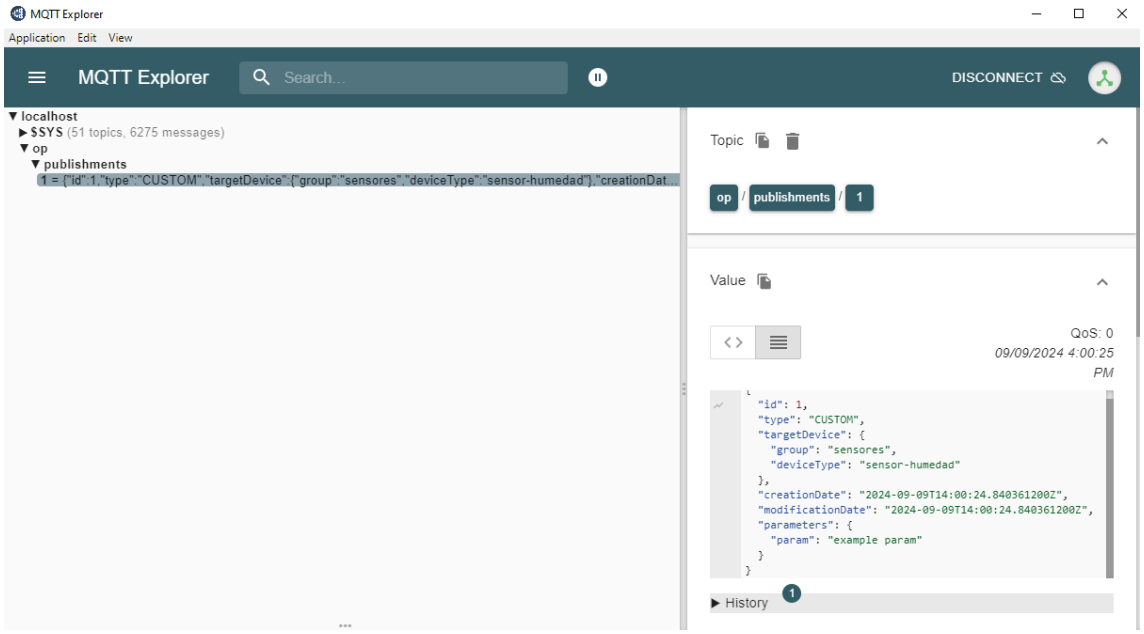

Ilustración 9 - Comprobación de la publicación de un mensaje de operación

Más tarde, enviamos un mensaje en el tópico que está escuchando la aplicación para esta operación:

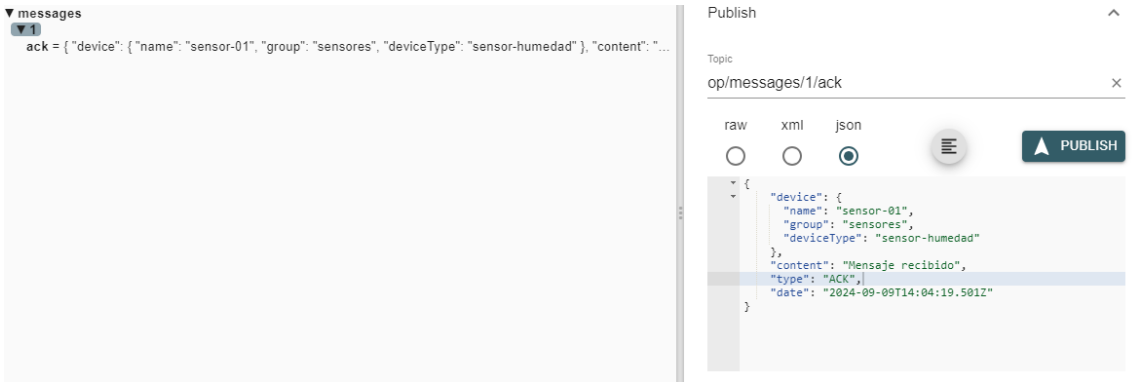

Ilustración 10 - Publicación de mensaje ACK

Y posteriormente, verificamos que el mensaje se ha guardado en la base de datos:

	id	content	date	device_type	device_group	name	type	operation_id
1	1	Mensaje recibido	2024-09-09 14:13:02.292196	sensor-humedad	sensores	sensor-01	0	1

Ilustración 11 - Registro en base de datos de mensaje

Por último, consultamos los mensajes de la operación con otra petición REST:

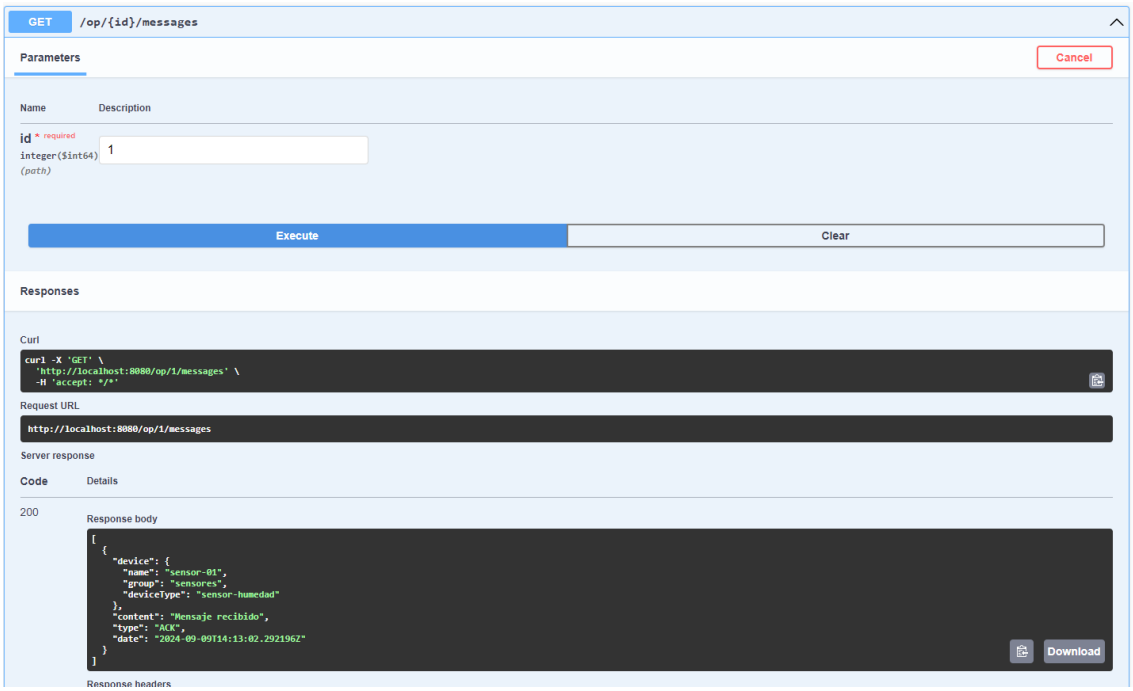

Ilustración 12 - Resultado a consulta de mensajes de una operación

Todo el flujo ha sido verificado por lo que se puede asegurar que la aplicación tiene el comportamiento esperado.

Desafíos encontrados

Durante el transcurso de la implementación de la solución se han encontrado varios desafíos ya que se han utilizado muchas herramientas potentes, pero tienen su complicación para integrarlas. Por ejemplo, MapStruct necesita de los métodos get y set de las clases del modelo de datos en sus mappers por lo que se necesitaba integrarla con Lombock para que no saltaran errores. Tras buscar se vio que era necesario un plugin del compilador de Maven para que se pudiera realizar la compilación llamado lombok-mapstruct-binding. [16]

Por otra parte, habilitar tres tipos de operaciones implementadas para añadir nuevas, ha sido un gran trabajo, ya que se han tenido que practicar todos los conocimientos sobre herencia (tanto en las librerías como en el Java) que se disponían, pero el resultado fue exitoso.

Por último, es necesario destacar el caso de uso de planificación ya que llegar a la idea de utilizar un CronTrigger y generarlo a partir de los datos que provienen de la creación de la operación requirió tiempo y búsqueda de información sobre como lograrlo en una aplicación como esta.

6. Implantación

La solución ya está implementada y probada por lo que ya podemos realizar una implantación en una infraestructura IoT existente para automatizar procesos relacionados con las meta-operaciones que no cubren los productos en el mercado.

Introducción

El SmartCity Lab del grupo Tatami tiene como objetivo implantar la solucion desarrollada para automatizar labores de gestión de la infraestructura de *smart city* que tienen actualmente. Este plan de implantación busca la forma de integrar los requisitos del laboratorio con los mecanismos que se ofrecen. Como hemos visto, se parte de que existen tres tipos de operaciones ya implementadas como son Script, Update y Custom las cuales otorgan una flexibilidad suficiente para automatizar tareas.

Objetivos

La elaboración del plan de implementación de la solución tiene los siguientes objetivos:

- Recoger los requisitos de automatización de tareas del laboratorio: la implementación de la solución mejorará todos los procesos requeridos.
- Adaptar la solución para el entorno actual del laboratorio: la solución deberá ser desplegable en el entorno del laboratorio.
- Automatizar la gestión de dispositivos IoT: la solución permitirá la automatización de tareas de mantenimiento y gestión.
- Mejorar la seguridad de la infraestructura: la solución debe dar pié a que la infraestructura sea más segura gracias a las nuevas automatizaciones.
- Simplificar el monitoreo de operaciones: gracias a la implementación de la solución el monitoreo de las operaciones se podrá hacer de forma sencilla y centralizada.

Requisitos

El laboratorio SmartCity lab del grupo Tatami tiene una serie de necesidades que la solución debe cubrir. A partir de los siguientes requisitos deberemos dar un marco de configuración de las operaciones implementadas para satisfacer esas necesidades:

- Ejecución remota de scripts: los usuarios del grupo Tatami deben poder ejecutar scripts de forma remota soportando formatos como bash o Python para automatizar tareas técnicas.
- Cambio de versión de aplicaciones, servicios o componentes: el laboratorio requiere la capacidad de seleccionar las versiones de diferentes tipos de instalaciones en dispositivos IoT.

- Actualización de certificados de seguridad: el sistema permitirá actualizar remotamente los certificados de seguridad.
- Modificación del archivo rc.local: se solicita la capacidad de hacer modificaciones en este archivo para configurar el inicio de los dispositivos IoT.
- Ejecución directa de comandos: es necesario que se puedan publicar directamente ejecuciones de comandos de bash. Por ejemplo, ejecutar "apt update" en cada dispositivo.
- Peticiones REST: la configuración de la infraestructura requiere que los dispositivos hagan peticiones REST en algunos casos de uso por lo que se necesita poder automatizar esta posibilidad.

Plan de despliegue

Para asegurar el éxito de la implantación de la solución es necesario planificar correctamente el despliegue ya que los errores en las primeras fases pueden generar desconfianza en un sistema que debe ser seguro y fiable. Por ello se hará el despliegue en tres fases:

Fase 1: Preparación

En esta fase se hará trabajo de campo examinando toda la infraestructura sobre la que se tiene que implementar la solución para descubrir sus particularidades y asegurarse de que la implementación es viable. Los pasos de esta fase son los siguientes:

1. Revisión de la infraestructura: verificar y listar todos los elementos que componen la infraestructura.
2. Configuración del entorno de pruebas: crear un entorno paralelo similar al actual donde se realizarán pruebas de la solución.
3. Conectividad: se debe asegurar que la red de la infraestructura es fiable y puede soportar comunicaciones MQTT con todos sus dispositivos.
4. Seguridad: ya que es requerido ejecutar scripts de forma remota se quiere garantizar que no vamos a inyectar vulnerabilidades en el sistema.
5. Configuración del equipo: se designará un equipo que se encargará de las labores de implementación conformado por usuarios del SmartCity Lab y responsables del producto.

Fase 2: Configuración inicial

En esta fase de darán los primeros pasos del despliegue como tal de la solución pues se empezarán a instalar los componentes en el entorno de pruebas. En el entorno de desarrollo se utilizará docker-compose para desplegar los componentes siendo clave el paso de revisión de la infraestructura para asegurar que las condiciones (nombres de servicio, redes). Los pasos de esta fase son los siguientes:

1. Elaborar el fichero docker-compose.yml: se creará un fichero YAML que contenga la especificación del despliegue en el entorno de pruebas. Este fichero tiene como objetivo tener en cuenta la estructura de la solución IoT de SmartCity Lab para evitar de antemano colisiones de puertos, problemas de red u otros inconvenientes que puedan surgir de conectividad.
2. Configuración de las operaciones: se deberá decidir qué tipo de operaciones se eligen para cada caso de uso. Es importante acertar ya que queremos evitar las máximas

modificaciones posibles en el firmware o rutinas de los dispositivos después del esfuerzo inicial de implementación de las operaciones en base a la estructura proporcionada.

3. Configuración de dispositivos IoT: se deberá implementar firmwares en los dispositivos para recibir, responder e interpretar los mensajes provenientes del microservicio. Habrá que tenerse en cuenta todas las particularidades de cada dispositivo involucrado como sistema operativo, capacidad de cómputo...
4. Despliegue entorno de pruebas: con los preparativos completados se ejecutará el comando de despliegue para que los componentes comiencen a desplegarse y los usuarios del laboratorio y los responsables de la implantación puedan probar la solución.

Fase 3: Testing

En esta fase el equipo de implantación repetirá la batería de pruebas de la fase de implantación en el entorno de pruebas y utilizará los nuevos recursos para asegurar que la solución puede ser funcional con las operaciones configuradas en la fase anterior y cumple con los requisitos. En paralelo se deben monitorizar continuamente los *logs* de los componentes para revisar el correcto funcionamiento.

Fase 4: Despliegue

En esta fase final es cuando, tras la exhaustiva batería de pruebas llevada a cabo y asegurarse de que todo es correcto desplegaremos la solución en el entorno real para realizar operaciones automáticas desde su plataforma. Los pasos del despliegue son:

- Elaboración del fichero docker-compose.yml final: se creará el fichero de despliegue en docker-compose que se conecte y opere en la red del entorno de producción. Si se ha hecho el archivo del entorno de pruebas compatible con el entorno final se evitarán muchas diferencias en el nuevo fichero respecto al anterior.
- Formación: se educará al personal del laboratorio en cómo utilizar la solución para que puedan conocer el sistema de mano de los desarrolladores y conozcan sus capacidades. Además, se debe impartir de forma muy exhaustiva un seminario de seguridad para asegurarnos de que el sistema no supone un problema de ciberseguridad.
- Puesta en marcha de sistema de monitoreo: se creará un sistema paralelo personalizado para el laboratorio que monitorice todo el entorno para detectar posibles problemas relacionados con la actualización de la infraestructura.
- Despliegue: se ejecuta el despliegue para que los componentes empiecen a funcionar en el entorno principal.
- Monitorización: se monitorizará exhaustivamente el entorno para observar si existen incidencias después del despliegue.
- (Opcional) Desmantelar entorno de pruebas: si el equipo del laboratorio no cree que necesite seguir actualizando la solución se eliminará el entorno de pruebas. No es recomendable ya que permite a los usuarios tener un entorno donde probar las operaciones antes de lanzarlas en producción, pero pueden existir límites de recursos ajenos al ámbito del proyecto por los cuales sea necesario.

Configuración de las operaciones

Como ya sabemos la aplicación ya viene con tres operaciones predefinidas (Script, Update y Custom) que nos dan posibles estructuras para implementar las operaciones que el laboratorio

necesita. En este apartado se va a explicar cómo utilizaremos el marco de configuración desarrollado para cumplir con las operaciones requisito del laboratorio. Se analizará como puede contribuir cada tipo de operación en la configuración de las operaciones requeridas

Operaciones SCRIPT

En el caso de las operaciones SCRIPT las utilizaremos para las siguientes tareas:

- Ejecución de scripts bash, Python u otros lenguajes: mediante la posibilidad de directamente redactar el script en el cuerpo del mensaje en el campo datos de la operación.
- Modificación del archivo rc.local: en relación con el punto anterior se configurará un script base que permita modificar el archivo local para que partiendo de ese script y realizando modificaciones en lo esencial se pueda modificar el archivo rc.local de forma segura.
- Actualizar certificados de seguridad: de la misma forma que para modificar el archivo local se creará un script base donde el usuario podrá definir la ruta y destino del nuevo certificado y que realice la sustitución para que lo incluya en el campo datos de la operación.

Estas operaciones pueden llegar a ejecutar comandos complejos por lo que es indispensable cifrar las comunicaciones debidamente para que no entren en escena actores inesperados.

Operaciones UPDATE

Mediante las operaciones UPDATE automatizaremos la siguiente tarea:

- Actualizar versiones de software de forma remota: en la estructura de la operación se definirán los paquetes que se deben actualizar y cuando el dispositivo la reciba hará su propia rutina de actualización conforme la tenga definida. Se puede combinar con otras operaciones que hagan posible el cambio de repositorios o rutinas para dotar de más flexibilidad al sistema.

Operaciones CUSTOM

Gracias a la personalización que ofrecen las operaciones de tipo CUSTOM por su atributo parámetros se puede crear una subestructura para implementar rutinas sin pasar por scripts previos o para adecuarla a las preferencias del equipo del laboratorio. Podría ser útil para satisfacer los siguientes requisitos:

- Ejecutar comandos directamente: se definirá una subestructura en la que dentro de los parámetros habrá uno que se llame método cuyo valor será “direct-command” y otro llamado comando en el que se incluirá el comando en texto. El dispositivo la recibirá y al interpretar que es un comando directo lo ejecutará directamente en su implementación. Para garantizar la seguridad se debería cifrar el contenido del mensaje.
- Hacer peticiones REST: se definirá una subestructura en la que en el cuerpo del atributo parámetros incluyamos todo lo necesario para hacer una petición REST como la ruta, método, proxy...para que luego el dispositivo la implemente con la librería que pueda utilizar o considere el desarrollador.

Plan de pruebas

Para evitar posibles fallos en el entorno principal es indispensable crear un plan de pruebas en el que comprobemos que la solución se acopla bien en el resto del sistema y cumple su cometido. Las pruebas que se realizarán son las siguientes:

- Funcionales: se verifica que todas las operaciones configuradas se ejecutan en los dispositivos en los que deberían y de forma correcta. Se reciben los mensajes correspondientes y no hay errores en el código.
- Seguridad: se asegura que las operaciones se realizan de forma cifrada y un actor de fuera de la red o que no posea autorización dentro de la red no puede comprometer el sistema.
- Escalabilidad: se realizarán pruebas de carga para ver hasta dónde puede llegar el sistema con los recursos del laboratorio.
- Integración: se revisará que los nuevos componentes se integran con los existentes.

Posibles riesgos y mitigaciones

Durante el proceso de implementación pueden surgir los siguientes riesgos que debemos tener en cuenta para poder mitigar:

- Problemas de conectividad: se puede encontrar una red muy saturada de dispositivos con unos recursos que no hagan posible una comunicación perfecta. Lo mitigaremos activando los mecanismos de calidad de servicio, retención de mensajes y auto reconexión del servicio de envío y recepción de mensajes.
- Seguridad: se puede detectar que las nuevas operaciones introducen vulnerabilidades en el sistema no previstas. Lo mitigaremos implementando mecanismos de cifrado y autenticación.
- Compatibilidad: algunos dispositivos pueden no llega a tener la capacidad de cómputo o la forma de implementar las operaciones. Lo mitigaremos explorando nuevos protocolos que se puedan utilizar en lugar de MQTT.

7. Conclusiones

Una vez finalizado el desarrollo y documentación del proyecto puedo decir que el resultado del proyecto es muy satisfactorio ya que se han cumplido todos los objetivos marcados al inicio del proyecto. He desarrollado un sistema integrable en todas las plataformas IoT que permiten realizar peticiones REST y eso hace que tengamos ya no solo una solución desarrollada en un proyecto formativo si no un producto en estado embrionario.

La aplicación permite orquestar operaciones de forma sencilla y efectiva lo cual da muchas posibilidades de automatización de tareas repetitivas que requerían de intervención humana para realizarse. Por ejemplo, a partir de ahora no necesitaríamos a nadie que esté pendiente de actualizar los paquetes de todos y cada uno de los dispositivos de una infraestructura IoT que utilicen dependencias o podríamos implementar rutinas de renovación de certificados más agresivas para proteger la seguridad de la red IoT.

Por otra parte, se ha conseguido el grado de personalización que buscábamos proporcionando tres tipos de operaciones que se pueden adaptar a cualquier ámbito. De hecho, la operación CUSTOM permite diseñar a cualquier usuario una estructura a su gusto para crear operaciones nuevas e innovadoras por lo que se puede anotar un éxito en el casillero del proyecto respecto a este objetivo.

El proyecto tiene un valor muy alto a nivel formativo ya que me ha permitido dominar tecnologías muy diversas que había utilizado poco como Spring Boot, Lombock, MapStruct, JUnit o Mockito y profundizar en mis conocimientos de tecnologías ya conocidas como Java, APIs REST, MQTT o Docker. Por otra parte, me ha enseñado mucho este desarrollo a configurar arquitecturas de forma eficiente ya que pasé de la propuesta de una API en cada dispositivo que a nivel de integración era muy ineficiente a una propuesta sólida como ha sido el diseño final. Además, he aprendido a utilizar el patrón de arquitectura hexagonal para el desarrollo que es tan demandado actualmente para desarrolladores back-end.

Por si no fuera suficiente el valor formativo también puedo decir que el valor profesional del desarrollo de este proyecto es muy alto ya que me ha dado las bases y las herramientas para dar mis primeros pasos profesionalmente en el mundo de la IoT y la automatización industrial contribuyendo a soluciones similares en un proyecto ambicioso como ASUMO de GMV y Red Eléctrica.

En definitiva, el proyecto ha sido un completo éxito tanto a nivel de objetivos como a nivel personal pues ha cumplido todos los propuestos y además me ha hecho crecer como alumno de la escuela y como profesional informático. Sin duda es un proyecto que merecerá la pena retomar para seguir evolucionándolo para quien sabe si obtener un producto que tenga un interés real por parte de la comunidad y el impacto esperado por él.

Trabajos futuros

Después de finalizar este proyecto me siguen surgiendo ideas de formas en las que se podría mejorar la aplicación:

Por ejemplo, limitarnos al protocolo MQTT puede hacer que perdamos muchos dispositivos IoT por el camino por lo que integrar la solución con protocolos industriales como OPC UA PubSub que también utiliza el patrón publicador suscriptor.

Otra mejora que se me ocurre es la de añadir una mejor sincronización entre las réplicas del microservicio a la hora de gestionar operaciones planificadas profundizando más en las librerías utilizadas o sustituyéndolas por una más compleja como Quartz.

Por otra parte, siempre podemos mejorar respecto a la seguridad por lo que podríamos añadir la posibilidad de integrar un gestor de identidades propio para la solución o hacerla más integrable con los que ya tengan las infraestructuras. Productos open-source como Keycloak podrían sernos útiles en una arquitectura nueva y más segura.

Por último, para liberar a usuarios menos experimentados de las dificultades de implementar las operaciones en los dispositivos se podría empezar una línea de trabajo paralela en la que se desarrollan poco a poco integraciones de las operaciones preimplementadas con dispositivos populares en el mercado como Raspberry Pi o Beagle Bone.

Relación del trabajo desarrollado con los estudios cursados

El trabajo desarrollado en este proyecto pone en práctica muchos de los conocimientos obtenidos en el proyecto formativo del MUInf como veremos a continuación:

La asignatura Planificación y Dirección de Proyectos de las TI proporciona un marco sobre se ha trabajado para planificar el proyecto, analizar sus riesgos e implantarlo. La planificación es muy importante en proyectos como este en los que la seguridad es crucial sobre todo si utilizamos la solución en infraestructuras críticas.

La asignatura de Redes y Seguridad es la que da la primera capa tecnológica del proyecto ya que en ella aprendí mucho sobre cómo gestionar la red de una infraestructura y se nos introdujo el protocolo MQTT y la contenedorización. Paralelamente en Servicios y Aplicaciones Distribuidas aprendimos a crear arquitecturas software como la que hemos visto en el proyecto. Posteriormente, Ciberseguridad me dio la concienciación sobre lo necesaria que es la seguridad en los sistemas informáticos que ha hecho que el desarrollo y el plan de implementación tengan uno de sus focos en la ciberseguridad.

A nivel de desarrollo también ha tenido una influencia muy positiva la asignatura Auditoría, Calidad y Gestión de los Sistemas de Información ya que desde el momento en el que asistí a las clases me di cuenta de lo importante que es el *testing* en el desarrollo software y di mis primeros pasos con Junit. Por estas razones se ha dedicado tiempo en el desarrollo a la confección de pruebas automáticas.

Por otra parte, Inteligencia Ambiental y Sistemas Informáticos para el Control de Instalaciones y Procesos han influido de forma clara en la confección de la solución ya que han dado el contexto temático e incluso filosófico que ha motivado el proyecto. También son donde se pusieron en práctica por primera vez los conocimientos técnicos de muchas de las tecnologías utilizadas en soluciones IoT como el protocolo MQTT, los brókeres de mensajería y las plataformas IoT.

Como hemos visto el proyecto, aunque tiene una tendencia clara hacia las asignaturas relacionadas con la IoT y la automatización industrial, es bastante transversal respecto a la cantidad de conocimientos de toda la formación que utiliza.

8. Referencias

[1] A. A. Laghari, K. Wu, R. A. Laghari, M. Ali y A. A. Khan, «A review and state of art of Internet of Things (IoT),» *Archives of Computational Methods in Engineering*, pp. 1-19, 2021.

[2] L. Gomez D'Orazio, S. Medina y D. M. Montezanti, «Integración de una red de sensores con una plataforma IoT para control inteligente de aulas,» de *XXVIII Congreso Argentino de Ciencias de la Computación (CACIC)*, La Rioja, 3 al 6 de octubre de 2022.

[3] C. Sobrino, «QUÉ SON LAS PLATAFORMAS IoT Y SUS APLICACIONES EN LA INDUSTRIA,» 27 01 2023. [En línea]. Available: https://www.captia.es/blog/plataformas-iot.html.

[4] alfaiot-webmaster, «¿Qué es una plataforma IOT? Guía completa,» 31 Enero 2023. [En línea]. Available: https://alfaiot.com/iot/que-es-una-plataforma-iot-guia-completa/.

[5] AWS, «IoT - Información general,» [En línea]. Available: https://aws.amazon.com/es/iot/.

[6] Thingsboard, «What is ThingsBoard?,» [En línea]. Available: https://thingsboard.io/docs/getting-started-guides/what-is-thingsboard/.

[7] Microsoft, «Azure IoT Hub,» [En línea]. Available: https://azure.microsoft.com/es-es/products/iot-hub.

[8] Red Hat, «¿Qué es una arquitectura de aplicaciones?,» 15 03 2023. [En línea]. Available: https://www.redhat.com/es/topics/cloud-native-apps/what-is-an-application-architecture#:~:text=Una%20arquitectura%20de%20aplicaciones%20describe,dise%C3%B1ar%20una%20aplicaci%C3%B3n%20bien%20estructurada..

[9] AWS, «¿Qué es una base de datos relacional?,» [En línea]. Available: https://aws.amazon.com/es/relational-database/.

[10] D. Soni y A. Makwana, «A survey on mqtt: a protocol of internet of things (iot),» de *International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017)* , 2027.

[11] AWS, «¿Qué es MQTT?,» [En línea]. Available: https://aws.amazon.com/es/what-is/mqtt/.

[12] E. Salguero, «Arquitectura Hexagonal,» 22 Junio 2018. [En línea]. Available: https://medium.com/@edusalguero/arquitectura-hexagonal-59834bb44b7f.

[13] Baeldung, «Hibernate Inheritance Mapping,» 11 Mayo 2024. [En línea]. Available: https://www.baeldung.com/hibernate-inheritance.

[14] Baeldung, «Quick Guide to MapStruct,» 18 Enero 2024. [En línea]. Available: https://www.baeldung.com/mapstruct.

[15] Baeldung, «A Guide to the Spring Task Scheduler,» 11 Mayo 2024. [En línea]. Available: https://www.baeldung.com/spring-task-scheduler.

[16] T. Bormans, «Medium,» 1 Abril 2024. [En línea]. Available: https://medium.com/@tijl.b/a-guide-to-mapstruct-with-spring-boot-vavr-lombok-d5325b436220.

9. Anexos

Glosario

A continuación, se presenta un glosario para aclarar términos que puede que el lector no conozca de antemano.

Término	Definición
API REST	Patrón de diseño de servicios web que permite la conexión mediante el protocolo HTTP.
Bash	Shell de comandos para sistemas operativos Linux y macOS.
Bróker MQTT	Servidor intermediario que se encarga de distribuir los mensajes publicados en un tópico MQTT a todos los suscriptores de ese tópico.
CRUD	Siglas de *create*, *read*, *update* y *delete* que se utilizan para agrupar estas operaciones básicas para gestionar registros en una base de datos.
DTO	Objeto que se utiliza para transportar datos entre capas de una aplicación.
Mapper	Componente software que transforma objetos de un tipo origen a otro destino.
Microservicio	Estilo arquitectónico software que consiste en desarrollar aplicaciones mediante colecciones de pequeños e independientes servicios.

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**			X	
ODS 3. **Salud y bienestar.**		X		
ODS 4. **Educación de calidad.**			X	
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**			X	
ODS 7. **Energía asequible y no contaminante.**		X		
ODS 8. **Trabajo decente y crecimiento económico.**		X		
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades**.			X	
ODS 11. **Ciudades y comunidades sostenibles**.	X			
ODS 12. **Producción y consumo responsables.**		X		
ODS 13. **Acción por el clima.**		X		
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**			X	
ODS 16. **Paz, justicia e instituciones sólidas.**			X	
ODS 17. **Alianzas para lograr objetivos.**			X	

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

El desarrollo de soluciones IoT tiene una relación significativa con varios de los Objetivos de Desarrollo Sostenible (ODS). El Proyecto tiene como resultado la extensión de una plataforma IoT para facilitar la gestión remota y automatizada de infraestructuras inteligentes. Se plantea la implantación en el contexto del SmartCity Lab del grupo Tatami el cuál necesita automatizar una serie de operaciones de plataforma. En este contexto podemos detectar implicaciones directas en la eficiencia de las operaciones, el uso responsable de los recursos, la innovación tecnológica y la mejora de la sostenibilidad en las ciudades. A continuación, vamos a ver qué objetivos están relacionados con el proyecto realizado.

ODS 9: Industria, Innovación e Infraestructuras
El ODS 9 tiene como objetivo la construcción de infraestructuras resilientes, la promoción de la industrialización inclusiva y sostenible, y el fomento de la innovación. Este trabajo se alinea fuertemente con este ODS, ya que la posibilidad de automatizar operaciones que previamente requerían de una intervención humana e incluso desplazamientos físicos permite que las ciudades e infraestructuras industriales puedan ser más eficientes a la hora de gestionar los recursos reduciendo los tiempos de operación y minimizando los costes de mantenimiento. Además, da un marco para la innovación en el que explorar nuevas posibilidades de aplicación de los principios de la industria 4.0, donde la automatización tiene un papel crucial.

ODS 11: Ciudades y Comunidades Sostenibles
Otro ODS clave relacionado con este trabajo es el 11 ya que promueve la construcción de ciudades y comunidades sostenibles. Las ciudades inteligentes juegan un papel clave en el futuro de estas junto con las soluciones IoT por lo que nuestro aporte en forma de capacidades de automatizar procesos puede reducir notablemente el impacto que tienen las ciudades sobre la salud del planeta mediante la optimización de recursos y la mejora de la gestión de dispositivos en entornos urbanos.

La posibilidad de ejecutar operaciones como la actualización de software automática garantiza que los dispositivos están siempre actualizados previniendo potenciales fallos que podrían afectar a infraestructuras críticas o servicios públicos lo cual aporta mucha fiabilidad y anima a los ciudadanos a utilizarlos. Por ejemplo, una actualización de las rutas del servicio de autobuses de una ciudad de forma continua mediante operaciones permite que se detecten las zonas con mayor tráfico en cada momento y da la posibilidad de reconfigurar los trayectos en tiempo real.

Además, las ciudades inteligentes pueden utilizar las plataformas IoT para monitorizar el uso de recursos como el agua o la energía proporcionando información con alto valor para la toma de decisiones. Por ejemplo, la automatización del sistema de regadío puede reducir el desperdicio de agua al ajustar dinámicamente la potencia en función de las circunstancias climatológicas en una zona lo cual contribuye a mejorar la sostenibilidad a largo plazo.

ODS 7: Energía Asequible y No Contaminante
El ODS 7 busca garantizar el acceso a una energía asequible, fiable, sostenible y moderna para todos. Aunque este trabajo no está directamente en la energía, la automatización de procesos tiene un impacto muy grande en la eficiencia energética abaratando el coste y garantizando el

suministro. Las infraestructuras de gestión de la energía que incorporen mecanismos IoT pueden optimizar el consumo de energía al ajustar automáticamente el consumo en función de la demanda y las condiciones operativas.

Por ejemplo, en una ciudad inteligente que utiliza dispositivos IoT para monitorizar y controlar el consumo energético de edificios, alumbrado público o transporte puede reducir el uso de energía cuando no sea necesaria lo cual disminuiría las emisiones de carbono y reduciría el coste de la energía.

En conclusión, el trabajo puede tener un impacto significativo respecto a los objetivos de desarrollo sostenible ya que ayuda a mejorar muchos procesos en cuando al uso de recursos y la reducción de costes gracias a la automatización que proporciona a las infraestructuras IoT y en concreto a las ciudades inteligentes como las que simula el SmartCity Lab.