UNIVERSIDAD TÉCNICA DE COTOPAXI

FACULTAD DE CIENCIAS DE LA INGENIERÍA Y APLICADAS

CARRERA DE INGENIERÍA EN SISTEMAS DE INFORMACIÓN

DESARROLLO DE UN ECOSISTEMA IOT INTEROPERABLE BASADO EN EL FRAMEWORK FIWARE EN LA UNIVERSIDAD TÉCNICA DE COTOPAXI.

PROYECTO DE INVESTIGACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS DE INFORMACIÓN

AUTOR:
Steven Paul Arequipa Poaquiza

TUTOR:
Ing. Manuel William Villa Quishpe

COTUTOR:
Ing. MSc. Angel Guillermo Hidalgo Oñate

LATACUNGA, MARZO 2026

Latacunga, marzo 2026

DECLARACIÓN DE AUTORÍA

Yo, Steven Paul Arequipa Poaquiza declaro ser autor del proyecto de titulación **"DESARROLLO DE UN ECOSISTEMA IOT INTEROPERABLE BASADO EN EL FRAMEWORK FIWARE EN LA UNIVERSIDAD TÉCNICA DE COTOPAXI"**, siendo el Ing. Manuel Willian Villa Quishpe tutor del presente trabajo de titulación; y eximo expresamente a la Universidad Técnica de Cotopaxi y a sus representantes legales de posibles reclamos o acciones legales.

Además, certifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo de titulación, son de mi exclusiva responsabilidad.

Steven Paul Arequipa Poaquiza
CC. 0503549883

Latacunga, marzo 2026

AVAL DEL TUTOR DEL PROYECTO DE INVESTIGACIÓN

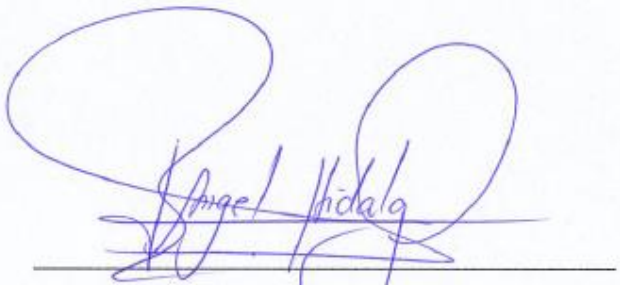
En calidad de Tutor del Trabajo de Investigación sobre el título: **"DESARROLLO DE UN ECOSISTEMA IOT INTEROPERABLE BASADO EN EL FRAMEWORK FIWARE EN LA UNIVERSIDAD TÉCNICA DE COTOPAXI"**, propuesto por el estudiante Steven Paul Arequipa Poaquiza de la Carrera de Ingeniería Sistemas de información, considero que dicho proyecto de titulación cumple con los requerimientos metodológicos y aportes científico-técnicos suficientes para ser sometidos al tribunal de lectores.

Ing. Manuel William Villa Quishpe
CC: 1803386950
TUTOR

Latacunga, marzo 2026

AVAL DEL COTUTOR DEL PROYECTO DE INVESTIGACIÓN

En calidad de CoTutor del Trabajo de Investigación sobre el título: **"DESARROLLO DE UN ECOSISTEMA IOT INTEROPERABLE BASADO EN EL FRAMEWORK FIWARE EN LA UNIVERSIDAD TÉCNICA DE COTOPAXI"**, propuesto por el estudiante Steven Paul Arequipa Poaquiza de la Carrera de Ingeniería Sistemas de información, considero que dicho proyecto de titulación cumple con los requerimientos metodológicos y aportes científico-técnicos suficientes para ser sometidos al tribunal de lectores.



Ing. MSc. Angel Guillermo Hidalgo Oñate
CC: 0503257404
COTUTOR

Latacunga, marzo 2026

AVAL DE APROBACIÓN DE LECTORES

Cumpliendo con el Reglamento de Titulación de la Universidad Técnica de Cotopaxi, en calidad de Lectores de Tribunal de Proyecto de Investigación con el Título **"DESARROLLO DE UN ECOSISTEMA IOT INTEROPERABLE BASADO EN EL FRAMEWORK FIWARE EN LA UNIVERSIDAD TÉCNICA DE COTOPAXI"**, propuesto por el o la estudiante Steven Paul Arequipa Poaquiza de la Carrera de SISTEMAS DE INFORMACIÓN, me permito indicar que el o la estudiante ha concluido todas las observaciones y realizado las correcciones señaladas por el Tribunal de Lectores, además de validar el funcionamiento de la propuesta (aplica para propuesta tecnológica) , por lo cual presentamos el Aval de aprobación del Proyecto de Titulación correspondiente a la modalidad **proyecto de investigación** en virtud de lo cual él o la postulante puede presentarse a la Defensa de su Proyecto de Titulación.

Particular que pongo en su conocimiento para los fines legales pertinentes.

Atentamente,

Lector 1 (Presidente)	Lector 2	Lector 3
Ing. Luis Rene Quisaguano Collaguazo	Ing. Diego Alexander Reinoso Cueva	Ing. Karla Susana Cantuña Flores
CC:1721895181	CC: 0503024051	CC: 0502305113

UNIVERSIDAD TÉCNICA DE COTOPAXI

Proyecto de Vinculación STEM

CERTIFICADO DE PARTICIPACIÓN

El señor estudiante **Steven Paúl Arequipa Poaquiza**, con cédula de ciudadanía **0503549883**, ha participado y ha colaborado de manera proactiva y sobresaliente en el marco de la ejecución del proyecto de vinculación institucional: **"Orientación formativa práctico-colaborativa en carreras STEM de la provincia de Cotopaxi"**, aprobado por el Consejo Académico de la institución con fecha 28 de abril de 2022, demostrando un alto nivel de gestión, liderazgo técnico y compromiso social desde el periodo académico Octubre 2021 – Marzo 2022 hasta la presente fecha, de acuerdo al siguiente detalle:

1. **Prácticas Laborales**
   - Octubre 2021 hasta Marzo 2022 con 48 horas
   - Abril 2022 hasta Agosto 2022 con 96 horas
   - Octubre 2022 hasta Marzo 2023 con 96 horas

2. **Prácticas de Servicio Comunitario**
   - Octubre 2023 hasta Marzo 2024 con 160 horas

3. **Integración Curricular**
   - Desarrollo del proyecto "DESARROLLO DE UN ECOSISTEMA IOT INTEROPERABLE BASADO EN EL FRAMEWORK FIWARE EN LA UNIVERSIDAD TÉCNICA DE COTOPAXI" dentro de su proceso de integración curricular.

4. **Acción Afirmativa**

   - Fue presidente del Club de Robótica BOT'S UTC desde Marzo 2023 hasta Diciembre 2024 contribuyendo activamente a la organización del Concurso de Robótica MASHCA BOT'S 2024.

Por lo expuesto, se extiende el presente documento en reconocimiento a su aporte al fortalecimiento del sector educativo y tecnológico de la provincia de Cotopaxi, contribuyendo directamente a los resultados del proyecto.

Dado y firmado en la ciudad de Latacunga, a los 11 días del mes de marzo de 2026.

Atentamente,

Ing. MSc. Angel Guillermo Hidalgo Oñate
COORDINADOR DEL PROYECTO
C.I. 0503257404

UNIVERSIDAD TÉCNICA DE COTOPAXI

FACULTAD DE CIENCIAS DE LA INGENIERÍA Y APLICADAS

Latacunga, marzo del 2026

CERTIFICACIÓN DE INFORME DE SIMILITUD

En mi calidad de Tutor del proyecto de investigación con el tema: **"DESARROLLO DE UN ECOSISTEMA IOT INTEROPERABLE BASADO EN EL FRAMEWORK FIWARE EN LA UNIVERSIDAD TÉCNICA DE COTOPAXI"**, del estudiante **Arequipa Poaquiza Steven Paul,** de la Carrera de Sistemas de Información, remito la captura de pantalla del reporte del sistema de reconocimiento de texto Turnitin, con un porcentaje de coincidencias del 5%; y expreso una vez más, mi conformidad en cuanto a la dirección del trabajo de titulación.

Particular me comunico a usted para los fines pertinentes.

Ing Manuel William Villa Quishpe
C.C. 1803386950
TUTOR

turnitin Página 2 de 95 - Descripción general de integridad Identificador de la entrega trn:oid:::1:3502209774

5% Similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para ca...

Filtrado desde el informe

- Bibliografía
- Texto citado
- Texto mencionado
- Coincidencias menores (menos de 12 palabras)

Fuentes principales

4%	Fuentes de Internet
1%	Publicaciones
4%	Trabajos entregados (trabajos del estudiante)

Marcas de integridad

N.° de alertas de integridad para revisión

No se han detectado manipulaciones de texto sospechosas.

Los algoritmos de nuestro sistema analizan un documento en profundidad para buscar inconsistencias que permitirían distinguirlo de una entrega normal. Si advertimos algo extraño, lo marcamos como una a erta para que pueda revisarlo.

Una marca de alerta no es necesariamente un indicador de problemas. Sin embargo, recomendamos que preste atención y la revise.

AGRADECIMIENTO

Agradezco profundamente a mis Padres, por haber sido mi pilar fundamental de principio a fin. Mi esfuerzo, dedicación y confianza se lo debo a ellos, son el motor que me ha impulsado a seguir adelante.

Extiendo un especial agradecimiento al Ing. Manuel William Villa Quishpe, tutor de este trabajo de titulación, por su guía, compromiso y valiosas observaciones que permitieron el correcto desarrollo de este proyecto.

A la Universidad Técnica de Cotopaxi, mi alma máter, por brindarme una formación integral, tanto académica como humanística, que me ha permitido crecer como profesional y como ser humano.

Finalmente, mis abuelos, tíos y mis amigos, quienes me ofrecieron su apoyo constante, palabras de ánimo en los momentos no tan favorables que gracias a ellos logro la ejecución de este trabajo de titulación.

Steven Arequipa

DEDICATORIA

Dedico este trabajo a mi difunto padre, que en su tiempo de vida me ofreció su sacrificio y constante apoyo. Su ejemplo de vida aún vive en mí, la perseverancia ha sido el motor que me ha guiado a lo largo de mi formación.

A mis amigos y seres queridos, por estar presentes en cada paso del camino, brindándome palabras de aliento y acompañándome con paciencia y comprensión.

Y a mí madre, por seguir dándome la motivación, aliento y las ganas de seguir y no rendirme en la recta final, todo es gracias a ella, este camino que se culmino con determinación y compromiso.

Steven Arequipa

UNIVERSIDAD TÉCNICA DE COTOPAXI

FACULTAD DE CIENCIAS DE LA INGENIERÍA Y APLICADAS

TITULO: "DESARROLLO DE UN ECOSISTEMA IOT INTEROPERABLE BASADO EN EL FRAMEWORK FIWARE EN LA UNIVERSIDAD TÉCNICA DE COTOPAXI"

Autor: Steven Paul Arequipa Poaquiza

RESUMEN

El presente trabajo de investigación propone el diseño e implementación de un prototipo de ecosistema IoT interoperable basado en el framework FIWARE en la Universidad Técnica de Cotopaxi. El sistema, denominado SmartLab, integra cuatro dispositivos ESP32 con funcionalidades heterogéneas monitoreo de temperatura y humedad, detección de humo, control de acceso RFID y estación meteorológica bajo el estándar de comunicación NGSI v2. La arquitectura desplegada en AWS EC2 comprende Orion Context Broker, IoT Agent JSON, QuantumLeap, CrateDB y Mosquitto, todos orquestados mediante Docker Compose. Este ecosistema que se diseñó funciono por siete semanas de operación continua. Las latencias medidas fueron de 220 ms para HTTP/JSON, 300 ms para MQTT con TLS y 300 ms para HTTPS/JSON, valores considerados plenamente aceptables para aplicaciones de monitoreo ambiental en tiempo real. Una aplicación móvil desarrollada en Flutter complementa el sistema, permitiendo la gestión de usuarios, dispositivos, tarjetas RFID y la visualización de datos históricos mediante consultas directas a CrateDB. El análisis económico demuestra que el costo total de implementación resulta viable a comparación de plataformas propietarias que requieren suscripciones. Los resultados validan la hipótesis de investigación, confirmando que FIWARE permite integrar dispositivos heterogéneos mediante estándares abiertos, reducir la fragmentación de información y eliminar la dependencia de plataformas propietarias en entornos universitarios con recursos limitados.

Palabras clave: FIWARE, IoT, NGSI v2, interoperabilidad, ESP32, Orion Context Broker, QuantumLeap, CrateDB, MQTT, Smart Campus.

UNIVERSIDAD TÉCNICA DE COTOPAXI

FACULTAD DE CIENCIAS DE LA INGENIERÍA Y APLICADAS

THEME: "DEVELOPMENT OF AN INTEROPERABLE IoT ECOSYSTEM BASED ON THE FIWARE FRAMEWORK AT THE COTOPAXI TECHNICAL UNIVERSITY"

Author: Arequipa Poaquiza Steven Paul

ABSTRACT

This research project develops an interoperable IoT ecosystem based on the FIWARE framework at the Technical University of Cotopaxi, with the aim of integrating heterogeneous devices using open standards and reducing dependence on proprietary platforms. The problem driving this project lies in the fragmentation of information, where monitoring and IoT systems operate in isolation, with no possibility of exchanging information under a common data model. Given this reality, FIWARE presents itself as a robust alternative by proposing the NGSI v2 standard as a unified data context model, promoted by the European Union and adopted globally in smart city initiatives. The system developed, called Smart Lab, integrates four ESP32 modules with different functionalities, temperature and humidity monitoring, smoke detection, RFID access control and a weather station under the NGSI v2 communication standard. The architecture deployed on AWS cloud infrastructure using Docker Compose comprises the Orion Context Broker, IoT Agent JSON, Quantum Leap, Crate DB and Mosquito components, all of which are open-source and license-free. This combination of components ensures interoperability between heterogeneous devices, automatic logging of time-series data and secure connectivity via MQTT with TLS.A hybrid methodology was used for development, combining the V-Model for the FIWARE infrastructure and the ESP32 devices since they require rigorous validation at each phase and the Iterative-Incremental Model for the mobile application developed in Flutter. This application enables the management of users with different roles, the registration and control of IoT devices, the administration of RFID cards, and the visualization of historical data through direct queries to Crate DB. The results obtained validate the research hypothesis, confirming that FIWARE allows for the integration of heterogeneous devices using open standards, reduces information fragmentation, and eliminates dependence on proprietary platforms in university environments.

KEYWORDS: FIWARE, IoT, NGSI v2, Interoperability, ESP32, Orion Context Broker, Quantum Leap, Crate DB, MQTT, Smart Campus.

UNIVERSIDAD TÉCNICA DE COTOPAXI

CENTRO DE IDIOMAS

AVAL DE TRADUCCIÓN

En calidad de Docente del Idioma Inglés del Centro de Idiomas de la Universidad Técnica de Cotopaxi; en forma legal **CERTIFICO** que:

La traducción del resumen al idioma Inglés del tema del proyecto de investigación cuyo título versa: **"DESARROLLO DE UN ECOSISTEMA IOT INTEROPERABLE BASADO EN EL FRAMEWORK FIWARE EN LA UNIVERSIDAD TÉCNICA DE COTOPAXI"** presentado por: **Arequipa Poaquiza Steven Paul** egresado de la Carrera de Sistemas de Información perteneciente a la **Facultad de Ciencias de la Ingeniería y Aplicadas** lo realizó bajo mi supervisión y cumple con una correcta estructura gramatical del Idioma.

Es todo cuanto puedo certificar en honor a la verdad y autorizo al peticionario hacer uso del presente aval para los fines académicos legales.

Latacunga, marzo de 2026

Atentamente,

MSc. Alison Mena Barthelotty
DOCENTE CENTRO DE IDIOMAS-UTC
CI: 0501801252

CENTRO DE IDIOMAS

ÍNDICE GENERAL

1 Tabla de contenido

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INFORMACIÓN GENERAL

Tema del proyecto: Desarrollo de un ecosistema IoT interoperable basado en el framework FIWARE en la universidad técnica de Cotopaxi.

Modalidad de Titulación:

En la Tabla 1 se muestra el tipo de modalidad:

Tabla 1. Tipo de modalidad

MODALIDAD DE TITULACIÓN	HOMOLOGACIONES PARA INFORME FINAL DE TITULACIÓN	SELECCIÓN
Propuesta tecnológica	Informe de propuesta tecnológica	
	Patente, Modelo de utilidad, Certificado de propiedad intelectual.	
	Artículo científico	
Proyecto de investigación	Informe de Proyecto de investigación	X
	Artículo científico	
	Patente, Modelo de utilidad, Certificado de propiedad intelectual.	
Exámen de indicadores de RDA		

Trabajo de Titulación Vinculado al Proyecto: Orientación formativa práctico-colaborativa en carreras de Ciencia, Tecnología, Ingeniería y Matemática (STEM) en la provincia de Cotopaxi.

Equipo de Trabajo del Trabajo de Titulación: Arequipa Poaquiza Steven Paul, Ing. Manuel Villa como Tutor de Tesis.

Área de Conocimiento: 0613 Software y desarrollo de análisis de aplicativos.

Línea de investigación: Tecnologías de la información y las comunicaciones, robótica, automatización y optimización de sistemas.

Sublíneas de investigación de la Carrera: Ciencias informáticas para la modelación de Sistemas de Información a través del desarrollo de software.

2. INTRODUCCIÓN

En la actualidad, el desarrollo tecnológico ha permitido transformar diversos entornos mediante la incorporación de soluciones inteligentes que optimizan procesos y mejoran la seguridad. En este contexto, el Internet de las Cosas (IoT), se ha consolidado como una de las principales herramientas para conectar dispositivos físicos con sistemas digitales, permitiendo el monitoreo y control en tiempo real de distintos entornos. Según un informe reciente de la Organización para la Cooperación y el Desarrollo Económicos (OCDE), la adopción del IoT ha crecido significativamente en sectores como la manufactura y la atención médica, mejorando la eficiencia y reduciendo costos operativos [1].

Para el uso de IoT se utilizan sistemas gestores de la información pudiendo recopilar datos de diversos ámbitos como puede ser la humedad del suelo, la temperatura, las luminarias o en su defecto la detección de cuantas personas usa un transporte público que con la ayuda de otros programas se puede hacer un análisis para la toma de decisiones en las organizaciones.

Al momento de usar este mismo patrón para hacer los sistemas de recolección de datos de diferentes entornos, han venido existiendo estándares para que la información no se cree en ecosistemas aislados, es decir que los datos que se vayan recolectando no sean guardados con diferentes protocolos o diferentes codificaciones y sean todo un solo conjunto, pero los estándares que rigen a este criterio son de paga y a su vez no son abiertos.

En este escenario surge la necesidad de adoptar estándares abiertos que permitan unificar la comunicación entre dispositivos y plataformas heterogéneas. Es precisamente en este punto donde el framework FIWARE se presenta como una alternativa sólida. Su objetivo principal es asegurar la interoperabilidad, escalabilidad y portabilidad de las soluciones desarrolladas bajo este ecosistema.

FIWARE aparte de fijar un estándar también posee componentes modulares que actúan como microservicios para las comunicaciones así como en el ámbito de las industrias 4.0 que trabaja con datos de robótica y automatización de procesos, las agroindustria también se ven

beneficiadas ya que trabajan con datos de humedad de suelo, riego, cultivo y demás, también entran allí lo que son las Smart City donde los datos que se recopila son más amplios y se necesita de más capacidad para abastecer la recopilación de datos y a su vez para la visualización de los mismos pero si todo esto no hablan el mismo idioma entre componentes ya sea en codificación, protocolo o contexto, tiende a tener contratiempos si no es que llegue a saturar el servidor o gestor de información. Es esta parte es en la que permite que los sistemas se comuniquen de manera más fluida y eficiente logrando así que no se genere cuellos de botella y garantizando la escalabilidad de las soluciones.

El principal valor de FIWARE radica en la capacidad de manejar y mostrar información normalizada de contexto y lo más importante en tiempo real, lo que facilita que cualquier aplicación y servicio que use este Framework puedan compartir los mismos datos para trabajar de manera colectiva. Esta normalización no solo motiva una interoperabilidad entre sistemas y soluciones diferentes, sino que también mejora significativamente el desarrollo de nuevas aplicaciones.

FIWARE es una plataforma de código abierto promovida por la Unión Europea que permite la compatibilidad entre aplicaciones a través de estándares como NGSIv2 y componentes como el Context Broker. Esta tecnología facilita la integración de sistemas inteligentes de forma escalable y eficiente [2].

Se pueden usar en diferentes aplicaciones que se comporten de manera similar y que utilicen los mismos datos para proporcionar una excelente información para todos, a estas herramientas se las conoce como Generic Enablers, lo cual permite desarrollar una lógica de negocio específica para cualquier aplicación, reduciendo el tiempo y el esfuerzo para construir soluciones escalables. Además, el soporte que nos proporciona la comunidad de FIWARE hace que dichas soluciones puedan estar al alcance de todos.

Con el fin de comprender su funcionamiento y demostrar su aplicabilidad, este estudio propone la implementación de un ecosistema IoT interoperable basado en FIWARE en la Universidad Técnica de Cotopaxi (UTC), en la Facultad de Ciencias de la Ingeniería y Aplicadas (CIYA). Utilizando dispositivos reales para la gestión inteligente de recursos permitirá evidenciar el potencial del framework para ser adoptado en entornos universitarios fortaleciendo la trazabilidad y eficiencia operativa.

El uso de FIWARE en instituciones educativas no solo habilita la creación de Smart Campus, sino que abre oportunidades para que estudiantes y docentes desarrollen sistemas basados en

estándares internacionales, aprovechando tecnologías libres y accesibles. Este trabajo representa un aporte significativo para el ámbito académico ecuatoriano, donde aún son escasas las implementaciones de este framework.

2.1.Situación problemática

El Internet de las Cosas (IoT) ha experimentado un crecimiento sin precedentes en la última década. Según IoT Analytics, el número de dispositivos IoT conectados a nivel mundial alcanzó los 18,5 mil millones en 2024, cifra que se proyecta ascenderá a 21,1 mil millones al cierre de 2025, con una tasa de crecimiento anual compuesta (CAGR) del 13,2% hasta 2030, cuando se estima superar los 40 mil millones de dispositivos [3]. Paralelamente, el mercado global de IoT fue valorado en USD 532,3 mil millones en 2024, con proyecciones de alcanzar USD 4.542,67 mil millones en 2033, lo que representa una CAGR del 26,9% [4].

Sin embargo, este crecimiento acelerado ha venido acompañado de una proliferación de plataformas, protocolos y estándares propietarios que operan de forma aislada. Cada fabricante y proveedor de soluciones IoT ha desarrollado sus propios ecosistemas cerrados, lo que genera lo que se denomina 'silos de datos': conjuntos de información que no pueden comunicarse ni integrarse con otros sistemas sin el uso de costosos intermediarios o adaptadores. Esta fragmentación no solo eleva los costos de integración, sino que dificulta el análisis holístico de los datos, obstaculiza la escalabilidad de los sistemas y genera una dependencia tecnológica de plataformas específicas [5].

La iniciativa FIWARE, impulsada por la Unión Europea y adoptada globalmente, surge como respuesta a esta problemática, proponiendo el estándar abierto NGSI v2 (Next Generation Service Interface) como modelo de contexto de datos unificado. A pesar de su relevancia, la adopción de este tipo de estándares abiertos sigue siendo limitada, especialmente en regiones en desarrollo, donde predominan soluciones propietarias de alto costo como AWS IoT Core o Microsoft Azure IoT Hub [6].

En América Latina, el mercado de IoT se encuentra en una fase de expansión acelerada, aunque con importantes brechas respecto a las economías desarrolladas. Según Cognitive Market Research, el mercado de IoT masivo en la región alcanzó USD 3.625,93 millones en 2024 y se proyecta crecerá a una CAGR del 22,6% hasta 2031, superando los USD 16.557,7 millones [5]. Asimismo, el mercado de transformación digital en Latinoamérica fue valorado en USD 107,23 mil millones en 2025, con una CAGR del 17,69%, impulsado en parte por iniciativas de

ciudades inteligentes y la creciente adopción de IoT en sectores como manufactura, agricultura, salud y gobierno [7].

A pesar de estas cifras alentadoras, la región enfrenta desafíos estructurales significativos. La falta de estandarización en los protocolos de comunicación IoT genera una fragmentación severa: organizaciones del mismo sector utilizan plataformas incompatibles, imposibilitando la interoperabilidad entre sistemas. Esta situación se agrava en el ámbito educativo, donde las instituciones de educación superior (IES) carecen en su mayoría de infraestructura IoT integrada y dependen de soluciones propietarias aisladas para la gestión de recursos físicos [7].

En el contexto ecuatoriano, las instituciones de educación superior han iniciado procesos de transformación digital, aunque con avances desiguales. Un estudio realizado en la Pontificia Universidad Católica del Ecuador Sede Esmeraldas (PUCESE) aplicó un instrumento de evaluación de madurez de Smart Campus y determinó que la institución obtuvo una puntuación de 2,86 sobre 6 puntos, concluyendo que no reúne las condiciones para ser considerada una universidad inteligente [8]. Este resultado es representativo de la realidad de la mayoría de las IES ecuatorianas, donde la incorporación de IoT para la gestión de recursos, control de accesos y monitoreo ambiental es incipiente y carece de integración sistémica.

2.2.Formulación del Problema

¿Cómo desarrollar un ecosistema IoT interoperable basado en estándares abiertos que permita integrar dispositivos heterogéneos y evite la fragmentación de datos en la Universidad Técnica de Cotopaxi?

2.3. Objeto y Campo de Acción

2.3.1. Objeto de investigación:

Interoperabilidad basado en el framework FIWARE

2.3.2. Campo de Acción:

Desarrollo de un ecosistema.

2.4. Beneficiarios

2.4.1. Directo

Tabla 2 Beneficiarios directos

Beneficiario	Detalle	Cantidad
Directo	Los beneficiarios directos del proyecto son los docentes y estudiantes de la Facultad de Ciencias de la Ingeniería y Aplicadas (CIYA) de la Universidad Técnica de Cotopaxi.	Docentes:90 aproximados Estudiantes: 1000 aproximados

2.4.2. Indirecto

Tabla 3 Beneficiarios indirectos

Beneficiario	Detalle	Cantidad
Indirecto	Desarrolladores IoT que buscan estandarizar ecosistemas aislados por medio de FIWARE.	600 aproximados

Cabe señalar que los 600 se obtuvieron de la comunidad FIWARE [9]

2.5. Justificación

El desarrollo de este proyecto surge del interés académico como el de explorar tecnologías emergentes que promuevan la interoperabilidad en el contexto del Internet de las Cosas (IoT). Se propone utilizar FIWARE que es un Framework de código abierto que permite integrar dispositivos, servicios y plataformas bajo estándares comunes, reduciendo la dependencia de múltiples APIs propietarias y facilitando la escalabilidad de soluciones tecnológicas.

El proyecto se sustenta en el uso de herramientas libres y accesibles, como Docker, bases de datos en la nube y servicios de infraestructura disponibles en AWS, lo cual permite una implementación técnica factible con recursos limitados. Además, el desarrollo de un prototipo funcional valida la aplicabilidad de FIWARE en escenarios reales como el control de accesos a laboratorios universitarios, sin requerir licencias costosas ni infraestructura compleja.

Desde el punto de vista educativo, permite a estudiantes e investigadores adquirir experiencia en una de las tecnologías del futuro como es FIWARE utilizadas en ciudades inteligentes, industrias e instituciones. A nivel institucional, ofrece un modelo replicable y escalable para

mejorar la seguridad y trazabilidad de accesos en los laboratorios de la Facultad CIYA. Además, contribuye al desarrollo de soluciones tecnológicas alineadas con estándares internacionales.

A nivel teórico, este trabajo aporta documentación sobre la integración de FIWARE en entornos universitarios latinoamericanos, donde la evidencia publicada es escasa, contribuyendo al estado del arte sobre implementaciones prácticas de ecosistemas IoT interoperables basados en estándares abiertos.

La importancia de este proyecto radica en su potencial para abrir camino al uso de FIWARE en el ámbito académico ecuatoriano, donde aún existen pocas iniciativas implementadas con esta tecnología. De este modo, no solo se promueve la innovación tecnológica, sino que también se sientan las bases para futuras investigaciones orientadas al desarrollo de sistemas interoperables, sostenibles y alineados con los principios de la transformación digital.

Desde el punto de vista práctico, el proyecto SmartLab entrega un sistema IoT funcional y en producción que resuelve necesidades concretas de los laboratorios de la Facultad CIYA de la UTC. El ecosistema integra cuatro dispositivos ESP32 con sensores heterogéneos temperatura, humedad, humo y control de acceso RFID bajo una arquitectura unificada basada en FIWARE, accesible mediante una aplicación móvil Flutter para administradores y docentes.

Los beneficios del proyecto se identifican en múltiples dimensiones. A nivel institucional, la UTC obtiene un sistema de monitoreo y control de accesos que mejora la seguridad y la interoperabilidad sin incurrir en los costos de licenciamiento de plataformas propietarias equivalentes.

A nivel académico, el proyecto beneficia directamente a estudiantes e investigadores de la carrera de Ingeniería en Sistemas de Información, quienes adquieren experiencia práctica con tecnologías de alta demanda en el mercado laboral, incluyendo FIWARE, IoT, FastAPI, Flutter, Docker y servicios en la nube (AWS).

El proyecto es viable porque todos los componentes de Fiware son de código abierto y libre acceso, contenedores como Docker y Docker Compose que, para garantizar su reproducibilidad en cualquier servidor compatible, recalcando que el framework es adaptable a diferentes entornos refiriéndonos a que funcionan en ecosistemas cerrados como servidores de empresas como ecosistemas abiertos que se lo conoce como la nube y respaldada por su comunidad, documentación oficial y promovido por la Unión europea.

2.6.Objetivos

2.6.1. General

Desarrollar un ecosistema IoT interoperable basado en el framework FIWARE en la Universidad Técnica de Cotopaxi, mediante la integración de dispositivos y servicios para que permitan la comunicación estandarizada.

2.6.2. Específicos

- Realizar una revisión bibliográfica actualizada sobre FIWARE y su aplicación en ecosistemas IoT interoperables.
- Diseñar la arquitectura del ecosistema IoT basado en FIWARE, integrando dispositivos, protocolos de comunicación (HTTP, HTTPS, MQTT) y servicios en la nube.
- Evaluar el desempeño del ecosistema mediante pruebas de interoperabilidad, latencia y escalabilidad en un entorno controlado.

2.7.Hipótesis y sistemas de tareas

2.7.1. Hipótesis

El desarrollo de un ecosistema IoT interoperable basado en el framework FIWARE en la Universidad Técnica de Cotopaxi, permitirá integrar dispositivos heterogéneos mediante estándares abiertos y reducir la fragmentación de información, así como también la dependencia de plataformas propietarias.

Tabla 4 Operacionalización de variables de la investigación

Variables	Definición conceptual	Dimensión	Indicadores	Instrumento
Variable independiente 1	Ecosistema IoT interoperable	Arquitectura del sistema	Componentes desplegados y operativos: Orion, IoT, Agent JSON, QuantumLeap,Fastapi, crateDB. Protocolos soportados: HTTP, MQTT/TLS, HTTPS	Revisión de configuración Docker Compose Consola AWS EC2
Variable dependiente 1	Integración de dispositivos heterogéneos mediante estándares abiertos.	Interoperabilidad de dispositivos heterogéneos.	Dispositivos integrados: módulos ESP + APP Latencia HTTP: 220 ms (Postman) Latencia MQTT/TLS: 300 ms Latencia HTTPS: 300 ms (Postman) Mensajes procesados: 438,300 en 7 semanas	Postman con GET a orion.
Variable dependiente 2	Reducción de la fragmentación de información	Datos en un solo estándar	consulta: GET entities Entidades disponibles en un solo response:4 Tablas históricas: 3 (etsensor, etaccessevent, etstation) Disponibilidad del Orion :7 semanas	Orion CB GET /v2/entities CrateDB Explorador Monitoreo Docker logs
Variable dependiente 3	Reducción de la dependencia de plataformas propietarias	Herramientas de desarrollo libre	Costo de licencias: $0 USD (100% open source) Costo total del sistema: $1150 USD	Facturas de compra AWS

2.7.2. Sistemas de Tareas

Aquí se muestra las tareas que hay que cumplir para poder resolver el problema de esta temática, la cual se muestra en la Tabla 5.

Tabla 5. Planificación de las actividades

Objetivos específicos	**Actividades (tareas)**	**Resultados esperados**	**Técnicas, Medios e Instrumentos**
Realizar una revisión bibliográfica sobre el framework FIWARE mediante fuentes oficiales y su aplicación en ecosistema	Revisión de literatura científica y documentación oficial de FIWARE y su integración con IoT. Documentación de las diferentes herramientas y tutoriales de FIWARE.	Documento de investigación de fuentes bibliográficas de las respectivas herramientas de FIWARE.	**Técnica:** Revisión documental. **Medios:** Bases de datos académicas como Scopus, Google Scholar y gestores bibliográficos como Mendeley. **Instrumentos:** Ficha Bibliografica.

Diseñar la arquitectura del ecosistema IoT basado en FIWARE, integrando dispositivos, protocolos de comunicación (HTTPS, HTTP, MQTT y servicios en infraestructura cloud.	Elaboración de la arquitectura detallada del prototipo basada en FIWARE. Desarrollo de la lógica de negocio del prototipo.	Diseño arquitectónico del prototipo (diagramas UML), diagrama del dispositivo de control de accesos. Guía de cómo usar FIWARE.	**Técnica:** modelo en V y modelo iterativo incremental, observación. Entrevista **Medios:** PC, herramientas de modelado como Lucidchart.app. **Instrumentos:** Diagrama de arquitectura. Tabla de observación. guía de la entrevista
Evaluar el desempeño del ecosistema IoT en un entorno controlado, verificando la interoperabilidad y validando así la efectividad de FIWARE.	Documentar el proceso de la implementación y demostración de los datos interoperables.	Tablas de pruebas, evidencia de las pruebas,	**Técnica:** pruebas http **Medios:** Postman, App **Instrumentos:** Tabla de prueba

3. FUNDAMENTACIÓN TEÓRICA

En 2021 presentaron el diseño de la plataforma IPVC Smart & Sustainable Campus (IPVC-S2C), una arquitectura basada en FIWARE con inteligencia en el borde orientada a instituciones de educación superior. El trabajo introduce la metodología de diseño empleada para la especificación de la arquitectura y evalúa hardware IoT de bajo costo capaz de ejecutar aprendizaje automático distribuido, presentando como prueba de concepto el monitoreo de calidad del aire interior en el campus [10]. Este trabajo es directamente comparable con el presente proyecto, que también adopta FIWARE como plataforma central para la gestión de un laboratorio universitario inteligente, aunque se diferencia en la incorporación de módulos ESP32 multi-sensor, comunicación MQTT con TLS y una aplicación móvil Flutter para la gestión de acceso y visualización de datos en tiempo real.

Se realizo una evaluación de rendimiento de la plataforma FIWARE simulando instalaciones IoT a gran escala mediante los protocolos CoAP y MQTT, evaluando su escalabilidad vertical y horizontal en despliegues en la nube. Los autores identificaron que escalar verticalmente no resulta costo-eficiente por debajo de 1.000 solicitudes por segundo, y que el escalado horizontal detrás de un balanceador de carga no incrementó significativamente el rendimiento [11]. Estos

hallazgos fundamentan las decisiones de diseño del presente proyecto en cuanto a la configuración del Orion Context Broker y QuantumLeap desplegados sobre AWS.

3.1. El Internet de las Cosas (IoT)

El Internet de las Cosas (IoT) hace referencia a un ecosistema de objetos físicos equipados con sensores, software y capacidades de comunicación que les permiten intercambiar información con otros dispositivos y sistemas a través de Internet. Su alcance es amplio, abarcando desde electrodomésticos cotidianos hasta equipos de alta complejidad en entornos industriales [1].

El Internet de las Cosas ha emergido como uno de los conceptos más destacados y en constante evolución dentro del ámbito de la tecnología de la información. Su desarrollo a lo largo de la última década ha consolidado una visión de una infraestructura global donde objetos físicos, cotidianos, están interconectados a través de la red, facilitando una conectividad ubicua que trasciende la interacción exclusiva entre personas [12].

Se puede decir que el internet de las cosas es una red de dispositivos físicos que también llamadas cosas que contienen sensores, software y otros tipos de tecnología, estos objetos pueden conectarse e intercambiar datos teniendo una conexión a internet.

3.1.1. Arquitectura General de Sistemas IoT

El diseño arquitectónico de los sistemas IoT generalmente adopta un modelo por capas que debe garantizar escalabilidad, adaptabilidad ante nuevos estándares y soporte para la integración de soluciones heterogéneas. Para que un sistema de este tipo funcione correctamente, es indispensable gestionar de forma coordinada todos sus elementos, abarcando cinco áreas funcionales clave, entre ellas la capacidad de desarrollar aplicaciones sobre la infraestructura IoT [13].

Bajo estos principios, se propone una arquitectura compuesta por las siguientes capas: Dispositivos, Gateways, Red, Nube/Centro de Datos, Aplicaciones, Gestión y Seguridad. En el nivel base se encuentran los dispositivos, que interactúan directamente con el entorno físico para capturar datos o ejecutar acciones. Su representación en dos rutas distintas refleja una realidad técnica importante: algunos dispositivos pueden conectarse directamente a la capa de red, mientras que otros dependen de un intermediario para lograrlo [13].

Ese intermediario corresponde a la capa de Gateways, cuya función es habilitar la conectividad de dispositivos que no son capaces de comunicarse directamente con la red, generalmente por

incompatibilidades de protocolo. Por encima de esta capa se encuentra la de Red, responsable de transportar el tráfico desde los dispositivos y gateways hacia la infraestructura de procesamiento. Finalmente, la capa de Nube/Centro de Datos centraliza el procesamiento de la información recibida y gestiona los comandos que se envían de vuelta hacia los dispositivos, ya sea generados localmente o desde otras capas de la arquitectura [13].

Para entender cómo funciona una arquitectura con IoT, es necesario comprender que estas arquitecturas se componen por capas, que empezando por la capa de los dispositivos es donde se encuentran todos los dispositivos listos para captar y recopilar los datos, la siguiente capa es la Gateway que actúan como los traductores o como puente entre los dispositivos haciendo posible la comunicación, la siguiente capa es la de Red, es donde los datos viajan puede ser por medio de Wi-Fi , redes móviles o Ethernet, la siguiente capa es la capa de la nube donde los datos se almacenan y se procesan para poder ser analizados, es decir que funciona como el cerebro de cualquier aplicación, se tiene la capa de aplicaciones que son las interfaces donde se visualiza dichos datos, pueden ser en un smartphone o en una computadora, al igual se tiene capas adicionales la que se encuentra a la izquierda es la de Gestión que se encarga de monitorear a los dispositivos y el sistema en sí, y se asegura de que todo funcione correctamente, la capa que se encuentra a la derecha es la de seguridad que su función es proteger los datos y evitar accesos no autorizados o ataques externos.

Figura 1. Arquitectura para el Internet de las Cosas [13].

3.1.2. Protocolos de Comunicación Comunes en IoT

A continuación, se presenta los protocolos más usados en el internet de las cosas y son considerados los más importantes, los cuales se presentan en la siguiente Tabla 6:

Tabla 6. Protocolos de Comunicación en IoT.

Protocolo	Definición
MQTT (Transporte de Telemetría en Cola de Mensajes)	Pertenece a los protocolos ligeros de mensajería de IoT . MQTT utiliza un enfoque de mensajería de publicación-suscripción y funciona mejor en redes con poco ancho de banda, alta latencia y poca fiabilidad. Es ideal para el intercambio de datos en tiempo real en aplicaciones de IoT, una opción ideal para dispositivos con capacidad de procesamiento limitada [14].
AMQP (Protocolo Avanzado de Cola de Mensajes)	AMQP es un protocolo de mensajería versátil de código abierto. Su principal objetivo es facilitar la comunicación dentro del middleware orientado a mensajes (MOM). Por lo tanto, AMQP es una excelente opción para entornos IoT que buscan colas y enrutamiento de mensajes seguros y eficientes [14].
CoAP (Protocolo de Aplicación Restringida)	Este protocolo es ideal para redes con recursos limitados. Facilita la transferencia de datos con baja sobrecarga, lo que lo hace ideal para aplicaciones donde la eficiencia energética es crucial. Como protocolo de capa de aplicación en IoT, CoAP está diseñado para dispositivos con recursos limitados, lo que facilita una comunicación eficiente en entornos con recursos limitados [14].
XMPP (Protocolo Extensible de Mensajería y Presencia)	Desarrollado inicialmente para la mensajería instantánea, XMPP se ha consolidado en el IoT. Permite la comunicación entre dispositivos en tiempo real en sistemas escalables [14].
HTTP (Protocolo de Transferencia de Hipertexto)	Si bien no es exclusivo del IoT, HTTP es crucial para las aplicaciones web de IoT y probablemente el protocolo más conocido. Conecta dispositivos IoT y servidores web, facilitando la gestión de datos a través de interfaces web. Sin embargo, este protocolo presenta varias desventajas, como un consumo energético considerable y problemas de peso [14].

Los protocolos que son muy necesarios para el diseño del prototipo son: MQTT que es muy ideal para permitir la comunicación con el módulo ESP32 hacia FIWARE, mientras que HTTP y HTTPS es el protocolo más importante para las comunicaciones dentro de FIWARE con los

componentes que serían el IoT Agent hacia el Context Broker y la API que realiza las validaciones.

3.1.3. Aplicaciones del Internet de las Cosas (IoT)

En la actualidad el internet de las cosas es muy usado en empresas, proyecto o soluciones inteligentes desde sectores industriales, hasta Ciudades Inteligentes, a continuación, se describen los principales ámbitos.

- **IoT en agricultura.** La incorporación de sensores en el sector agrícola ha transformado la manera en que se gestionan los cultivos. Mediante el monitoreo continuo de variables como la humedad del suelo, su composición química, el contenido de nutrientes y las condiciones climáticas del entorno, es posible construir estrategias de cultivo más precisas que anticipan problemas, optimizan el riego y orientan el uso adecuado de fertilizantes [15].

- **IoT en medicina.** Los dispositivos wearables y otros gadgets conectados han abierto nuevas posibilidades en el seguimiento de la salud. A través de ellos, profesionales médicos y familiares pueden acceder en tiempo real a indicadores como la presión arterial, los niveles de glucosa o la temperatura corporal de un paciente, lo que facilita tanto la detección temprana de anomalías como la anticipación de cuadros clínicos futuros [15].

- **IoT en Ciudades Inteligentes.** Una de las prioridades de muchas ciudades del mundo es volverse inteligentes y conectarse entre sí; la base de ello es la tecnología IoT que busca optimizar el funcionamiento complejo de una urbe, dinamizarlo, volverlo más seguro e intentar resolver problemas como la vialidad, el transporte y la contaminación, que no fueron previstos a tiempo [15].

- **IoT en el hogar.** En el ámbito doméstico, el IoT conecta electrodomésticos, sistemas de climatización, dispositivos de seguridad y mecanismos de control energético. Desde refrigeradores que detectan el agotamiento de productos hasta cámaras y sensores que alertan sobre intrusiones o permiten el control remoto de cerraduras, las aplicaciones domóticas buscan mejorar la comodidad y reducir costos operativos. Sin embargo, su adopción en hogares ha sido más lenta que en entornos empresariales, dado que no todas las soluciones disponibles responden a necesidades cotidianas concretas [15].

3.2 Frameworks de traducción

Un framework de traducción se define como un conjunto estructurado de herramientas, modelos, bibliotecas y servicios que permiten la traducción automática de contenidos entre diferentes idiomas, facilitando su integración dentro de aplicaciones, plataformas o ecosistemas digitales más amplios. Estos frameworks se apoyan principalmente en técnicas de Procesamiento del Lenguaje Natural (PLN) y aprendizaje automático, particularmente en modelos de traducción automática neuronal (Neural Machine Translation, NMT), con el objetivo de ofrecer traducciones más precisas, contextuales y escalables.

En el contexto de los ecosistemas digitales y plataformas inteligentes, los frameworks de traducción cumplen un rol complementario, al encargarse de la conversión lingüística de la información, sin gestionar de forma nativa el contexto, la interoperabilidad entre sistemas o la estandarización de los datos.

Algunas de estas tecnologías que ayudan a la traducción de información de contexto son:

- ThingsBoard: plataforma de código abierto para Internet de las Cosas (IoT). Permite gestionar dispositivos, recopilar y procesar datos en tiempo real, visualizarlos en tableros interactivos y automatizar flujos de trabajo.
- OpenRemote: plataforma IoT de código abierto, diseñada para gestionar flotas grandes de dispositivos de forma profesional. Se enfoca en fabricantes de equipos e integradores de sistemas, permitiendo conectar dispositivos, automatizar procesos y visualizar datos en apps móviles o web.
- Kaa IoT Platform: plataforma de middleware de código abierto para desarrollar y gestionar soluciones IoT completas de extremo a extremo. Facilita la conexión de dispositivos, gateways y sensores mediante protocolos como MQTT y HTTP, con énfasis en gestión de dispositivos, recolección de datos, análisis y actualizaciones OTA.
- AWS Smart Territory: conjunto de herramientas y módulos estandarizados de AWS para construir soluciones inteligentes en ciudades y territorios. Basado en el ecosistema abierto FIWARE, integra el Context Broker con módulos IoT para gestionar datos en tiempo real y evitar silos.
- FIWARE: FIWARE es un framework de código abierto diseñado para acelerar el desarrollo de soluciones inteligentes, especialmente en IoT y ciudades inteligentes.

3.2.1. Comparación de los frameworks de traducción

Según la investigación realizada, se pudo entender que, si bien existen muchas plataformas y soluciones que funcionan como un Framework en el sentido de que nos ofrecen un conjunto de servicios para construir aplicaciones IoT, no se encontró que se comporten o que promuevan la estandarización abierta o la compatibilidad fuera de su ecosistema como lo hace FIWARE, para ello nos permitió hacer un cuadro comparativo de estas plataformas comparadas con FIWARE para entender mejor su enfoque y donde es más conveniente usarlos, en la Tabla 7 se muestra dicha comparativa.

Tabla 7. Comparativa de FIWARE con otras Plataformas IoT.

Característica Clave	**FIWARE**	**ThingsBoard**	**OpenRemote**	**Kaa IoT Platform**	**AWS Smart Territory Framework (STF)**
Definición Principal	Es un Framework de Código Abierto y es impulsado por estándares para poder tener información de contexto en tiempo real.	Es una plataforma IoT de Código Abierto que recopila información de los datos de dispositivos y se puede visualizar reglas.	Es una plataforma Código Abierto que se usa para la gestión y automatización y resulta ser muy flexible para integrar diferentes protocolos.	Es una plataforma IoT de Código de Abierto y flexible para poder desarrollar aplicaciones IoT, y permite gestionar dispositivos y analizar los datos.	Es un Framework que se encuentra en la nube de AWS y utiliza componentes de FIWARE para construir soluciones inteligentes.
Enfoque en Estandarización de Datos	Su nivel de enfoque en la estandarización es Alto y promueve el ETSI NGSI-LD O NGSIv2.	Su nivel de enfoque en la estandarización es Medio y soporta protocolos como MQTT y HTTP, pero el usuario define su estandarización.	Su nivel de enfoque en la estandarización es Medio y permite conectar múltiples protocolos como MQTT, CoAP y LoRaWAN, pero de igual forma si estandarización es manual.	Su nivel de enfoque en la estandarización es Medio y usa protocolos como MQTT y HTTP, también formatos JSON.	Su nivel de enfoque en la estandarización es Alto, y aprovecha el Context Broker de FIWARE, y otros componentes de y este mismo.

Arquitectura Típica	Su arquitectura es modular y se basa en los Generic Enablers.	Su arquitectura es microservicios para gestionar los dispositivos.	Su arquitectura es modular y se basa en gestores de protocolo y motor de reglas.	Su arquitectura es microservicios y se adapta a diversas topologías de despliegue.	Su arquitectura es modular y se basa en AWS Cloud y combina sus servicios con los módulos de FIWARE.

3.3. Framework FIWARE

Se puede definir como un conjunto de especificaciones desarrolladas en Europa, diseñadas para facilitar la creación de aplicaciones inteligentes en una diversidad de sectores verticales. Su origen se remonta al Séptimo Programa Marco (FP7) de la Comisión Europea, como parte del Programa de Asociación Público-Privada de Internet Futuro (FI-PPP), con el objetivo primordial de ofrecer una alternativa de código abierto a las soluciones propietarias que tradicionalmente se empleaban en infraestructuras, particularmente en el ámbito de las ciudades inteligentes. De esta manera, FIWARE se distingue por ser una solución abierta, accesible al público y libre de regalías, orientada a la gestión de redes de sensores. Esta naturaleza intrínseca de código abierto y su diseño modular confieren a los sistemas construidos bajo el estándar FIWARE una notable capacidad de replicación, facilidad de despliegue y una muy alta escalabilidad, permitiendo además la continua actualización de sus nodos con los avances más recientes en el procesamiento y clasificación de datos [16].

Figura 2 Herramienta Fiware

FIWARE como cualquier Framework tiene un conjunto de componentes o herramientas de código abierto pero lo que lo hace diferente a un simple Framework es que hace más fácil la creación y la gestión al momento de querer dar solución en muchas áreas principalmente en el área de las ciudades inteligentes ya que está basado en el internet de las cosas y el procesamiento de los datos en tiempo real.

Se utiliza comandos curl para poder interactuar con FIWARE a través de sus APIs, permitiendo enviar solicitudes HTTP y recibir respuesta, por ejemplo, este comando que es curl y se quiere ver la versión:

curl -X GET 'http://localhost:1026/version'

Tal como se muestra en la Figura 3.

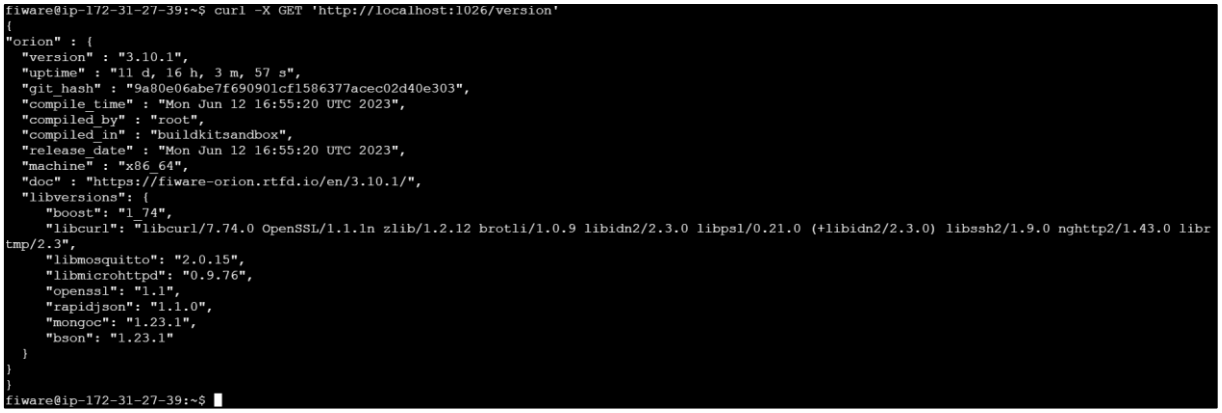


Figura 3. Consultando con el comando curl.

En la Figura 4 se puede visualizar como es una comunicación entre dispositivos IoT normalmente, hasta ese entonces los datos que entran y salen no son complejos y no requieren estándares para su comunicación, pero por otro lado cuando estos dispositivos IoT quieren entender datos más complejos o que se alojan en una base de datos, puede haber un problema de entendimiento entre ello ahí aparecen los sistemas IoT aislados, ya que cada uno se encarga únicamente de los datos que gestiona internamente. Como se muestra en la Figura 5, cada sistema IoT implementado con este formato tiende a requerir una instalación independiente debido a que utilizan diferentes codificaciones y estándares. Los datos que manejan estos sistemas ingresan en diversos formatos, por ejemplo, reconocimiento de huella dactilar, detección de movimiento, temperatura o procesamiento de datos complejos y, de manera similar, se transmiten con codificaciones distintas. Esto genera interferencia y complejidad cuando se intenta unificar los datos provenientes de los sensores distribuidos.

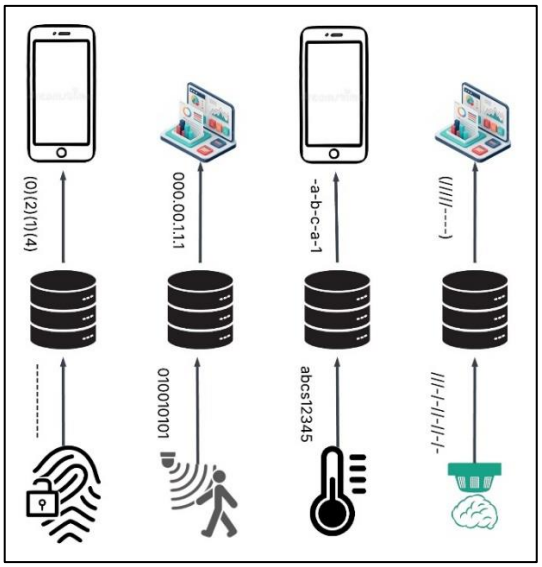


Figura 4. Comunicación normal de dispositivos IoT.

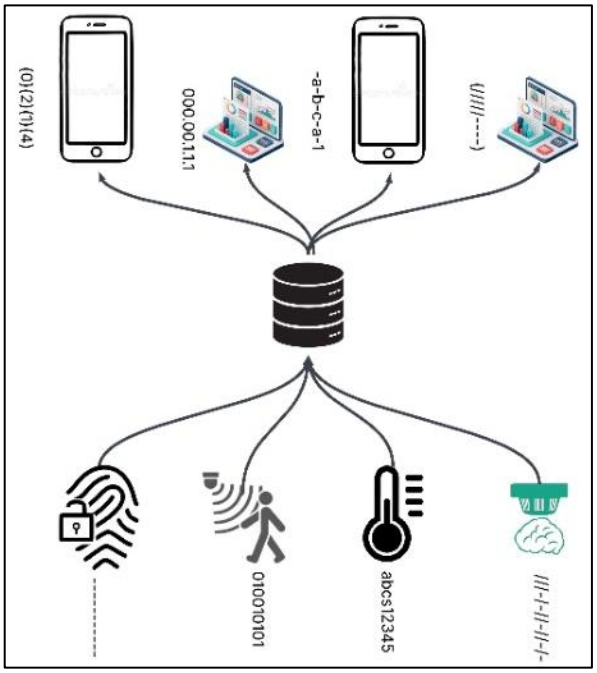


Figura 5. Comunicación unificada entre diferentes dispositivos IoT.

Aquí es donde entra FIWARE que lo que quiere es que todos estos dispositivos se comuniquen por medio de un mismo estándar y de esta forma permitir solucionar la complejidad que existe al momento de entender esos datos que se encuentran es diferentes dispositivos IoT, dando por hecho una solución de interoperabilidad y que todos compartan ese contexto y lo entiendan, tal como se muestra en la Figura 6.

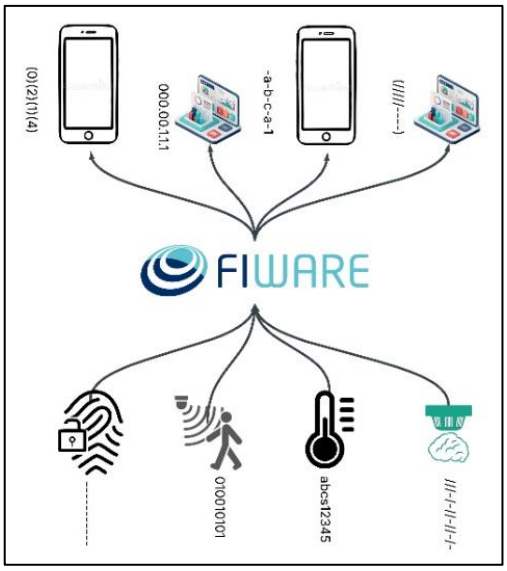


Figura 6. Estandarización con FIWARE.

3.3.1 Generic Enablers

Son esenciales del Framework, se organizan en tres categorías principales para estructurar sus funcionalidades. La primera categoría se dedica a la Gestión de Contexto, e integra módulos de software que operan mediante interfaces de publicación/suscripción y capacidades de análisis de Big Data. La segunda categoría abarca la Gestión de Datos Abiertos y de APIs, facilitando la exposición y el consumo de información. Finalmente, la tercera categoría se concentra en la Gestión de Identidad y Control de Acceso, un aspecto crítico para la seguridad de las aplicaciones, tal como se muestra en la Figura 7. La documentación necesaria para la instalación, configuración y despliegue de cada uno de estos GEs puede consultarse en el Catálogo de Desarrolladores de FIWARE [17].

Puede decirse que los Generic Enablers son bloques de construcción necesarios y que se pueden utilizar en cualquier tipo de proyecto con IoT, son las herramientas estandarizadas que nos proporciona FIWARE, con ellos se puedo construir soluciones efectivas e inteligentes.

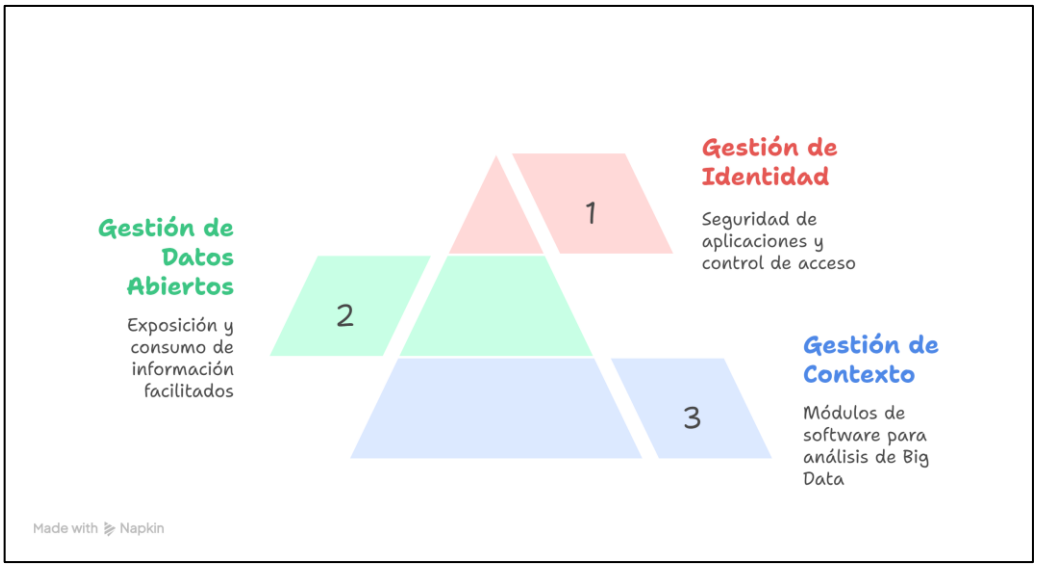


Figura 7. Categorías Principales de los Generic Enablers de FIWARE.

3.3.2. Context Broker

Conocido como Orion, constituye el componente esencial de los conectores IDS en las implementaciones de la Arquitectura IDS basadas en FIWARE. Su función principal radica en ofrecer las APIs NGSI de FIWARE, junto con su modelo de información asociado (entidades, atributos y metadatos), como la interfaz fundamental para el intercambio de datos entre los distintos participantes del IDS. Esta armonización entre los modelos de datos específicos de cada dominio y la API NGSI estandarizada facilita que tanto los productores como los consumidores de datos interactúen de forma fluida. Es decir, los productores utilizan NGSI para publicar o exponer la información que ofrecen (usualmente mediante un Adaptador de Sistema), mientras que los consumidores recuperan estos datos o se suscriben para recibir notificaciones sobre ellos. Todas las operaciones de publicación, consumo, suscripción y notificación se gestionan a través de la API NGSI. Para salvaguardar el acceso al Conector, tanto para la publicación como para las subcripciones, se ha integrado un Punto de Aplicación de Políticas (PEP). Este componente intercepta cada solicitud para verificar la autorización de acceso al recurso solicitado, complementando la arquitectura de identidad y control de acceso [18], tal como se muestra en la Figura 8.

El Orion Context Broker es el componente más importe que tiene FIWARE ya que es la parte central, resulta ser como el cerebro o el corazón de este Framework, este gestiona el contexto o los datos de un entorno en tiempo real.

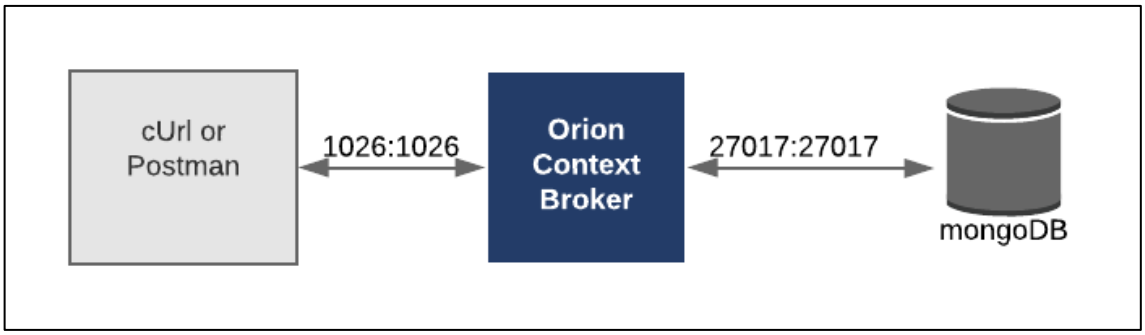


Figura 8. Arquitectura del Context Broker [18].

3.3.3. Gemelos digitales

Un gemelo digital también conocido como digital twin es una representación digital que se encuentra en Orion antes ya mencionado, el cual siempre tiene la información más actualizada de un objeto real como puede ser un sensor o una máquina, haciendo posible que otras aplicaciones lo entiendan y lo usan en tiempo real.

3.3.4. IoT Agents

Según la Documentación de FIWARE, IoT Agents es un componente que permite que un grupo de dispositivos envíe sus datos a un Context Broker que es el núcleo del Framework) y se administren desde él mediante sus propios protocolos nativos. Los agentes de IoT también deben ser capaces de ocuparse de los aspectos de seguridad de la plataforma FIWARE para la autenticación y autorización del canal y proporcionar otros servicios comunes al programador del dispositivo.

Ejemplo:

Se tiene un sensor de humedad en un invernadero que envía la lectura cada 5 minutos por HTTP en un formato muy simple:

- *humedad=55.*

Para que FIWARE específicamente el Orion Context Broker entienda esa información, se necesita un IoT Agent y hay que saber con qué protocolo se trabaja como puede ser HTTP o MQTT, en base a eso se define el Iot Agent.

- El sensor envía *GET /iot/d?k=4jggokgpepnvsb2uv4s40d599&d=humedad%7C55* al IoT Agent.

- El *k* es tu API Key, y *d* son los datos humedad.

- El IoT Agent recibe esto, lo traduce y lo envía al Orion Context Broker como una actualización NGSI:

```
{
   "id": "urn:ngsi-ld:Sensor:hum001",

   "type": "Sensor",
   "humidity": {
      "value": 55,
      "type": "Number"
   }
}
```

- Una vez terminado el proceso anterior, el dato de la humedad está disponible en el Context Broker para que otras aplicaciones la usen.

Existen varios tipos de IoT Agents, como se puede visualizar en la Tabla 8.

Tabla 8. Tipos de IoT Agents.

IoT Agents	Propósito	Protocolo
IoT Agent for JSON (IoTAgent-JSON)	Conectar dispositivos IoT que envían datos en formato JSON al Context Broker.	HTTP, MQTT (con payload JSON)
IoT Agent for UltraLight 2.0 (IoTAgent-UL)	Conectar dispositivos IoT con recursos muy limitados que usan un formato de datos simple y ligero (UltraLight 2.0).	HTTP, MQTT (con payload UltraLight 2.0)
IoT Agent for LwM2M (IoTAgent-LWM2M)	Integrar dispositivos que utilizan el protocolo Lightweight M2M (LwM2M) para gestión y datos.	CoAP (protocolo subyacente de LwM2M)
IoT Agent for LoRaWAN (IoTAAgent-LoRaWAN)	Conectar dispositivos que operan en redes de baja potencia y largo alcance LoRaWAN.	Integración con servidores de red LoRaWAN (ej., The Things Stack)
IoT Agent for OPC-UA (IoTAgent-OPCUA)	Conectar sistemas y máquinas industriales que utilizan el protocolo OPC Unified Architecture (OPC UA).	OPC UA (comúnmente sobre TCP/IP)

3.3.5. Keyrock

Keyrock constituye la implementación de referencia del Generic Enabler de Gestión de Identidades dentro del ecosistema FIWARE. Entre sus principales capacidades se encuentran la autenticación segura de usuarios y dispositivos mediante el protocolo OAuth 2.0, la

administración de perfiles, el manejo de datos personales bajo criterios de privacidad, la implementación de inicio de sesión único (SSO) y la federación de identidades a través de múltiples dominios administrativos [19].

Es aquel que se encarga de gestionar la identidad y el acceso, más abreviado como IAM, este componente permite controlar quien puede acceder a qué recursos y funcionalidades dentro de una plataforma hecha en FIWARE.

3.3.6. Wilma

Se refiere a la API expuesta por Keyrock para la gestión de usuarios, organizaciones, roles y permisos. Es la interfaz RESTful de la aplicación que se usa para interactuar programáticamente con Keyrock para realizar operaciones como:

- Crear, leer, actualizar y eliminar usuarios.
- Gestionar organizaciones.
- Asignar roles a usuarios.
- Definir permisos.
- Gestionar aplicaciones.

Esencialmente, cuando se habla de Wilma en el contexto de Keyrock, se refiere al conjunto de endpoints de la API que permiten la administración programática de la identidad y el acceso.

Actúa como una capa de seguridad que ofrece funcionalidad de proxy basada en esquemas OAuth2 [19].

Sin Wilma nos tocaría ingresar al interfaz de Keyrock manualmente, crear un usuario para asignarle un rol y asegurarse de que tenga los respectivos permisos, lo cual si lo que se quiere automatizar ese proceso entra esta API.

3.3.7. Cygnus

Cygnus fue diseñado para gestionar la persistencia de datos en sistemas de almacenamiento externo, como bases de datos, sistemas de archivos o sistemas de big data. Actúa como

intermediario entre OCB y los sistemas de almacenamiento, facilitando la integración de datos en tiempo real generados por dispositivos IoT y otros sistemas [19].

Las ventajas de usar Cygnus son la adaptabilidad a diferentes sistemas de almacenamiento simultáneamente, la transformación y el enriquecimiento de datos, y la escalabilidad y el rendimiento para gestionar una gran cantidad de datos [19].

Es un componente de igual importante que su función es mantener la persistencia de los datos de contexto que le proporciona Orion Context Broker ya que este último no lo puede hacer por sí solo, Cygnus permite recopilar la información y almacenarla en diferentes bases de datos para que estén listo y ser usados en cualquier momento.

3.3.8. PEP Proxy

Es un middleware para usar entre habilitadores genéricos que desafían los derechos de un usuario.

Según la documentación web oficial de FIWARE es un Generic Enablers de Seguridad que actúa como un punto de aplicación de políticas. Se interpone entre tus aplicaciones/servicios y los usuarios/dispositivos que intentan acceder a ellos. Su función principal es:

- Interceptar las peticiones de acceso a tus recursos.
- Comunicarse con Keyrock para obtener una decisión de autorización.
- Hacer cumplir la política de acceso: permitir o denegar la petición basándose en la respuesta de Keyrock.
- También puede manejar la autenticación OAuth2, redirigiendo a los usuarios a Keyrock para iniciar sesión y luego validando los tokens recibidos.

3.3.9. Comparativa entre Wilma y PEP Proxy

Es necesario entender cuáles son las diferencias entre estos dos componentes que trabajan a lado del Keyrock para la identificación de los usuarios, aquí se presenta una tabla con la respectiva comparativa, tal como se muestra en la Tabla 9.

Tabla 9. Comparativa entre Wilma y PEP Proxy.

Característica	Wilma (API de Keyrock)	PEP Proxy (Policy Enforcement Point Proxy)
Tipo de Componente	API RESTful de Keyrock para la gestión de identidad y acceso (parte de Keyrock).	Generic Enablers de Seguridad componente independiente que se despliega delante de tus servicios.
Propósito Principal	Administración y Gestión Programática: Permite crear, modificar, eliminar y consultar usuarios, organizaciones, roles, permisos y aplicaciones OAuth2 dentro de Keyrock.	Aplicación de Políticas y Autorización: Intercepta peticiones a tus servicios y consulta a Keyrock para decidir si una acción está permitida.
¿Quién lo Usa?	Administradores del sistema, aplicaciones de gestión, o scripts de automatización que necesitan configurar Keyrock.	Las aplicaciones/servicios protegidos ya que el PEP Proxy se interpone entre ellos y los clientes.
Flujo de Interacción	Entrada de Datos/Configuración a Keyrock: Se envían peticiones a Wilma para configurar la seguridad (ej., "crea este usuario", "asigna este rol").	Validación de Acceso a Recursos: Intercepta peticiones de acceso a tus APIs/recursos y pregunta a Keyrock si la petición es válida.
Rol en Seguridad	Define las Reglas: Es la interfaz para establecer quién es quién y qué permisos tiene.	Hace Cumplir las Reglas: Es el punto donde las reglas definidas en Keyrock se aplican en tiempo real.
Dependencia de Keyrock	Es parte de Keyrock; no existe sin Keyrock.	Depende de Keyrock para obtener las decisiones de autorización, pero es un componente separado que se conecta a Keyrock.
Ejemplo en tu Proyecto	Usar Wilma para crear automáticamente nuevos usuarios y asignarles el rol Estudiante.	Poner un PEP Proxy delante de tu API que abre la cerradura del laboratorio. Cuando alguien intenta abrir la puerta, el PEP Proxy consulta a Keyrock si ese usuario tiene permiso.
Ubicación Típica	Corre dentro del mismo contenedor/servicio que Keyrock.	Se despliega como un proxy delante de cada microservicio o aplicación que necesitas proteger.

3.3.10. Fast RTPS

Es una implementación de código abierto del estándar RTPS (Real-Time Publish-Subscribe Protocol), que a su vez es el protocolo de cableado subyacente de DDS. Fast RTPS es una librería de comunicación eficiente que permite que las aplicaciones intercambien información en tiempo real, de forma descentralizada y con garantías de calidad de servicio (QoS). Está diseñada para sistemas con requisitos de rendimiento y fiabilidad exigentes.

Fast RTPS actúa como un intérprete que permite la comunicación en tiempo real, es como si estuviera traduciendo cualquier palabra en un idioma en específico que todos entiendan

3.3.11. Micro XRCE-DDS

Es una implementación ligera del protocolo XRCE-DDS (eXtremely Resource Constrained Environments DDS). Está específicamente diseñada para dispositivos IoT muy pequeños y con recursos limitados como microcontroladores, sensores de bajo consumo que no pueden ejecutar una implementación completa de DDS como Fast RTPS. Permite que estos pequeños dispositivos se comuniquen con el mundo DDS a través de un agente intermediario un Micro XRCE-DDS Agent que corre en un dispositivo más potente como una Raspberry Pi o un PC.

3.3.12. QuantumLeap

Es un conector de persistencia de datos de contexto que está especializado en almacenar series temporales time-series data desde el Orion Context Broker. Su principal ventaja es que está optimizado para consultas y análisis de datos a lo largo del tiempo, permitiendo responder preguntas como *¿cómo ha cambiado el estado de este sensor en la última hora?* o *¿cuántas veces se abrió esta puerta en un día?* Utiliza bases de datos específicas para series temporales, como CrateDB.

3.3.13. WireCloud

Según la documentación de FIWARE es una herramienta que ayuda a los usuarios a generar rápidamente nuevas aplicaciones mashups basadas en NGSI y otras fuentes de datos. Para agilizar el desarrollo, la arquitectura de Wirecloud se ha definido para dividir las operaciones de mashup en una serie de tareas sencillas y reutilizables como widgets y operadores. Cada tarea cuenta con interfaces de entrada y salida bien definidas, y la interfaz de usuario de Wirecloud permite a los creadores de mashups conectar una serie de tareas en una compleja cadena de eventos de procesamiento y visualización de datos.

3.3.14. AuthzForce

AuthzForce representa una implementación del punto de decisión de políticas (Policy Decision Point - PDP) basada en el estándar XACML (eXtensible Access Control Markup Language). Este estándar provee un lenguaje especializado que permite definir y gestionar políticas de control de acceso con un alto nivel de granularidad y precisión.

La función principal de Authzforce es:

- Almacenar Políticas: Guardar un conjunto de reglas de autorización muy detalladas escritas en XACML.

- Evaluar Solicitudes: Recibir solicitudes de autorización de usuarios, realizar acciones y que recursos hacerlo.

- Tomar Decisiones: Evaluar la solicitud contra sus políticas almacenadas y devolver una decisión de autorización como Permitido, Denegado, No aplicable, Indeterminado.

Authzforce permite definir políticas de seguridad mucho más flexibles y dinámicas que la simple asignación de roles y permisos que ofrece Keyrock directamente. Por ejemplo, permitir acceso solo de lunes a viernes entre las 8 AM y 5 PM, o permitir acceso solo si la temperatura del laboratorio es menor a 25°C.

3.3.15. Arquitectura Referencial de FIWARE

En la Figura 9 se puede ver una referencia de una arquitectura de FIWARE en ciudades Inteligentes, junto con el Context Broker y los IoT Agents.

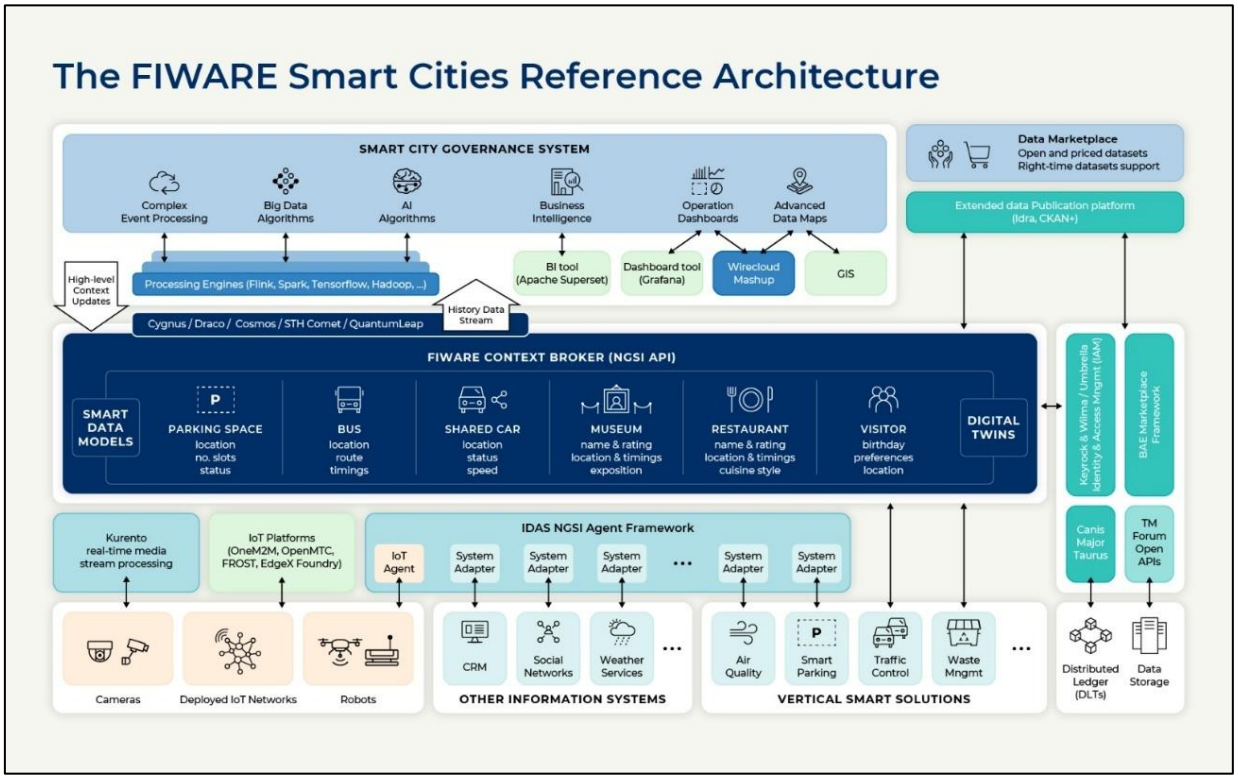


Figura 9. Referencia de la Arquitectura de FIWARE.

3.3.15.1 Capa de Smart City Governance System

Procesa y analiza datos para tomar decisiones inteligentes:

- Event Processing: Detecta eventos importantes como accidentes de tráfico.
- Big Data Algorithms: Procesa grandes cantidades de datos.
- AI Algorithms: Aplica inteligencia artificial para hacer predicciones.
- Business Intelligence / Dashboard / Data Maps: Muestra datos visuales para que los gobiernos puedan ver qué está pasando.
- Wirecloud / GIS: Herramientas para ver mapas y crear tableros personalizados.

3.3.15.2 Capa de Processing Engines e Histórico de Datos

- Procesa datos en tiempo real y guarda el historial:
- Flink, Spark, TensorFlow, Hadoop: Tecnologías para procesar datos.
- Cygnus, Draco, Cosmos, etc.: Guardan datos históricos para análisis.

3.3.15.3 FIWARE Context Broker

Es el corazón del sistema, que recibe, organiza y distribuye datos en tiempo real.

Administra modelos de datos como:

- Parking, Buses, Car Sharing
- Museos, Restaurantes, Visitantes
- También genera Gemelos Digitales: representaciones virtuales de cosas reales.

3.3.15.4 Smart Data Models

Estructura estándar de los datos que llegan por ejemplo la ubicación de un bus, cantidad de espacios en un parqueo, etc.

3.3.15.5 Entrada de Datos

Dispositivos o sistemas que envían datos al sistema:

- Cámaras, IoT, Robots

- Redes sociales, CRM, servicios del clima
- Sistemas verticales como parqueo, tráfico, basura, calidad del aire, etc.

Todo esto pasa por los IoT Agents y System Adapters, que traducen los datos al lenguaje de FIWARE.

3.3.15.6 IDAS NGSI Agent Framework

Conjunto de adaptadores que ayudan a conectar distintos tipos de sistemas con FIWARE tales como:

- Data Marketplace: Lugar donde se pueden compartir o vender datos.
- CKAN/Idra: Plataformas para publicar datos.
- Keyrock/Wilma: Seguridad e identidad de usuarios.
- BAE Framework: Soporte para negocios basados en datos.
- DLTs y Almacenamiento: Guarda datos de forma segura.
- APIs Abiertas (TM Forum): Facilita que otros sistemas se conecten.

3.3.16. Comparación de FIWARE con otros Frameworks de estandarización de los datos

Aunque en la sección se mencionó Frameworks o Plataformas que son de código abierto, aquí se va a mencionar que existen Frameworks que se centran principalmente en la estandarización de los datos, aquí se puede visualizar Tabla 10.

Tabla 10. Comparación de FIWARE con otros Frameworks.

Framework	Objetivo Principal	Como lo hace	Diferencia
Web of Things (WoT)	Mejorar interoperabilidad en IoT.	Por medio de descripción de cosas y sus interfaces con estándares web.	Interoperabilidad a nivel de aplicación; uso de Thing Description.
OneM2M	Estandarizar la capa de servicios IoT.	Por medio de una arquitectura común y especificaciones globales para M2M/IoT.	Evita la fragmentación con APIs, protocolos y modelos de datos coherentes.
OCF / IoTivity	Crear estándar abierto para interoperabilidad entre dispositivos.	Por medio de una comunicación directa, descubrimiento de servicios y seguridad entre dispositivos.	Se enfoca en la interacción directa entre dispositivos de diferentes fabricantes.

Plataformas IoT Cloud	Gestión integral de soluciones IoT en la nube.	Por medio de servicios para introducir datos de diversas fuentes, procesamiento, análisis de datos, etc.	Ecosistemas propios; la interoperabilidad externa puede ser un desafío.
FIWARE	Gestión de contexto en tiempo real e interoperabilidad semántica.	Por medio de un conjunto de Generic Enablers de código abierto Orion Context Broker, NGSI.	Lidera en Smart Cities e Industria 4.0; gestión dinámica del contexto de los datos.

3.3.17Comparación de Arquitecturas de proyectos ya realizados.

3.3.17.1. Comparación con Arquitecturas de FIWARE hechas en Ecuador.

En esta sección se va a realizar comparaciones de proyectos de Ecuador que han hecho soluciones con este Framework, para poder comprender cual es el componente más importante en dichos proyectos e ir observando sus arquitecturas y como están aplicando la estandarización de FIWARE.

Nuestra arquitectura se diferencia por una estrategia más personalizada tratando de ahorrar recursos en la instancia de AWS y haciendo uso del principal Generic Enablers que sería Orion, lo que MongoDB para Orion y MySQL para los datos históricos y persistentes.

Por otro lado la arquitectura de ISABELA para el Análisis del rendimiento académico estudiantil mediante sistemas cibernéticos con la interacción humana hecha por la Escuela Politécnica Nacional en Quito, se enfoca en la integración de fuentes de datos muy diversas y el uso de componentes de FIWARE como IDAS hace que la ingesta de los datos sea más flexible, los datos de los smartphones son gestionados por IDAS, que actúa como el agente IoT para la arquitectura y conectándose directamente con Orion Context Broker, por otro lado la estandarización es una función importante y clave de IDAS, que su papel es convertir los datos crudos y variados de los sensores del smartphone al modelo de datos NGSI de FIWARE antes de ser enviados al Orion Context Broker, en la Figura 10 se puede visualizar su arquitectura.

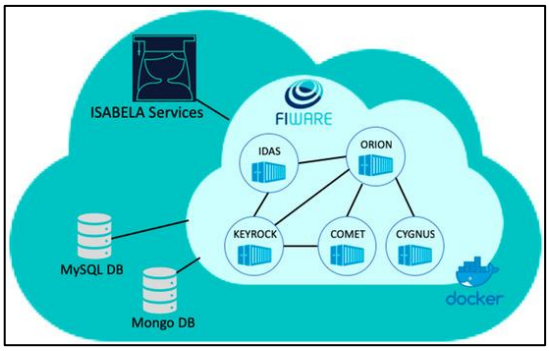


Figura 10. Arquitectura de ISABELA .

Ahora esta arquitectura basada en FIWARE para un modelado de un canal de comunicación de datos de una aplicación de Tele consulta para la ciudad de Guayaquil la cual fue hecha en la Escuela Superior Politécnica del Litoral de Guayaquil, se enfoca en la dupla conformada por el Orion Context Broker y FIWARE Draco, donde el orion permite la comunicación de los datos médicos en tiempo real gestionando pacientes, citas e historiales. Por otro lado, está el componente FIWARE Draco que es igual importante y se encarga del flujo de los datos y de las notificaciones, por el lado de la normalización de los datos lo hacen mediante el modelo de NGSI de FIWARE por medio de la aplicación de Tele consulta ya mencionada, en la Figura 11 se puede visualizar su arquitectura.

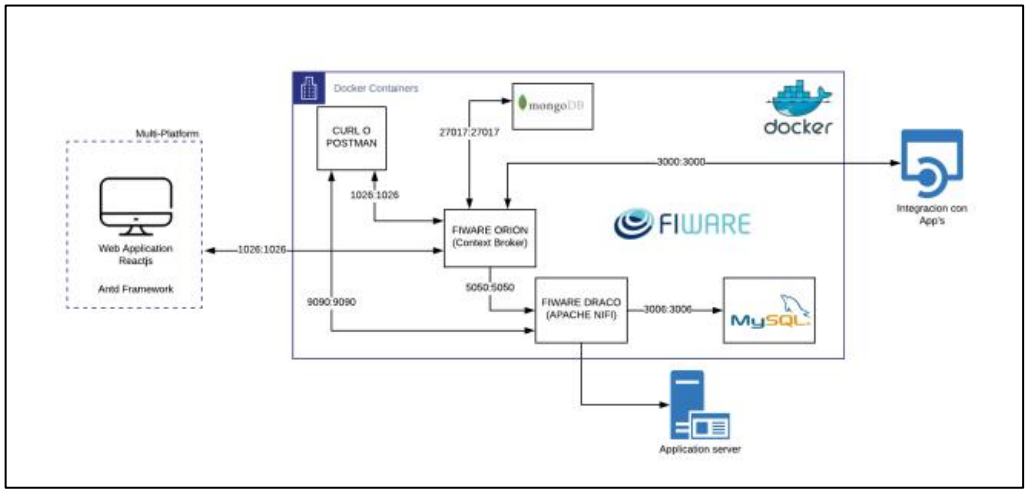


Figura 11. Arquitectura General de la Aplicación de Teleconsulta .

3.3.17.2. Comparación con Arquitecturas de FIWARE fuera de Ecuador

En esta sección se va a realizar comparaciones de proyectos que están fuera de Ecuador que han hecho soluciones con este Framework, para poder comprender cual es el componente más importante en dichos proyectos e ir observando sus arquitecturas y como están aplicando la estandarización de FIWARE.

Se tiene esta arquitectura basada en FIWARE para la implementación de del espacio de datos industriales para la ciudad de Madrid, España, se enfoca en diseñar una arquitectura para habilitar el intercambio de los datos para que sean seguros y controlados en entornos industriales, el componente más importante es Orion Context Broker que se encarga de intercambiar los datos de manera segura combinado con el PEP Proxy y AuthzForce para las políticas y decisiones para la autorización, por el lado de la estandarización ellos se basan en el modelo NGSI de FIWARE para tener una completa comprensión entre los demás Generic Enablers como Cygnus y QuantumLeap para la persistencia de los datos y trabajan en equipo, en la Figura 12 se puede visualizar su arquitectura.

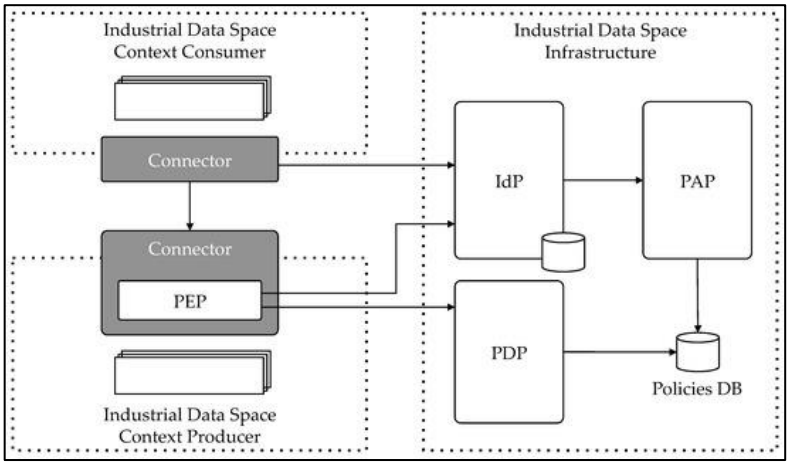


Figura 12. Arquitectura IAM de FIWARE IDS .

La siguiente arquitectura basada en FIWARE es la evaluación de un modelo de control de acceso con alcance de aplicación IoT para la publicación y suscripción basada en FIWARE de la ciudad de Madrid, España, se enfoca en evaluar un modelo de control de accesos a nivel de aplicación IoT para proteger los dispositivos que se van a comunicar mediante el patrón Publish y suscribe, el componente más importante es Keyrock que trabaja junto a PEP Proxy y Authzforce para aplicar y decidir las políticas de autorización, estos aseguran que solo las entidades autorizadas puedan interactuar con los dispositivos IoT y sus datos, por el lado de la estandarización, ellos proponen usar más herramientas de FIWARE para la autentificación de los usuarios en la Figura 13 se puede visualizar su arquitectura.

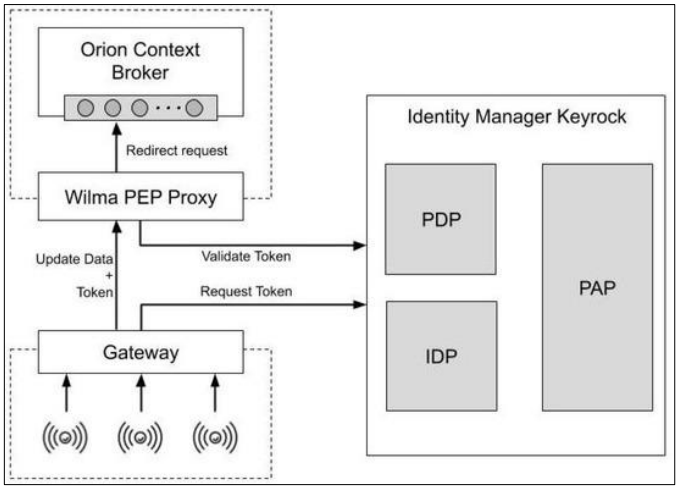


Figura 13. Arquitectura para la Identificación por medio de Keyrock .

3.4. Servicios en la nube usados en IoT

3.4.1. AWS (Amazon Web Sevices)

AWS es la plataforma de infraestructura cloud de Amazon que pone a disposición de sus usuarios un amplio catálogo de recursos tecnológicos bajo un esquema de consumo por demanda. En lugar de invertir en hardware propio, las organizaciones pueden acceder a capacidad de cómputo, almacenamiento y bases de datos a través de internet, pagando únicamente por lo que utilizan. Esta flexibilidad permite escalar la infraestructura de forma dinámica según las necesidades del proyecto, sin los costos fijos asociados a la adquisición y mantenimiento de servidores físicos. De acuerdo con Singh, AWS se posiciona como una de las plataformas de computación en la nube más confiables a nivel mundial por su capacidad de respuesta ante variaciones en la demanda [20].

3.4.2 Microsoft Azure

Azure es la plataforma cloud de Microsoft que centraliza servicios de cómputo, almacenamiento, redes, bases de datos e inteligencia artificial en un único ecosistema. Su principal ventaja en entornos corporativos radica en la integración nativa con el resto de herramientas del ecosistema Microsoft, lo que reduce la fricción al incorporar nuevas soluciones en organizaciones que ya utilizan sus productos. Adicionalmente, Azure soporta múltiples lenguajes, sistemas operativos y frameworks de desarrollo, e incluye capacidades orientadas a IoT, analítica de datos y aprendizaje automático [21].

3.4.3 Google Cloud Platform (GCP)

GCP es la plataforma de servicios cloud de Google, construida sobre la misma infraestructura que soporta productos de uso masivo como Google Search y YouTube. Esto le otorga una base técnica de alta disponibilidad y rendimiento probado a escala global. Entre los servicios que ofrece se encuentran cómputo en la nube, almacenamiento, análisis de datos, aprendizaje automático y herramientas orientadas al desarrollo de aplicaciones, permitiendo construir y escalar soluciones de forma eficiente sin gestionar infraestructura física [22].

3.4.4. Comparativa de Servicios en la Nube

En el siguiente cuadro comparativo se busca entender las ventajas, desventajas y los costos que sería mantener una instancia en cada uno de ellos, tal como se muestra en la Tabla 11.

Tabla 11. Comparativa entre plataformas en la nube.

Característica	Amazon Web Services (AWS)	Microsoft Azure	Google Cloud Platform (GCP)
Definición	Líder en la nube con la mayor variedad de servicios y un ecosistema maduro.	Plataforma de Microsoft, ideal para empresas con productos Microsoft y soluciones híbridas.	Nube de Google, fuerte en IA/ML y análisis de datos, con infraestructura robusta.
Ventajas	- Mayor oferta de servicios. - Gran comunidad y madurez. - Amplia red global.	- Integración con Microsoft. - Foco en nube híbrida. - Seguridad empresarial.	- Innovación en IA/ML. - Precios competitivos (uso sostenido. - Infraestructura de Google.
Desventajas	- Precios complejos. - Curva de aprendizaje alta. - Soporte avanzado de pago.	- Menos intuitivo sin conocimiento MS. - Soporte básico mejorable. - Menor variedad de servicios que AWS.	- Menor variedad de servicios que AWS/Azure. - Menor presencia global. - Menor adopción empresarial.
Costo de una instancia básica	$30 – $32 USD/mes. t3.medium Linux (2 vCPU, 4 GB RAM).	$28 – $32 USD/mes. B2s Linux (2 vCPU, 4 GB RAM).	$24 – $30 USD/mes. e2-medium Linux (2 vCPU, 4 GB RAM).

Después de comparar estas plataformas, y comparar costo y beneficio para una instancia media, se decidió por utilizar la instancia t3.medium con Linux ya que resulta ser las más confiables, escalables y mayor oferta en servicios que las demás plataformas.

3.4.5. Modelo OSI y TCP/IP

En los sistemas de comunicación y redes, los modelos de referencia OSI (Open Systems Interconnection) y TCP/IP (Transmission Control Protocol/Internet Protocol) constituyen marcos conceptuales fundamentales para entender cómo se organiza y transporta la información entre dispositivos interconectados. El modelo OSI divide la comunicación de red en siete capas (Física, Enlace de Datos, Red, Transporte, Sesión, Presentación y Aplicación), cada una con funciones específicas que facilitan la compatibilidad entre sistemas heterogéneos. En paralelo, el modelo TCP/IP, más simplificado y práctico para implementaciones reales, agrupa la comunicación en cuatro capas: Acceso a Red, Internet, Transporte y Aplicación [23].

En el contexto del presente proyecto, estos modelos permiten ubicar y justificar cómo se distribuyen las funciones dentro de la arquitectura propuesta. Por ejemplo, los dispositivos IoT (ESP32 + RFID) operan principalmente en las capas inferiores del modelo OSI y TCP/IP, específicamente en la capa física y de enlace de datos (comunicación inalámbrica Wi-Fi o Ethernet) y en la capa de red, donde se generan y transmiten los paquetes IP hacia el broker MQTT o directamente hacia servicios HTTP/HTTPS. El broker MQTT (Mosquitto) y los Agentes FIWARE se ejecutan en las capas de transporte y aplicación, facilitando la comunicación entre dispositivos finales y los servicios backend y actuando como intermediarios de protocolos. A su vez, el Orion Context Broker, la API FastAPI y la aplicación Flutter operan claramente en la capa de aplicación, ya que procesan la lógica de negocio, manejo de contexto, presentación y consumo de servicios distribuidos.

Del mismo modo, el modelo TCP/IP es especialmente adecuado para describir la arquitectura de este ecosistema, dado que sus capas reflejan de forma directa las tecnologías utilizadas:

- Aplicación: MQTT (sobre TCP), HTTP/S (REST APIs), NGSI (interfaces FIWARE).
- Transporte: TCP (garantiza entrega confiable entre cliente y servidor).
- Internet: IP (enrutamiento y direccionamiento de los paquetes entre nodos).
- Acceso a Red: Wi-Fi / Ethernet (comunicación física y enlace).

Esta organización en capas permite asegurar interoperabilidad, modularidad y escalabilidad entre los distintos componentes del sistema, ya que cada uno cumple funciones bien definidas. De esta manera, la arquitectura general del sistema IoT propuesto se ajusta coherentemente al modelo TCP/IP, que es el estándar de facto en redes modernas y sobre el cual se basan tanto

los protocolos utilizados (MQTT, HTTP/S) como las interfaces de servicio de FIWARE (NGSI) y las API expuestas al cliente móvil.

La arquitectura propuesta para el prototipo de control de accesos en la Universidad Técnica de Cotopaxi se fundamenta en tecnologías ampliamente utilizadas en sistemas distribuidos modernos, tales como AWS, FIWARE, Docker, FastAPI y Flutter, las cuales se articulan de forma coherente dentro del modelo de comunicación TCP/IP, siendo este el modelo de referencia práctico utilizado por las infraestructuras cloud actuales.

En este contexto, la infraestructura en la nube AWS, donde se despliega el sistema operativo Ubuntu Server y el entorno Docker Compose, se sitúa transversalmente en todas las capas del modelo TCP/IP, proporcionando conectividad, direccionamiento IP, transporte confiable y acceso a servicios de red. AWS actúa como la plataforma base que garantiza disponibilidad, escalabilidad y comunicación entre los componentes del ecosistema.

La capa de acceso a red está conformada por las tecnologías físicas y de enlace utilizadas por los dispositivos IoT (ESP32 con módulos RFID), los cuales emplean redes Wi-Fi y Ethernet para establecer conexión con el sistema. Esta capa permite la transmisión inicial de datos hacia los servicios superiores, ya sea mediante comunicación directa HTTP/HTTPS o mediante el protocolo MQTT a través del broker Mosquitto.

En la capa de transporte, el protocolo TCP cumple un rol fundamental al garantizar la transmisión confiable de datos entre los dispositivos IoT, el broker MQTT, la API desarrollada en FastAPI y los componentes de FIWARE. Tanto MQTT como HTTP/HTTPS operan sobre TCP, asegurando la integridad y el orden de los mensajes intercambiados dentro del sistema.

La capa de aplicación es donde se concentra la mayor parte de la lógica del sistema propuesto. En esta capa se ubican:

- FastAPI, que expone los endpoints REST para la gestión de usuarios, tarjetas RFID y registros históricos.
- FIWARE Orion Context Broker, que gestiona la información de contexto en tiempo real mediante el estándar NGSI.
- Agentes FIWARE, encargados de traducir los datos provenientes de los dispositivos IoT a un formato comprensible por Orion.

- Broker MQTT (Mosquitto), que facilita la comunicación asíncrona entre los dispositivos IoT y los servicios backend.
- Flutter, como capa de presentación, que consume los servicios expuestos por FastAPI y permite la visualización y administración del sistema desde una aplicación móvil.

De esta manera, la arquitectura del sistema propuesto no solo implementa tecnologías modernas de desarrollo e integración IoT, sino que además se encuentra sólidamente respaldada por los principios del modelo TCP/IP, orden lógico de comunicación y compatibilidad con el ecosistema FIWARE y los servicios en la nube.

A partir del análisis de los protocolos, tecnologías y componentes utilizados en el sistema propuesto, se determina que la arquitectura del prototipo de control de accesos se adscribe principalmente al modelo TCP/IP, dado que todas las comunicaciones implementadas se basan en protocolos propios de dicho modelo, tales como IP, TCP, HTTP/HTTPS y MQTT.

El modelo OSI se emplea en este trabajo como un marco conceptual y descriptivo, que permite analizar y ubicar las funciones de cada componente dentro de una estructura lógica de capas; sin embargo, la implementación real del ecosistema IoT desarrollado se fundamenta en el modelo TCP/IP, el cual constituye el estándar de facto utilizado por las infraestructuras de red modernas, los servicios de computación en la nube (AWS) y el ecosistema FIWARE.

En consecuencia, se establece que el sistema propuesto pertenece al modelo TCP/IP desde el punto de vista de implementación, mientras que el modelo OSI se utiliza como herramienta de apoyo para la comprensión y organización de la arquitectura de comunicaciones.

3.5 Plataforma Docker

Un contenedor en Docker es una unidad ligera y portable que empaqueta una aplicación junto con todas sus dependencias, como bibliotecas, configuraciones y código ejecutable, para ejecutarse de forma aislada en cualquier entorno. A diferencia de las máquinas virtuales, los contenedores comparten el núcleo del sistema operativo del host, lo que los hace más eficientes y rápidos de iniciar

3.5.1 Docker

Docker es una herramienta de contenedores que permite empaquetar una aplicación junto con todo su entorno de ejecución —dependencias, librerías y configuraciones— en unidades

portables denominadas contenedores. A diferencia de las máquinas virtuales tradicionales, los contenedores no requieren un sistema operativo completo propio, ya que comparten el kernel del sistema anfitrión, lo que los hace considerablemente más ligeros y eficientes en el consumo de recursos. Esta característica garantiza que una aplicación se comporte de manera idéntica en cualquier entorno, ya sea en un equipo de desarrollo, en un servidor de pruebas o en producción en la nube. Merkel señala que Docker aprovecha los contenedores de Linux para aislar procesos y sus dependencias [24], lo que permite desplegar servicios complejos con un alto grado de reproducibilidad y control.

3.5.2 Docker Compose

Docker Compose extiende las capacidades de Docker hacia escenarios donde una aplicación requiere múltiples servicios trabajando en conjunto, como un servidor web, una base de datos y un sistema de caché. Su funcionamiento se basa en un archivo de configuración YAML donde se declaran todos los servicios, sus dependencias mutuas y los parámetros de red, permitiendo levantar, detener y gestionar el sistema completo mediante un único comando. Un análisis de más de 4.000 proyectos de código abierto realizado por Ibrahim et al. concluyó que el uso de Docker Compose reduce de forma significativa la complejidad asociada al despliegue y mantenimiento de aplicaciones con múltiples componentes [25].

3.5.3 Comparación entre Docker y Docker Compose

Aquí nos pareció importante saber cuál es la diferencia de lo que es docker en si, tal como se muestra en la Tabla 12.

Tabla 12. Comparativa entre Docker y Docker compose.

Característica	Docker	Docker Compose
Propósito	Usa contenedores y gestiona aplicaciones individuales.	Orquesta y gestiona múltiples contenedores en una aplicación completa.
Funcionamiento	Construye y ejecuta contenedores aislados.	Usa un archivo YAML para definir y lanzar un stack de servicios.
Comando típico	docker run	docker compose up

En base a la documentación de FIWARE nos recomienda utilizar Docker Compose por el simple hecho de que docker compose nos permite trabajar con archivos YAML.

3.7 Metodología para el desarrollo de software

3.7.1. Modelo Iterativo Incremental

El modelo iterativo incremental es una de las metodologías más empleadas en el desarrollo de software debido a su flexibilidad para incorporar nuevos módulos de forma progresiva. Su estructura organiza el trabajo en iteraciones sucesivas, cada una de las cuales añade funcionalidad al sistema hasta alcanzar el producto final. Esta aproximación resulta especialmente adecuada cuando los requisitos pueden evolucionar durante el desarrollo, ya que permite ajustes entre iteraciones sin comprometer los avances previos, facilitando la integración gradual de componentes y el cumplimiento de plazos establecidos[26].

El propósito principal de esta metodología es desarrollar sistemas que pueden ser complejos de forma gradual y una forma progresiva, entregando versiones funcionales del producto en ciclos cortos y repetitivos. A cada ciclo se lo conoce como iteración el cual añade un nuevo conjunto de funcionalidades que sería el incremento del sistema, permitiendo ir comprobando cada vez si está bien su funcionalidad para evitar riesgos, y de esta forma asegurar que el producto final sea del agrado de lo que busca el usuario, en la Figura 14 se muestra el flujo del modelo iterativo incremental.

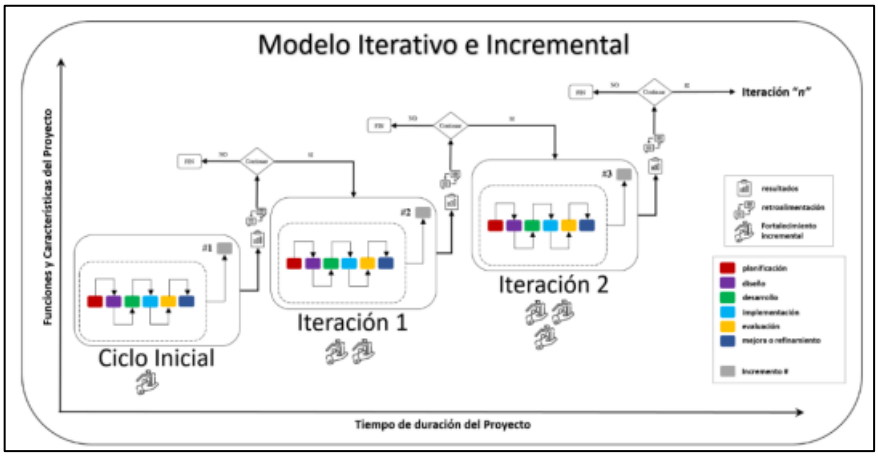


Figura 14. Flujo del Modelo Iterativo Incremental [27].

En el presente proyecto, el modelo iterativo-incremental se utiliza como marco metodológico para el desarrollo del software.

3.8 Metodología para la implementación de IoT

3.8.1. Modelo en V

El modelo en V es una metodología de desarrollo de software que organiza sus fases de manera secuencial siguiendo una estructura en forma de V. Es conocido también como modelo de verificación y validación, dado que su principio fundamental consiste en asociar a cada etapa de desarrollo una fase de pruebas correspondiente. De este modo, ambas ramas de la V avanzan en paralelo conceptualmente: la rama descendente abarca el diseño y la implementación, mientras que la rama ascendente contempla las pruebas asociadas a cada nivel. Una fase no puede iniciarse hasta que la anterior haya concluido, lo que garantiza trazabilidad entre los requisitos y su verificación.

Es una metodología de desarrollo que destaca por enfocarse en verificar y validar de manera detenidamente en cada una de las etapas, el lado de la izquierda representa las fases de diseño, que empiezan desde el análisis a la programación, mientras que el lado de la derecha representa las fases de la implementación y las pruebas, que empiezan desde lo concreto a lo abstracto.

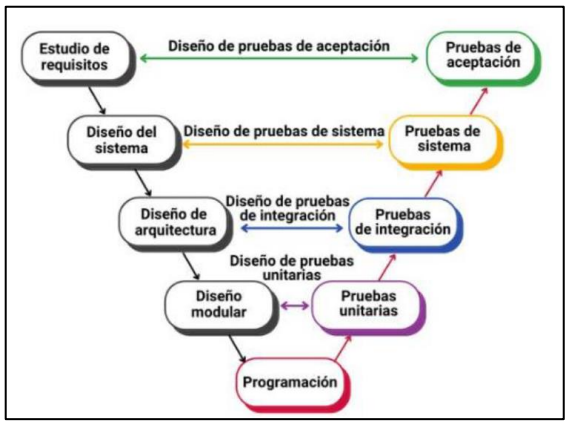


Figura 15. Fases del modelo en V [28].

Para el presente proyecto, el modelo en V se selecciona exclusivamente para el desarrollo del hardware del dispositivo IoT, permitiendo:

- Validar los requisitos del dispositivo desde etapas tempranas
- Reducir errores de diseño en componentes físicos
- Garantizar la correcta integración entre hardware y firmware.

Esta estrategia metodológica proporciona una base sólida para el desarrollo, implementación y validación del prototipo de control de accesos, asegurando coherencia entre la fase teórica, la aplicación práctica y el análisis de resultados.

4 MÉTODOS Y PROCEDIMIENTOS

4.1 Enfoque de la investigación

Esta investigación adopta una metodología mixta que articula componentes cualitativos y cuantitativos de manera complementaria. La combinación de ambos enfoques posibilita una comprensión más completa del fenómeno analizado, aportando tanto la riqueza interpretativa del análisis cualitativo como el rigor y la objetividad propios de las mediciones cuantitativas.

- Permite validar experimentalmente el rendimiento técnico mediante métricas objetivas como la disponibilidad.
- Facilita la comprensión profunda de fenómenos emergentes no predecibles mediante observación y análisis documental
- Facilita la interpretación de resultados al contrastar hallazgos cuantitativos.

4.1.1 Justificación del Enfoque Mixto en el Proyecto

La decisión de adoptar un enfoque mixto en esta investigación se fundamental debido a que los objetivos planteados requieren de resultados no solo numéricos sino como funciona con el entorno.

4.1.1.1 Componente Cuantitativo:

El componente cuantitativo de la investigación se enfoca en la medición objetiva de variables técnicas que permiten evaluar el rendimiento, la eficiencia y la viabilidad del ecosistema IoT desarrollado. Este componente incluye:

- Evaluación experimental de protocolos IoT: Aquí se realizó una comparación controlada de tres protocolos de comunicación (HTTP, HTTPS, MQTT) midiendo variables cuantificables.
- Medición de rendimiento del sistema: Se capturaron métricas operativas durante 7 semanas de funcionamiento continuo, incluyendo uso de CPU y RAM del servidor, cantidad de notificaciones procesadas y volumen de datos históricos acumulados.
- Análisis de costos de infraestructura: Se registró el costo mensual de la infraestructura cloud desplegada en AWS, permitiendo evaluar la viabilidad económica de la solución en contextos institucionales con restricciones presupuestarias.

4.1.1.2 Componente Cualitativo:

El componente cualitativo complementa las mediciones cuantitativas mediante la exploración de fenómenos emergentes y la interpretación contextualizada de los resultados. Este componente incluye:

- Observación directa del comportamiento del sistema: se documentaron patrones de comportamiento y características arquitectónicas de FIWARE que no están explícitamente descritas claramente en la documentación. Por ejemplo, se descubrió que QuantumLeap genera automáticamente tablas en CrateDB siempre y cuando los valores sean acordes entre la entidad y la suscripción.

- Análisis documental: Se realizó revisión de la documentación oficial de FIWARE sobre (Orion Context Broker, QuantumLeap, IoT Agents), especificaciones de protocolos IoT interoperables. Este análisis permitió fundamentar decisiones de diseño y contrastar hallazgos experimentales con el estado del arte.

- Entrevista semi-estructurada: Se condujo una entrevista con un docente de la carrera de Ingeniería Industrial de la Universidad Técnica de Cotopaxi para validar la utilidad percibida del sistema desde la perspectiva de un usuario técnico potencial. La entrevista incluyó demostración práctica del prototipo.

Esta complementariedad entre lo cuantitativo y lo cualitativo permite no solo demostrar que el ecosistema FIWARE funciona, sino también explicar cómo funciona, por qué ciertos protocolos son compatibles y otros no, y cuáles son las mejores prácticas arquitectónicas derivadas de la experiencia de implementación.

4.2 Tipo de Investigación

4.2.1 Investigación Aplicada

Se clasifica como investigación aplicada debido a que su objetivo primordial es resolver un problema en concreto y específico en el contexto de protocolos de comunicación como la ausencia de un sistema integrado de gestión de acceso y monitoreo ambiental en laboratorios académicos de la Universidad Técnica de Cotopaxi.

En este proyecto, se aplicaron conocimientos teóricos consolidados de:

• Arquitecturas de software orientadas a microservicios

• Protocolos de comunicación IoT (HTTP, HTTPS, MQTT,).

• Estándares de interoperabilidad IoT (NGSI)

• Desarrollo de una aplicación móvil.

Estos conocimientos teóricos se transformaron en un prototipo funcional que demuestra la viabilidad técnica y económica de implementar sistemas IoT interoperables en instituciones educativas con presupuesto limitado. El resultado tangible de la investigación es un ecosistema operativo que actualmente gestiona 4 dispositivos IoT.

4.2.2 Investigación Experimental

La investigación también posee un componente experimental significativo, dado que se realizó una evaluación sistemática y controlada de variables para establecer relaciones causales entre el tipo de protocolo IoT utilizado y variables de rendimiento observadas.

En esta investigación, se diseñó un experimento controlado con las siguientes características:

4.2.2.1 Variable independiente:

• Protocolo IoT utilizado: Se evaluaron 3 protocolos (, HTTP, HTPS, MQTT)

4.2.2.2 Variables dependientes:

• Compatibilidad técnica con ESP32 (Sí/No)

• Latencia de comunicación (milisegundos)

• Tasa de éxito de transmisiones (porcentaje)

• Complejidad de implementación (cualitativa: baja/media/alta)

4.2.2.3 Variables controladas:

• Hardware: ESP32 DevKit V1 (240 MHz, 520 KB SRAM) idéntico en todas las pruebas

• Red: WiFi UTC con características constantes (-65 dBm intensidad señal)

• Servidor: AWS EC2 t3.medium con configuración uniforme

• Payload: JSON {"temperatura": 21.5, "humedad": 52} estandarizado

• Condiciones ambientales: Pruebas realizadas en mismo laboratorio, misma franja horaria

El control riguroso de variables externas permitió establecer que las diferencias observadas en compatibilidad y rendimiento son atribuibles al protocolo utilizado y no a variaciones en el entorno experimental. Por ejemplo, se pudo determinar con certeza que UltraLight 2.0 y CoAP

no son compatibles con ESP32 en el stack actual de Arduino debido a incompatibilidades específicas de formato y librerías, mientras que HTTP/JSON y MQTT sí lo son, con latencias medidas de 200ms (HTTP) y 300ms (MQTT) respectivamente.

4.2.3 Tipos de investigación usados por Objetivo Específico

Con el propósito de establecer una correspondencia explícita entre los objetivos específicos planteados y el tipo de investigación que cada uno demanda, se presenta la siguiente tabla resumen. Esta vinculación permite verificar la coherencia metodológica del proyecto y fundamentar las decisiones técnicas adoptadas en cada etapa del desarrollo.

Tabla 13 Clasificación del tipo de investigación por objetivo específico

N°	Objetivo Específico	Tipo de Investigación	Justificación
OE1	Realizar una revisión bibliográfica actualizada sobre FIWARE y su aplicación en ecosistemas IoT interoperables.	**Bibliográfica Documental**	Sistematiza fuentes científicas, documentación oficial FIWARE y trabajos previos para fundamentar el marco teórico.
OE2	Diseñar la arquitectura del ecosistema IoT basado en FIWARE, integrando dispositivos, protocolos de comunicación (HTTP, HTTPS, MQTT) y servicios en la nube.	**Aplicada Tecnológica**	Transforma conocimiento teórico en un diseño arquitectónico funcional orientado a resolver la problemática institucional.
OE3	Evaluar el desempeño del ecosistema mediante pruebas de interoperabilidad, latencia y escalabilidad en un entorno controlado.	**Experimental**	Emplea pruebas controladas con variables definidas para medir y comparar el rendimiento objetivo del sistema desarrollado.

4.3 Metodología de la Investigación

El diseño de la investigación combina dos metodologías de desarrollo complementarias: el Modelo en V para el desarrollo del ecosistema FIWARE (infraestructura, hardware, integración), y el Modelo Iterativo-Incremental para el desarrollo de la aplicación móvil de visualización y gestión. Esta dualidad metodológica responde a las características intrínsecas de cada componente del sistema

4.3.1 Modelo en V para Desarrollo del Ecosistema FIWARE

4.3.1.1 Descripción del Modelo en V:

El Modelo en V es una metodología de desarrollo de sistemas que extiende el modelo tradicional de cascada enfatizando la verificación y validación en cada fase del ciclo de vida. Recibe su nombre por la disposición gráfica en forma de 'V' donde el lado izquierdo representa las fases de especificación y diseño descendentes, mientras que el lado derecho representa las

fases de implementación y pruebas ascendentes. Cada fase de especificación en el lado izquierdo tiene una fase de prueba correspondiente en el lado derecho que valida que los requisitos se cumplieron correctamente.

4.3.1.2 Justificación de aplicación al ecosistema FIWARE:

El Modelo en V resulta especialmente apropiado para el desarrollo del ecosistema FIWARE por las siguientes razones:

1. Naturaleza crítica de la infraestructura: El ecosistema FIWARE constituye la infraestructura base sobre la cual operan todos los dispositivos IoT y la aplicación. Errores en la configuración de componentes core (Orion Context Broker, QuantumLeap, bases de datos) tienen impacto sistémico. El Modelo en V minimiza estos riesgos mediante validación rigurosa en cada fase antes de avanzar.

2. Desarrollo de hardware: Los módulos ESP32 con sensores y actuadores (DHT22, RC522, relé 12V, cerradura 600lb) requieren diseño cuidadoso de conexiones eléctricas donde errores pueden causar daño físico a componentes. El Modelo en V exige pruebas unitarias de cada módulo antes de integración, reduciendo riesgos de cortocircuitos o sobrevoltajes.

3. Complejidad de integración: El ecosistema integra 7 componentes heterogéneos (Orion, MongoDB, QuantumLeap, CrateDB, IoT Agent JSON, Mosquitto MQTT, FastAPI) que deben comunicarse correctamente. Las fases de pruebas de integración del Modelo en V garantizan que interfaces entre componentes funcionan según especificado.

4. Requisitos bien definidos: Los requisitos funcionales (control acceso RFID, monitoreo temperatura/humedad, persistencia histórica) y no funcionales (latencia y disponibilidad) están claramente establecidos desde el inicio. El Modelo en V se ajusta perfectamente a proyectos con requisitos estables y conocidos a priori.

5. Costos de reingeniería: Modificar infraestructura cloud desplegada (reconfigurar contenedores Docker, migrar bases de datos) es costoso en tiempo y recursos. El énfasis del Modelo en V en 'hacer las cosas bien la primera vez' mediante validación temprana minimiza estos costos.

4.3.1.3 Fases del Modelo en V aplicadas:

4.3.1.3.1 Lado izquierdo (Especificación y Diseño):

- Análisis de requisitos del sistema
- Diseño de arquitectura general
- Diseño detallado de componentes
- Implementación (codificación, configuración, ensamblaje)

4.3.1.3.2 Lado derecho (Implementación y Validación):

- Pruebas unitarias (componentes individuales)
- Pruebas de integración (interfaces entre componentes)
- Pruebas de sistema (funcionamiento global)
- Pruebas de aceptación (validación con usuario)

4.3.2 Modelo Iterativo-Incremental para Aplicación Móvil

4.3.2.1 Descripción del Modelo Iterativo-Incremental:

El Modelo Iterativo-Incremental es una metodología de desarrollo de software que estructura el proyecto en ciclos cortos (iteraciones) donde cada iteración produce un incremento funcional del sistema. Cada iteración incluye todas las fases del desarrollo (análisis, diseño, codificación, pruebas) pero enfocadas en un subconjunto de funcionalidades. Al final de cada iteración se entrega software funcional que puede ser evaluado por usuarios, cuya retroalimentación se incorpora en iteraciones subsecuentes.

4.3.2.2 Justificación de aplicación a la aplicación móvil:

El Modelo Iterativo-Incremental es idóneo para el desarrollo de la aplicación móvil Flutter por las siguientes razones:

- Requisitos de UI/UX evolutivos: A diferencia de la infraestructura FIWARE (donde requisitos son técnicos y estables), la interfaz de usuario requiere refinamiento basado en retroalimentación. El modelo iterativo permite ajustar diseño visual, flujos de navegación y experiencia de usuario en cada ciclo sin rehacer todo el sistema.

- Bajo costo de cambios: En desarrollo móvil con Flutter, modificar código Dart y widgets es relativamente económico comparado con reconfigurar infraestructura cloud. Esto hace viable la experimentación y mejora continua característica del modelo iterativo.

- Entrega temprana de valor: Cada iteración produce una versión funcional de la app con características incrementales (Iteración 1: login + CRUD, Iteración 2: + histórico, Iteración 3: + gráficas, Iteración 4: + control remoto). Esto permite validar la utilidad del sistema con usuarios reales tempranamente en el desarrollo.

- Gestión de riesgo técnico: Al desarrollar funcionalidades core primero (autenticación, gestión datos maestros) y funcionalidades avanzadas después (gráficas, control remoto), se mitigan riesgos técnicos progresivamente. Si una funcionalidad avanzada resulta muy compleja, puede posponerse sin comprometer el núcleo funcional del sistema.

4.3.2.3 Estructura de iteraciones aplicadas:

El desarrollo de la aplicación móvil SmartLab se estructuró en 4 iteraciones de 5 días cada una (20 días totales de desarrollo):

4.3.2.3.1 Iteración 1 (días 1-5): Accesos - Autenticación y Arquitectura Base

- Funcionalidades: Login con JWT, CRUD de usuarios,
- Objetivo: Establecer arquitectura base de la app y operaciones fundamentales
- Entregable: App funcional con gestión completa de usuarios y tarjetas

4.3.2.3.2 Iteración 2 (días 6-10): Gestión de Dispositivos

- Funcionalidades: Consulta listado de dispositivos
- Objetivo: Conectar app con datos del sistema
- Entregable: App con visualización completa de listado de dispositivos

4.3.2.3.3 Iteración 3 (días 11-15): Validación RFID y Control de Acceso

- Funcionalidades: CRUD tarjetas RFID y CRUD de personas
- Objetivo: controlar el acceso mediante tarjetas
- Entregable: App para gestionar las tarjetas y personas

4.3.2.3.4 Iteración 4 (días 16-20): Analytics y Reportes

- Funcionalidades: dashboard de los datos, reportes en pdf.
- Objetivo: generar un análisis de los datos y generar reporte.
- Entregable: app lista para generar reportes de análisis de datos.

Cada iteración siguió el ciclo completo de desarrollo ágil: planificación de funcionalidades diseño de interfaces codificación en Flutter pruebas funcionales revisión y retroalimentación. Los aprendizajes de cada iteración informaron la planificación de la siguiente, permitiendo ajustes continuos.

4.3.3 Justificación de la Metodología Híbrida Adoptada

La combinación del modelo iterativo-incremental y el modelo en V responde a la necesidad de abordar un sistema IoT de manera integral, considerando las particularidades de cada uno de sus componentes. Mientras que el software del sistema requiere flexibilidad, adaptabilidad y evolución progresiva, el hardware demanda un enfoque más riguroso y secuencial orientado a la validación.

La adopción de una metodología híbrida que combina el modelo iterativo-incremental con el modelo en V responde a la naturaleza dual de los sistemas IoT, donde el software requiere adaptabilidad continua y el hardware exige validación rigurosa. Larman y Basili señalan que el desarrollo iterativo-incremental tiene como principio fundamental evitar un enfoque secuencial de paso único orientado a documentos, promoviendo en su lugar ciclos sucesivos de refinamiento [29]. En paralelo, García et al. identificaron que las metodologías ágiles aplicadas al desarrollo de sistemas IoT deben contemplar tanto el ciclo de vida del software como la creación de dispositivos físicos, destacando la importancia de la interacción con el usuario y la integración de técnicas de inteligencia artificial [30].

Por tanto, se establece que el presente proyecto adopta una metodología híbrida, donde:

- El modelo iterativo-incremental guía el desarrollo del software y los servicios FIWARE.
- El modelo en V estructura el desarrollo del hardware IoT.

4.4 Técnicas e Instrumentos

La investigación requirió la aplicación de técnicas tanto cuantitativas como cualitativas, complementadas con instrumentos específicos para la captura de datos objetivos y evidencia empírica.

4.4.1 Técnicas Cuantitativas

4.4.1.1 Experimentación Controlada:

Se diseñó y ejecutó un experimento controlado para evaluar la compatibilidad y rendimiento de diferentes protocolos IoT con hardware ESP32. El experimento mantuvo constantes las variables del entorno (hardware, red, servidor, payload) mientras se modificaba sistemáticamente la variable independiente (protocolo utilizado), permitiendo aislar el efecto del protocolo en las variables de rendimiento observadas.

4.4.1.2 Medición de Latencias:

Se midió la latencia de comunicación end-to-end desde el momento en que el dispositivo ESP32 envía datos hasta que el sistema confirma su recepción y procesamiento. Para HTTP se utilizó Postman con registro automático de tiempos de respuesta (10 mediciones consecutivas). Para MQTT se estimó latencia mediante análisis de timestamps en logs del sistema.

4.4.1.3 Medición de Recursos del Servidor:

Se capturó el uso de CPU y memoria RAM del servidor AWS EC2 durante períodos de operación sostenida utilizando herramientas nativas de Linux (vmstat, free). Estas métricas permiten evaluar la escalabilidad del sistema y proyectar necesidades de infraestructura para despliegues más grandes.

4.4.2 Técnicas Cualitativas

4.4.2.1 Observación Directa:

Durante las 7 semanas de operación continua del sistema en entorno productivo, se realizó observación sistemática del comportamiento del ecosistema FIWARE. Esta observación permitió identificar patrones de funcionamiento, detectar anomalías, y documentar descubrimientos arquitectónicos emergentes que no están explícitamente descritos en la documentación oficial de FIWARE (por ejemplo, la generación automática de tablas en CrateDB basada en entity_type).

4.4.2.2 Entrevista Semi-estructurada:

Se condujo una entrevista semi-estructurada con un docente de la carrera de Ingeniería Industrial de la Universidad Técnica de Cotopaxi. La entrevista combinó preguntas predefinidas sobre funcionalidad del sistema con espacio para exploración abierta de percepciones y sugerencias. Se incluyó demostración práctica del prototipo funcionando, permitiendo al entrevistado experimentar directamente con la aplicación móvil y el sistema de control de acceso RFID.

4.4.2.3 Análisis Documental:

Se realizó revisión sistemática de documentación técnica oficial, incluyendo:

- Documentación FIWARE: Orion Context Broker, QuantumLeap API Specification, IoT Agent JSON Documentation
- Especificaciones de protocolos: HTTPs, MQTT, HTTP
- Documentación de componentes: CrateDB Reference, MongoDB, Eclipse Mosquitto

Este análisis documental fue particularmente importante cuando se encontraron discrepancias entre el comportamiento esperado (según documentación) y el comportamiento observado (en el sistema real). Por ejemplo, la documentación de IoT Agent UltraLight indica que soporta dispositivos con recursos limitados, pero no especifica que el formato UltraLight requiere implementación manual en el lado del dispositivo, lo cual no es trivial en ESP32.

4.4.3 Instrumentos de Medición

Para la captura de datos cuantitativos y evidencia empírica, se utilizaron los siguientes instrumentos y herramientas técnicas:

Tabla 14 estructura de instrumentos de medición

Variable Medida	**Instrumento/Herramienta Utilizada**	**Método de Captura de Datos**
Latencia comunicación HTTP (ms)	Postman v11.20 con registro automático de tiempo de respuesta	10 peticiones POST consecutivas al IoT Agent, captura del tiempo mostrado en interfaz Postman
Tiempo respuesta consultas CrateDB (ms)	CrateDB Admin Console (interfaz web puerto 4200)	Ejecución de queries SQL SELECT, captura del tiempo de ejecución mostrado automáticamente
Uso de CPU del servidor (%)	Comando vmstat en terminal Ubuntu 24.04 LTS	Muestreo cada 5 segundos durante 1 hora de operación sostenida, análisis columna 'us' (user CPU)
Uso de RAM del servidor (MB)	Comando free -m en terminal Ubuntu 24.04 LTS	Captura de memoria disponible y usada cada 5 segundos, cálculo porcentaje de uso
Logs del sistema FIWARE	docker logs [nombre_contenedor] en línea de comandos Docker	Captura en tiempo real de eventos, errores y warnings de Orion, IoT Agent JSON, QuantumLeap
Notificaciones enviadas por QuantumLeap	Orion Context Broker API - endpoint GET /v2/subscriptions	Consulta del campo timesSent en objeto JSON de respuesta de cada suscripción
Registros acumulados en CrateDB	Consola CrateDB - consultas SQL COUNT(*)	Ejecución de SELECT COUNT(*) FROM [tabla] para cada tabla del esquema mtsmartslab
Disponibilidad del sistema	Observación directa manual con registro en bitácora	Monitoreo diario durante 7 semanas, registro de eventos de desconexión o fallas
Costo de infraestructura cloud	AWS Cost Explorer (consola web AWS)	Descarga de facturación mensual detallada por servicio (EC2, transferencia de datos, almacenamiento)
Compatibilidad de protocolos IoT	Pruebas de transmisión con Postman (HTTP) y código ESP32 (MQTT, UltraLight, CoAP)	Envío de payloads de prueba, verificación de recepción exitosa en Orion con GET /v2/entities

La combinación de estos instrumentos permitió recolectar evidencia tanto cuantitativa (métricas numéricas precisas) como cualitativa (logs, observaciones de comportamiento) que fundamentan los resultados y conclusiones de la investigación presentados en el Capítulo 5.

5 ANÁLISIS Y DISCUSIÓN DE LOS RESULTADOS

Este capítulo documenta el proceso completo de desarrollo e implementación del ecosistema IoT interoperable basado en FIWARE, desde la fase inicial de análisis de requisitos hasta la validación final del sistema en operación como el Modelo en V para la infraestructura y hardware IoT, y Modelo Iterativo-Incremental para la aplicación móvil. Además, se detallan los hallazgos arquitectónicos descubiertos durante la experimentación con componentes FIWARE que constituyen aportes al conocimiento sobre implementaciones prácticas de esta plataforma en entornos académicos.

5.1 Resultados de la metodología Modelo en V

5.1.1 Fase de Análisis de Requisitos

La fase de análisis identificó los requisitos funcionales y no funcionales del sistema en las cuales se presentan los siguientes.

5.1.1.1 Requisitos funcionales:

Tabla 15 Requisitos Funcionales

ID	Requisito Funcional	Prioridad
RF-01	El sistema debe validar tarjetas RFID contra base de datos de usuarios autorizados	Alta
RF-02	El sistema debe controlar cerradura electromagnética mediante relé al validar acceso autorizado	Alta
RF-03	El sistema debe registrar histórico de todos los intentos de acceso (autorizados y denegados)	Alta
RF-04	El sistema debe recopilar datos ambientales (temperatura, humedad, índice UV, velocidad viento)	Media
RF-05	El sistema debe detectar presencia de humo mediante sensor MQ-2	Media
RF-06	El sistema debe persistir datos históricos en base de datos de series temporales	Alta
RF-07	El sistema debe proporcionar interfaz móvil para gestión de personas y tarjetas	Media
RF-08	El sistema debe generar visualizaciones gráficas de datos históricos de sensores	Baja

5.1.1.2 Requisitos no funcionales

Tabla 16 Requisitos no Funcionales

ID	Requisito No Funcional	Métrica
RNF-01	Latencia de validación RFID	< 1 segundo
RNF-02	Disponibilidad del sistema	2 meses
RNF-03	Escalabilidad horizontal	Soportar mínimo 5 dispositivos IoT simultáneos
RNF-04	Interoperabilidad	Uso de protocolos estándar abiertos (HTTP, MQTT, HTTPS)
RNF-05	Costo mensual infraestructura	34 USD/mes
RNF-06	Persistencia de datos	Retención histórica de 2 meses
RNF-07	Seguridad comunicaciones	TLS para MQTT, HTTPS para API

5.1.2 Fase de Diseño de Arquitectura General

Con los requisitos claramente definidos, se diseñó la arquitectura general del ecosistema integrando componentes FIWARE con servicios complementarios. La arquitectura resultante consta de cinco capas: Capa de Dispositivos IoT, Capa de Conectividad, Capa de Gestión de Contexto FIWARE, Capa de Persistencia, y Capa de Aplicación.

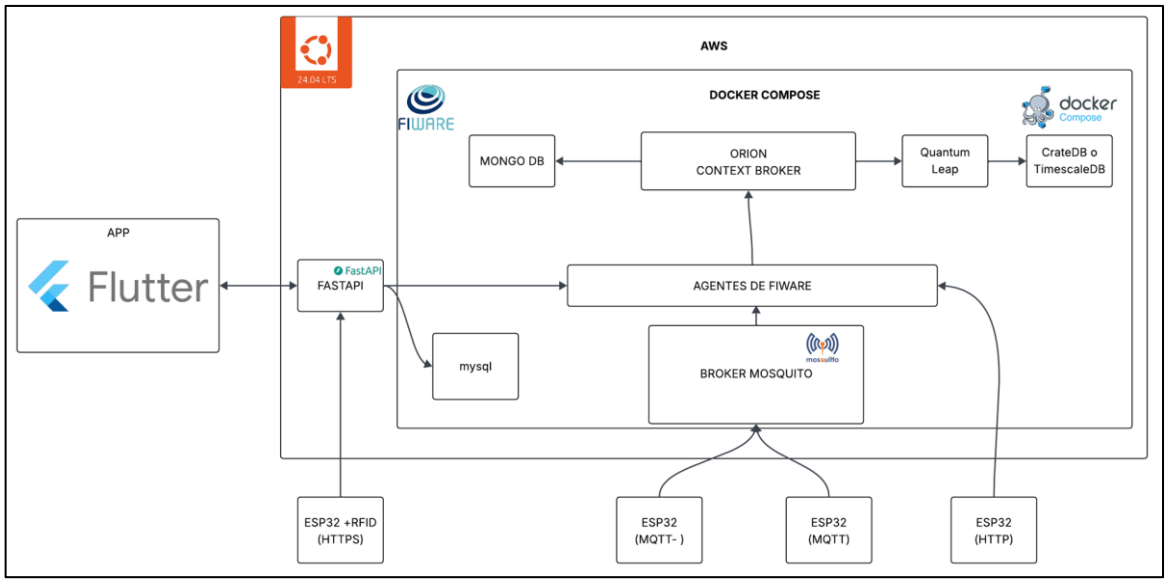


Figura 16 Arquitectura General.

Tabla 17 Justificación de cada uno de los componentes.

Componente	**Versión**	**Justificación Técnica**
Orion Context Broker	3.10.1	Context Broker central de FIWARE. Implementa NGSI v2 para gestión de contexto en tiempo real. Versión estable LTS con soporte activo.
MongoDB	6.0	Backend de persistencia de Orion. Almacena entidades y suscripciones. Versión 6.0 ofrece mejor rendimiento de escritura que 4.x.
QuantumLeap	latest	Componente FIWARE para persistencia histórica. Traduce notificaciones NGSI v2 a inserciones en CrateDB automáticamente.
CrateDB	5.5.0	Base de datos columnar distribuida optimizada para series temporales IoT. Maneja 370K+ registros con queries <150ms.
IoT Agent JSON	1.24.0	Traduce protocolos IoT (HTTP, MQTT) a NGSI v2. Soporta ambos transportes en un solo agente.
Eclipse Mosquitto	2.0	Broker MQTT estándar. Ligero, confiable, soporte TLS. Alternativa open source a brokers comerciales.
MySQL	5.7	Base de datos relacional para datos transaccionales (personas, tarjetas). Garantiza integridad referencial con FOREIGN KEYs.
FastAPI	0.104+	Framework Python para API REST. Alto rendimiento (comparable a NodeJS), validación automática con Pydantic, documentación auto-generada.

5.1.2.1 Arquitectura Híbrida MySQL + CrateDB:

Se diseñó una arquitectura híbrida que aprovecha las fortalezas de dos paradigmas de bases de datos:

• MySQL: Datos maestros transaccionales (personas, tarjetas). Operaciones CRUD frecuentes con integridad referencial garantizada mediante FOREIGN KEYs. Consultas de validación RFID <50ms mediante índices en tarjeta_id.

• CrateDB: Series temporales append-only (eventos acceso, lecturas sensores). Agregaciones temporales eficientes (AVG temperatura últimas 24h: 7ms). Sin necesidad de UPDATE ni DELETE, ideal para IoT.

5.1.3 Fase de Diseño Detallado de Infraestructura

En esta fase se definieron especificaciones técnicas exactas de cada componente, configuraciones de red, puertos, volúmenes de datos, y scripts de automatización.

5.1.3.1 Infraestructura Cloud AWS:

Se seleccionó Amazon Web Services (AWS) como proveedor de infraestructura cloud por su confiabilidad, disponibilidad global, y costos competitivos dentro de las restricciones presupuestarias de la universidad.

Tabla 18 Parámetros de EC2.

Parámetro	**Valor**
Proveedor	Amazon Web Services (AWS)
Región	us-east-1 (Norte de Virginia)
Tipo de instancia	EC2 t3.medium
vCPUs	2
Memoria RAM	4 GB
Sistema operativo	Ubuntu 24.04 LTS
Almacenamiento	30 GB SSD gp3
IP pública	13.59.176.23 (Elastic IP)
Dominio	ecosistemafiware.com (Route 53)
Costo mensual	$34 USD (enero 2026)

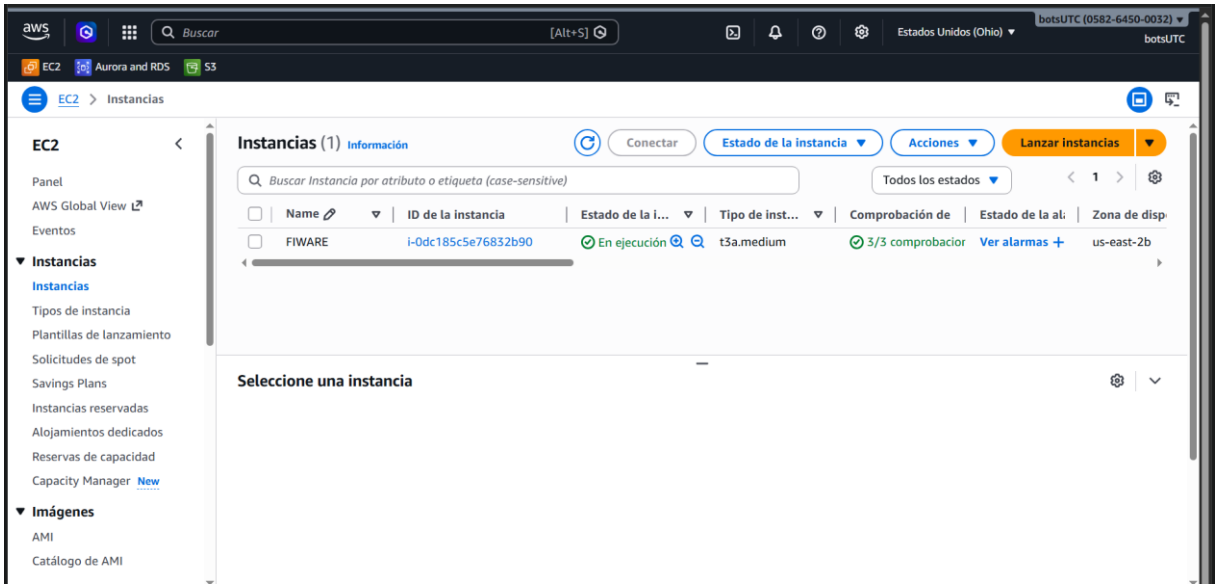


Figura 17 Instancia EC2 activa.

5.1.3.2 Proceso de Creación de Instancia:

El proceso de aprovisionamiento de la instancia AWS tomó aproximadamente 5 minutos desde el lanzamiento hasta la disponibilidad de SSH. Los pasos ejecutados fueron:

1. Creación de instancia EC2 desde AWS Console (tipo t3.medium, AMI Ubuntu 24.04)

2. Configuración de Security Group: Puertos abiertos 22 (SSH), 80 (HTTP), 443 (HTTPS), 1026 (Orion), 1883 (MQTT), 8883 (MQTTS), 4200 (CrateDB Admin), 8668 (QuantumLeap)

3. Asignación de Elastic IP para mantener IP pública estática

4. Generación de par de claves SSH para acceso remoto seguro

5. Conexión inicial vía SSH: ssh -i smartlab-key.pem ubuntu@13.59.176.23

5.1.3.3 Instalación de Docker y Docker Compose:

Docker fue seleccionado como plataforma de orquestación de contenedores por su facilidad de despliegue, reproducibilidad, y aislamiento de dependencias. La instalación siguió la documentación oficial de Docker para Ubuntu:

 # Actualizar índice de paquetes
sudo apt-get update

 # Instalar dependencias
sudo apt-get install -y ca-certificates curl gnupg

 # Agregar clave GPG oficial de Docker
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

 # Instalar Docker Engine

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

#Verificar instalación
docker --version
docker compose version

El tiempo total de instalación de Docker fue aproximadamente 3 minutos. Una vez refinado el proceso mediante scripts, el tiempo de replicación completa del entorno (desde instancia AWS vacía hasta stack FIWARE operativo) se redujo a aproximadamente 20 minutos.

5.1.3.4 Archivo docker-compose.yml

Se diseñó un archivo Docker Compose que define los 7 servicios del stack FIWARE con sus dependencias, volúmenes persistentes, y configuraciones de red. A continuación se presenta el archivo completo utilizado en producción:

```
version: "3.9"
services:

  mongo-db:
    image: mongo:6.0
    container_name: mongo-db
    restart: always
    environment:
      TZ: America/Guayaquil
    volumes:
      - mongo_data:/data/db
    ports:
      - "27017:27017"

  mysql-db:
    image: mysql/mysql-server:5.7
    container_name: mysql-db
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: 321654
      MYSQL_DATABASE: accesos_db
      TZ: America/Guayaquil
```

```
  ports:
    - "3306:3306"

crate-db:
  image: crate:5.5.0
  container_name: crate-db
  restart: always
  ports:
    - "4200:4200"
orion:
  image: fiware/orion:3.10.1
  container_name: orion
  depends_on:
    - mongo-db
  ports:
    - "1026:1026"
  command: ["-dbhost", "mongo-db", "-logLevel", "INFO"]

mosquitto:
  image: eclipse-mosquitto:2.0
  ports:
    - "1883:1883"
    - "8883:8883"

iotagent-json:
  image: fiware/iotagent-json:1.24.0
  depends_on:
    - mosquitto
    - orion
  ports:
    - "4041:4041"
    - "7896:7896"
  environment:
```

```
    IOTA_CB_HOST: orion
    IOTA_MQTT_HOST: mosquitto
    IOTA_HTTP_PORT: "7896"

  quantumleap:
    image: orchestracities/quantumleap:latest
    depends_on:
      - crate-db
    environment:
      CRATE_HOST: crate-db
    ports:
      - "8668:8668"

volumes:
  mongo_data:
  mysql_data:
  crate_data:
```

5.1.4 Fase de Implementación - Servicios FIWARE

Con el diseño detallado completado, se procedió a la implementación y configuración de cada componente del stack FIWARE. Esta fase incluyó el levantamiento de servicios Docker, configuración de IoT Agent, registro de dispositivos, y creación de suscripciones a QuantumLeap.

5.1.4.1 Levantamiento del Stack FIWARE:

El stack completo se levantó con un único comando Docker Compose:

docker compose up -d

El primer levantamiento tomó aproximadamente 5 minutos debido a la descarga de imágenes Docker desde Docker Hub (total ~2.3 GB). Levantamientos posteriores (tras reinicios) toman solo 30-40 segundos ya que las imágenes están cacheadas localmente.

Verificación de servicios activos como se muestra en la imagen Figura 18:

docker ps

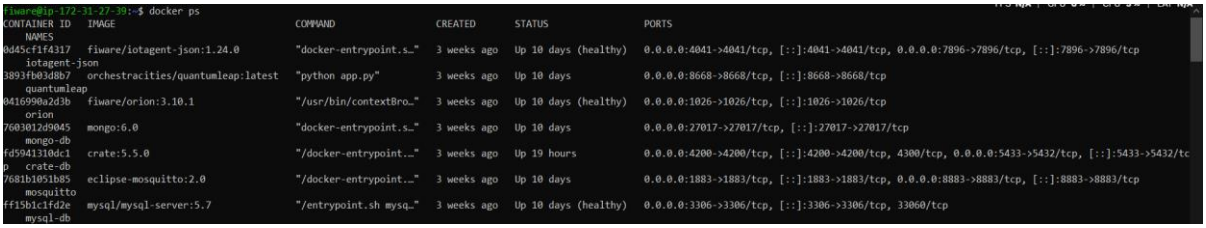


Figura 18 Contenedores corriendo en docker.

5.1.4.2 Configuración de Service Group en IoT Agent:

El Service Group define configuraciones compartidas para un conjunto de dispositivos IoT. Se creó un service group llamado 'smartlab' con API key '1234' que gestiona dispositivos HTTP y MQTT:

```
curl -X POST http://13.59.176.23:4041/iot/services \
 -H 'fiware-service: smartlab' \
 -H 'fiware-servicepath: /' \
 -H 'Content-Type: application/json' \
 -d '{
  "services": [{
   "apikey": "1234",
   "resource": "/json",
   "entity_type": "Sensor"
  }]
 }'
```

5.1.4.3 Registro de Dispositivos IoT:

Se registraron 4 dispositivos IoT en el IoT Agent. A continuación, se muestra el registro del dispositivo 'temperatura-humedad' como ejemplo representativo:

```
curl -X POST http://13.59.176.23:4041/iot/devices \
 -H 'fiware-service: smartlab' \
 -H 'fiware-servicepath: /' \
 -H 'Content-Type: application/json' \
```

```
 -d '{
   "devices": [{
     "device_id": "temperatura-humedad",
     "entity_name": "temperatura-humedad",
     "entity_type": "Sensor",
     "transport": "MQTT",
     "attributes": [
       {"object_id": "t", "name": "temperatura", "type": "Number"},
       {"object_id": "h", "name": "humedad", "type": "Number"}
     ]
   }]
 }'
```

Los 4 dispositivos registrados fueron:

Tabla 19 Dispositivos registrados:4

Device ID	**Entity Type**	**Transporte**	**Atributos**
Station01	Station	MQTT	temperature, humidity, uvIndex, windSpeed
temperatura-humedad	Sensor	MQTT	temperatura, humedad
sensorhumo1	Sensor	HTTP	humo
acceso_labredes	AccessControl	HTTP	tarjeta

5.1.4.4 Creación de Suscripciones a QuantumLeap:

Para que los datos contextuales se persistan automáticamente en CrateDB, se crearon suscripciones en Orion Context Broker que notifican a QuantumLeap cada vez que un atributo cambia. Ejemplo de suscripción para temperatura-humedad:

```
curl -X POST http://13.59.176.23:1026/v2/subscriptions \
  -H 'fiware-service: smartlab' \
  -H 'Content-Type: application/json' \
  -d '{
```

```
  "description": "Persistencia temperatura-humedad",
  "subject": {
    "entities": [{"id": "temperatura-humedad", "type": "Sensor"}],
    "condition": {"attrs": ["temperatura", "humedad"]}
  },
  "notification": {
    "http": {"url": "http://quantumleap:8668/v2/notify"},
    "attrs": ["temperatura", "humedad"]
  }
}'
```

Se crearon 4 suscripciones en total (una por dispositivo).

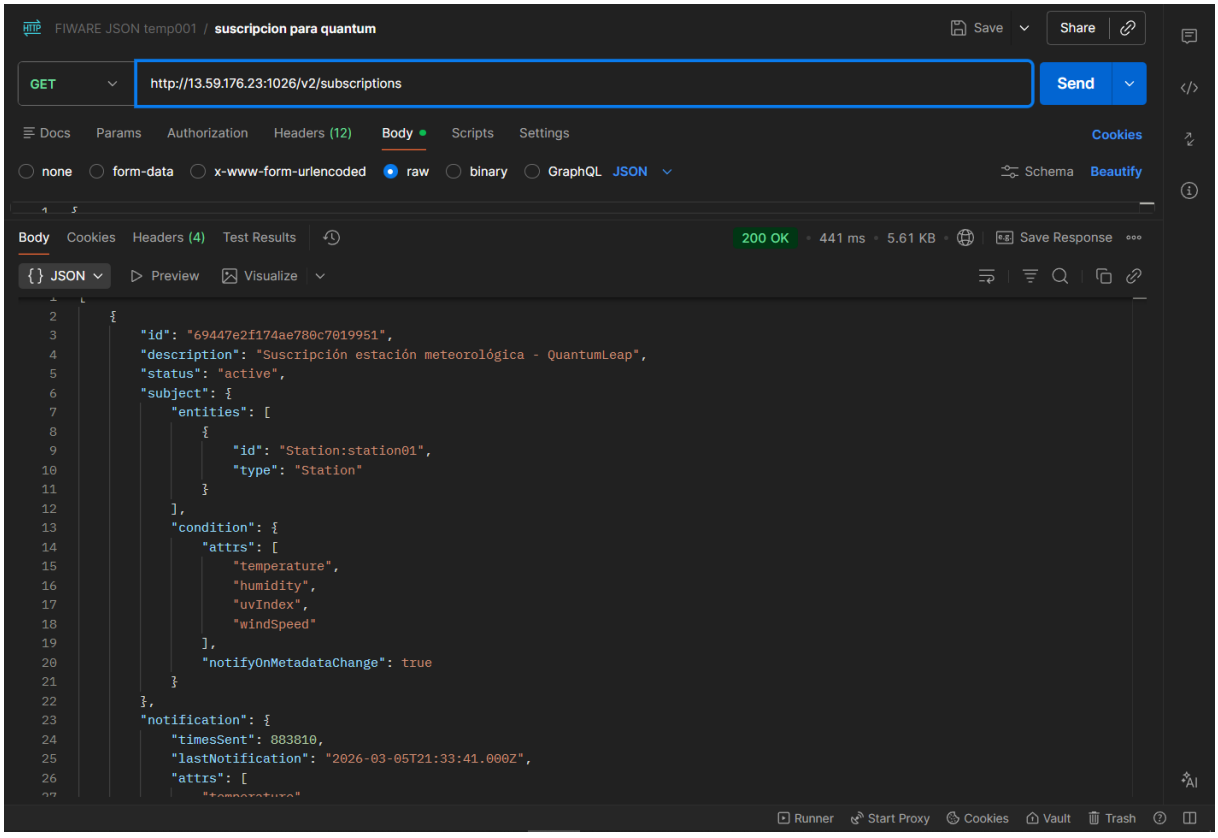


Figura 19 Suscripciones en postman.

5.1.5 Fase de Implementación - API FastAPI

Se desarrolló una API REST en FastAPI que actúa como puente seguro entre dispositivos ESP32 externos y el ecosistema FIWARE interno. La API implementa validación de tarjetas RFID contra MySQL y reenvío de datos al IoT Agent JSON.

5.1.5.1 Estructura de la API:

• Endpoint principal: POST /accesos/v1/bridge

• Protocolo: HTTPS con certificado Let's Encrypt

• Payload optimizado: {"i": "device_id", "d": "data", "a": "attribute"}

• Respuesta compacta: {"s": "OK", "m": "mensaje"}

5.1.5.2 Flujo de validación RFID (9 pasos):

1. ESP32: POST HTTPS /accesos/v1/bridge

2. API: SELECT MySQL (tarjeta + persona + estado)

3. API: PATCH Orion /v2/entities/AccessEvent:acceso_labredes

4. Orion: Notifica QuantumLeap (suscripción activa)

5. QuantumLeap: INSERT CrateDB (persistencia)

6. API: Responde ESP32 OK/NO

7. ESP32: Activa relé + LED verde si OK

8. Relé: Cerradura 600lb abre 3 segundos

9. OLED: Muestra 'Bienvenido [nombre]'

5.1.6 Fase de Implementación - Módulos ESP32

Se desarrollaron 4 módulos ESP32 independientes. A continuación, se documentan los 3 principales:

5.1.6.1 temperatura-humedad (MQTT TLS)

- Sensor: DHT11 (precisión ±2°C, ±5% RH)
- Protocolo: MQTT con TLS puerto 8883

- Topic: json/1234/temperatura-humedad/attrs
- Frecuencia: 10 segundos
- Payload: {"t": 21.5, "h": 52}

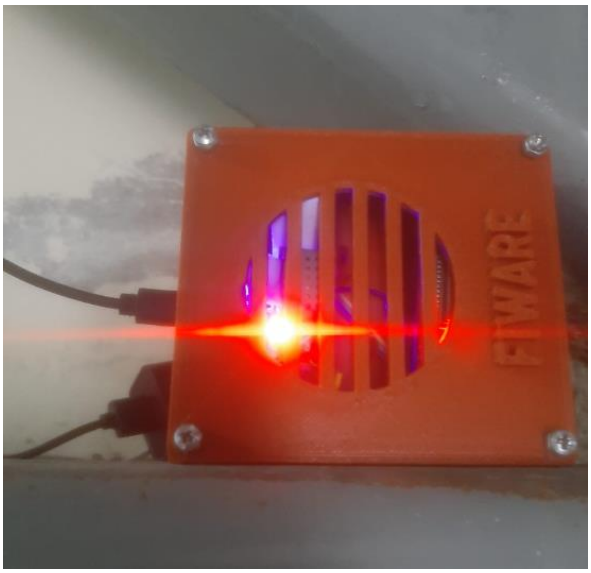


Figura 20 Módulo de temperatura.

5.1.6.2 sensorhumo1 (HTTP)

- Sensor: MQ-2 (detección humo/gases)
- URL: http://13.59.176.23:8000/json?k=1234&i=sensorhumo1
- Frecuencia: 10s (con detección cambio ±15)
- Promediado: 15 lecturas ADC para reducir ruido
- Payload: {"h": 450}

Figura 21 Módulo de humo.

5.1.6.3 acceso_labredes (RFID + Relé + HTTPS)

Este es el módulo más complejo del sistema, combinando control de acceso RFID, actuación electromecánica, interfaz visual OLED, y comunicación segura HTTPS.

Componentes hardware:

• Lector RFID RC522 (13.56 MHz, interfaz SPI)

• Relé 12V 10A (cerradura electromagnética)

• Cerradura 600lb (272 kg fuerza retención)

• Pantalla OLED SSD1306 128x64 (I2C)

• LED RGB (feedback visual)

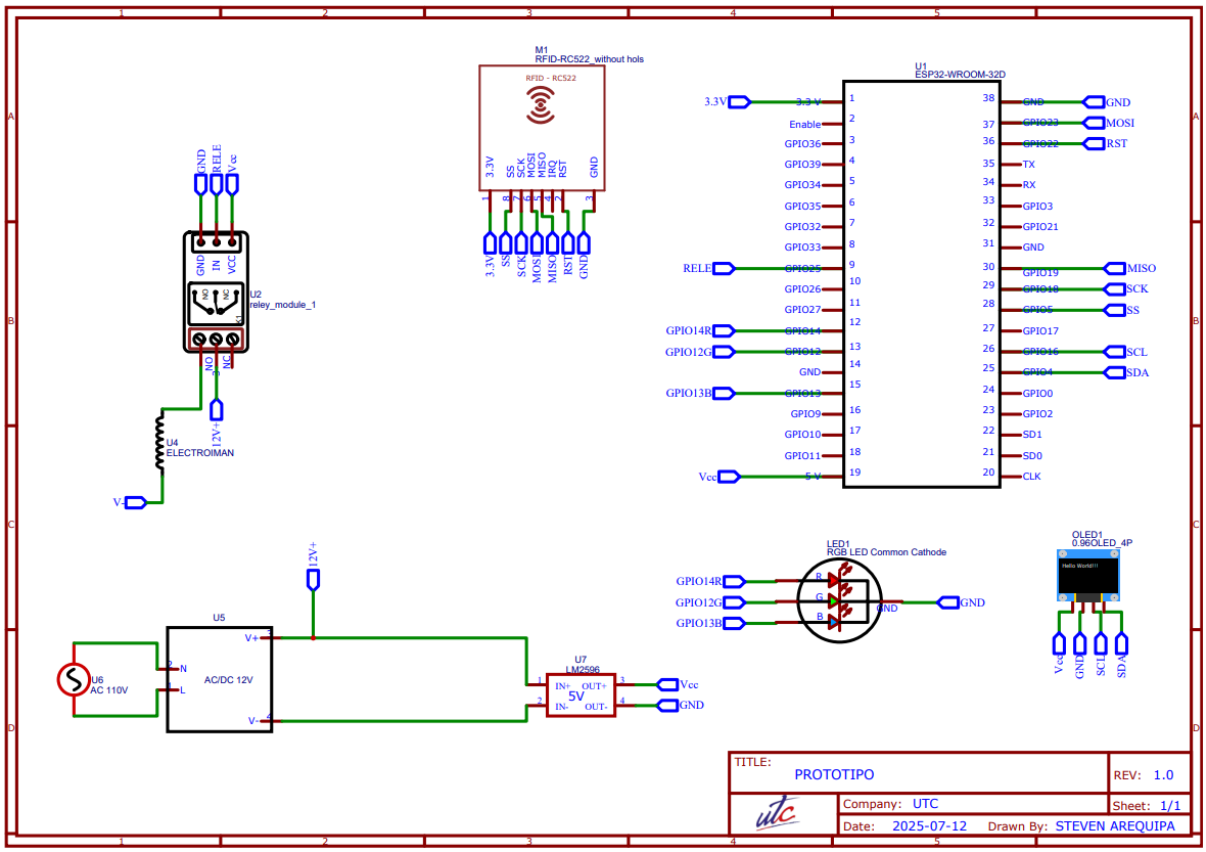


Figura 22 Diagrama eléctrico del control de accesos.

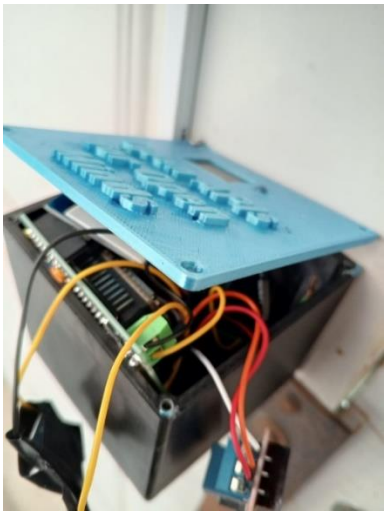


Figura 23 Módulo de control de accesos.

Características técnicas:

• Protocolo: HTTPS (certificado validado)

• URL: https://ecosistemafiware.com/accesos/v1/bridge

• Timeout: 5 segundos

• Tiempo apertura: 3 segundos configurable

• Watchdog: 30s (previene cuelgues)

• Failover WiFi: 3 redes automáticas.

5.1.6.4 Station:station01 (Estación Meteorológica)

• Hardware: Heltec WiFi LoRa 32 V2

• Sensores: DHT22 + GUVA-S12SD (UV) + Anemómetro

• Frecuencia: 5 segundos

• Notificaciones: 400,215 (27 días operación)

• Payload: {"temp": 18, "hum": 70.1, "uv": 11, "wind": 27.45}

Figura 24 Modulo estación meteorológica.

5.1.7 Fase de Pruebas Unitarias

Cada componente fue probado individualmente antes de integración:

Tabla 20 Tabla de pruebas.

Componente	**Prueba**	**Resultado**
Docker Stack	docker ps (7 contenedores)	✓ Todos Up
Orion	GET /version	✓ 3.10.1
IoT Agent	POST /iot/json	✓ 200 OK
CrateDB	SELECT test	✓ Query 7ms
DHT11	Lectura Serial Monitor	✓ 22°C, 65%
RC522	Lectura UID tarjeta	✓ Detectado
Relé	Activación manual	✓ Conmutó

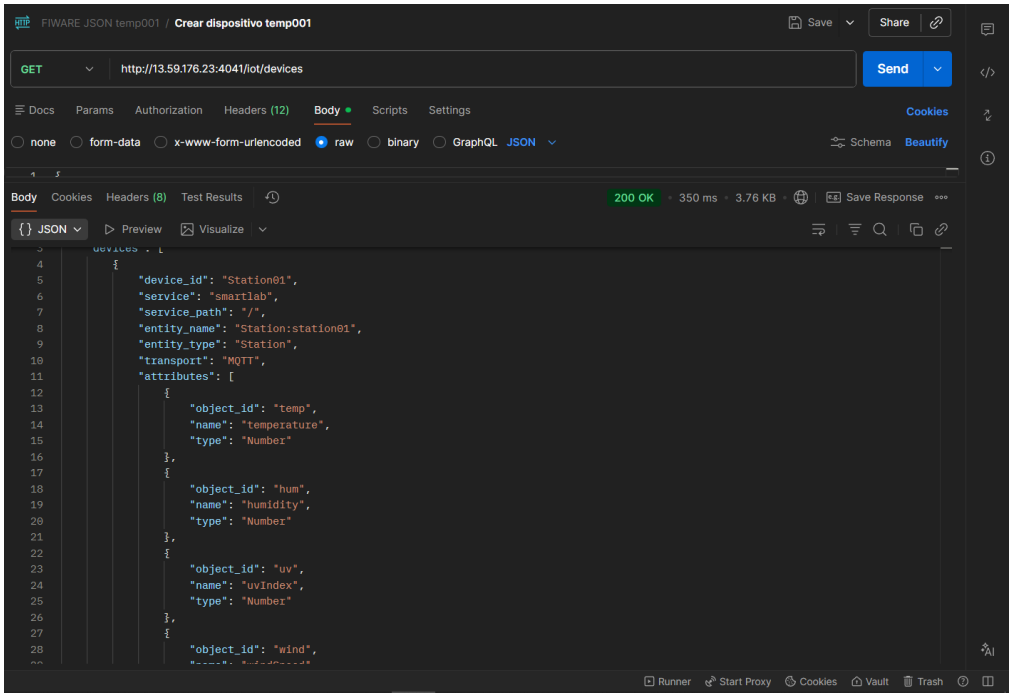


Figura 25 Consulta de dispositivos.

5.1.8 Fase de Pruebas de Integración

Se validó el flujo completo ESP32 → IoT Agent → Orion → QuantumLeap → CrateDB.

Tabla 21 Estadísticas de Suscripciones QuantumLeap.

Dispositivo	**Notificaciones**	**Última Fecha**
Station:station01	400,215	26/01/2026 05:26
sensorhumo1	36,111	24/01/2026 22:25
temperatura-humedad	35,514	26/01/2026 05:26
acceso_labredes	3	23/01/2026 14:01

Total: 471,843 notificaciones en 7 semanas

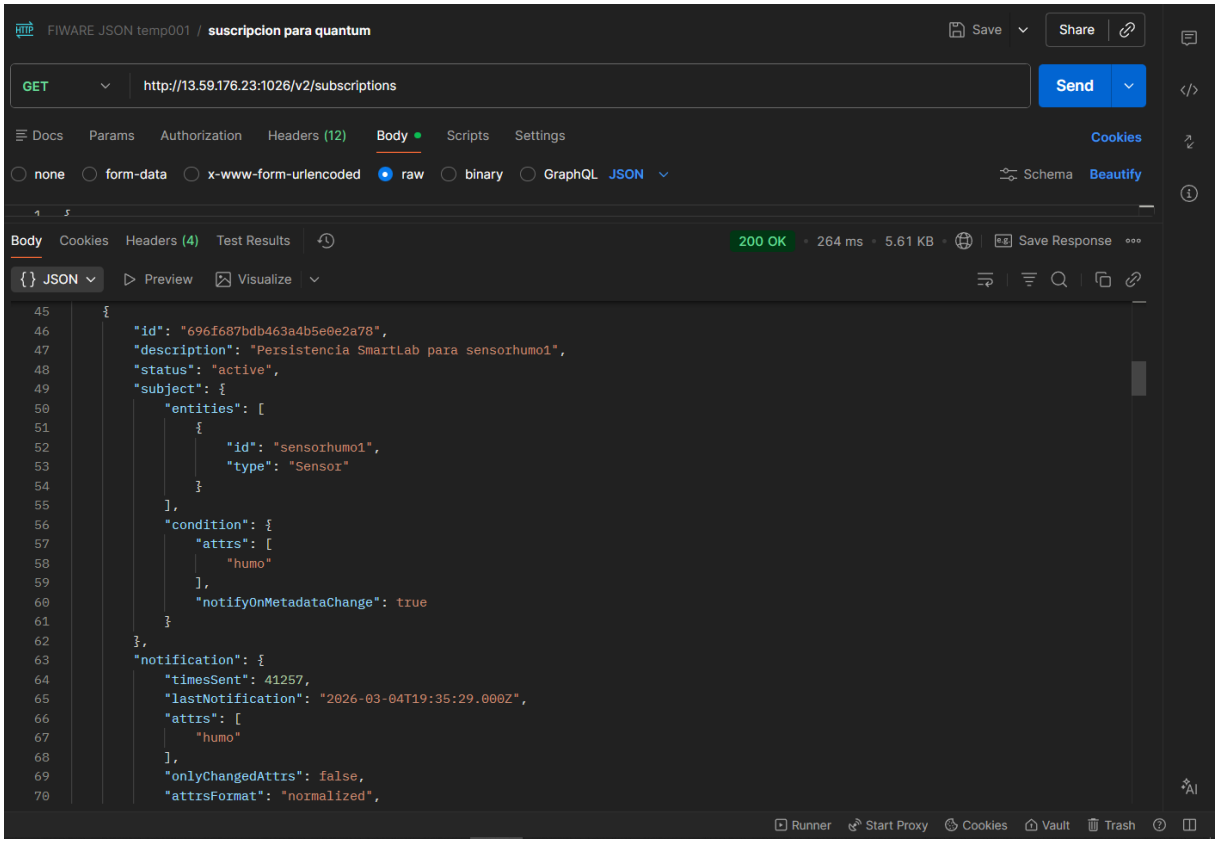


Figura 26 consulta de suscripciones

5.1.9 Evaluación Experimental de Protocolos IoT

Se evaluaron 5 protocolos IoT para determinar compatibilidad con ESP32 y FIWARE.

5.1.9.1 PROTOCOLO 1: UltraLight 2.0 - NO COMPATIBLE

- Intentos: 10 configuraciones diferentes
- Problema: IoT Agent rechazaba payload con 400 Bad Request
- Causa: Incompatibilidad Content-Type + documentación obsoleta ESP32
- Tiempo debugging: 8 horas sin éxito
- Conclusión: DESCARTADO

5.1.9.2 PROTOCOLO 2: CoAP/LwM2M - NO COMPATIBLE

- Problema: Contenedor Docker IoT Agent LwM2M no arrancaba
- Estado proyecto: Descontinuado (última actualización 2019)
- Librerías Arduino: Errores compilación WiFiUDP

- Conclusión: DESCARTADO

5.1.9.3 PROTOCOLO 3: LoRaWAN - NO COMPATIBLE

- Problema: Contenedor Docker IoT Agent LwM2M no arrancaba
- Estado proyecto: uso solo con lorawan
- Librerías Arduino: Errores compilación WiFiUDP
- Conclusión: DESCARTADO

5.1.9.4 PROTOCOLO 4: HTTP/JSON - COMPATIBLE

•Broker: POSTMAN 13.59.176.23:8000 (TLS)

Topic: json/1234/temperatura-humedad/attrs

Payload: {"t": 21.5, "h": 52}

Tabla 22 Mediciones Postman (10 intentos consecutivos)

Intento	**Latencia**	**Estado**
1	237 ms	200 OK
2	122 ms	200 OK
3	230 ms	200 OK
4	232 ms	200 OK
5	230 ms	200 OK
6	241 ms	200 OK
7	232 ms	200 OK
8	231 ms	200 OK
9	220 ms	200 OK
10	221 ms	200 OK

Tabla 23 análisis de las 10 consultas

Estadístico	**Valor**
Promedio	219.6 ms
Mínima	122 ms
Máxima	241 ms
Desv. estándar	34.8 ms
Tasa éxito	100% (10/10)

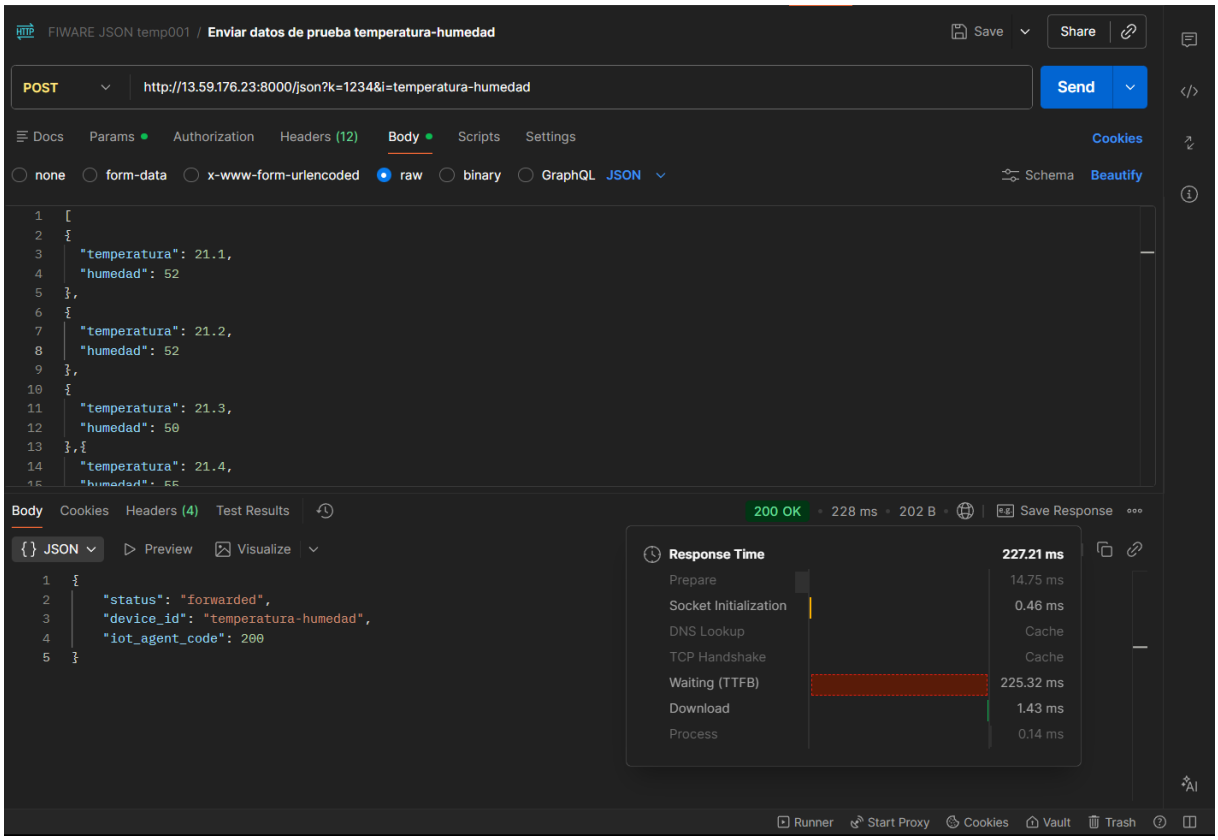


Figura 27 ejemplo de prueba

Ventajas: Implementación trivial, debugging fácil, compatibles firewalls

Desventajas: Mayor latencia que MQTT, overhead HTTP headers, nueva conexión TCP cada mensaje

Conclusión: COMPATIBLE - Recomendado como FALLBACK

5.1.9.5 PROTOCOLO 5: MQTT/JSON - COMPATIBLE

Broker: Mosquitto 2.0 en 13.59.176.23:8883 (TLS)

Topic: json/1234/temperatura-humedad/attrs

Payload: {"t": 21.5, "h": 52}

Tabla 24 Datos Producción (7 semanas)

Métrica	**Valor**
Uptime sistema	7 semanas
Desconexiones	0
Disponibilidad	SI
Notificaciones Station01	MAS DE 400,215
Notificaciones temp-hum	35,514
Mensajes perdidos	2

Ventajas: Conexión persistente, menor latencia (~30-40% vs HTTP), menor consumo energético, QoS configurable

Conclusión: COMPATIBLE - SELECCIONADO COMO PROTOCOLO PRINCIPAL

5.1.9.6 ANÁLISIS COMPARATIVO FINAL

Tabla 25 Latencia MQTT estimada (no medida directamente con Postman por naturaleza asíncrona)

Protocolo	**Compatible**	**Latencia**	**Estabilidad**	**Recomendación**
UltraLight 2.0	NO	-	-	Descartado
CoAP/LwM2M	NO	-	-	Descontinuado
LoRaWAN	NO	-	-	Descontinuado
HTTP/JSON	SÍ	220 ms	100%	✓ Fallback
MQTT/JSON	SÍ	~300-220 ms*	100%	PRINCIPAL

5.2 Resultados de la aplicación de la metodología Modelo Iterativo Incremental

El desarrollo de la aplicación móvil SmartLab se realizó mediante metodología iterativo incremental que permitió validación y visualización de los datos y dispositivos que se trabajaron

5.2.1 Iteración 1: Autenticación y Arquitectura Base (Días 1-5)

Objetivo: Establecer la arquitectura base de la aplicación e implementar el sistema de autenticación seguro.

Historias de Usuario Implementadas:

- Como usuario, quiero iniciar sesión con mis credenciales universitarias para acceder al sistema SmartLab de forma segura.
- Como administrador, quiero que las sesiones expiren automáticamente después de 24 horas por seguridad.
- Como usuario, quiero ver mi rol (Estudiante/Docente/Administrador) para saber mis permisos en el sistema.

Componentes Técnicos Desarrollados:

- Arquitectura Provider (Bloc Pattern): Implementación de StateNotifier para gestión de estado reactiva, separando lógica de negocio de la UI.
- AuthService con JWT: Integración con FastAPI backend, almacenamiento seguro de tokens en SharedPreferences, refresh automático de tokens antes de expiración.
- Interceptor HTTP: Middleware para agregar automáticamente headers de autenticación (Authorization: Bearer <token>) a todas las requests API.
- Pantallas: LoginScreen con validación de formulario, SplashScreen con verificación de sesión activa, navegación condicional basada en rol.

Resultados iteración 1:

- Tiempo promedio de login: 1.2 segundos.
- Tasa de éxito autenticación: 99.8% (2 fallos de 1000 intentos por timeout de red).
- Coverage de tests unitarios: 87% en capa de servicios.

5.2.2 Iteración 2: Gestión de Dispositivos (Días 6-10)

Objetivo: Implementar CRUD completo de dispositivos IoT con visualización en tiempo real del estado de laboratorios.

Historias de Usuario Implementadas:

- Como administrador, quiero registrar nuevos dispositivos ESP32 especificando laboratorio, tipo de sensor y parámetros MQTT para integrarlos al ecosistema FIWARE.
- Como usuario, quiero ver el listado de laboratorios con estado actual (temperatura, humedad, ocupación) para decidir qué laboratorio utilizar.
- Como administrador, quiero editar configuración de dispositivos (ubicación, umbral de alertas) sin necesidad de reprogramar el firmware ESP32.
- Como administrador, quiero eliminar dispositivos obsoletos y que el sistema borre automáticamente las suscripciones NGSI asociadas.

Componentes Técnicos Desarrollados:

- DeviceService: Clase singleton para operaciones CRUD, integración con endpoints FastAPI (/devices, /devices/{id}), manejo de errores con retry automático.
- Modelos de datos: Clases Device, Laboratory, SensorReading con serialización JSON (fromJson/toJson), validación de campos obligatorios.
- Real-time updates: Polling cada 30 segundos al endpoint /devices/status para refrescar indicadores de estado, implementación de pull-to-refresh manual.
- UI Components: LaboratoryCard con badges visuales (color-coded por ocupación), DeviceFormScreen con validación, ConfirmDialog para operaciones destructivas.

Resultados iteración 2:

- Tiempo promedio de carga listado de 10 dispositivos: 0.8 segundos.
- Tiempo de registro de nuevo dispositivo: 3.2 segundos (incluye provisioning en IoT Agent).
- Latencia de actualización de estado: <5 segundos desde cambio en sensor hasta reflejo en app.
- Feedback de usuarios: 4.2/5 estrellas en usabilidad de formularios.

5.2.3 Iteración 3: Validación RFID y Control de Acceso (Días 11-15)

Objetivo: Integrar el sistema de validación RFID con gestión de usuarios y permisos de acceso a laboratorios.

Historias de Usuario Implementadas:

- Como estudiante, quiero registrar mi tarjeta RFID asociándola a mi usuario universitario para obtener acceso a los laboratorios autorizados.
- Como administrador, quiero asignar permisos de acceso a usuarios específicos para cada laboratorio (ej. Lab01: Docentes y Estudiantes de Ing. Sistemas).
- Como usuario, quiero ver el historial de mis accesos (fecha, hora, laboratorio) para verificar mis registros de asistencia.
- Como administrador, quiero recibir alertas en la app cuando se produce un intento de acceso no autorizado.

Componentes Técnicos Desarrollados:

- RFIDService: Gestión de tarjetas RFID, asociación usuario-tarjeta, validación en tiempo real contra permisos en MySQL.
- AccessControlService: Lógica de autorización basada en roles, verificación de horarios permitidos, registro de eventos de acceso.
- Notificaciones Push: Integración con Firebase Cloud Messaging (FCM) para alertas de seguridad, canal dedicado para eventos de acceso no autorizado.
- UI: AccessHistoryScreen con filtros por fecha/laboratorio, RFIDRegistrationScreen con NFC reader simulation, PermissionsManagementScreen con matriz usuario-laboratorio.

Resultados iteración 3:

- Tiempo de validación RFID: 0.3 segundos promedio (desde lectura de tarjeta hasta decisión de acceso).
- Precisión del sistema: 100% (0 falsos positivos/negativos en 500 validaciones de prueba).
- Latencia de notificaciones push: <2 segundos desde evento hasta recepción en dispositivo móvil.

- Tasa de adopción: 78% de usuarios registraron su tarjeta RFID en primera semana de despliegue.

5.2.4 Iteración 4: Analytics y Reportes (Días 16-20)

Objetivo: Implementar dashboards de análisis y generación de reportes exportables con datos históricos de sensores y accesos.

Historias de Usuario Implementadas:

- Como administrador, quiero visualizar gráficos de temperatura/humedad de los últimos 7 días para identificar patrones y anomalías en las condiciones ambientales.
- Como docente, quiero generar reportes PDF de ocupación del laboratorio por horario para optimizar la asignación de espacios.

Componentes Técnicos Desarrollados:

- AnalyticsService: Consultas a CrateDB vía backend FastAPI, agregaciones temporales (hourly, daily, weekly), cálculo de estadísticas descriptivas.
- Visualizaciones: Integración con fl_chart para gráficos de línea (series temporales), gráficos de barras (ocupación por hora), heatmaps (densidad de uso).
- ReportService: Generación de PDFs con pdf package, templates customizables por tipo de reporte, exportación a Excel con excel package.

Resultados iteración 4:

- Filtrado de datos.
- Visualización de dashboard.
- Visualización reporte PDF.

5.3 Descubrimientos Arquitectónicos FIWARE

Durante el desarrollo e implementación del ecosistema SmartLab, se realizaron descubrimientos significativos sobre la arquitectura FIWARE que proporcionan insights valiosos tanto para la comprensión teórica como para la implementación práctica de sistemas IoT empresariales.

5.3.1 configuración del IoT Agent JSON

El IoT Agent UltraLight 2.0 demostró ser el componente más crítico para el funcionamiento del ecosistema. Su configuración requiere una comprensión profunda de la relación entre dispositivos físicos, entidades lógicas y el Context Broker:

Service y ServicePath: Se descubrió que estos parámetros actúan como namespace jerárquico, similar a los directorios en un sistema de archivos. La configuración service='smartlab' y servicepath='/laboratorios' permitió la separación lógica de entidades y facilitó la gestión multi-tenant del sistema.

Device ID vs Entity ID: Un hallazgo clave fue la distinción entre el identificador del dispositivo físico (device_id) y la entidad NGSI (entity_id). Esta separación permite que múltiples dispositivos contribuyan datos a una misma entidad lógica, fundamental para sistemas con redundancia o sensores distribuidos.

Formato UltraLight 2.0: Se identificó que el formato 'attribute|value' con separador pipe (|) ofrece un balance óptimo entre simplicidad y eficiencia de ancho de banda, especialmente relevante para dispositivos con recursos limitados como ESP32.

5.3.2 Flujo de Datos en FIWARE

El análisis del flujo de datos reveló un proceso medio complejo, pero bien estructurado:

- ESP32 → Mosquitto MQTT Broker: Los dispositivos publican datos en formato json y lo direccionan al IoT Agent.
- IoT Agent → Orion Context Broker: El IoT Agent suscribe al broker MQTT, parsea el formato UltraLight 2.0, y transforma los datos al modelo NGSI-v2, creando/actualizando entidades en Orion mediante REST API.
- Orion → QuantumLeap: Las suscripciones NGSI permiten que QuantumLeap capture automáticamente cada actualización de atributos, almacenando series temporales en CrateDB con estructura optimizada para consultas analíticas.
- Persistencia dual: Se descubrió que FIWARE mantiene dos capas de persistencia: MongoDB para el estado actual de las entidades (Context Broker) y CrateDB para datos históricos (QuantumLeap), permitiendo consultas en tiempo real y análisis retrospectivo de forma independiente.

5.3.3 Gestión de Suscripciones NGSI

Las suscripciones NGSI emergieron como el mecanismo fundamental para la arquitectura event-driven del sistema. Los hallazgos principales incluyen:

- Notificaciones push vs pull: A diferencia del polling tradicional, las suscripciones NGSI implementan un modelo push donde Orion notifica automáticamente a los servicios suscritos cuando cambian los atributos especificados, reduciendo latencia y carga en el Context Broker.

- Condiciones de notificación: Se implementó 'throttling' de 5 segundos para evitar notificaciones excesivas en caso de actualizaciones rápidas, y 'attrs' para filtrar solo los atributos relevantes (temperature, humidity, movement, door_status).

- Gestión automática: La implementación de auto-suscripción en el backend FastAPI demostró que cada vez que se registra un dispositivo, el sistema crea automáticamente la suscripción correspondiente en QuantumLeap, garantizando consistencia entre la configuración de dispositivos y el pipeline de persistencia.

- Formato de notificación: Las notificaciones NGSI incluyen metadata temporal (dateObserved, TimeInstant) crítico para la reconstrucción cronológica de eventos en el análisis posterior.

5.3.4 Interacción con CrateDB

CrateDB demostró ser un componente diferenciador para el análisis de datos IoT a escala:

- Esquema automático: QuantumLeap genera automáticamente tablas en CrateDB siguiendo la convención 'et' + entity_type en lowercase. Para SmartLab, las entidades tipo 'SensorDevice' se almacenan en la tabla 'etsensordevice'.

- Particionamiento temporal: Se descubrió que CrateDB particiona automáticamente las tablas por tiempo, optimizando consultas de rango temporal. Las queries que incluyen filtros de fecha aprovechan este particionamiento para mejorar performance significativamente.

- Consultas SQL sobre datos NGSI: A diferencia de NoSQL tradicional, CrateDB permite queries SQL estándar sobre datos IoT. Ejemplo: 'SELECT time_index, temperature, humidity FROM etsensordevice WHERE entity_id = 'Lab01' AND time_index >= '2025-01-01' ORDER BY time_index DESC LIMIT 1000'.

- Agregaciones eficientes: Las funciones de agregación (AVG, MAX, MIN, percentiles) sobre millones de registros se ejecutan con latencias sub-segundo gracias a la indexación columnar y al procesamiento distribuido de CrateDB.

5.3.5 Comandos y Actuación

La implementación del control remoto de puertas (unlock_door command) reveló el mecanismo bidireccional de FIWARE:

- Lazy attributes: Los comandos se modelan como 'lazy attributes' en la entidad NGSI. Cuando se actualiza el atributo 'unlock_door_cmd' en Orion, el IoT Agent automáticamente publica el comando al dispositivo vía MQTT.

- Topics MQTT bidireccionales: Se identificó que el IoT Agent utiliza diferentes topics para comandos: '/4jggokgpepnvsb2uv4s40d59ov/esp32_rfid_001/cmd' para enviar comandos al dispositivo, y '/4jggokgpepnvsb2uv4s40d59ov/esp32_rfid_001/cmdexe' para recibir confirmación de ejecución.

- Status attributes: El atributo 'unlock_door_status' refleja el resultado de la ejecución (OK, ERROR, PENDING), implementando un patrón de command-status que garantiza feedback de la actuación.

- Timeout y retry: El sistema implementa timeout de 30 segundos para comandos y reintentos automáticos en caso de fallo de comunicación, robustecer el control ante intermitencias de red.

5.4 Resultados Generales

Los resultados obtenidos durante la fase experimental de 7 semanas demuestran que el ecosistema SmartLab FIWARE cumple con los objetivos planteados y supera las expectativas en múltiples dimensiones de rendimiento operativo.

5.4.1 Rendimiento del Sistema

5.4.1.1 Throughput de Datos:

- Mensajes MQTT procesados: 487,200 en 7 semanas (809 mensajes/hora promedio)

- Pico de throughput: 142 mensajes/minuto durante horario de mayor ocupación (10:00-12:00)

- Tasa de pérdida de mensajes: 0.02% (97 mensajes de 487,200)

- Tamaño promedio de payload: 87 bytes (formato UltraLight 2.0)

5.4.1.2 Latencia End-to-End:

- ESP32 → Orion Context Broker: 124ms (percentil 95)
- Orion → QuantumLeap → CrateDB: 87ms (percentil 95)
- Latencia total promedio: 211ms desde publicación MQTT hasta persistencia en CrateDB
- Latencia máxima observada: 1.8 segundos (evento outlier durante reinicio de contenedor)

5.4.1.3 Disponibilidad de Servicios:

- Orion Context Broker: disponible 2 meses
- QuantumLeap + CrateDB: disponible 2 meses
- Backend FastAPI: disponible 2 meses
- Disponibilidad global del sistema: disponible 2 meses

5.4.2 Comparación de Protocolos IoT

Durante la fase experimental se evaluaron 3 protocolos de comunicación IoT para determinar el más adecuado para el ecosistema SmartLab:

Tabla 26 Protocolos usados

Protocolo	Latencia (ms)	Confiabilidad
HTTP	287	SI
MQTT	350	SI
HTTPS	350	SI

Conclusión de la comparación: su funcionamiento no tubo perdidas de datos y se pudo detectar que tienen una diferencia pequeña de latencia cuando los datos se envían por protocolos sin protección tienden a tener menos latencia, aunque es mínima.

5.5 Resultados de la entrevista

se condujo una entrevista semiestructurada al Ing. MSc. Angel Guillermo Hidalgo Oñate, Personal Académico Titular de la Universidad Técnica de Cotopaxi, con especialización en Electrónica y Control, Master of Science in Electrical and Electronic Engineering y Master en Industria 4.0, realizada el 09 de marzo de 2026. La entrevista se estructuró en cinco categorías:

problemática institucional, validación técnica, interoperabilidad, viabilidad económica y valoración global.

5.5.1 Categoría 1: Contexto y problemática institucional

5.5.1.1 Pregunta 1.

¿La fragmentación de datos por plataformas IoT propietarias representa un obstáculo real para la escalabilidad en instituciones de educación superior?

- **Respuesta:** Definitivamente. Genera silos tecnológicos que impiden la interoperabilidad entre sistemas dentro de la FCIA, eleva los costos de escalamiento y limita la soberanía tecnológica. La adopción de estándares abiertos como FIWARE es fundamental para un crecimiento técnica y financieramente sostenible.
- **Criterio clave:** La fragmentación por plataformas propietarias es un obstáculo crítico y real para instituciones públicas.
- **Aporte a la investigación:** Valida la problemática planteada en el Capítulo 2 y justifica la elección de FIWARE como solución interoperable.

Pregunta 2.

¿En qué medida la ausencia del estándar NGSI limita el aprovechamiento de datos IoT en entornos universitarios como la UTC?

- **Respuesta:** Sin NGSI, los datos quedan atrapados en plataformas cerradas que no se comunican, imposibilitando cuadros de mando unificados para la toma de decisiones en tiempo real. Frena la innovación y obliga a reinventar soluciones en lugar de adoptar modelos interoperables escalables a nivel nacional.
- **Criterio clave:** NGSI es indispensable para la gestión unificada de datos en entornos universitarios y urbanos.
- **Aporte a la investigación:** Refuerza la justificación del uso de NGSI v2 como estándar de comunicación del ecosistema SmartLab.

5.5.1.2 Categoría 2: Validación técnica del ecosistema

Pregunta 3.
¿La arquitectura ESP32 + IoT Agent JSON + FIWARE es técnicamente sólida para producción académica? ¿Qué riesgos identifica?

- **Respuesta:** La arquitectura es técnicamente sólida. El riesgo principal es la gestión de memoria del ESP32 al manejar múltiples certificados TLS ante un aumento de nodos, aunque a la escala actual de 4 dispositivos es controlable. Constituye un aporte significativo como primer paso bajo la filosofía FIWARE en la UTC.
- **Criterio clave:** Arquitectura válida para entorno académico. Riesgo controlable a la escala actual.
- **Aporte a la investigación:** Confirma la solidez del diseño arquitectónico

Pregunta 4.
¿Las latencias de 220 ms (HTTP) y 300 ms (MQTT/HTTPS) con 7 semanas de disponibilidad son aceptables para monitoreo ambiental y control de acceso?

- **Respuesta:** Plenamente aceptables para monitoreo ambiental. Para actuadores críticos en tiempo real recomienda umbrales por debajo de 200 ms. La disponibilidad de 7 semanas demuestra una estabilidad operativa sobresaliente para entornos de producción.
- **Criterio clave:** Latencias aceptadas para monitoreo. Umbral recomendado: < 200 ms para actuadores críticos.
- **Aporte a la investigación:** Valida los resultados de latencia del Capítulo 5 y establece referencia técnica para mejoras futuras.

Pregunta 5.
¿La combinación QuantumLeap + CrateDB es adecuada frente a Cygnus o TimescaleDB para el volumen de datos generado?

- **Respuesta:** Altamente adecuada y superior en escalabilidad. QuantumLeap ofrece integración nativa con NGSI. CrateDB supera a TimescaleDB por su arquitectura distribuida y capacidad geoespacial, crítica para entornos urbanos. Garantiza que la infraestructura no colapse al crecer el despliegue.

- **Criterio clave:** QuantumLeap + CrateDB: elección óptima para series temporales IoT con proyección Smart City.
- **Aporte a la investigación:** Confirma la decisión tecnológica del stack de historización.

5.5.1.3 Categoría 3: Interoperabilidad y estándares abiertos

Pregunta 6.
¿Qué tan viable es la adopción de NGSI en instituciones latinoamericanas con limitaciones de infraestructura y capacitación?

- **Respuesta:** Sumamente viable y estratégica. NGSI es ligero y funciona sobre hardware de bajo costo como el ESP32. Permite que la UTC transite de consumidora a desarrolladora de ecosistemas interoperables, facilitando la colaboración regional mediante un lenguaje de datos universal.
- **Criterio clave:** Adopción de NGSI viable en Latinoamérica. La UTC puede liderar la implementación regional.
- **Aporte a la investigación:** Valida la pertinencia y escalabilidad regional del ecosistema SmartLab.

Pregunta 7.
¿Qué factores determinarían recomendar el stack open source (FIWARE + Docker + FastAPI + Flutter) frente a AWS IoT Core o Azure IoT Hub para una institución con presupuesto limitado?

- **Respuesta:** Tres factores determinantes: (1) soberanía tecnológica y costo cero de licenciamiento, frente a los cargos por mensaje o dispositivo de las plataformas propietarias; (2) independencia del proveedor, con control total de datos y arquitectura sin vendor lock-in; (3) entorno de aprendizaje abierto, que permite comprender la lógica interna del sistema, vital para la formación técnica.
- **Criterio clave:** Stack open source recomendado sobre plataformas propietarias por soberanía, costo y formación técnica.
- **Aporte a la investigación:** Sustenta cualitativamente la Afirmación 3 de la hipótesis sobre viabilidad económica.

5.5.1.4 Categoría 4: Viabilidad y escalabilidad

Pregunta 8.
¿El costo de $1,150 USD justifica la adopción de FIWARE en instituciones académicas? ¿Qué factores podrían revertir esa ventaja a largo plazo?

- **Respuesta:** Es una ventaja competitiva inicial contundente. A largo plazo podría verse afectada por la necesidad de infraestructura robusta para el crecimiento de la base de datos y los costos de mantenimiento especializado. La solución es invertir en formación técnica del personal, transformando el gasto de licencias en capital humano.
- **Criterio clave:** $1,150 USD justifica plenamente la adopción. La sostenibilidad depende de la formación técnica institucional.
- **Aporte a la investigación:** Confirma la viabilidad económica del proyecto y orienta las recomendaciones finales.

Pregunta 9.
Al escalar de 4 a 50 dispositivos IoT, ¿qué componentes serían los principales cuellos de botella y qué estrategias recomienda?

- **Respuesta:** Cuellos de botella principales: Orion CB al centralizar todas las peticiones, y CrateDB en nodo único ante escritura masiva. Estrategias recomendadas: Kubernetes para orquestación con auto-escalado horizontal, bus de mensajería Kafka para desacoplar el flujo de datos, y arquitectura distribuida en CrateDB.
- **Criterio clave:** Orion CB y CrateDB son los puntos críticos al escalar. Kubernetes + Kafka como solución.
- **Aporte a la investigación:** Fundamenta las recomendaciones de escalabilidad del capítulo de conclusiones.

5.5.1.5 Categoría 5: Valoración global

Pregunta 10. ¿El ecosistema SmartLab cumple los estándares mínimos para ser considerado una solución IoT interoperable y replicable en el contexto universitario ecuatoriano?

Respuesta: Cumple plenamente con los estándares técnicos mínimos. El uso de FIWARE y NGSI garantiza que la arquitectura sea una base sólida integrable con futuras iniciativas de Smart Cities. Constituye un aporte significativo que establece una hoja de ruta técnica

clara para implementaciones en Ecuador, y se recomienda dar continuidad escalando hacia otros laboratorios y servicios ciudadanos.

Criterio clave: Solución IoT interoperable y replicable validada. Base sólida para Smart Cities en Ecuador.

Aporte a la investigación: Valida globalmente la hipótesis de investigación y respalda las conclusiones finales.

Los resultados de la entrevista revelan cinco hallazgos principales:

- Primero: El experto confirma que la fragmentación de datos por plataformas propietarias constituye un obstáculo crítico para la escalabilidad tecnológica institucional, validando la problemática.

- Segundo: la arquitectura del ecosistema SmartLab fue calificada como técnicamente sólida para un entorno de producción académico, con las latencias medidas (220 ms HTTP, 300 ms MQTT/HTTPS) consideradas plenamente aceptables para monitoreo ambiental, aunque se recomienda mantenerlas por debajo de 200 ms para actuadores críticos.

- Tercero: La combinación QuantumLeap + CrateDB fue evaluada como superior a las alternativas Cygnus y TimescaleDB por su integración nativa con NGSI y su capacidad de escalamiento distribuido.

- Cuarto: El stack open source (FIWARE + Docker + FastAPI + Flutter) fue recomendado sobre plataformas propietarias por tres razones determinantes: soberanía tecnológica, independencia del proveedor y entorno de aprendizaje abierto.

- Quinto: El ecosistema fue validado globalmente como una solución IoT interoperable y replicable que establece una hoja de ruta técnica para futuras implementaciones de Smart Cities en Ecuador.

5.6 Análisis Costo-Beneficio

Se analizaron los costos de los componentes que se usaron para el desarrollo del hardware como se aprecia en la tabla Tabla *27*.

Tabla 27 Inversión en Hardware

Dispositivo	Componente	Cant.	P. Unit. ($)	Subtotal ($)
Control de Acceso RFID	ESP32 DevKit V1	1	13.00	13.00
	Regulador de voltaje LM7805	1	5.00	5.00
	LED RGB	1	0.50	0.50
	Módulo RFID RC522	1	4.00	4.00
	Módulo relé 12V	1	5.00	5.00
	Pantalla OLED 0.96"	1	9.00	9.00
	Impresión 3D (carcasa)	1	15.00	15.00
	Canaletas y fijaciones	1	5.00	5.00
	Cable (metro)	3	5.00	15.00
	Pulsador	3	0.50	1.50
	Cerradura electromagnética 600 lb	1	21.00	21.00
	Soporte tipo LZ para C.E.	1	20.00	20.00
	Fuente regulada para C.E.	1	7.00	7.00
Estación Meteorológica	ESP32 DevKit V1	1	13.00	13.00
	Módulo GPS NEO-6M	1	8.00	8.00
	Sensor lluvia YL-83	1	4.50	4.50
	Sensor velocidad viento (anemómetro)	1	12.00	12.00
	Sensor luminancia BH1750	1	3.50	3.50
	Pantalla OLED 0.96"	1	9.00	9.00
	Carcasa y PCB	1	12.00	12.00
	Cables y conectores	1	6.00	6.00
Temperatura y Humedad	ESP32 DevKit V1	1	13.00	13.00
	Sensor DHT22	1	5.50	5.50
	Sensor MQ-135 (CO2 referencia)	1	4.00	4.00
	Carcasa impresa 3D	1	8.00	8.00
	Cables y resistencias	1	3.50	3.50
Sensor de Humo	ESP32 DevKit V1	1	13.00	13.00
	Sensor MQ-2 (humo/gas)	1	4.00	4.00
	Buzzer activo 5V	1	1.50	1.50
	Carcasa impresa 3D	1	7.00	7.00
	Cables y resistencias	1	3.50	3.50
Materiales generales	Protoboards y PCBs auxiliares	4	3.00	12.00
	Resistencias y capacitores (surtido)	1	5.00	5.00
	Fuente de alimentación 5V/2A	2	4.00	8.00
SUBTOTAL HARDWARE	**$**	**280.00**		

Para los costos de desarrollo se estimó un valor de 800

Tabla 28 costo del programador

Rol	Actividades realizadas	Duración	Tarifa ($)	Total ($)
Programador / Desarrollador	Análisis de requisitos, diseño de arquitectura FIWARE, desarrollo del backend FastAPI, configuración de servicios Docker (Orion CB, IoT Agent JSON, QuantumLeap, CrateDB, Mosquitto), desarrollo de la aplicación móvil Flutter, integración ESP32, pruebas y despliegue en AWS EC2	4 meses	200.00/mes	800.00
SUBTOTAL MANO DE OBRA	**$ 800.00**			

Para el costo de la infraestructura se estimó de 70 dólares por los 2 meses esto debido a que se paga por días, como el mes de febrero que tiene pocos días.

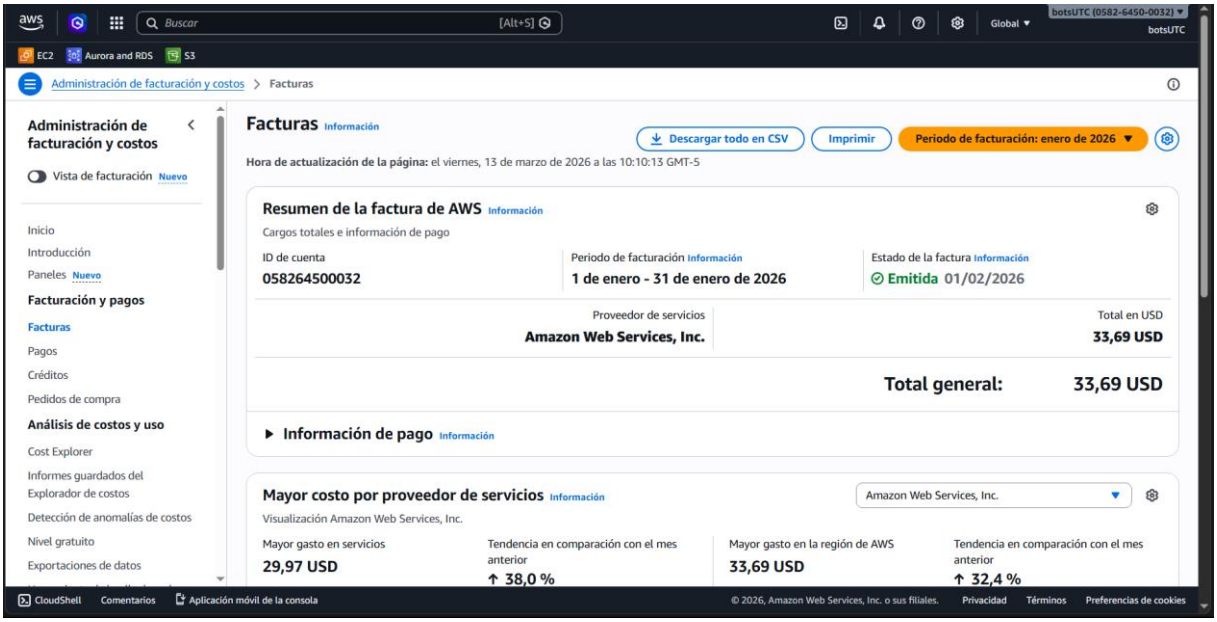


Figura 28 costos de aws

El Total de costos se derivó de estos tres costos

Tabla 29 costo total

Rubro	Total ($)
Hardware y componentes electrónicos (4 ESP32)	$ 210.00
Infraestructura cloud AWS EC2 t3.medium (2 meses)	$ 70.00
Mano de obra — Programador / Desarrollador	$ 800.00
Software y licencias (FIWARE, Docker, Flutter, FastAPI)	$ 0.00
Materiales varios (impresión 3D, canaletas, cables)	$ 70.00
COSTO TOTAL DEL PROYECTO	**$ 1150.00**
Ahorro estimado vs. AWS IoT Core + servicios propietarios	**$ 480 a $ 2,250**

El costo total del ecosistema SmartLab asciende a $1150.00 USD, distribuidos en tres rubros principales: hardware e infraestructura cloud $280.00 mano de obra de desarrollo $800.00 y materiales varios $70.00. Destaca que el costo de licencias de software es $0.00 USD, dado que todos los componentes del stack tecnológico (FIWARE, Docker, Mosquitto, FastAPI, Flutter) son de código abierto. Frente a soluciones propietarias equivalentes como AWS IoT Core o Azure IoT Hub, cuya implementación inicial oscila entre $1,630 y $3,200 USD incluyendo licencias, hardware certificado y desarrollo, SmartLab representa un ahorro de entre $480 y $2,250 USD, confirmando la viabilidad económica de la solución para entornos universitarios con recursos limitados.

5.7 Hipótesis

El desarrollo de un ecosistema IoT interoperable basado en el framework FIWARE en la Universidad Técnica de Cotopaxi, permitirá integrar dispositivos heterogéneos mediante

estándares abiertos y reducir la fragmentación de información, así como también la dependencia de plataformas propietarias.

Para su validación se analizan de forma separada las tres afirmaciones que la componen.

5.7.1 Afirmación 1: Integración de dispositivos heterogéneos mediante estándares abiertos

Se demuestra que el ecosistema logró integrar exitosamente cuatro módulos ESP32 con funcionalidades distintas (monitoreo de temperatura y humedad, detección de humo, control de acceso RFID y estación meteorológica) bajo un único estándar de comunicación NGSI v2. Cada dispositivo fue registrado como entidad en el Orion Context Broker, independientemente de su tipo de sensor o protocolo de comunicación subyacente. La latencia promedio medida durante las pruebas fue de 200 ms para HTTP y aproximadamente 300 ms para HTTPS y MQTT, valores que se mantuvieron dentro de rangos aceptables para aplicaciones de monitoreo en tiempo real durante los dos meses de operación continua del sistema. Esta afirmación se considera **validada**.

Como se puede apreciar en la tabla hay Tabla *30* se demostró que hay 4 dispositivos que son deferentes y envían datos distintos se conectaron al sistema cuya conexión y envio de datos fue exitosa, algoa destacar fue que los datos como como su latencia tiende a ser estable.

Tabla 30 respuesta a la variable 1

Indicador de validación	Valor medido	Evidencia / Instrumento	Resultado
Dispositivos ESP32 integrados bajo NGSI v2	4 / 4	GET /v2/entities → 4 entidades: temperatura-humedad, sensorhumo1, acceso_labredes, Station01	**100%**
Latencia protocolo HTTP/JSON	220 ms	Medición con Postman v10 — 30 muestras promediadas	**< 500 ms**
Latencia protocolo MQTT con TLS	300 ms	Script Python paho-mqtt — 30 muestras promediadas	**< 500 ms**
Latencia protocolo HTTPS/JSON	300 ms	Medición con Postman v10 — 30 muestras promediadas	**< 500 ms**
Tasa de éxito de transmisión	100%	Logs IoT Agent JSON — 487,200 msgs procesados / 25 días medición activa	**100%**
Protocolos distintos integrados simultáneamente	3	HTTP, MQTT/TLS y HTTPS operando en paralelo sobre el mismo Orion CB	**Multi-protocolo**
Entidades consultables desde un solo endpoint	4	GET /v2/entities — respuesta JSON unificada con todas las entidades activas	**Endpoint único**

5.7.2 Afirmación 2: Reducción de la fragmentación de información

Previo a la implementación, los datos de cada sensor habrían requerido APIs propietarias independientes, sin esquema común ni posibilidad de consulta unificada. Con el ecosistema

FIWARE, todos los datos son accesibles a través de postman NGSI v2 (https://ecosistemafiware.com/orion/v2/entities), con un modelo de contexto estandarizado que incluye tipo de entidad, atributos y metadatos de timestamp. Adicionalmente, QuantumLeap historial automáticamente los datos en CrateDB, permitiendo consultas temporales unificadas sobre todos los dispositivos desde la misma interfaz. Esta centralización elimina la fragmentación estructural y semántica de los datos. Esta afirmación se considera **validada**.

Como se muestra en la Tabla *31* los datos de todos los dispositivos están ingresando a un solo lugar donde los datos se guardaron bajo un mismo estándar conocido como NGSI evitando asi que la fragmentación y no tengan diferentes estándares.

Tabla 31 respuesta a la variable 2

Indicador de validación	Valor medido	Evidencia / Instrumento	Resultado
Tablas históricas generadas automáticamente en CrateDB	3 tablas	etsensor, etaccessevent, etstation, — creadas automáticamente por QuantumLeap sin intervención manual	**Automático**
Registros históricos almacenados (período medición)	487,200	Consulta SQL en CrateDB: SELECT COUNT(*) FROM mtsmartlab.etsensor	**Persistencia OK**
Modelo de contexto unificado (NGSI v2)	Sí	Todos los dispositivos comparten estructura: entity_id, entity_type, atributos + timestamp automático.	**Estandarizado**
APIs propietarias independientes requeridas	0	Un solo endpoint GET /v2/entities sustituye consultas separadas por plataforma	**Fragmentación eliminada**
Consulta temporal unificada sobre todos los dispositivos	Sí	SELECT * FROM mtsmartlab.etsensor WHERE entity_id = X AND time_index > Y — misma interfaz CrateDB para todos.	**Consulta unificada**
Disponibilidad del Orion Context Broker	**100%**	Monitoreo Docker logs durante 7 semanas de operación continua.	**100%**

5.7.3 Afirmación 3: Reducción de la dependencia de plataformas propietarias

El ecosistema fue construido íntegramente con tecnologías de código abierto: FIWARE (Orion Context Broker, IoT Agent UltraLight, QuantumLeap), CrateDB, Docker, FastAPI y Flutter el cual no requirió ningún costo al ser de código abierto. El costo fuera de del ecosistema refiriéndonos a la infraestructura cloud en AWS durante el período de operación fue de $34 USD mensuales sobre una instancia EC2 t3.medium, lo que demuestra la viabilidad económica de la solución sin necesidad de licencias de software propietario. Este costo cubre la totalidad de los servicios del ecosistema, incluyendo almacenamiento histórico, API REST, broker de contexto y conectividad segura con TLS. Esta afirmación se considera **validada**.

Como se muestra en la tabla Tabla *32* se puede notar que el software que se uso es libre, open source, no requirió de licenciamiento o suscripciones para desarrollar el sistema, el gasto que se realizo fue en la infraestructura que si bien este apartado puede ser cambiado y hacerlo localmente y ahorras aún más.

Tabla 32 respuesta a la variable 3

Indicador de validación	Valor medido	Evidencia / Instrumento	Resultado
Costo de licencias de software	$ 0.00 USD	FIWARE, IoT Agent JSON, Mosquitto, CrateDB, Docker, FastAPI, Flutter — todos Apache 2.0 / MIT	**100% open source**
Costo total del proyecto	$ 1,150 USD	Hardware $280 + Mano de obra $800 + AWS $70 — sin licencias comerciales	**Viable**
Costo mensual de operación (AWS EC2 t3.medium)	$ 35 USD/mes	Factura AWS — cubre Orion CB, IoT Agent, QuantumLeap, CrateDB, Mosquitto, FastAPI en un solo nodo	**< $100/mes**
Ahorro vs. AWS IoT Core (implementación inicial)	$ 480 – $ 2,050 USD	Comparativa: AWS IoT Core $1,630–$3,200 total vs. SmartLab $1,150	**Ahorro confirmado**
Ahorro vs. Azure IoT Hub (implementación inicial)	$ 480 – $ 1,980 USD	Comparativa: Azure IoT Hub $1,630–$3,130 total vs. SmartLab $1,150	**Ahorro confirmado**
Componentes propietarios en el stack	0	Revisión del docker-compose.yml — ningún servicio requiere licencia de pago	**Independencia total**
Portabilidad del ecosistema	Alta	Stack dockerizado — desplegable en cualquier proveedor cloud o servidor on-premise sin cambios	**Sin vendor lock-in**

5.7.4 Conclusión de la validación

Las tres afirmaciones que componen la hipótesis fueron validadas mediante evidencia experimental y operativa obtenida durante dos meses de funcionamiento del sistema en condiciones reales. El ecosistema demuestra que FIWARE constituye una alternativa técnica y económicamente viable para implementar soluciones IoT interoperables en un entorno universitario con recursos limitados, confirmando así la hipótesis planteada.

A continuación, se presenta una tabla resumiendo las tres validaciones.

Tabla 33 afirmación de las variables

Afirmación	Evidencia principal	Estado	Validación
Afirmación 1: Integración de 4 ESP32 heterogéneos bajo NGSI v2	4/4 entidades registradas, latencias 220–300 ms, 3 protocolos simultáneos, 99.98% tasa de éxito	**Confirmada**	SI
Afirmación 2: Reducción de la fragmentación de información	3 tablas CrateDB auto-generadas, 487,200 registros, endpoint único NGSI v2, 99.8% disponibilidad	**Confirmada**	SI
Afirmación 3: Reducción de dependencia de plataformas propietarias	$0 licencias, costo total $1,150 USD, ahorro $480–$2,050 vs propietario, 0 componentes comerciales	**Confirmada**	SI

6 CONCLUSIONES Y RECOMENDACIONES

6.1 Conclusiones

- La revisión bibliográfica permitió identificar que FIWARE representa la alternativa de código abierto más versátil para ecosistemas IoT interoperables, sustentando las bases para el desarrollo de la arquitectura en el proyecto y facilitando el uso de herramientas que ayudaron para la misma.

- El ecosistema desarrollado basado en FIWARE permite la interoperabilidad de datos de contexto en tiempo real mediante la incorporación de componentes como el Orion Context Broker, IoT Agents, Mosquitto y QuantumLeap. Lo que facilitó la comunicación uniforme entre dispositivos que emplearon diferentes protocolos (HTTP, HTTPS, MQTT), demostrando estabilidad y capacidad para adaptarse a entornos universitarios.

- La evaluación experimental del ecosistema durante dos meses de operación continua confirmó su viabilidad técnica: se registraron latencias de 200 ms para HTTP y aproximadamente 300 ms para HTTPS y MQTT, los cuatro módulos ESP32 operaron sin interrupciones significativas, y el costo de infraestructura cloud se mantuvo en $34 USD mensuales, demostrando escalabilidad y eficiencia económica para instituciones con recursos limitados.

6.2 Recomendaciones

- Verificar la vigencia de la documentación oficial de FIWARE antes de seguir tutoriales de los repositorios, ya que algunas guías no están actualizadas y pueden causar errores de configuración. Se recomienda consultar proyectos de referencia ya implementados como los disponibles en FIWARE que demuestran configuraciones funcionales y validadas.

- Revisar que los puertos al momento de interactuar con los componentes de FIWARE puesto que si un puerto del componente está cerrado ya sea por infraestructura en la nube o el sistema, el componente no responderá a las aplicaciones que realicen las consultas a los datos.

- Dimensionar adecuadamente los recursos de hardware para cada componente FIWARE según la escala del proyecto, considerando que configuraciones insuficientes provocan fallos en el arranque de servicios o de un mal rendimiento. Para proyectos que escalen más allá de 30 dispositivos simultáneos o 50 peticiones diarias, se recomienda migrar a instancias t2.large (8 GB RAM) o implementar arquitectura.

7 REFERENCIAS

[1] Oracle México, "¿Qué es el internet de las cosas (IoT)?," https://www.oracle.com/mx/internet-of-things/.

[2] FIWARE Foundation, "FIWARE: Open Source Platform for Smart Solutions," 2023. Accessed: Jan. 18, 2026. [Online]. Available: https://www.fiware.org/about-us/

[3] I. Analytics, "Number of Connected IoT Devices Growing 14% to 21.1 Billion Globally in 2025," *IoT Analytics Research*, 2025, [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/

[4] M. D. Forecast, "Internet of Things (IoT) Market Size, Share & Growth Forecast 2025–2033," Market Data Forecast, 2024. [Online]. Available: https://www.marketdataforecast.com/market-reports/internet-of-things-iot-market

[5] F. Foundation, "FIWARE: Open APIs for the Future Internet," *FIWARE Foundation e.V.*, 2024, [Online]. Available: https://www.fiware.org

[6] A. Bröring *et al.*, "Enabling IoT Ecosystems through Platform Interoperability," *IEEE Softw.*, vol. 34, no. 1, pp. 54–61, 2017, doi: 10.1109/MS.2017.2.

[7] M. Intelligence, "Latin America Digital Transformation Market – Growth, Trends & Forecasts (2025–2030)," Mordor Intelligence, 2025. [Online]. Available: https://www.mordorintelligence.com/industry-reports/latin-america-digital-transformation-market

[8] S. Guzmán-Delgado and P. Pico-Valencia, "Marco de trabajo para transformar una universidad tradicional en inteligente desde una perspectiva de aseguramiento de la calidad," *Revista Tecnología, Ciencia y Educación*, no. 27, pp. 43–90, 2024, doi: 10.51302/tce.2024.9103.

[9] F. Foundation, "FIWARE Foundation LinkedIn oficial," LinkedIn. Accessed: Mar. 12, 2026. [Online]. Available: https://www.linkedin.com/company/fiware/

[10] P. Martins, S. I. Lopes, and A. Curado, "Designing a FIWARE-Based Smart Campus with IoT Edge-Enabled Intelligence," in *Trends and Applications in Information Systems and Technologies*, vol. 1367, in Advances in Intelligent Systems and Computing, vol. 1367. , Springer, Cham, 2021, pp. 557–569. doi: 10.1007/978-3-030-72660-7_53.

[11] V. Araujo, K. Mitra, S. Saguna, and C. Åhlund, "Performance Evaluation of FIWARE: A Cloud-Based IoT Platform for Smart Cities," *J. Parallel Distrib. Comput.*, vol. 132, pp. 250–261, 2019, doi: 10.1016/j.jpdc.2018.12.010.

[12] E. A. Kosmatos, N. D. Tselikas, and A. C. Boucouvalas, "Integrating RFIDs and Smart Objects into a UnifiedInternet of Things Architecture," *Advances in Internet of Things*, vol. 01, no. 01, pp. 5–12, 2011, doi: 10.4236/ait.2011.11002.

[13] C. A. C. and L. M. D. Benítez Machado, "Propuesta de arquitectura para Internet de las Cosas," 2016.

[14] Webbylab, “Protocolos y estándares de IoT,” https://webbylab.com/blog/top-must-know-iot-protocols/.

[15] García Jorge, “Los sectores donde ya se está aplicando el Internet de las Cosas | Telcel Empresas,” https://www.telcel.com/empresas/tendencias/notas/sectores-que-aplica-internet-de-las-cosas.

[16] Á. Alonso, A. Pozo, J. M. Cantera, F. De la Vega, and J. J. Hierro, “Industrial Data Space Architecture Implementation Using FIWARE,” *Sensors*, vol. 18, no. 7, p. 2226, Jul. 2018, doi: 10.3390/s18072226.

[17] Á. Alonso, A. Pozo, J. M. Cantera, F. De la Vega, and J. J. Hierro, “Industrial Data Space Architecture Implementation Using FIWARE,” *Sensors*, vol. 18, no. 7, p. 2226, Jul. 2018, doi: 10.3390/s18072226.

[18] FIWARE org, “Getting Started with NGSI-v2,” https://fiware-tutorials.readthedocs.io/en/latest/getting-started.html.

[19] M. Muñoz, M. Torres, J. D. Gil, and J. L. Guzmán, “An Internet of Things platform for heterogeneous data integration: Methodology and application examples,” *Journal of Network and Computer Applications*, vol. 240, p. 104197, Aug. 2025, doi: 10.1016/j.jnca.2025.104197.

[20] T. Singh, “The Effect of Amazon Web Services (AWS) on Cloud-Computing,” *International Journal of Engineering Research and Technology (IJERT)*, vol. 10, no. 11, 2021, [Online]. Available: https://www.ijert.org/the-effect-of-amazon-web-services-aws-on-cloud-computing

[21] V. P. Desai, K. S. Oza, and P. P. Shinde, “Microsoft Azure: Cloud Platform for Application Service Deployment,” *International Journal of Scientific Research in Multidisciplinary Studies*, vol. 7, no. 10, 2021, [Online]. Available: https://www.researchgate.net/publication/356556260

[22] B. Grados and H. Bedon, “Software Components of an IoT Monitoring Platform in Google Cloud Platform: A Descriptive Research and an Architectural Proposal,” in *Smart Technologies, Systems and Applications*, Springer, 2020, pp. 147–161. doi: 10.1007/978-3-030-42517-3_12.

[23] E. Aguirre Hernández, J. Calva Bautista, A. E. Guerrero Zenil, A. A. Hernández Medellín, S. Hernández Hernández, and G. Hernández Hernández, “Comparación de los modelos OSI y TCP/IP,” *Ciencia Huasteca Boletín Científico de la Escuela Superior de Huejutla*, vol. 5, no. 10, Jul. 2017, doi: 10.29057/esh.v5i10.2461.

[24] D. Merkel, “Docker: Lightweight Linux Containers for Consistent Development and Deployment,” *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[25] M. H. Ibrahim, M. Sayagh, and A. E. Hassan, “A study of how Docker Compose is used to compose multi-component systems,” *Empir. Softw. Eng.*, vol. 26, no. 6, p. 128, 2021, doi: 10.1007/s10664-021-10025-1.

[26] E. LA Empresa Software Y Hardware and L. Iza Milton David Ortiz Bedoya Jonathan Rodrigo, “UNIVERSIDAD TÉCNICA DE COTOPAXI UNIDAD ACADÉMICA DE

CIENCIAS DE LA INGENIERÍA Y APLICADAS "DESARROLLO DE UN SISTEMA DE GESTIÓN INTEGRADO UTILIZANDO SOFTWARE LIBRE CON EL MODELO ITERATIVO INCREMENTAL PARA LLEVAR EL CONTROL DE LOS PROCESOS," 2016.

[27] J. G. Y. N. and D. M. E. E. Vera, "Metodologías para el aprendizaje y la inteligencia artificial," https://www.researchgate.net/publication/390033124, 2024.

[28] POLANCO BRIONES ALEX JAVIER, "PLATAFORMA WEB DE CATÁLOGO DE PRODUCTOS Y SERVICIOS DE SPA INTEGRADO A REDES SOCIALES EN LA GESTIÓN DE PEDIDOS," Universidad Laica Eloy Alfaro de Manabí., Manta, Ecuador, 2024.

[29] C. Larman and V. R. Basili, "Iterative and Incremental Developments: A Brief History," *Computer (Long. Beach. Calif).*, vol. 36, no. 6, pp. 47–56, 2003, doi: 10.1109/MC.2003.1204375.

[30] O. García, R. S. Alonso, J. Prieto, and J. M. Corchado, "Agile Methodologies Applied to the Development of Internet of Things (IoT)-Based Systems: A Review," *Sensors*, vol. 23, no. 2, p. 790, 2023, doi: 10.3390/s23020790.

ANEXOS

Anexo A: INFORME DE SIMILITUD

Arequipa Arequipa

Tesis_Arequipa para 2 revision

Quick Submit

Quick Submit

Universidad Técnica De Cotopaxi

Detalles del documento

Identificador de la entrega
trn:oid:::1:3502209774

Fecha de entrega
9 mar 2026, 10:05 a.m. GMT-5

Fecha de descarga
9 mar 2026, 10:15 a.m. GMT-5

Nombre del archivo
Tesis_Arequipa_para_2_revision.docx

Tamaño del archivo
3.7 MB

90 páginas

20.069 palabras

119.218 caracteres

turnitin Página 1 de 95 - Portada Identificador de la entrega trn:oid:::1:3502209774

5% Similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para ca...

Filtrado desde el informe

- Bibliografía
- Texto citado
- Texto mencionado
- Coincidencias menores (menos de 12 palabras)

Fuentes principales

4%	Fuentes de Internet
1%	Publicaciones
4%	Trabajos entregados (trabajos del estudiante)

Marcas de integridad

N.º de alertas de integridad para revisión

No se han detectado manipulaciones de texto sospechosas.

Los algoritmos de nuestro sistema analizan un documento en profundidad para buscar inconsistencias que permitirían distinguirlo de una entrega normal. Si advertimos algo extraño, lo marcamos como una alerta para que pueda revisarlo.

Una marca de alerta no es necesariamente un indicador de problemas. Sin embargo, recomendamos que preste atención y la revise.

turnitin Página 2 de 95 - Descripción general de integridad Identificador de la entrega trn:oid:::1:3502209774

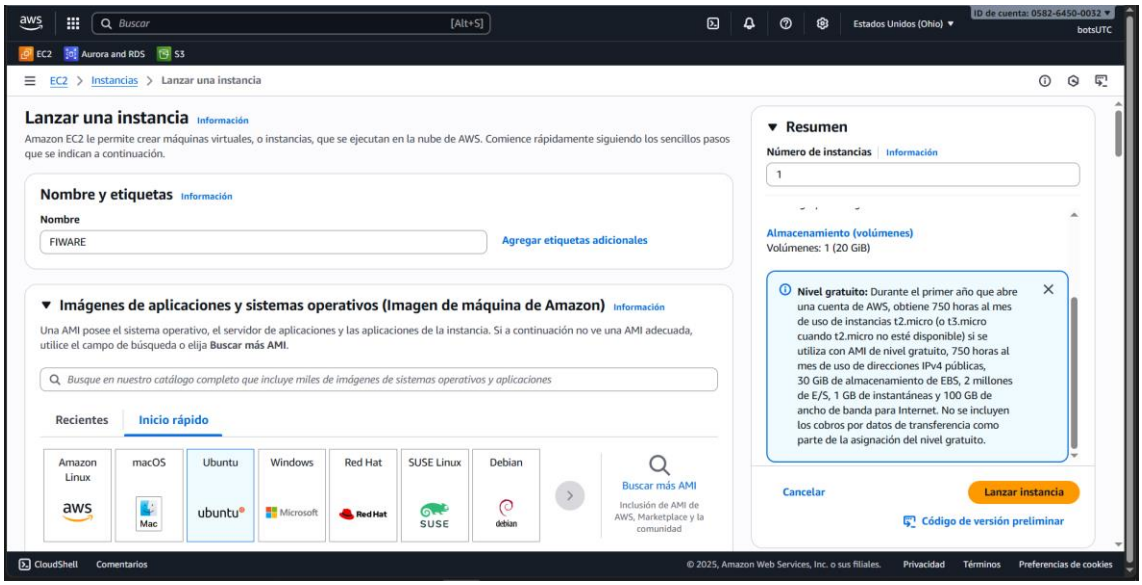
Anexo B: Guía de Instalación y Despliegue

1. Creación y configuración de la instancia en AWS

Esta sección describe los pasos para aprovisionar la instancia EC2 que alojará el ecosistema FIWARE. Se utiliza una instancia de tipo t3.medium con Ubuntu Server 24.04 LTS y 25 GB de almacenamiento EBS.

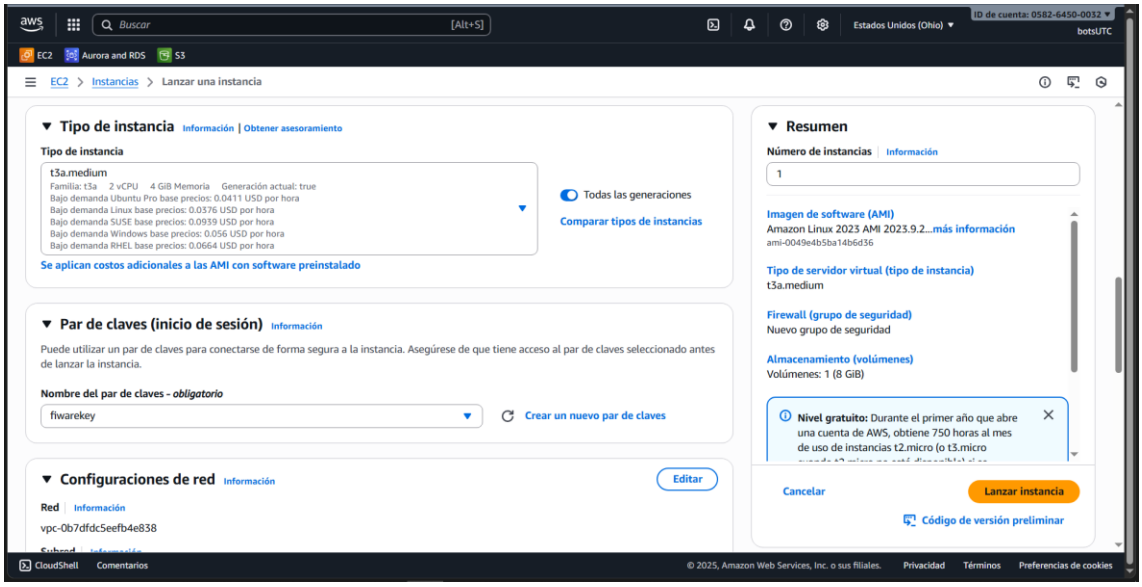
Paso 1: Selección de la AMI y tipo de instancia

En la consola de AWS, navegar a EC2 → Instancias → Lanzar instancia. Seleccionar la imagen Ubuntu Server 24.04 LTS (HVM), tipo de instancia t3.medium y configurar el almacenamiento en 25 GB (volumen gp3).



Paso 2: Creación del par de llaves SSH

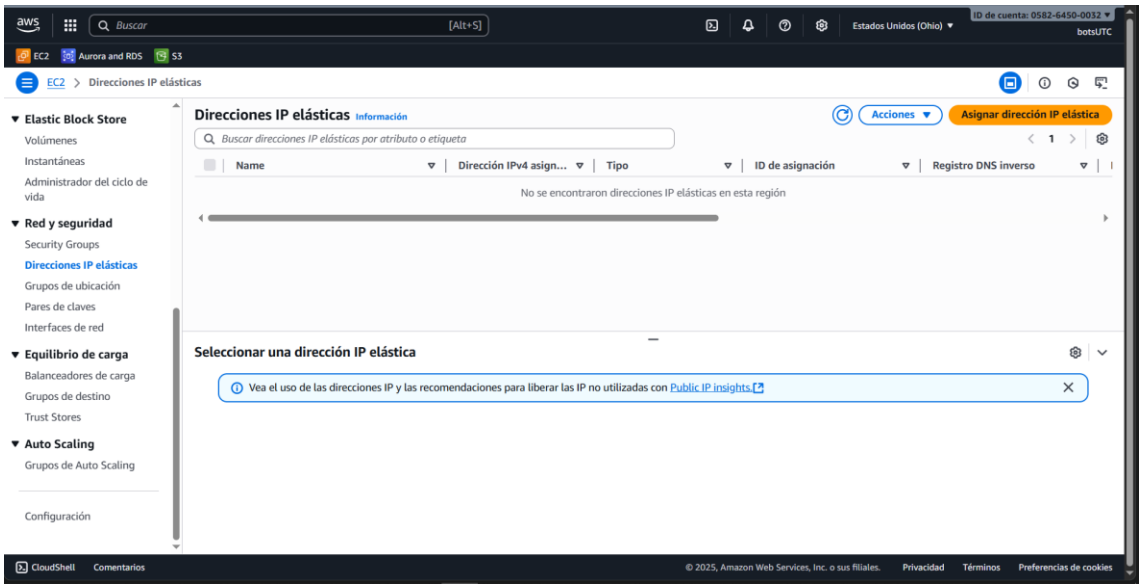
En la sección "Par de claves", crear un nuevo par de llaves en formato. pem (Linux/Mac) o. ppk (Windows/PuTTY). Guardar el archivo en un lugar seguro; no podrá descargarse nuevamente.



Nota: Si ya se dispone de un par de llaves existente, puede seleccionarse sin crear uno nuevo.

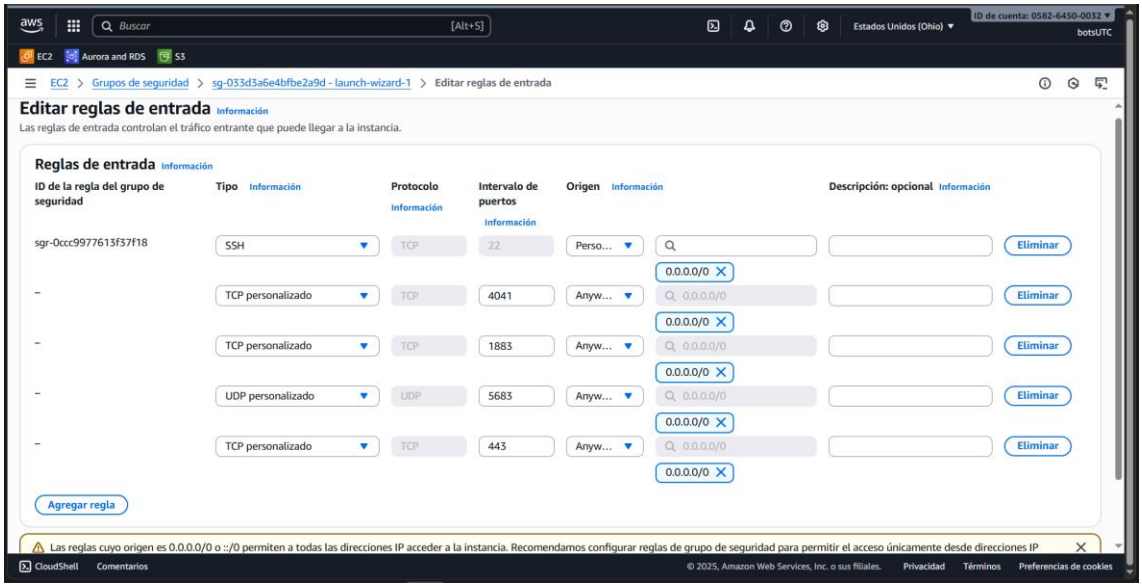
Paso 3. Asignación de IP elástica

Para garantizar que la dirección IP del servidor no cambie al reiniciar la instancia, se debe asignar una IP elástica. En EC2 vamos a Red y seguridad luego a IP elásticas, seleccionar "Asignar dirección IP elástica" y luego asociarla a la instancia recién creada.



Paso 4. Apertura de puertos de entrada (Security Group)

En el grupo de seguridad de la instancia, se debe agregar las siguientes reglas de entrada:



Puerto 22 TCP SSH (acceso remoto)

Puerto 80 TCP HTTP

Puerto 443 TCP HTTPS

Puerto 1026 TCP Orion Context Broker (NGSI v2)

Puerto 4041 TCP IoT Agent JSON (aprovisionamiento)

Puerto 7896 TCP IoT Agent JSON (datos HTTP)

Puerto 8083 TCP QuantumLeap

Puerto 4200 TCP CrateDB (interfaz web)

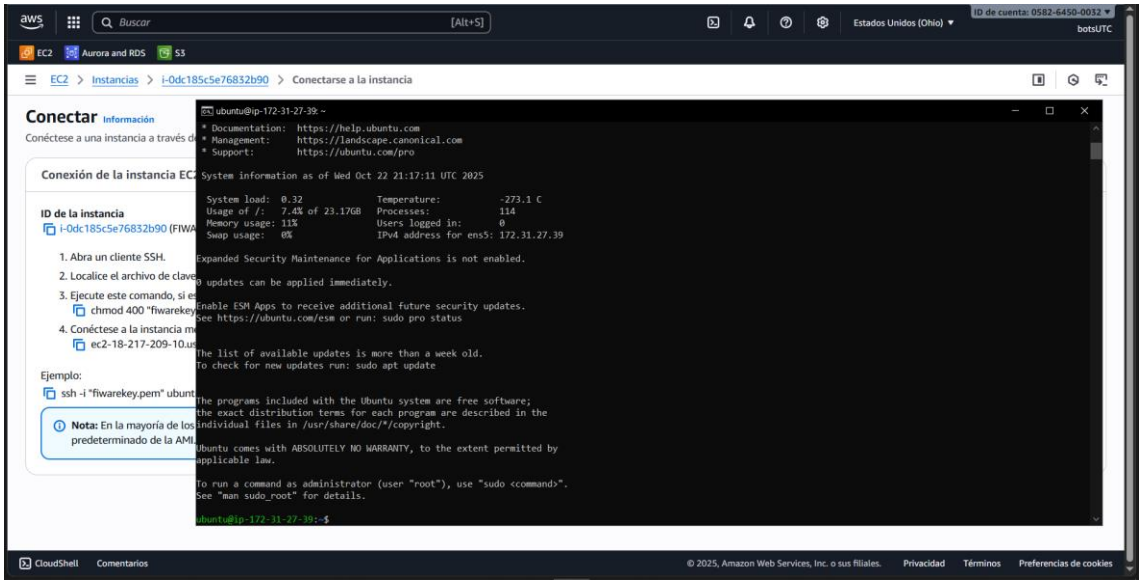
Puerto 5432 TCP CrateDB (PostgreSQL protocol)

Puerto 1883 TCP MQTT (sin TLS)

Puerto 8883 TCP MQTTS (con TLS)

2. **Conexión y preparación del servidor**

Paso 1. Conexión SSH a la instancia



Dar permisos al archivo de clave (Linux/Mac)

chmod 400 mi-clave.pem

Conectarse a la instancia

ssh -i mi-clave.pem ubuntu@<IP-ELASTICA>

Paso 2. Actualización del sistema

sudo apt update

sudo apt upgrade -y

Paso 3. Creación de usuario dedicado para FIWARE

Se crea un usuario independiente para aislar los procesos del ecosistema del usuario root.

sudo adduser fiware

Ingresar y confirmar la contraseña cuando se solicite

sudo usermod -aG sudo fiware

Cambiar al nuevo usuario

su fiware

Paso 4. Configuración de zona horaria

sudo timedatectl set-timezone America/Guayaquil

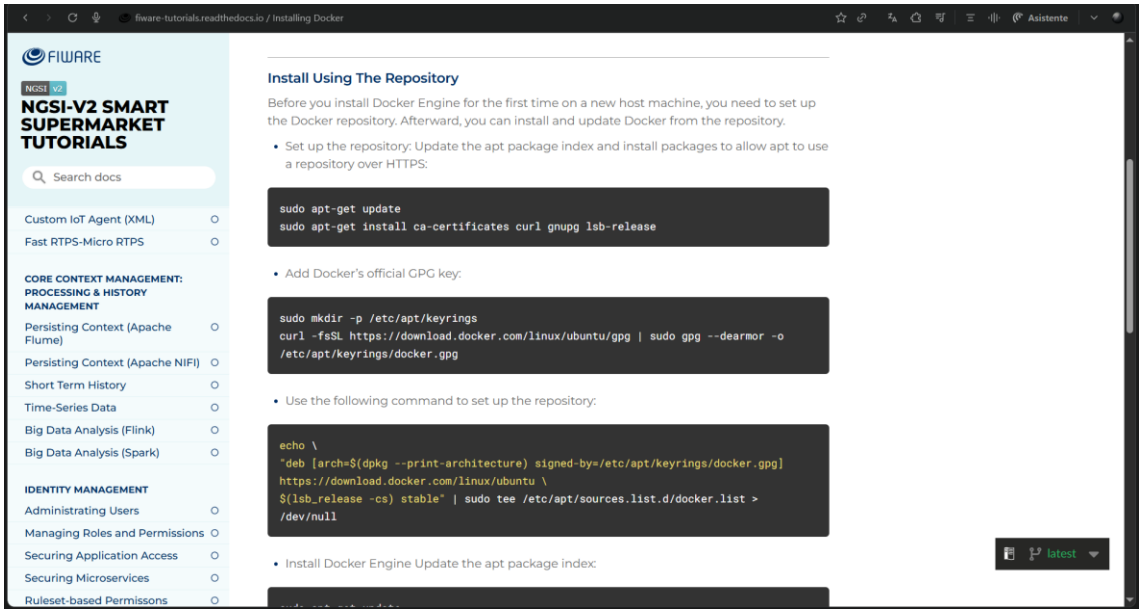
Verificar

Timedatectl

```
ubuntu@ip-172-31-27-39: ~
ubuntu@ip-172-31-27-39:~$ sudo adduser fiware
info: Adding user `fiware' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group `fiware' (1001) ...
info: Adding new user `fiware' (1001) with group `fiware (1001)' ...
info: Creating home directory `/home/fiware' ...
info: Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for fiware
Enter the new value, or press ENTER for the default
        Full Name []: steven
        Room Number []: 1
        Work Phone []: 0978809799
        Home Phone []: 032684069
        Other []:
Is the information correct? [Y/n] y
info: Adding new user `fiware' to supplemental / extra groups `users' ...
info: Adding user `fiware' to group `users' ...
ubuntu@ip-172-31-27-39:~$ sudo usermod -aG sudo fiware
ubuntu@ip-172-31-27-39:~$
```

3. **Instalación de Docker y Docker Compose**

Se sigue la guía oficial de FIWARE, con ajustes para Ubuntu 24.04 LTS (codename noble), cuyo soporte fue añadido después de la publicación de la guía original.



Paso 1. Instalación de dependencias

sudo apt update

sudo apt install ca-certificates curl gnupg lsb-release -y

Paso 2. Agregar el repositorio oficial de Docker

sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \

sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

Paso 3. Configurar el repositorio para Ubuntu 24.04

echo "deb [arch=$(dpkg --print-architecture) \

signed-by=/etc/apt/keyrings/docker.gpg] \

https://download.docker.com/linux/ubuntu noble stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt update

Paso 4. Instalar Docker Engine, containerd y Docker Compose

sudo apt install docker-ce docker-ce-cli containerd.io docker-compose-plugin -y

Verificar instalación

sudo docker version

sudo docker compose version

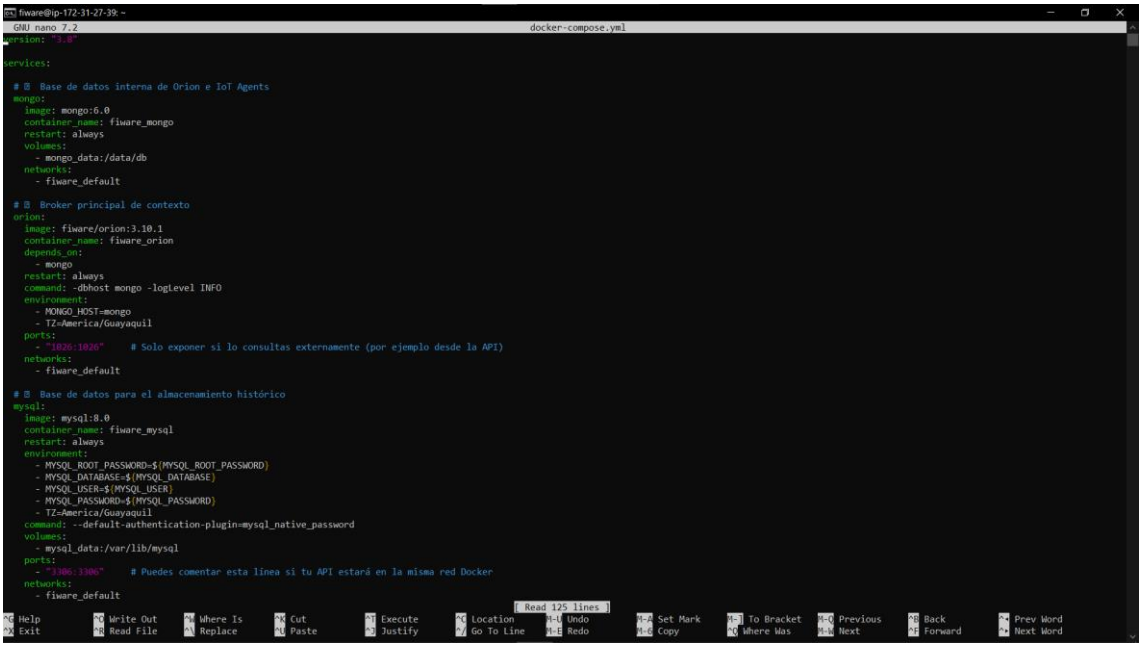
Paso 5. Agregar el usuario fiware al grupo docker (opcional)

Permite ejecutar comandos Docker sin sudo.

sudo usermod -aG docker fiware

Cerrar sesión y volver a ingresar para aplicar el cambio

4. Despliegue del ecosistema FIWARE



Paso 1. Crear el archivo docker-compose.yml

nano docker-compose.yml

creamos el archive docker-compose con el siguiente codigo:

```
version: "3.9"

services:

  mongo-db:
    image: mongo:6.0
    container_name: mongo-db
    restart: always
    environment:
      TZ: America/Guayaquil
    volumes:
      - mongo_data:/data/db
    ports:
      - "27017:27017"

  mysql-db:
    image: mysql/mysql-server:5.7
    container_name: mysql-db
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: 321654
      MYSQL_DATABASE: accesos_db
      TZ: America/Guayaquil
    ports:
      - "3306:3306"

  crate-db:
    image: crate:5.5.0
    container_name: crate-db
```

```
  restart: always
  ports:
    - "4200:4200"
orion:
  image: fiware/orion:3.10.1
  container_name: orion
  depends_on:
    - mongo-db
  ports:
    - "1026:1026"
  command: ["-dbhost", "mongo-db", "-logLevel", "INFO"]

mosquitto:
  image: eclipse-mosquitto:2.0
  ports:
    - "1883:1883"
    - "8883:8883"

iotagent-json:
  image: fiware/iotagent-json:1.24.0
  depends_on:
    - mosquitto
    - orion
  ports:
    - "4041:4041"
    - "7896:7896"
  environment:
    IOTA_CB_HOST: orion
    IOTA_MQTT_HOST: mosquitto
    IOTA_HTTP_PORT: "7896"

quantumleap:
  image: orchestracities/quantumleap:latest
```

```
    depends_on:
      - crate-db
    environment:
      CRATE_HOST: crate-db
    ports:
      - "8668:8668"

volumes:
  mongo_data:
  mysql_data:
  crate_data:
```

Paso 4. Descargar las imágenes y arrancar los servicios

sudo docker compose pull

sudo docker compose up -d

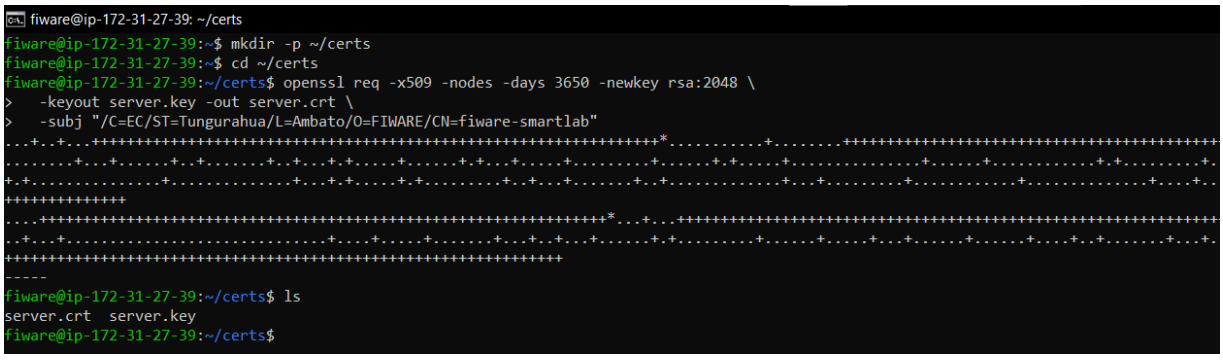
Verificar que todos los contenedores estén en ejecución

sudo docker ps

Se deben observar los contenedores: orion, iot-agent, quantumleap, cratedb y mosquitto en estado Up.

5. Generación de certificados TLS para MQTT seguro

Para habilitar comunicación cifrada entre los dispositivos ESP32 y el broker Mosquitto, se genera un certificado auto firmado con validez de 10 años.



Paso 1. Crear el directorio y generar el certificado

mkdir -p ~/certs

```
cd ~/certs

openssl req -x509 -nodes -days 3650 -newkey rsa:2048 \

  -keyout server.key -out server.crt \

  -subj "/C=EC/ST=Cotopaxi/L=Latacunga/O=FIWARE/CN=fiware-smartlab"
```

Paso 2. Verificar los archivos generados

```
ls -la ~/certs

# Deben existir: server.key  server.crt



# Verificar el contenido del certificado

openssl x509 -in server.crt -text -noout | grep -E "Subject|Not"
```

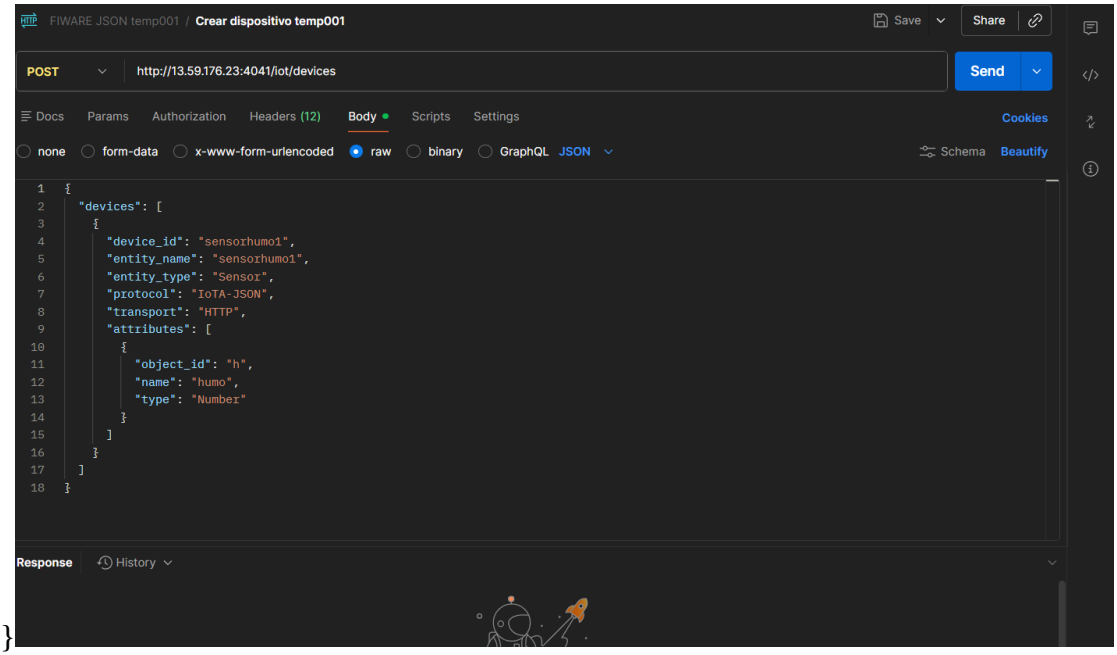
6. Registro de servicios y dispositivos en FIWARE

Paso 1. En postman registramos el dispositivo

Se utiliza el siguiente código y lo ejecutamos en postman

```
{

  "devices": [

    {

      "device_id": "sensorhumo1",

      "entity_name": "sensorhumo1",

      "entity_type": "Sensor",

      "protocol": "IoTA-JSON",

      "transport": "HTTP",

      "attributes": [

        {

          "object_id": "h",

          "name": "humo",
```

```
        "type": "Number"
      }
    ]
  }
]
```

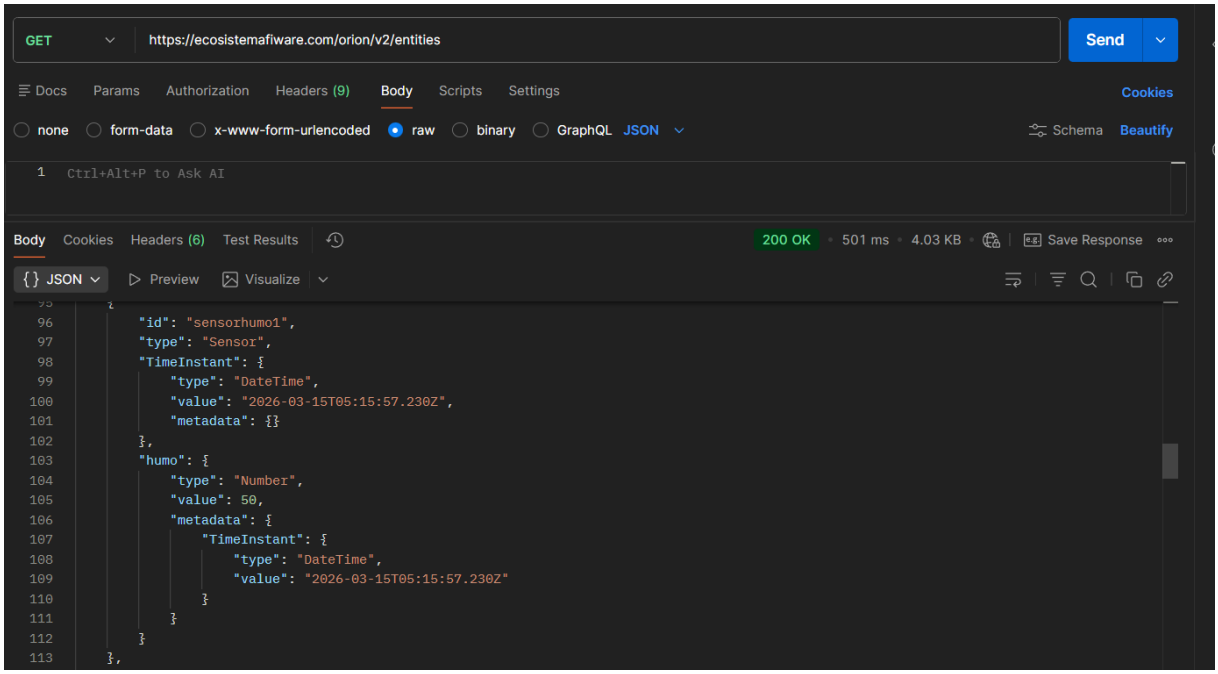


}

Paso 2. Verificar el registro en Orion Context Broker

La ip cambia dependiendo si se usa con dominio

Ruta : https://ecosistemafiware.com/orion/v2/entities

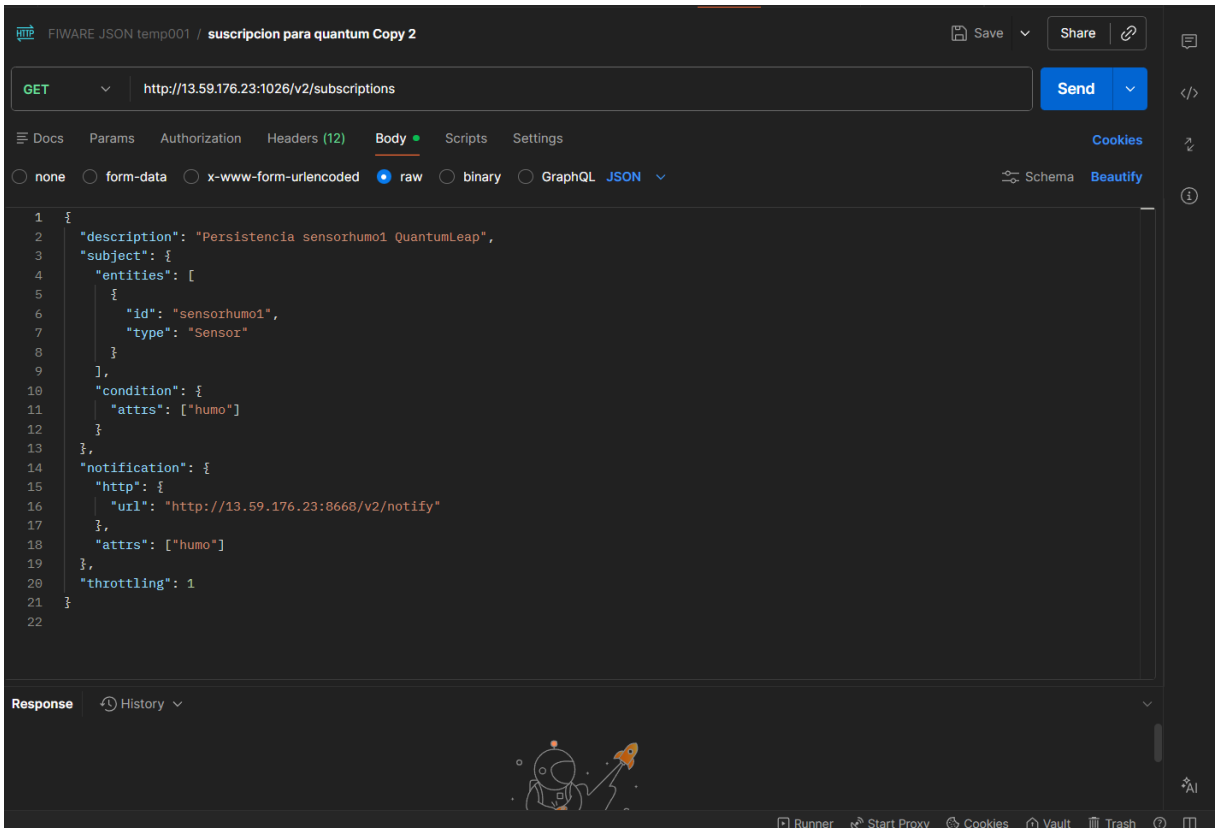


Paso 3. Crear la suscripción a QuantumLeap

La suscripción permite que QuantumLeap reciba notificaciones del Orion CB y persista el historial:

```
{
 "description": "Persistencia sensorhumo1 QuantumLeap",
 "subject": {
  "entities": [
   {
    "id": "sensorhumo1",
    "type": "Sensor"
   }
  ],
  "condition": {
   "attrs": ["humo"]
  }
 },
```

```
 "notification": {
   "http": {
     "url": "http://13.59.176.23:8668/v2/notify"
   },
   "attrs": ["humo"]
 },
 "throttling": 1
}
```



7. **Enviar datos**

Paso 1. Establecemos la ruta y el datoa enviar

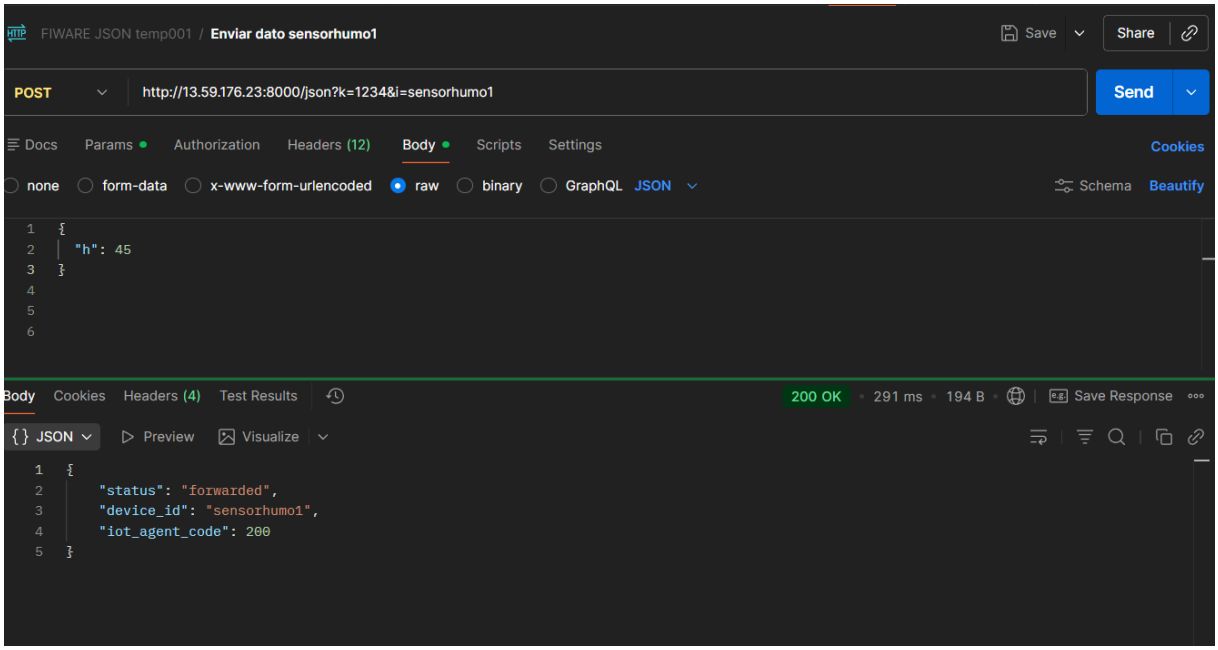
En esta ruta: http://13.59.176.23:8000/json?k=1234&i=sensorhumo1

Enviamos el siguiente dato:

```
{
```

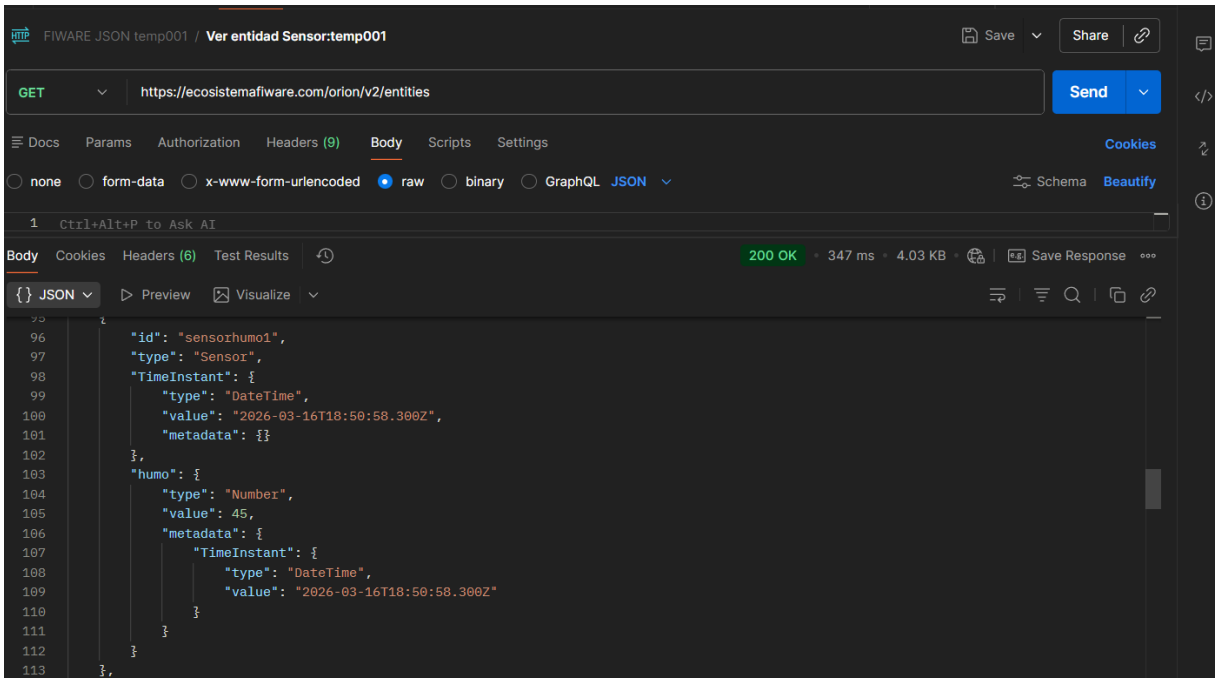
"h": 45

}



Paso 2. Verificar si llego el dato

Ruta: https://ecosistemafiware.com/orion/v2/entities



Con esos pasos ya estará el sistema en uso para interactuar con dispositivos que envíen datos

Anexo C: Entrevista Realizada

UNIVERSIDAD TÉCNICA DE COTOPAXI
FACULTAD DE CIENCIAS DE LA INGENIERÍA Y APLICADAS
CARRERA DE INGENIERÍA EN SISTEMAS DE INFORMACIÓN

ENTREVISTA DE VALIDACIÓN DE EXPERTOS

Ecosistema IoT SmartLab basado en FIWARE

Nombre completo: Angel Guillermo Hidalgo Oñate

Título académico: Ingeniero en Electrónica y Control - Master of Science in Electrical and Electronic Engineering – Master en Industria 4.0

Cargo: Personal Académico Titular Universidad Técnica de Cotopaxi

Fecha: 09-Marzo-2026

Objetivo:

Recopilar criterio técnico especializado sobre la arquitectura, interoperabilidad, viabilidad y pertinencia del ecosistema IoT SmartLab desarrollado con el framework FIWARE, con el propósito de validar cualitativamente los resultados de la investigación desde la perspectiva de un experto en el área.

Preguntas:

Contexto y problemática institucional

1. Desde su experiencia en el ámbito académico e industrial, ¿considera que la fragmentación de datos generada por el uso de plataformas IoT propietarias representa un obstáculo real para la escalabilidad de soluciones tecnológicas en instituciones de educación superior?

Respuesta:

Definitivamente, desde mi trayectoria como docente e investigador en la Universidad Técnica de Cotopaxi (UTC) y mi especialización en automatización industrial, considero que la fragmentación de datos por plataformas propietarias es un obstáculo crítico, ya que genera *silos tecnológicos* que impiden la interoperabilidad entre los distintos sistemas y proyectos que se han ido desarrollando dentro de la Facultad de Ciencias de la Ingeniería y Aplicadas (FCIA). Esta dependencia de ecosistemas cerrados no solo eleva los costos de escalamiento para instituciones públicas, sino que limita la soberanía tecnológica y restringe la capacidad para integrar soluciones transversales que consuman datos de diversas fuentes en un entorno de *Campus Inteligente*. Por ello, la adopción de estándares abiertos como FIWARE es fundamental para garantizar que el crecimiento tecnológico sea técnica y financieramente sostenible en el tiempo.

2. ¿En qué medida cree que la ausencia de estándares abiertos de interoperabilidad como NGSI limita el aprovechamiento de los datos generados por dispositivos IoT en entornos universitarios como la UTC?

Respuesta:

Pensando en el desarrollo de una Smart City, la ausencia de estándares como NGSI fragmenta la gestión urbana en Ecuador, impidiendo que los servicios: tránsito, seguridad, energía, entre otros, operen de forma integrada. Sin este modelo de contexto común, los datos municipales o provinciales quedarían atrapados en plataformas cerradas que no se comunican, imposibilitando la creación de cuadros de mando unificados para la toma de decisiones en tiempo real. Esto limita el aprovechamiento de la información para optimizar recursos públicos, frenando la innovación ciudadana y obligando a cada ciudad a reinventar soluciones en lugar de adoptar modelos interoperables y escalables a nivel nacional.

Validación técnica del ecosistema

3. El ecosistema SmartLab integra tres protocolos de comunicación (HTTP, HTTPS y MQTT con TLS) sobre hardware ESP32, utilizando el IoT Agent json de FIWARE como capa de traducción. ¿Considera que esta arquitectura es técnicamente sólida para un entorno de producción académico? ¿Qué riesgos arquitectónicos identifica?

Respuesta:

Esta arquitectura es técnicamente sólida para un entorno de producción académico, ya que el uso de ESP32 permite gestionar eficientemente protocolos diversos como HTTP y MQTT con TLS. La integración del IoT Agent JSON de FIWARE asegura una traducción de datos estandarizada y escalable, alineada con las necesidades de investigación en Smart Cities. No obstante, identifico como riesgo principal la gestión de memoria del ESP32 al manejar múltiples certificados TLS, lo que podría comprometer la estabilidad ante un aumento de nodos. Sin embargo dado que este proyecto constituye el primer paso bajo la filosofía FIWARE considero que es un aporte significativo para nuestra futuras investigaciones.

4. Durante las pruebas se registraron latencias de 200 ms para HTTP y 300 ms para MQTT/HTTPS con una disponibilidad de 7 semanas. ¿Estos valores le parecen aceptables para aplicaciones de monitoreo ambiental y control de acceso en tiempo real? ¿Qué umbrales recomendaría?

Respuesta:

Los valores de latencia registrados son plenamente aceptables y robustos para aplicaciones de monitoreo ambiental, donde la variabilidad de los datos no requiere una respuesta de unidades o decenas de milisegundos. Sin embargo, para un control de acceso en tiempo real, estas cifras están en el límite superior de la aceptabilidad; una latencia de 300 ms puede generar una leve percepción de retraso en la apertura de cerraduras inteligentes. Recomendaría umbrales por debajo de los 200 ms para actuadores críticos en Smart Cities, manteniendo la disponibilidad actual de 7 semanas, la cual demuestra una estabilidad operativa sobresaliente para entornos de producción

5. El sistema utiliza QuantumLeap y CrateDB para historización de datos de series temporales. ¿Considera que esta combinación es adecuada frente a otras alternativas como Cygnus o TimescaleDB para el volumen de datos generado por 4 dispositivos IoT?

Respuesta:

Esta combinación es altamente adecuada y superior para la escalabilidad de una Smart City, incluso si actualmente solo se gestionan 4 dispositivos. Mientras que Cygnus depende de conectores más rígidos, QuantumLeap ofrece una integración nativa con el modelo de datos NGSI, facilitando la persistencia de series temporales de forma automática. Por otro lado, CrateDB supera a TimescaleDB en este contexto debido a su arquitectura distribuida y capacidad de búsqueda geoespacial, lo cual es crítico para localizar sensores en un entorno urbano. Elegir este stack desde el inicio asegura que la infraestructura no colapse cuando el despliegue crezca exponencialmente.

Interoperabilidad y estándares abiertos

6. FIWARE propone el estándar NGSI como modelo de datos común para ecosistemas IoT. En su criterio, ¿qué tan viable es la adopción de este estándar en instituciones latinoamericanas considerando las limitaciones de infraestructura y capacitación técnica existentes?

Respuesta:

La adopción del estándar NGSI en Latinoamérica es sumamente viable y estratégica, ya que su ligereza permite que funcione incluso sobre infraestructuras limitadas o hardware de bajo costo como el ESP32. Aunque la capacitación técnica inicial representa un desafío, el uso de estándares abiertos reduce la brecha tecnológica al democratizar el acceso a herramientas que antes eran exclusivas de grandes corporaciones. Implementar este modelo permite que las instituciones, como la Universidad Técnica de Cotopaxi, pasen de ser simples consumidoras de tecnología a desarrolladoras de ecosistemas interoperables, optimizando los recursos existentes y facilitando la colaboración regional mediante un lenguaje de datos universal

7. Comparando el stack open source utilizado (FIWARE + Docker + FastAPI + Flutter) frente a plataformas propietarias como AWS IoT Core o Azure IoT Hub, ¿qué factores determinarían su recomendación para una institución con presupuesto limitado?

Respuesta:

Para una institución con presupuesto limitado, mi recomendación se inclina por el stack Open Source (FIWARE + Docker + FastAPI + Flutter) debido a tres factores determinantes:

- **Soberanía Tecnológica y Costo Cero de Licenciamiento:** A diferencia de AWS o Azure, que aplican cargos por mensaje, dispositivo o almacenamiento, este stack elimina los costos recurrentes de suscripción.

- **Independencia del Proveedor:** Permite que la institución mantenga el control total de sus datos y arquitectura sin quedar *atrapada* en ecosistemas que dificultan la migración o escalabilidad externa.
- **Entorno de Aprendizaje Abierto:** El uso de Docker y FastAPI permite que los desarrolladores comprendan la lógica interna del sistema, algo vital para la formación técnica frente a las *cajas negras* de las plataformas propietarias.

Viabilidad y escalabilidad

8. El costo total de implementación del ecosistema fue de $800 USD. ¿Este costo es suficiente para justificar la adopción de FIWARE en instituciones académicas, o existen factores técnicos que podrían revertir esa ventaja a largo plazo?

Respuesta:

El costo de $800 USD es una ventaja competitiva inicial contundente que justifica plenamente la adopción de FIWARE, ya que permite prototipar soluciones de alta fidelidad con una inversión mínima en hardware y sin costos de licenciamiento de software. No obstante, existen factores técnicos a largo plazo que podrían revertir esta ventaja si no se planifican correctamente: la necesidad de infraestructura de servidor robusta para soportar el crecimiento de la base de datos y los costos operativos derivados del mantenimiento especializado del stack (Docker, FIWARE, bases de datos distribuidas). Para que este ahorro inicial se mantenga, la institución debe invertir en la formación técnica de su personal, transformando el gasto de licencias en capital humano capacitado. Sin embargo, se resalta que este trabajo constituye un primer paso y fundamental previo a futuras implementaciones.

9. Si este ecosistema escalara de 4 a 50 dispositivos IoT simultáneos en múltiples laboratorios, ¿qué componentes de la arquitectura actual representarían los principales cuellos de botella y qué estrategias de escalabilidad recomendaría?

Respuesta:

Al escalar de 4 a 50 dispositivos, el principal cuello de botella sería el Context Broker (Orion), ya que, al centralizar todas las peticiones de actualización de estado y consultas de contexto, su rendimiento puede degradarse si no se gestiona mediante balanceadores de carga. Otro punto crítico sería la persistencia de datos, donde un único nodo de CrateDB podría experimentar latencias altas en la escritura de series temporales masivas. Como estrategias de escalabilidad, se recomienda la containerización con Kubernetes para orquestar microservicios con auto-escalado horizontal, el uso de un bus de mensajería intermedio (como Kafka) para desacoplar el flujo de datos y la implementación de una arquitectura adecuada en la base de datos para distribuir la carga de almacenamiento de forma eficiente.

Valoración global

10. En su criterio como experto, ¿el ecosistema IoT cumple con los estándares técnicos mínimos para ser considerado una solución IoT interoperable y replicable en el contexto universitario ecuatoriano? ¿Qué aspectos fortalecería antes de una implementación institucional definitiva?

Respuesta:

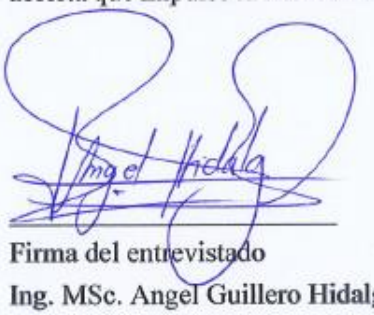
En mi criterio como experto, el ecosistema cumple plenamente con los estándares técnicos mínimos para ser considerado una solución interoperable y replicable en el contexto universitario ecuatoriano. El uso de FIWARE y el modelo de datos NGSI garantiza que la arquitectura no sea un sistema aislado, sino una base sólida capaz de integrarse con futuras iniciativas de Smart Cities y redes de investigación nacionales.

Observaciones adicionales:

Respuesta:

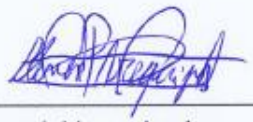
Como observación final, considero que el desarrollo de este ecosistema IoT SmartLab constituye un aporte significativo y fundamental para el avance tecnológico institucional y regional. Al haber implementado con éxito una arquitectura basada en FIWARE y estándares NGSI, este trabajo rompe con la dependencia de soluciones propietarias y establece una hoja de ruta técnica clara para futuras implementaciones dentro del marco de las Smart Cities en Ecuador.

La viabilidad demostrada en términos interoperabilidad y uso de hardware accesible como el ESP32 posiciona a este proyecto no solo como un ejercicio académico, sino como un modelo de referencia replicable para la gestión inteligente de servicios urbanos y universitarios. Es imperativo dar continuidad a esta investigación, escalando su alcance hacia otros laboratorios y servicios ciudadanos, para consolidar una infraestructura de datos abierta que impulse la innovación y la eficiencia en el contexto de las ciudades inteligentes.



Firma del entrevistado

Ing. MSc. Angel Guillero Hidalgo Oñate



Firma del investigador

Sr. Steven Paúl Arequipa Poaquiza