Pipeline para la ingesta de datos desde un UNS a una plataforma de datos en la nube

Cristian Fernando Rugeles Blanco

Trabajo de Grado para optar por el título de Ingeniero Electrónico

Director

Juan Manuel Rey López

Doctor en Ingeniería Electrónica

Co-Director

Iván Darío Hernández Rodríguez

Tutor de la empresa Dautom S.A.S

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

2025

Lista de contenido

Lista de figuras

Lista de tablas

Lista de apéndices

Resumen

Título: Pipeline para la ingesta de datos desde un UNS a una plataforma de datos en la nube[*]

Autor: Cristian Fernando Rugeles Blanco[**]

Palabras clave: IoT, MQTT, Ignition, Industria 4.0

Descripción: En el panorama tecnológico actual, la información se ha convertido en un recurso central, impulsando la necesidad de organizarla, transportarla y usarla eficientemente desde su origen: los datos. Bajo esta necesidad cobra relevancia el concepto de "*Pipeline de datos*", un proceso mediante el cual se define y estructura el flujo de datos, incluyendo su generación, organización, transformación y transporte. Un pipeline puede ser tan simple como una ruta directa entre fuente y destino, o tan sofisticado como para incluir lógica condicional o procesamiento en función del contenido. Para garantizar la organización y acceso a datos industriales, una estrategia es el Unified Namespace (UNS), una capa centralizada que unifica los datos producidos desde distintos puntos de la organización, convirtiéndose en un "hub de datos" donde cada dispositivo y sistema puede publicar o consumir información de forma desacoplada. El UNS conlleva un modelado de datos que permite a los consumidores comprender su origen y propósito dentro de un contexto operativo. Para este modelado, una alternativa popular es la norma ISA-95, que permite representar la jerarquía física de una empresa, asociando cada productor de datos con un nivel en ella. Este proyecto aprovecha una implementación operativa de UNS para construir un pipeline que envía datos hacia *Snowflake*, una plataforma de datos en la nube. El objetivo es agregar valor a la información generada mediante las capacidades de análisis, procesamiento y consulta de las plataformas en la nube. Durante el desarrollo se contemplan los desafíos de interoperabilidad y escalabilidad de la Industria 4.0, contribuyendo a la transformación digital en el sector industrial colombiano.

[*] Trabajo de grado.

[**] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería eléctrica, electrónica y de telecomunicaciones. Director: Juan Manuel Rey Lopez. PhD. Codirector: Iván Darío Rodríguez Hernández. Especialista.

Abstract

Title: Pipeline for data ingestion from a UNS to a cloud data platform*

Author: Cristian Fernando Rugeles Blanco**

Keywords: IoT, MQTT, Ignition, Industria 4.0

In today's technological landscape, information has become a central resource, driving the need to organize, transport, and efficiently use data from its origin. Within this context, the concept of "*data pipeline*" becomes especially relevant, referring to the process by which the flow of data is defined and structured, including its generation, organization, transformation, and transport. A pipeline can be as simple as a direct path between a source and a destination, or as complex as including conditional logic or content-based processing.

As a first step, it is essential to ensure proper organization and access to industrial data. There are various strategies to achieve this; one of them is the Unified Namespace (UNS). The UNS acts as a centralized layer that unifies data produced across different points in the organization, functioning as a data hub where each device and system can publish or consume information in a decoupled manner. Additionally, the UNS involves data modeling that enables consumers to understand the origin and purpose of the information within an operational context. A common trend in Industry 4.0 is to structure this model based on the ISA-95 standard, which defines the physical hierarchy of a company by associating each data source with a level in that structure.

This project leverages an existing UNS implementation to build a pipeline that sends data to *Snowflake*, a cloud-based data platform. The main objective is to add value to the generated information by using the cloud's capabilities for analysis, processing, and querying. Throughout the development, the project addresses key challenges related to interoperability and scalability in Industry 4.0 environments, contributing to the digital transformation of the Colombian industrial sector.

---

* Degree work.

**Faculty of Physicomechanical Engineering. School of Electrical, Electronic and Telecommunications Engineering. Advisor: Juan Manuel Rey López, PhD. Co-advisor: Iván Darío Hernández Rodríguez, Specialist.

Introducción

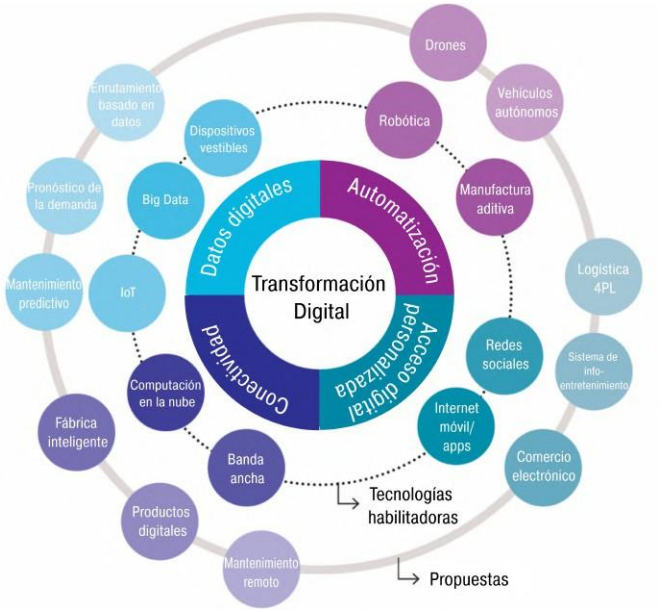
Industria 4.0

La llegada de la cuarta revolución industrial, también conocida como Industria 4.0, ha impulsado el desarrollo de un ecosistema digitalizado que ha transformado profundamente la forma en que se diseñan, controlan y optimizan los procesos industriales. En este nuevo paradigma, los datos se consolidan como un recurso clave para la generación de valor (Gölzer, Cato, y Amberg, 2015), lo que ha desencadenado una carrera tecnológica a nivel global centrada en su adquisición, procesamiento y análisis. Esto se debe a que su aprovechamiento efectivo permite a las organizaciones anticipar fallas, optimizar la producción, mejorar la trazabilidad y tomar decisiones más informadas.

Esta transformación ha llevado a las organizaciones a replantear sus procesos y modelos de negocio, dando paso a lo que se conoce como **transformación digital**. Este proceso implica la integración de tecnologías habilitadoras, como la inteligencia artificial, el IIoT y la computación en la nube (Figura 1), con el objetivo de satisfacer las necesidades particulares de gestión y aprovechamiento de la información en cada empresa (Dritsas y Trigka, 2025).

Figura 1

Infografía transformación digital. Adaptado de: Bloching y cols. (2015)



Como resultado de este proceso, se ha hecho evidente la necesidad de reemplazar arquitecturas industriales tradicionales, caracterizadas por su rigidez, aislamiento y baja escalabilidad. En respuesta, han surgido enfoques más integrados y dinámicos, entre los cuales destaca el **UNS**. Esta arquitectura propone la unificación de datos en una única fuente de verdad: una estructura común y accesible a todos los niveles de la organización. Para lograrlo, el UNS propone el uso de un esquema de publicación/suscripción desacoplado, generalmente utilizando **MQTT** como protocolo de comunicación. A diferencia de los enfoques tradicionales basados en conexiones punto a punto o en historizadores, el UNS permite centralizar el acceso a los datos industriales, facilitando su distribución dentro de la organización. Este enfoque se apoya en estándares como la norma ISA-95, diseñada para facilitar la integración entre los sistemas de control de planta (como SCADA o PLCs) y los sistemas empresariales (como ERP o MES). La ISA-95 define una jerarquía funcional y una jerarquía física con base en la ubicación donde ocurren las actividades (empresa, área, sitio, línea, activo), permitiendo organizar y contextualizar la información. En este marco, el UNS actúa como un hub centralizado desde el cual se producen y consumen los datos, ofreciendo una solución moderna para la integración efectiva y escalable en entornos industriales conectados, volviéndose un modelo cada vez más relevante para las compañías manufactureras (Péter y Werner, 2024).

A la vez que se establecen arquitecturas alineadas con los principios de la cuarta revolución industrial, surge una nueva preocupación: *¿cómo manejar de forma efectiva los datos generados*? Esta pregunta plantea decisiones clave sobre dónde almacenar, procesar y analizar la información. En otras palabras, se trata de un problema relacionado con los recursos computacionales, los costos operativos y la accesibilidad. En este escenario, las plataformas de datos en la nube surgen como respuesta, posicionándose como la alternativa más popular gracias a su flexibilidad, escalabilidad y a la reducción en la complejidad operativa frente a soluciones *on-premise* (AWS Editorial Team, 2023; Grigoriou y Fink, 2023).

La incorporación de estas tecnologías ha despertado interés tanto en pequeñas como

en grandes empresas, debido a su potencial para mejorar procesos productivos y administrativos (Velásquez Ruiz, 2021). A pesar del interés manifestado en sectores como la extracción de materia prima, los servicios y la industria manufacturera, la adopción de estas tecnologías en Colombia aún enfrenta retos significativos. Entre los obstáculos más comunes se encuentran la falta de conocimiento técnico, la escasez de talento especializado y las limitaciones presupuestales (Muñoz Pinzón, Valencia Rivero, y Castro, 2024). Este desafío no es exclusivo de Colombia ni de América Latina. Diversos estudios destacan que migrar hacia arquitecturas centradas en los datos implica riesgos considerables, incluyendo la pérdida de tiempo, recursos y capital si el proceso no se gestiona adecuadamente (Ian Colotla y Spindelndreier, s.f.). Ante esta necesidad, el presente proyecto, desarrollado en colaboración entre el grupo de investigación GISEL y la empresa DAUTOM, busca explorar alternativas para el manejo eficiente de la información y su conexión con la nube. DAUTOM aporta, como caso de estudio, una implementación en curso de una arquitectura basada en principios de IIoT y Unified Namespace, que permite contextualizar y validar la propuesta en un entorno industrial real. Con ello, el proyecto busca hacer tangible la transformación digital en el contexto local, empoderando a las empresas colombianas en esta carrera tecnológica.

Objetivo General

Desarrollar un pipeline de datos que integre un Unified Name Space con una plataforma en la nube para la ingesta de datos.

Objetivos específicos

1. Establecer las especificaciones mínimas en conjunto con la entidad interesada.
2. Evaluar alternativas de tecnologías abiertas para los nodos funcionales del pipeline.
3. Diseñar la arquitectura del pipeline que responda a las necesidades de gestión de ingesta de datos.
4. Realizar pruebas funcionales de la arquitectura para verificar la continuidad del flujo de información y la robustez ante errores de transmisión.

1: Análisis del problema y definición de criterios

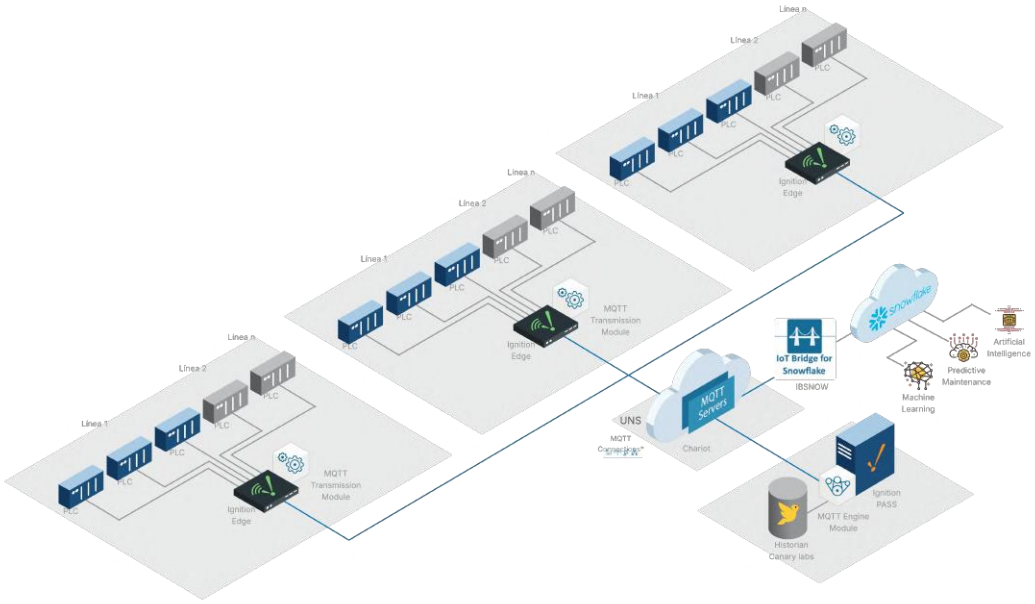
1.1 Contexto

DAUTOM es una empresa santandereana integradora de sistemas, especializada en soluciones de Smart Manufacturing, infraestructura OT y transformación digital. Su trabajo se enfoca en hacer realidad la transformación digital, conectando los mundos IT y OT para impulsar procesos industriales más eficientes, escalables y sostenibles. Actualmente, DAUTOM cuenta con distintos casos de arquitecturas de IIoT basadas en el modelo de Unified Namespace (UNS) utilizando Ignition como plataforma SCADA. Estas arquitecturas no solo responden a las necesidades operativas de supervisión y control, sino que también proporcionan la escalabilidad y modularidad requeridas por los principios de la Industria 4.0. Ante la creciente demanda de sus clientes por integrar soluciones en la nube, la compañía ha iniciado el desarrollo de estrategias para el transporte eficiente de datos hacia dichas plataformas. Para esto, se ha hecho uso de Snowflake, una plataforma de almacenamiento, procesamiento y análisis de datos diseñada para ejecutarse completamente sobre infraestructura en la nube. Su arquitectura híbrida, que combina rasgos de los modelos *shared-disk* y *shared-nothing*, permite el procesamiento de consultas usando MPP (*massively parallel processing*) mientras mantiene un repositorio central de datos accesible y escalable (*Snowflake Documentation*, 2025). Estas y otras características la convierten en una de las plataformas más populares para *Data Warehousing* (*Snowflake - Market Share, Competitor Insights in Data Warehousing*, 2025), volviéndose una alternativa sólida a largo plazo para llevar a cabo procesos de transformación digital. Previo al presente proyecto, la empresa llevó a cabo un primer intento de desarrollar un pipeline hacia Snowflake, en el cual se probó el uso del módulo para Ignition "*IoT Bridge for Snowflake*" desarrollado por *Cirrus Link*. Este módulo se encarga de transmitir automáticamente los datos, consumiendo mensajes en formato MQTT SparkplugB y transformándolos en un formato compatible con Snowflake, mientras estructura los datos y enriquece con metadatos. Además, al ser una solución diseñada específicamente para Ignition, ofrece una integración nativa y robusta, garantizando estabilidad operativa. La eva-

luación del módulo de Cirrus Link se llevó a cabo bajo condiciones operativas definidas: el flujo de datos desde una arquitectura UNS implementada para tres *sites* de una empresa del sector energético (Figura 2). En cada uno de estos sites los datos operativos provenientes de PLCs o dispositivos con soporte MQTT nativo son recibidos por Ignition, donde se realiza un modelado de datos de acuerdo con la ISA-95 para posteriormente ser procesados y enviados a Snowflake por el módulo "*IoT Bridge for Snowflake*". El resultado de esta primera prueba fue una base de datos en Snowflake conectada directamente con Ignition mediante la inserción periódica por *Microbatching*.

Figura 2

Arquitectura usando IoT bridge for Snowflake



Las condiciones operativas, costos y criterios presentes en el resultado reportado por la empresa sobre el uso del módulo, serán usados como referencia para este proyecto. El reporte del costo anual por el uso del módulo "*IoT Bridge for Snowflake*" se presenta en la Tabla 1.

Tabla 1

Estimación de costos anuales del módulo IBSnow de Cirrus Link

Concepto	**Valor**
Costo por hora del módulo "*IoT Bridge for Snowflake*"	1.25 USD/h
Horas operativas anuales estimadas	8,760 h
Costo anual del módulo	10,950 USD
Costo anual por uso de cómputo en Snowflake	26,280 USD
Costo anual estimado del broker MQTT	3,600 USD
Costo total anual	**40,830 USD**

1.2 Necesidades y criterios para el diseño del pipeline

Si bien la arquitectura y *pipeline* de datos probado usando el módulo de *Cirrus Link* cumplen con múltiples criterios importantes en la industria 4.0, su uso supone un problema principal y es un costo elevado. Esto es un factor limitante para desarrollar un producto disponible para un amplio abanico de clientes. Además, su función es exclusivamente el envío de datos hacia Snowflake desde Ignition, por lo que se pierde la centralización de los datos en la arquitectura. Por esto, se planteó reemplazar esta etapa, ya sea definiendo la estructura del proceso como ELT, o bien optando por un ETL. Debido al gran y creciente interés de conectar la industria con la nube, existen numerosas alternativas posibles para llevar a cabo este reemplazo en la integración con Snowflake. Para abordar este proceso se definieron criterios para la exploración y selección del software. Inicialmente, se aplicó el método MoSCoW como herramienta para definir los criterios básicos en la exploración de alternativas, como será explicado en la siguiente sección ("MoSCoW" es un acrónimo para cada categoría del marco de priorización "**M**ust have", "**S**hould have", "**C**ould have" y "**W**on't have") .

1.2.1 Método MoSCoW

Este enfoque permite identificar qué características o capacidades son imprescindibles (Must Have), cuáles son deseables pero no críticas (Should Have), aquellas que pueden considerarse como adicionales o complementarias (Could Have) y las que no serán consideradas en esta etapa (Won't Have).

- ***M**ust have*
  - **Modo de transmisión**: La solución debe cumplir con al menos uno de los siguientes modos de transmisión: (1) Streaming en tiempo real, o (2) Microbatching con intervalos de carga configurables.
  - **Compatibilidad de protocolos**: Debe tener compatibilidad con MQTT SparkplugB y MQTT Vanilla ya que son los principales protocolos de comunicación en la arquitectura.
  - **Integración directa con Snowflake**: La solución debe proporcionar integración directa con Snowflake, evitando añadir múltiples herramientas intermedias.
  - **Costo total de implementación y operación**: Debe ser notablemente inferior a la suma del costo del módulo de *Cirrus Link* y los costos generados por operación, por ejemplo, el cómputo en *warehouse*.
  - **Facilidad de adopción**: La solución debe ser fácil de adoptar por parte del equipo técnico, minimizando la curva de aprendizaje necesaria para su uso y mantenimiento.
- ***S**hould have*
  - **Interfaz Gráfica**: Si bien es posible la adopción de soluciones basadas en archivos de configuración y scripting, esto no se considera óptimo.
  - **Integración a distintas plataformas de nube**: La necesidad inmediata es la conexión con Snowflake, pero incluir otras plataformas permite cumplir con la

modularidad y flexibilidad que exige el mercado.

- **Portabilidad del despliegue**: Las configuraciones deben poder exportarse con el fin de facilitar nuevos despliegues.

- ***C**ould have*
  - **Sistema de seguridad externo:** La compatibilidad con OAuth, LDAP, entre otros, es un atributo deseable pero no es crítico.
  - **Herramientas de Monitoreo**: Sistemas de alarma, notificación e historial de errores o pérdida de datos.
  - **Retención de datos**: La capacidad de almacenar localmente datos recientes como estrategia para mitigar la pérdida de datos.
- ***W**on't Have*
  - **Operaciones y transformaciones avanzadas**: La conversión a distintos formatos, extracción de información, el uso de lógica condicional o el enriquecimiento con datos externos, no serán necesarios pues pueden ser realizados en Snowflake.

2: Exploración de alternativas y selección de Software

El enfoque inicial de este proyecto fue comprender las distintas formas en que puede llevarse a cabo la integración con la nube. Para ello, se aplicó un primer filtro que permitió obtener un listado de tecnologías que cumplieran con los criterios fundamentales. Dado que todas ellas superan con amplio margen los requisitos técnicos mínimos (latencia, throughput, tasa de error, consumo de recursos, etc.) estos aspectos se detallarán en una sección posterior.

2.1 Tecnologías candidatas

Con base en la exploración realizada, se propone la siguiente clasificación, enfocada en distinguir el rol que puede asumir cada software dentro de la arquitectura, así como el modo de transmisión de datos que utiliza.

2.1.1 Brokers MQTT con capacidades de integración

Dada la creciente popularidad del protocolo MQTT y su adopción como uno de los más utilizados en entornos industriales James R. Koelsch (2022), muchos brokers han incorporado extensiones a su función principal. Estas extensiones permiten la transmisión de datos hacia plataformas de almacenamiento, procesamiento y otros servicios en la nube. Los principales brokers que resultaron de la exploración fueron: **EMQX**, **HiveMQ**, **Cedalo Pro Mosquitto**.

2.1.2 Herramientas intermedias para orquestación y transformación de datos

Otra alternativa común y asequible son los programas intermediarios diseñados para extraer datos desde múltiples fuentes, con capacidades para transformar, limpiar, enriquecer y enrutar información. Entre las plataformas destacadas se encuentran: **NodeRed**, **Apache NiFi** y **HighByte**.

2.1.3 Plataformas de procesamiento de datos en streaming

Esta categoría agrupa tecnologías orientadas al procesamiento continuo y en tiempo real de los datos, abarcando desde su extracción hasta su distribución. Entre las plataformas evaluadas se encuentran **Confluent**, **RedPanda**.

2.2 Modo de ingesta de datos

A partir de esta exploración, también es posible clasificar las tecnologías de acuerdo al mecanismo de ingesta de datos en Snowflake. Esta clasificación resulta importante para seleccionar una u otra tecnología, pues la inserción de datos en Snowflake genera un costo basado en los recursos utilizados. Existen dos modos principales de facturación: uno basado en créditos consumidos por el uso de recursos computacionales durante la inserción y otro que depende directamente del volumen de datos y la cantidad de archivos cargados.

2.2.1 Drivers y conectores

Cedalo Pro Mosquitto, EMQX y las Herramientas intermedias suelen usar *drivers* (JDBC, ODBC, Node.js) provistos por Snowflake. En este caso, las aplicaciones definen su lógica de carga mediante comandos SQL como *INSERT INTO* ejecutados directamente en Snowflake. En el caso de los brokers MQTT, estos pueden incluir la configuración de un intervalo de tiempo mínimo entre cada carga. Además, EMQX cuenta con un motor de reglas que permite personalizar más el proceso de ingesta. Cada comando SQL ejecutado requiere activar un *warehouse* en Snowflake. El tiempo de cómputo de la *warehouse* se factura en función de su tamaño. Cada activación conlleva un mínimo de 60 segundos facturables. Por lo tanto, si se programan inserciones pequeñas de forma continua, el costo acumulado puede aumentar significativamente, ya que será equivalente a estar usándolo de manera constante *Warehouse considerations | Snowflake Documentation* (2025).

2.2.2 Snowpipe y Snowpipe Streaming

Mientras tanto, las plataformas de procesamiento de datos en streaming y el broker HiveMQ usan *Snowpipe* y *Snowpipe Streaming*, servicios de Snowflake para facilitar la carga de información sin necesidad de activar un warehouse. Snowpipe funciona bajo el modelo de micro-batching, es decir, agrupa datos en pequeños lotes que se cargan tan pronto llegan a un almacenamiento externo (como S3). Esto se logra gracias a un sistema de notificaciones que avisa automáticamente a Snowflake cuando hay archivos nuevos, lo que permite iniciar el proceso de carga casi de inmediato. Aunque hay una ligera demora, esta opción es ideal

para cargas frecuentes que no requieren inmediatez total. Por otro lado, Snowpipe Streaming permite que los datos lleguen a Snowflake fila por fila, de forma continua, sin necesidad de crear archivos ni esperar notificaciones. Este enfoque se parece más a un flujo en tiempo real y resulta especialmente útil cuando se necesita baja latencia y respuesta inmediata. Ambos mecanismos gestionan los recursos del servidor automáticamente sin depender de un *warehouse*. En consecuencia, se reduce significativamente el costo de computo en Snowflake Huang (2023)

2.3 Selección

Tras la evaluación inicial, se decidió descartar Confluent y RedPanda, ya que, aunque ofrecen funciones robustas para el diseño de pipelines, su alto costo y la subutilización de sus capacidades las convierten en opciones con bajo retorno de inversión. Se optó entonces por tecnologías que ofrecen mayor margen de escalabilidad sin incrementar significativamente los costos, y que además pueden integrarse directamente con Snowflake. A continuación se describen los criterios evaluados para el proceso de selección.

Costo total:

a) **Costo de software:** El costo total por licencia y operación del software, algunas de las opciones consideradas, además de cobrar una licencia con características definidas, incluyen modelos *"Pay as you go"* o cobro adicional por tráfico.

b) **Costo operativo en Snowflake:** Se refiere al costo generado al cargar datos en Snowflake en un intervalo de tiempo, en este caso se define como referencia el costo diario considerando la distinción en los costos de usar *Snowpipe, Snowpipe streaming* y *Conectores o Drivers*, siendo estos últimos los menos eficientes.

2. **Escalabilidad:** La escalabilidad de un sistema puede definirse como la capacidad de funcionar correctamente al incrementar la carga de trabajo a la que se le somete. Algunas características que aportan escalabilidad en los softwares explorados son:

a) **Despliegue en Clusters y balanceo de carga:** El uso de varios nodos de procesamiento coordinados para la ejecución permite contar con mecanismos de *Failover*, *High Availability* y balanceo de cargas entre nodos para mejorar el rendimiento.

b) **Facilidad de configuración:** A medida que el sistema crece, la integración de nuevos flujos de datos debe realizarse con agilidad. Por ello, contar con un software de configuración sencilla es un requisito clave.

3. **Integración multicloud:** La compatibilidad con otras plataformas en la nube como Amazon Redshift y Google Cloud Platform permite tener alternativas a futuro para ofrecer distintos servicios sin necesidad de incorporar otro programa intermediario.

4. **Funciones incorporadas:** Hace referencia a funcionalidades adicionales no directamente asociadas al procesamiento o transmisión de datos, pero que contribuyen al mantenimiento, supervisión y seguridad del sistema. Entre estas se incluyen:

   - Visualización de métricas operativas (conexiones, errores, latencia, throughput).
   - Registro de eventos y logs para depuración.
   - Notificaciones o alertas programables basadas en eventos del sistema.
   - Mecanismos de autenticación y autorización: compatibilidad con TLS/mTLS, LDAP, OAuth, JWT, entre otros. Además, control de acceso mediante ACL o RBAC para la administración del programa.

5. **Rendimiento:** Algunas métricas importantes del rendimiento usadas para la evaluación de *pipelines* son:

a) **Maximum Sustainable Throughput:** La máxima cantidad de mensajes por segundo que puede manejar manteniendo estabilidad operativa.

b) **Latencia promedio:** El tiempo que tarda cada mensaje desde su entrada al pipeline hasta su llegada al destino.

c) **Consumo de recursos:** Pensando en despliegues *On premise* y *Edge* es preferible usar software ligero, es decir, con bajo consumo de RAM y exigencia en núcleos de CPU.

6. **Soporte y documentación:** Pensando en la facilidad de adopción y en el tiempo requerido para la implementación es importante contar con buena documentación y soporte de los desarrolladores.

A partir de estos criterios se construye una matriz de decisión (Mostrada en la Tabla 3) asignando una calificación de 0 a 5 para obtener el software con mejor puntuación en correspondencia a las necesidades específicas de DAUTOM.

Tabla 2

Comparativa de precios entre distintas tecnologías

Tecnología	**Costo de uso (USD/año)**
EMQX	20,000
HiveMQ	18,000
Cedalo Pro Mosquitto	4,940
Node-RED	0 (open source)
Apache NiFi	0 (open source)
HighByte	>17,500

Debido a la naturaleza competitiva de los productos evaluados, la mayor parte de la información utilizada para la asignación de puntajes provino de las páginas oficiales de cada proveedor. Para obtener una visión más imparcial, la información oficial se complementó con discusiones y análisis en foros técnicos especializados en ingeniería y automatización. La evaluación del criterio de costos presentó desafíos particulares, derivados de las diferencias en los esquemas de licenciamiento.

Tabla 3

Matriz de decisión ponderada

Criterio	Peso (%)	Cedalo	EMQX	HiveMQ	Node-Red	Apache NiFi	HighByte
Costo total	25	4	2	2	**5**	**5**	1
Escalabilidad	20	**5**	**5**	**5**	3	4	**5**
Integración multicloud	15	4	**5**	**5**	**5**	3	**5**
Facilidad de adopción	15	4	4	4	3	2	4
Funciones incorporadas	10	3	4	3	2	2	**5**
Rendimiento	10	4	**5**	4	2	3	4
Soporte y documentación	5	4	4	4	**5**	2	4
Score ponderado		**4.1**	**3.95**	**3.75**	**3.7**	**3.4**	**3.7**

Algunos proveedores requerían una definición completa de la arquitectura para emitir cotizaciones, lo cual restringía la posibilidad de proyectar soluciones escalables o adaptables. Estas cotizaciones (Tabla 2), al estar ligadas a condiciones altamente específicas, limitaban el margen de flexibilidad en el diseño. En contraste, Cedalo Pro Mosquitto ofrecía modelos de precios más modulares, basados, por ejemplo, en el número de conexiones concurrentes y tráfico mensual, lo que facilitó su comparación y permitió mayor libertad en el diseño de la arquitectura.

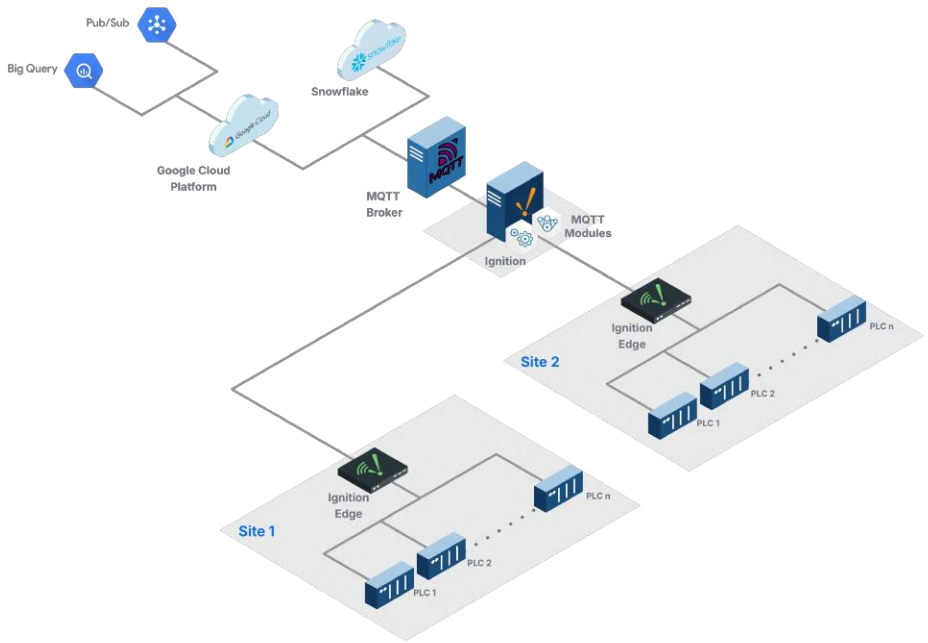
Un aspecto notable es que HighByte destacó como una herramienta robusta para el diseño de pipelines, con soporte para múltiples integraciones y con un énfasis notable en el modelado de datos, sin embargo, debido a que no cuenta con un licenciamiento modular, mantiene un costo elevado, resaltando la rentabilidad de Cedalo. En general, la opción de usar brokers MQTT para el proyecto de integración obtuvo los mejores puntajes. Esto se debe a que las transformaciones necesarias pueden realizarse directamente en Snowflake utilizando SQL y Python, sin requerir la orquestación de flujos complejos con condiciones lógicas o eventos. Dentro de las tres opciones exploradas, múltiples benchmarks apuntan a HiveMQ y EMQX como las alternativas más robustas, debido a su baja tasa de errores, alta capacidad de manejo de throughput y baja latencia en escenarios intensivos *MQTT Benchmark Suite* (2023), Koziolek, Grüner, y Rückert (2020). Sin embargo, como se evidencia en el criterio de costo total, EMQX y HiveMQ obtuvieron una valoración muy baja, costando aproximadamente 20,000 USD y 18,000 USD respectivamente. Esto permitió destacar al broker MQTT Pro Mosquitto de Cedalo, el cual fue finalmente seleccionado para implementar el pipeline, contando con mejor puntuación frente al resto de opciones, y destacando entre los 3 brokers MQTT explorados, como se observa en la Tabla 3, gracias a su buen rendimiento y bajo consumo de recursos, considerándose un broker ultraligero. Además, se considera que el caso de uso implica un volumen de datos moderado, lo que permite mantener baja latencia, al nivel del resto de brokers mientras ofrece características clave para la escalabilidad y seguridad *MQTT at the Edge, which do you choose??* (2025). Por último, cabe destacar que escoger un broker como solución implica una ligera modificación a la arquitectura inicial (Figura 2), ya que evita la necesidad de incorporar una etapa intermedia entre el broker MQTT y Snowflake, o entre Ignition y Snowflake. En su lugar, basta con sustituir el broker previamente utilizado. La elección de Cedalo Pro Mosquitto está respaldada no solo por la documentación, pruebas y análisis de facturación de las distintas alternativas, sino también por tendencias observadas en redes y foros especializados en automatización industrial *Thred and CedaloA Use Case for Industrial Data Optimization* (s.f.).

3: Implementación del pipeline

El pipeline de datos comienza con la generación de datos en máquinas y dispositivos industriales ubicados en distintos sites de la empresa. A través de protocolos como OPC UA, Modbus TCP/RTU, Ethernet/IP o Profinet, los datos son adquiridos por Ignition desde el Edge. Desde allí se envían a un servidor de *Ignition Standard*, donde se usa la jerarquía propuesta por la norma ISA-95 para organizar las fuentes de datos con respecto a su posición dentro de la empresa manufacturera (*Enterprise/Site/Area/Line/Cell/Device*) HiveMQ (2024). Es el conjunto de dispositivos inteligentes, brokers MQTT y plataformas IIoT lo que apalanca el UNS, habilitando una única fuente de verdad en tiempo real.

Figura 3

Arquitectura de la plataforma IIoT integrada con la nube



En la arquitectura modificada, Ignition continúa siendo el punto central de recepción de la información proveniente de PLCs y dispositivos con soporte nativo para MQTT. Sin embargo, al ser un broker MQTT esto añade la versatilidad para conectar nuevos dispositivos y clientes, expandiendo la funcionalidad del UNS.

3.1 Instalación y despliegue

Si bien el broker Pro Mosquitto ofrece opciones de despliegue en la nube con funcionalidades equivalentes a su versión *on-premise*, en este proyecto se optó por una implementación local. Esta decisión se basó tanto en consideraciones de costo como en el hecho de que los requerimientos de conectividad, latencia y seguridad podían ser satisfechos completamente desde un entorno local. En total, no se identificaron ventajas significativas en utilizar un broker basado en la nube. Una vez definido el tipo de despliegue, se seleccionó la opción basada en Docker, entre las modalidades de instalación ofrecidas.Docker fue escogido por su facilidad de instalación, alta portabilidad del despliegue y compatibilidad con distintos sistemas operativos, lo que resultó útil durante la etapa de desarrollo.

Como se indicó previamente, un criterio importante en la selección incluía la posibilidad del despliegue en *Clusters* para *HA* (*High Availability*), lo cual es una característica incluida en la versión de Mosquitto desarrollada por Cedalo. No obstante, se consideró suficiente realizar un despliegue en un único nodo (*Single Node*). Al igual que el uso de *HA*, muchas de las características y funciones ofrecidas por el broker no serán usadas durante este proyecto, ya que se busca preservar la posibilidad de habilitarlas en escenarios futuros, cuando la arquitectura lo requiera. Los requerimientos de hardware para este tipo de despliegue se muestran en la Tabla 4. El levantamiento del contenedor se realizó mediante el archivo YAML “docker-compose.yml” incluido en el paquete de instalación. La aplicación se compone de dos servicios, uno relacionado con la ejecución del broker, y otro con la plataforma web para su administración. En este caso se mantuvieron las configuraciones predeterminadas, alterando solo la sección de mapeo de puertos (*Port mapping*) como se muestra en el Apéndice A. Este mapeo se realiza para evitar conflictos con otros servicios locales que accedan a estos puertos predeterminados. El segundo archivo relevante para la configuración es "mosquitto.conf", en él se definen los puertos de escucha del servicio (*Port Listeners*), la configuración de plugins, TLS, persistencia de mensajes, acceso, APIs y declaración de ubicación de archivos de configuración. En la primera etapa de exploración, se

usó MQTT sin TLS conservando las demás configuraciones por defecto (Apéndice B). Tras acabar la primera etapa, se configuró la conexión por TLS en el puerto 8883 como se ve en el Apéndice D.

Tabla 4

Requisitos de hardware para el broker. Fuente: Hardware Requirements | Cedalo MQTT Platform, *2024*.

Recurso	Mínimo	Recomendado	Stream Processing
RAM	128MB	4GB	16GB
CPU	1 núcleo ARM de 32 bits	CPU AMD/Intel de 4+ núcleos reciente	CPU AMD/Intel de 4+ núcleos reciente
Storage	20MB	200MB	40GB

Debido a que se utilizó la versión gratuita (trial), algunas características de la plataforma no están disponibles, sin embargo, como se ve en el Apéndice D, en la barra de navegación lateral, se incluyen herramientas de monitoreo, medidas básicas de seguridad y un cliente gráfico para pruebas *Publish/Subscribe*.

Desde la pestaña de ***Broker Insights***, mostrada en el Apéndice E, se pueden apreciar las métricas disponibles, relacionadas con el tráfico de mensajes MQTT. Estas métricas permiten evaluar la carga operativa del broker en términos de volumen y frecuencia de mensajes procesados. Para comprobar el funcionamiento del broker se realizaron pruebas de conexión desde distintos dispositivos en la red usando clientes desarrollados en Python (Apéndice C). Tras finalizar la prueba confirmando la operación normal se procedió con la activación del plugin de Snowflake siguiendo la documentación provista por Cedalo.

3.2 Configuración Plugin: Snowflake Bridge

Para el uso del plugin es necesario configurar los recursos que serán usados en Snowflake, esto incluye usuario, rol, warehouse, base de datos, schema y tabla. Aunque es posible

usar cualquier usuario, se considera buena práctica asignar recursos específicos para mantener un control estricto sobre el acceso y consumo de recursos. A continuación se muestran los recursos asignados:

```
user      = "CEDALO_INGEST"
role      = "CED_BRIDGE_PROCESS_RL"
warehouse = "CED_BRIDGE_INGEST_WH"
database  = "CEDALO_BRIDGE_STAGE_DB"
schema    = "CEDALO_TESTING"
```

Además, se requiere de una tabla objetivo donde se almacenarán los contenidos de cada mensaje, creada en el *schema* y *database* seleccionados, con columnas definidas de acuerdo al formato de datos enviados, en este caso la llamaremos "SPARKPLUG_RAW". Toda la configuración mencionada se evidencia en el Apéndice F .

```
1 plugin /usr/lib/cedalo_snowflake_bridge.so
2 persistence_location /mosquitto/data
```

Código 1: Activación del plugin de Snowflake en el archivo mosquitto.conf

Una vez se estableció lo necesario en el lado de Snowflake, el siguiente paso es la configuración del plugin, esto se hace añadiendo las líneas mostradas en el Código 1 en el archivo mosquitto.conf. La configuración del plugin se realizó siguiendo la documentación oficial de Cedalo, la cual proporciona el esquema JSON requerido para definir los parámetros de conexión a Snowflake, la configuración operativa del plugin, el mapeo de tópicos MQTT a tablas y el mapeo de los campos del mensaje a las columnas de la tabla destino. En el Apéndice G, se muestra el JSON de configuración usado. En esta primera prueba se especifican las siguientes opciones para la configuración operativa del plugin *Snowflake Bridge | Cedalo MQTT Platform* (2024):

- **bufferSize:** Define la cantidad de mensajes MQTT que puede almacenar el buffer antes de iniciar el proceso de inserción en la tabla, en este caso se definen 10000 mensajes.

- **batchSize:** Divide los mensajes almacenados en el buffer en lotes, en este caso se definen 1000 mensajes, lo que implica que, si el buffer se llena, se tendrán 10 batches de mensajes insertados en secuencia. Es posible usar la misma cantidad definida en bufferSize para enviar todos los mensajes almacenados.

- **timeoutMs:** Especifica el tiempo máximo en milisegundos que puede pasar antes de que el contenido del buffer se inserte en la tabla, incluso si el buffer no está lleno, en este caso se definen 10 s a modo de prueba inicial.

Con esta configuración se evidencia la inserción de mensajes en la base de datos, confirmada desde la opción de *Data preview* para la tabla "SPARKPLUG_RAW" mostrado en la Figura 4. Esta primera prueba permitió confirmar el funcionamiento básico del Plugin. Sin embargo, debido a la frecuencia de inserción de datos, especificada por el parámetro **timeoutMs** el consumo de la Warehouse es equivalente a usarla de manera constante las 24 horas del día, esto produce un costo anual de 26,280 $. Con el fin de encontrar una estrategia para reducir el costo se decide hacer un ajuste en las opciones del plugin. Con este fin se modifica el parámetro **timeoutMs**, definiendo un intervalo de 5 minutos de recolección de datos antes de cada carga, mientras se mantiene el **bufferSize** y **batchSize**. Tras haber implementado esta configuración durante una semana, se encontró que el consumo promedio del día es de 5.32 créditos. Siguiendo los cálculos mostrados en la Tabla 5,al hacer los créditos por día igual a 5.32 se encuentra un costo anual estimado de 5,825 $ *Understanding overall cost | Snowflake Documentation* (s.f.). De acuerdo al soporte técnico de Cedalo, la capacidad del buffer llega a los cientos de Megabytes y considerando que 3 días de datos produjeron alrededor de 144 MB para este caso, se concluye que existe un margen amplio para ajustar los parámetros del buffer, buscando un equilibrio entre el costo y el requerimiento de baja latencia en la disponibilidad de datos para el cliente.

Tabla 5

Cálculo estimado de costos anuales usando una warehouse X-Small en Snowflake

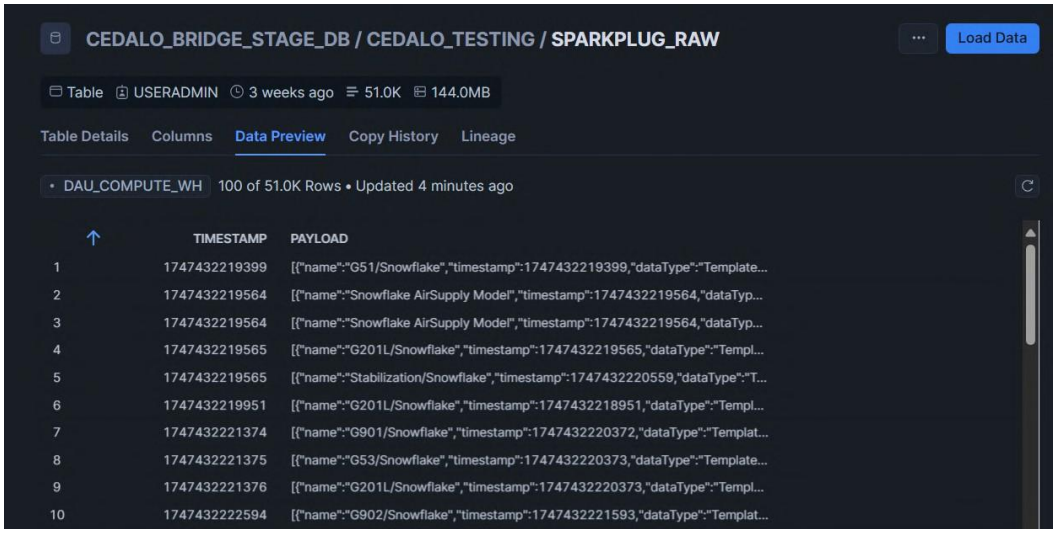
Parámetro	**Valores con batches de 10 segundos**	**Valores con batches de 5 minutos**
Tamaño de warehouse	X-Small	X-Small
Consumo por hora	1 crédito/h	0.221 crédito/h
Horas de operación diaria	24 h	24 h
Créditos por día	1 crédito/h × 24 h = 24 créditos/día	0.221 crédito/h × 24 h = 5.32 créditos/día
Créditos por año	24 créditos/día × 365 días = 8,760 créditos/año	5.32 créditos/día × 365 días = 1,942 créditos/año
Costo por crédito	3 USD/crédito	3 USD/crédito
Costo anual de cómputo	**26,280 USD/año**	**5,825.4 USD/año**
Licencia anual Broker	**4,940 USD/año**	**4,940 USD/año**
Costo total anual	**31,220 USD/año**	**10,765.4 USD/año**

Considerando estos resultados, el escenario más eficiente en términos de costo —dado este volumen de datos— es operar en modalidad de *batching*, es decir, almacenando mensajes durante intervalos de varios minutos u horas antes de su inserción. Además, el broker puede manejar exitosamente cargas en modalidad de *micro-batching*, (intervalos de recolección de datos en lotes que van desde cientos de milisegundos hasta varios minutos) *Data Ingestion Patterns in AWS: A Practical Guide* (2025). Este atributo lo hace una solución versátil considerando las condiciones de facturación de Snowflake y esta versatilidad puede extenderse

aún más para la ingesta en otras plataformas en la nube, cuyo modelo de facturación depende más del volumen de datos insertado que del tiempo de cómputo, por ejemplo Google Cloud Platform.

Figura 4

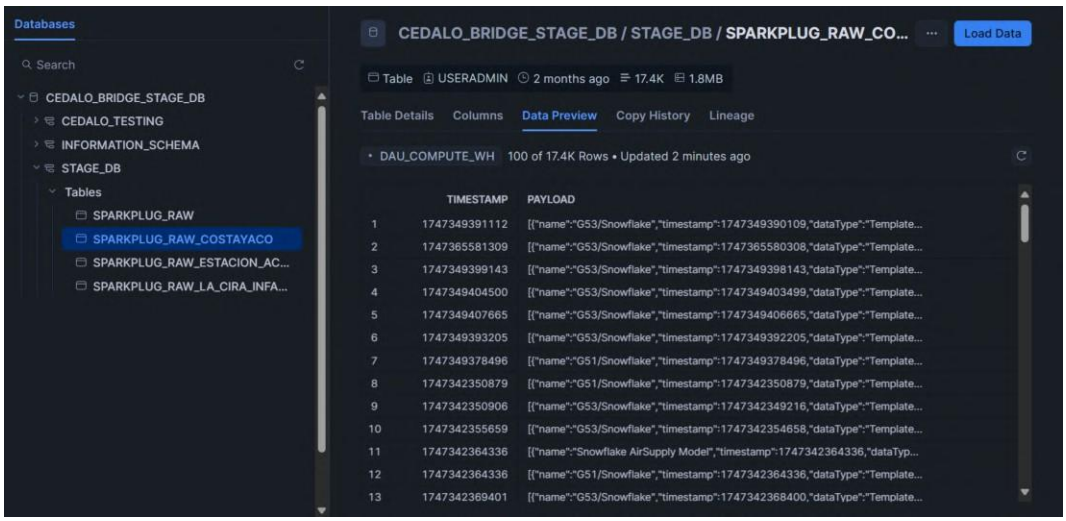
Visualización de datos en tabla objetivo



Adicional a ambos resultados del pipeline, se comprobó la capacidad del plugin para realizar mapeos 1:n y n:1 de tópicos MQTT a tablas en Snowflake (Figura 5).

Figura 5

Visualización de datos en tablas objetivo, con mapeo de múltiples tópicos y múltiples tablas



4: Validación del pipeline

En el contexto de este proyecto de integración, el criterio más importante para la validación del pipeline es la baja o nula pérdida de mensajes. Esta condición es fundamental para preservar la integridad de los datos y asegurar la confiabilidad de los procesos de análisis llevados a cabo en Snowflake. Para evaluar este aspecto, así como para medir la calidad de la solución propuesta como alternativa al módulo de Cirrus Link, se llevó a cabo una comparación directa entre tres días de datos almacenados por el pipeline basado en el broker Pro Mosquitto y los correspondientes registros procesados por la solución de Cirrus Link. Aunque los tiempos de inserción difieren entre ambas plataformas, se parte de la premisa de que los mensajes transmitidos deben contener la misma información.

4.1 Metodología

Debido a que los mensajes presentan diferencias estructurales entre plataformas, se seleccionó una porción del dataset para analizar usando las consultas mostradas en el Apéndice H. Posteriormente, se buscó una estrategia para la comparación hallando elementos en común dentro de la estructura de ambos JSON (consultar Apéndice K, figuras 10 y 11). Con base en esto, se decidió usar los pares clave-valor del arreglo "*metrics*", el cual se nombró como "submétricas" para distinguirlo. Tras desarrollar distintos códigos, se observó que en la mayoría de los casos el módulo de Cirrus Link y el broker Pro Mosquitto podían enviar el mismo contenido empaquetado en un mensaje con distinto *timestamp* (como se muestra en los Apéndice K, figuras 10 y 11). Sin embargo, en algunas ocasiones una métrica se podría encontrar empaquetada en otro mensaje, por lo que la comparación mensaje a mensaje se descartó. En su lugar, se desarrolló el código mostrado en el Apéndice L, que extrae todas las submétricas de cada mensaje en ambos dataset almacenándolas en dos conjuntos (sets) de Python, para posteriormente hacer la diferencia entre ellos, obteniendo las métricas únicas en cada conjunto. El proceso descrito puede consultarse en el repositorio de GitHub https://github.com/Seidenki/Data-Analysis-Pipeline, Rugeles (2025).

4.2 Resultados de comparación de submétricas únicas

Se compararon las submétricas únicas reportadas por Cirrus Link y Cedalo. Los resultados fueron los siguientes:

- Total de submétricas únicas en Cirrus: **871596**
- Total de submétricas únicas en Cedalo: **871465**
- Diferencia (submétricas en Cirrus que no están en Cedalo): **131**

No se identificaron submétricas que fueran enviadas por el broker pero omitidas por el bridge. Para esclarecer la causa de las 131 submétricas faltantes, se realizó una revisión con el fin de determinar si se trataba de un fallo del broker o de un error de configuración en el módulo de transmisión de Ignition. Esta revisión reveló que todas las métricas ausentes correspondían a un único *tag* de prueba llamado *timestamp*, el cual representa la hora en formato Unix epoch (milisegundos desde el 1 de enero de 1970 a las 00:00:00 UTC). Dado que cada mensaje MQTT ya contiene su propio timestamp, este *tag* adicional se considera redundante y no esencial para su almacenamiento en la nube. Finalmente, tras una inspección breve de la configuración del módulo de transmisión en Ignition, no se encontraron errores en la selección de *tags* ni en su configuración. Debido al nulo valor que aporta dicho *tag* y al tiempo que implicaría un proceso de *debugging* más exhaustivo, no se consideró necesario determinar si el error se produce en la comunicación Ignition-Broker o Broker-Snowflake.

Tras haber realizado la validación mencionada, el broker Pro Mosquitto de Cedalo se considera viable para su uso en producción para la conexión con Snowflake. Una vez los datos se encuentran en esta plataforma, la cuestión de transformarlos para su consumo resulta sencilla haciendo uso de Python y SQL. Con esto, se considera completada la implementación del pipeline hacia Snowflake.

5: Configuración Plugin: Google Pub/Sub

Entre las alternativas populares a Snowflake para Data Warehousing está Google Cloud Platform (GCP), una plataforma que ofrece distintos servicios orientados al uso de datos. Entre estos servicios se incluyen visualización para inteligencia de negocios, análisis de datos, almacenamiento, procesamiento e incluso orquestación de pipelines. Entre los servicios principales que podrían reemplazar en primera instancia a Snowflake, se encuentran Google Pub/Sub, un servicio de mensajería asincrónico que sigue el modelo Publish/Subscribe y Google BigQuery, un servicio de almacenamiento para grandes volúmenes de datos empresariales. Una integración inicial con esta plataforma puede hacerse a través del envío de mensajes desde un tópico MQTT a un tópico en Google Pub/Sub, el cual a su vez puede configurarse para redirigir cada mensaje a una base de datos en Google BigQuery.

El broker cuenta con un plugin para la integración con Google Pub/Sub, el cual se puede habilitar desde el archivo mosquitto.conf como se muestra en el Código 2. Estas líneas indican la dirección y nombre de los dos archivos necesarios para la configuración.

```
1 plugin /usr/lib/cedalo_google_pubsub.so
2 plugin_opt_key_file pubsub-key.json
3 plugin_opt_config_file pubsub-config.json
4 persistence_location /mosquitto/data
```

Código 2: Activación del plugin Pub/Sub en el archivo mosquitto.conf

El primer archivo necesario es “pubsub-key.json” el cual se genera desde el centro de administración de cuentas de servicio en la consola de GCP. Este archivo contiene información del proyecto asociado a la cuenta vinculada y sirve para brindar un usuario exclusivo que el broker utiliza para realizar la carga de datos. El segundo archivo “pubsub-config.json” (Apéndice Ñ) incluye la configuración de las acciones (Publish o Pull) y el mapeo de tópicos MQTT a tópicos en Google Pub/Sub. Es importante mencionar que, debido a que el plugin es reciente y se encuentra en etapa de desarrollo, la estructura del JSON utilizada fue provista por el soporte de Cedalo debido a que la documentación usada en el momento no

se encontraba actualizada, por esta razón, la estructura puede variar para futuras consultas. Otro aspecto en desarrollo son los métodos que puede usar el plugin para comunicarse vía API con Google Pub/Sub.

- Publish: Permite la inserción de mensajes en un tópico en Pub/Sub,
- Pull: Permite la extracción de mensajes desde un tópico en Pub/Sub basado en solicitudes desde el cliente
- Push: Permite a Google Pub/Sub enviar solicitudes a un servidor del cliente para entregar los mensajes de manera automática.

Durante la realización del proyecto de integración, se dio prioridad al uso de Publish, pues el objetivo principal es la ingesta a la nube. Sin embargo, el plugin también cuenta con el método Pull como método para la extracción, bajo la consideración de que se encuentra en fase BETA. Con base a lo mencionado previamente, se creó un tópico en Pub/Sub llamado "cedalo_bridge_bigquery" con la opción de "Exportar datos a BigQuery" habilitada (Apéndice M, Figura 12). Para confirmar la correcta configuración del tópico y de la cuenta de servicio se creó un cliente en Python, haciendo uso de la librería google.cloud (Apéndice N, Código 3)

Tras confirmar la llegada de datos se creó una base de datos en BigQuery, con el esquema mostrado en la Figura 6. Para poblarla se usó un código en Python similar al mencionado previamente (Código 4, Apéndice N), esta vez publicando periódicamente mensajes con datos simulando un sensor. Este mensaje se estructuró en formato JSON con el fin de confirmar la carga correcta de datos a BigQuery de acuerdo al esquema establecido.

Los resultados de ambas pruebas se muestran en la Figura 7, con esto se facilita el proceso de debugging al confirmar que todas las configuraciones necesarias dentro de GCP fueron realizadas como lo espera el plugin.

Figura 6

Base de datos en BigQuery

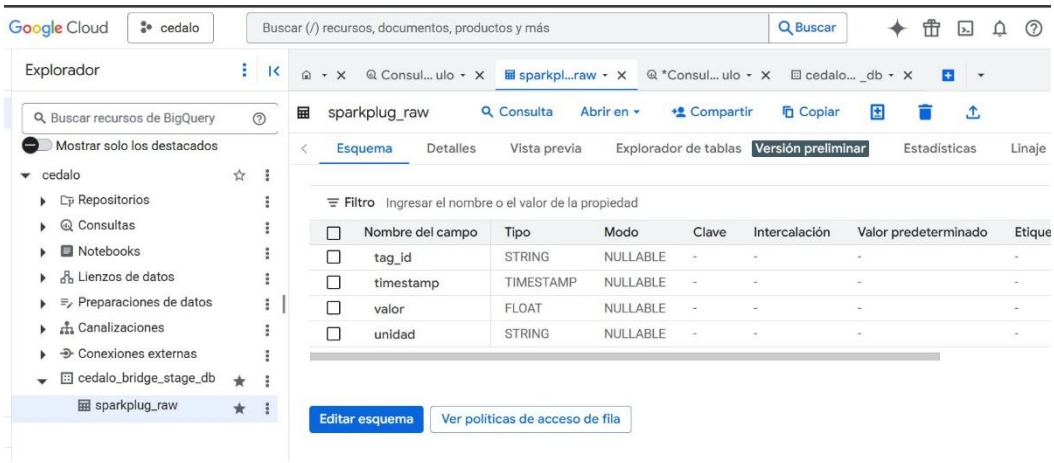
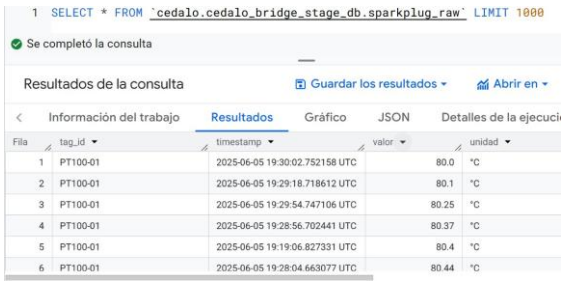


Figura 7

Resultados de envío de mensajes vía cliente Python

(a) *Confirmación configuración cuenta de servicio y tópico*



(b) *Confirmación base de datos y suscripción a BigQuery*

Para confirmar el funcionamiento del plugin se prueba la conexión modificando el código de Cliente Python (Apéndice C) para encajar con el tópico del archivo pubsub-config.json, usando un delay de 7 segundos para publicar cada mensaje. Se confirma la llegada de los mensajes al broker usando la herramienta de suscripción de la plataforma (Figura 8a) y su correcta llegada al tópico (Figura 8b).

Figura 8

Confirmación funcionamiento del plugin

(a) Herramienta de suscripción en Cedalo MQTT Platform

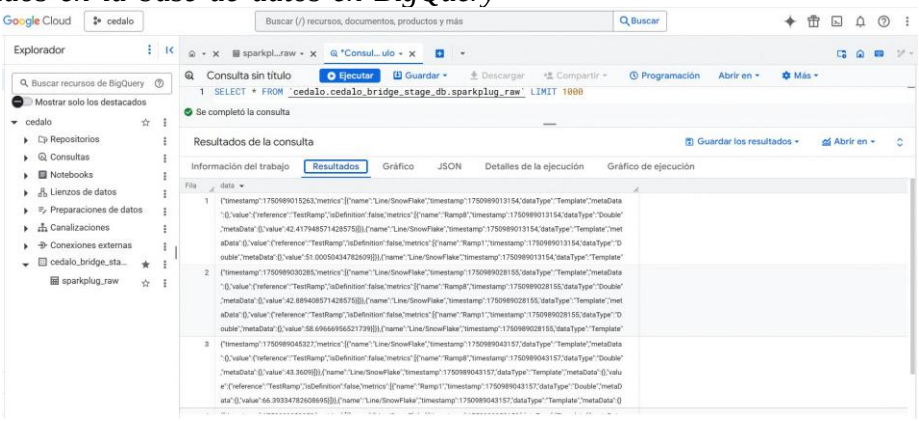
(b) Llegada de mensajes al tópico en Google Pub/Sub

De igual manera, se confirmó lallegada de datos a la tabla en la base de datos cumpliendo con el esquema establecido. El último paso fue la confirmación del envío de datos desde Ignition hasta BigQuery. Con este fin se crea una estructura de tags (Apéndice M, Figura 13) y se configura el transmisor MQTT para enviarlos. En este caso el nombre del tópico de interés es "spBv1.0/Enterprise/DDATA/Site/Area" por lo que se debe modificar el archivo pubsub-config.json de configuración para el plugin teniendo en cuenta esta consideración. Además, debido a que se requiere un esquema válido para realizar la inserción de datos en BigQuery, se decidió crear la tabla "sparkplug_raw" nuevamente con la columna "data" como su único campo, de este modo es posible crear una suscripción sin necesidad de descomponer la estructura del mensaje al marcar la opción "No usar un esquema".

Finalmente, se evidencia la inserción de mensajes en formato string dentro de la tabla objetivo (Figura 9). Dado que la exploración de esta plataforma en la nube no fue un objetivo principal del proyecto, la implementación del plugin se detiene en este punto, sentando las bases para un desarrollo futuro.

Figura 9

Datos insertados en la base de datos en BigQuery



6: Conclusiones

El desarrollo del presente proyecto permitió abordar de manera integral el diseño e implementación de un pipeline de datos desde una arquitectura basada en Unified Namespace (UNS) hasta plataformas de datos en la nube, particularmente Snowflake y, de forma preliminar, Google Cloud Platform, resultando en una solución técnicamente sólida y económicamente viable.

Tras una evaluación comparativa, el broker Pro Mosquitto de Cedalo demostró su eficacia en transmisión, escalabilidad y bajo costo operativo, superando a soluciones como HiveMQ y EMQX en el contexto analizado. Su enfoque ELT y capacidades de integración permitieron una ingesta eficiente, sin modificar drásticamente la arquitectura existente.

El pipeline desarrollado no solo replicó la funcionalidad del módulo comercial ofrecido por *Cirrus Link*, sino que además alcanzó una eficiencia operativa superior al optimizar el uso de recursos computacionales en la nube. Mediante pruebas de configuración del buffer y ajustes en los parámetros de inserción, se obtuvo una reducción significativa del costo anual, estimado en aproximadamente 5,825 USD, sin comprometer la disponibilidad ni la confiabilidad del sistema. Comparado con el costo total del módulo comercial "*IoT Bridge for Snowflake*" , estimado en 40,830 USD/año bajo un escenario de operación 24/7 en mo-

dalidad *micro-batching*, esta solución representa un ahorro anual de 9,610 USD (23.53 %). Adicionalmente, al modificar los parámetros del plugin para operar con intervalos de envío mayores, la reducción alcanza los 30,065 USD/año (73.63 %), lo que proporciona un amplio margen de ajuste según las necesidades específicas de cada cliente.

En cuanto a la confiabilidad del sistema, los resultados obtenidos en la validación muestran que el pipeline implementado es robusto en cuanto a la entrega de datos, presentando una pérdida de apenas 131 submétricas (0.015 %) sobre un total de más de 871 mil, todas correspondientes a un tag redundante.

Este proyecto demuestra que un diseño basado en arquitecturas y protocolos abiertos, como MQTT y el UNS, facilita significativamente la integración tecnológica. En este caso, fue posible conectar dos plataformas en la nube sin necesidad de realizar cambios drásticos en la arquitectura. Esta capacidad multicloud, si bien no era un objetivo principal, representa un valor agregado importante al ofrecer flexibilidad para adaptarse a diversas necesidades de clientes y escenarios de uso futuros.

El desarrollo exitoso del pipeline de datos entre el UNS y la infraestructura en la nube, junto con el cumplimiento de los objetivos específicos planteados, representa un aporte significativo al ecosistema de transformación digital local. Esta solución, diseñada como una alternativa asequible y sostenible para el contexto industrial colombiano, no solo aborda las necesidades actuales de integración de datos, sino que establece los cimientos necesarios para la implementación futura de tecnologías avanzadas de la industria 4.0. Al proporcionar una infraestructura de datos confiable y estandarizada, el trabajo habilita el desarrollo de aplicaciones más sofisticadas como análisis de anomalías, optimización de producción, IA generativa, lo que promueve la adopción tecnológica en la industria a mediano y largo plazo.

Referencias

AWS Editorial Team. (2023). *Solving scalability challenges in industry 4.0 with a cloud provider agnostic edge solution.* Descargado de https://aws.amazon.com/es/blogs/industries/solving-scalability-challenges-in-industry-4-0-with-a-cloud-provider-agnostic-edge-solution/ (Accedido el 21 de mayo de 2025)

Bloching, B., Leutiger, P., Oltmanns, T., Rossbach, C., Schlick, T., Remane, G., . . . Shafranyuk, O. (2015, feb). *The digital transformation of industry - how important is it? who are the winners? what must be done?* https://www.rolandberger.com/en/Insights/Publications/The-digital-transformation-of-industry.html. (Roland Berger Report)

Data Ingestion Patterns in AWS: A Practical Guide. (2025). https://medium.com/%40data.dev.backyard/data-ingestion-patterns-in-aws-a-practical-guide-234897e9de57. ([Accessed 23-07-2025])

Dritsas, E., y Trigka, M. (2025, febrero). A survey on the applications of cloud computing in the industrial internet of things. *Big Data Cogn. Comput., 9*(2), 44.

Grigoriou, N. N., y Fink, A. (2023). Cloud computing: Key to enabling smart production and industry 4.0. En O. Madsen, U. Berger, C. Møller, A. Heidemann Lassen, B. Vejrum Waehrens, y C. Schou (Eds.), *The future of smart production for smes: A methodological and practical approach towards digitalization in smes* (pp. 315–322). Cham: Springer International Publishing. Descargado de https://doi.org/10.1007/978-3-031-15428-7_26 doi: 10.1007/978-3-031-15428-7_26

Gölzer, P., Cato, P., y Amberg, M. (2015). Data processing requirements of industry 4.0 - use cases for big data applications. En *Ecis 2015 research-in-progress papers.* AIS Electronic Library (AISeL).

Hardware Requirements | Cedalo MQTT Platform. (2024). https://docs.cedalo.com/mosquitto/deployment/on-premises/prerequisites/hardware-requirements. ([Accessed 30-06-2025])

HiveMQ. (2024). *Smart Manufacturing Using ISA95, MQTT Sparkplug and the Unified Namespace.* https://www.hivemq.com/resources/smart-manufacturing-using-isa95-mqtt-sparkplug-and-uns/. ([Accessed 30-06-2025])

Huang, X. (2023). *Snowpipe Streaming Demystified — medium.com.* https://medium.com/snowflake/snowpipe-streaming-demystified-e1ee385c6d9c. ([Accessed 30-06-2025])

Ian Colotla, C. K., Damon Bland, y Spindelndreier, D. (s.f.). *Avoiding the Hidden Hazards of Industry 4.0 — bcg.com.* https://www.bcg.com/publications/2018/avoiding-hidden-hazards-industry-4.0. ([Accessed 22-04-2025])

James R. Koelsch, c. w. (2022). *Why the MQTT Protocol is So Popular — automationworld.com.* https://www.automationworld.com/factory/iiot/article/22080946/why-the-mqtt-protocol-is-so-popular. ([Accessed 30-06-2025])

Koziolek, H., Grüner, S., y Rückert, J. (2020). A comparison of MQTT brokers for distributed IoT edge computing. En *Software architecture* (pp. 352–368). Cham: Springer International Publishing.

MQTT at the Edge, which do you choose?? (2025). https://www.portainer.io/blog/mqtt-at-the-edge-which-do-you-choose. ([Accessed 30-06-2025])

Mqtt benchmark suite. (2023). https://github.com/emqx/mqttbs. ([Accessed 30-06-2025])

Muñoz Pinzón, D. S., Valencia Rivero, K. T., y Castro, C. (2024, marzo). Estado actual de la adopción de la industria 4.0 en pymes colombianas: desafíos y oportunidades. *Revista Politécnica*, *20*(39), 99–118. Descargado de http://dx.doi.org/10.33571/rpolitec.v20n39a7 doi: 10.33571/rpolitec.v20n39a7

Péter, , y Werner, S. (2024). *The impact of unified namespace in industry 4.0* (Master's Thesis). Lund University, Sweden.

Rugeles, C. (2025, 5). *Data Analysis Pipeline.* Descargado de https://github.com/Seidenki/Data-Analysis-Pipeline (GitHub repository)

Snowflake Bridge | Cedalo MQTT Platform. (2024). https://docs.cedalo.com/

mosquitto/bridges/snowflake-bridge. ([Accessed 30-06-2025])

Snowflake Documentation. (2025). https://docs.snowflake.com/en/user-guide/intro-key-concepts. ([Accessed 25-05-2025])

Snowflake - Market Share, Competitor Insights in Data Warehousing. (2025). https://6sense.com/tech/data-warehousing/snowflake-market-share#free-plan-signup. ([Accessed 25-05-2025])

Thred and CedaloA Use Case for Industrial Data Optimization. (s.f.). https://iotpicks.com/iot-solutions/thred-and-cedalo-a-use-case-for-industrial-data-optimization. ([Accessed 30-06-2025])

Understanding overall cost | Snowflake Documentation. (s.f.). https://docs.snowflake.com/en/user-guide/cost-understanding-overall. ([Accessed 30-06-2025])

Velásquez Ruiz, J. (2021). *Rol de las tecnologías emergentes "i4.0" en las medianas y grandes empresas en colombia* (Tesis de Maestría en Ingeniería). Universidad de La Sabana, Chía, Colombia.

Warehouse considerations | Snowflake Documentation. (2025). https://docs.snowflake.com/en/user-guide/warehouses-considerations. ([Accessed 30-06-2025])

Apéndice A

Port Mapping

Snippet del archivo YAML para el levantamiento del contenedor, se muestra el mapeo de puertos.

```
ports:
    mosquitto:
        - 1813:1883 # MQTT no TLS
        - 8813:8883 # MQTT with TLS (needs to be configured in mosquitto.conf)
        - 8010:8090 # Websocket no TLS (needs to be configured in mosquitto.conf)
    platform:
        -3010:3000
```

Apéndice B

Default mosquitto.conf

Archivo .conf con la configuración del broker pro mosquitto usada.

```
# Listener for port 1883
listener 1883
# Unauthenticated access
allow_anonymous true
# Persistent Message Store
persistence true
persistence_location /mosquitto/data/
# Broker Control API enabling `$CONTROL/broker/v1`
enable_control_api true
# Dynamic security plugin
plugin /usr/lib/mosquitto_dynamic_security.so
plugin_opt_config_file /mosquitto/data/dynamic-security.json
# Certificate Management plugin
plugin /usr/lib/cedalo_certificate_management.so
# Stream processing plugin
plugin /usr/lib/cedalo_stream_processing.so
plugin_opt_data_dir /mosquitto/data
# Client inspection plugin
plugin /usr/lib/cedalo_inspect.so
plugin_opt_disconnected_clients_ttl 300
# Client control plugin
plugin /usr/lib/cedalo_client_control.so
# Topic Tree plugin
plugin /usr/lib/cedalo_topic_tree.so
```

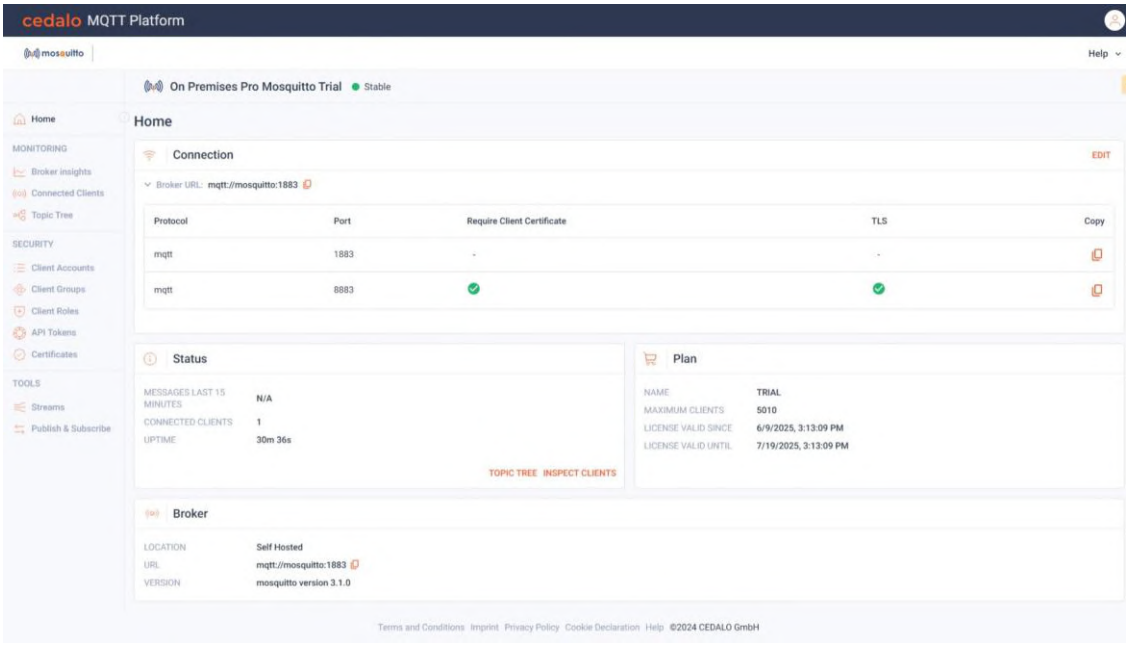
Apéndice C

Python client code

Código usado para probar la conexión desde un cliente publicando datos periódicamente.

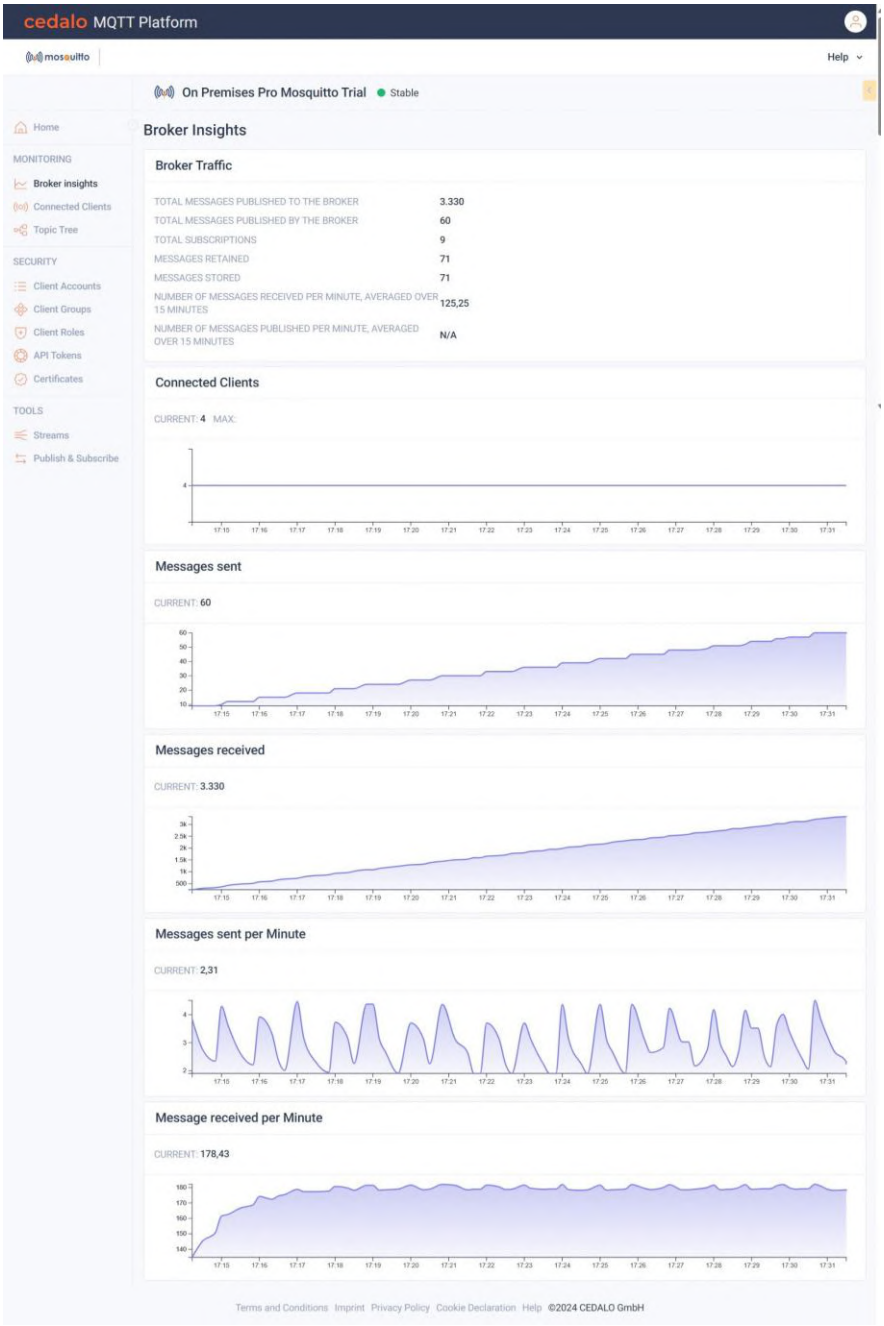
```python
import paho.mqtt.client as mqtt
import json
import datetime
import time
import random
# Configura el broker MQTT
broker_address = "localhost"
port = 1813 #Puerto MQTT (1813 sin TLS )
username = "code"
password = "******"
topic = "Prueba/sensor"# El topic MQTT al que se publicara
def on_connect(client, userdata, flags, rc):
    global connected
    if rc == 0:
        print("Conectado al broker MQTT")
        connected = True
    else:
        print(f"o la conexion, codigo de error: {rc}")
connected = False
# Configura cliente y credenciales
client = mqtt.Client()
client.username_pw_set(username, password)
client.on_connect = on_connect
client.connect(broker_address, port)
client.loop_start()
while not connected:
    print("Esperando conexion al broker...")
    time.sleep(1)
try:
    while True:
        mensaje = {
            "tag_id": "PT100-01",
            "timestamp": datetime.datetime.utcnow().isoformat(),
            "valor": round(random.uniform(80, 90), 2),
            "unidad": "C"
        }
        mensaje_raw = json.dumps(mensaje)
        result = client.publish(topic, mensaje_raw)
        if result.rc == mqtt.MQTT_ERR_SUCCESS:
            print(f"Mensaje enviado: {mensaje_raw}")
        else:
            print(f"Error al enviar mensaje: {result.rc}")
        time.sleep(2)
except KeyboardInterrupt:
    print("Publicacion interrumpida por el usuario.")
finally:
    client.loop_stop()
    client.disconnect()
```

Apéndice D
Interfaz de usuario de la plataforma



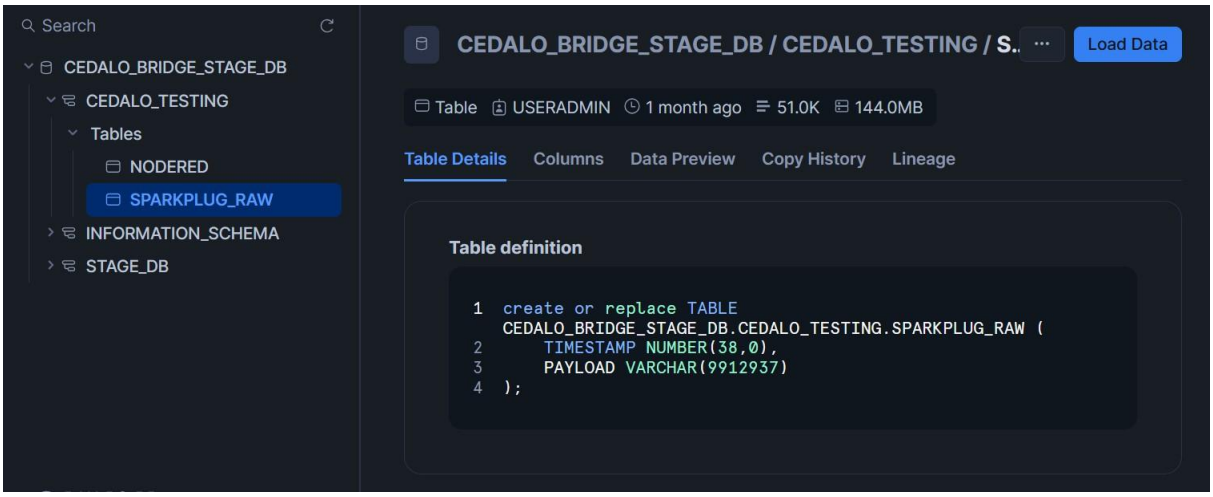
Apéndice E
Métricas desde Broker Insight

Herramienta de visualización de métricas dentro de la plataforma del broker Cedalo Pro Mosquitto, en el se pueden ver estadísticas de los mensajes enviados y recibidos en la implementación final.



Apéndice F
Configuración de la tabla objetivo y base de datos

Se muestra la definición de la tabla estándar usada en Snowflake contenida en el Schema y Base de datos



Apéndice G
Archivo JSON de configuración para el plugin a Snowflake

Archivo .json usado para configurar las conexiones, mapeo de tópicos MQTT a bases de datos y opciones operativas del plugin.

```
{    "version": "1",
     "options": {
          "driverDebug": true},
     "connections": [
          {    "name": "snowflake-dautom",
               "connection": {
                    "accountId": "********",
                    "credentials": {
                         "username": "CEDALO_INGEST",
                         "password": "********"},
                    "lazyConnect": true,
                    "debug": true},
               "options": {
                    "bufferSize": 10000,
                    "batchSize": 1000,
                    "timeoutMs": 4000,
                    "queueMaxSize": 10000,
                    "maxRetries": 15,
                    "retryDelayMs": 1000},
               "topicMappings": [
                    {    "name": "Raw",
                         "schemaMapping": "schema-mapping-Raw",
                         "target": "SPARKPLUG_RAW",
                         "options": {
                              "database": "CEDALO_BRIDGE_STAGE_DB",
                              "schema": "CEDALO_TESTING",
                              "uppercase": true},
                         "mqttTopics": [
                              "+/Copower/#"]
                    }
               ]
          }
     ],
     "schemaMappings": [
          {    "name": "schema-mapping-Raw",
               "mapping": [
                    {
                         "source": "[payload][timestamp]",
                         "target": "TIMESTAMP"
                    },
                    {
                         "source": "[payload][metrics]",
                         "target": "PAYLOAD"
                    }
               ]
          }
     ]
}
```

Apéndice H
Consultas SQL Para la extracción de dataset

Consulta usada para extraer datasets con el fin de usar los códigos desarrollados para validar la llegada de mensajes a Snowflake. Estas consultas siriveron como base, a las cuales se les modificó el limite de mensajes por consulta y las fechas analizadas, generando así multiples dataset.

```
SELECT inserted_at,
msg
FROM CL_BRIDGE_STAGE_DB.STAGE_DB.SPARKPLUG_RAW
WHERE
    GROUP_ID = 'Copower'
    AND CONVERT_TIMEZONE('UTC', 'America/Bogota', TO_TIMESTAMP_LTZ(inserted_at /
        1000))
        BETWEEN '2025-05-26 00:00:00' AND '2025-05-26 23:59:59'
ORDER BY
    inserted_at ASC
LIMIT 10000;

SELECT
    timestamp,
    TRY_PARSE_JSON(payload) AS payload_json
FROM
    CEDALO_BRIDGE_STAGE_DB.STAGE_DB.SPARKPLUG_RAW
WHERE
    CONVERT_TIMEZONE('UTC', 'America/Bogota', TO_TIMESTAMP_LTZ(Timestamp / 1000))
    BETWEEN '2025-05-26 00:00:00' AND '2025-05-26 23:59:59'
ORDER BY
    timestamp ASC
    LIMIT 10000
    ;
```

Apéndice I

Ejemplo mensaje enviado por Cedalo

Ejemplo de un mensaje enviado por el broker de Cedalo.Corresponde a una fila de un archivo dentro de los datasets generados.


```
[
  {
    "dataType": "Template",
    "name": "G53/Snowflake",
    "timestamp": 1748235600668,
    "value": {
      "isDefinition": false,
      "metrics": [
        {
          "dataType": "DateTime",
          "name": "Timestamp",
          "timestamp": 1748235600666,
          "value":  1748235600666
        }
      ],
      "reference":  "Snowflake_Copower_Genset_Model"
    }
  },
  {
    "dataType": "Template",
    "name": "G52/Snowflake",
    "timestamp": 1748235600668,
    "value": {
      "isDefinition": false,
      "metrics": [
        {
          "dataType": "DateTime",
          "name": "Timestamp",
          "timestamp": 1748235600666,
          "value":  1748235600666
        }
      ],
      "reference":  "Snowflake_Copower_Genset_Model"
    }
  },
  {
    "dataType": "Template",
    "name": "G51/Snowflake",
    "timestamp": 1748235600671,
    "value": {
      "isDefinition": false,
      "metrics": [
        {
          "dataType": "DateTime",
          "name": "Timestamp",
          "timestamp": 1748235600669,
          "value":  1748235600669
        }
      ],
      "reference":  "Snowflake_Copower_Genset_Model"
    }
  }
]
```

Apéndice J
Ejemplo mensaje enviado por Cirrus Link

Ejemplo de un mensaje enviado por el módulo *Iot Bridge for Snowflake* de Cirrus Link.Corresponde a una fila de un archivo dentro de los datasets generados.


```
{"metrics": [
    {
      "dataType": "Template",
      "name": "G53/Snowflake",
      "timestamp": 1748235600667,
      "value": {
        "isDefinition": false,
        "metrics": [
          {
            "dataType": "DateTime",
            "name": "Timestamp",
            "timestamp": 1748235600666,
            "value":  1748235600666
          }
        ],
        "reference": "Snowflake_Copower_Genset_Model",
        "version": ""
      }
    },
    {
      "dataType": "Template",
      "name": "G52/Snowflake",
      "timestamp": 1748235600667,
      "value": {
        "isDefinition": false,
        "metrics": [
          {
            "dataType": "DateTime",
            "name": "Timestamp",
            "timestamp": 1748235600666,
            "value":  1748235600666
          }
        ],
        "reference": "Snowflake_Copower_Genset_Model",
        "version": ""
      }
    },
    {
      "dataType": "Template",
      "name": "G51/Snowflake",
      "timestamp": 1748235600671,
      "value": {
        "isDefinition": false,
        "metrics": [
          {
            "dataType": "DateTime",
            "name": "Timestamp",
            "timestamp": 1748235600669,
            "value":  1748235600669
          }
        ],
        "reference": "Snowflake_Copower_Genset_Model",
        "version": ""
      }
    }],
  "seq": 39,
  "timestamp": 1748235601667}
```

Apéndice K
Estructura de los mensajes JSON enviados

Estructura de ambos mensajes describiendo la jerarquía de acuerdo al tipo de dato anidado.

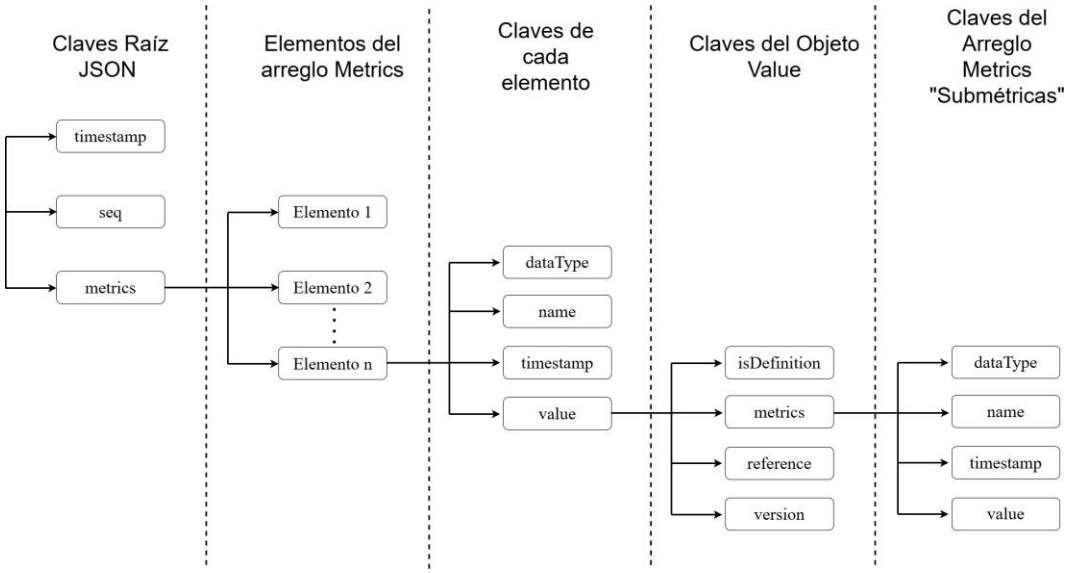


Figura 10
Estructura del JSON para Cirrus Link

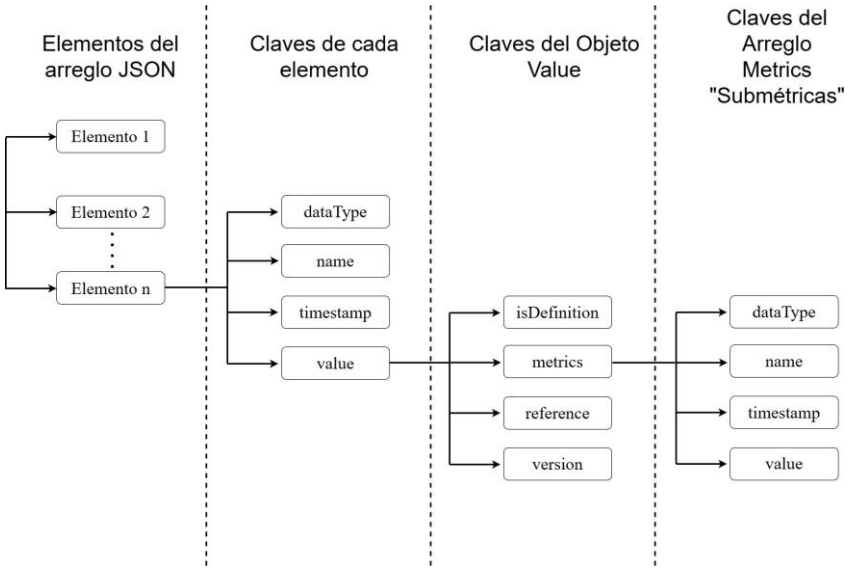


Figura 11
Estructura del JSON para Cedalo

Apéndice L
Código usado para la validación

Código en Python encargado de extraer todas las submétricas de dos datasets correspondientes a 1000 mensajes de Cedalo y 1000 del módulo *IoT Bridge*, agrupandolos en "sets" para operar la diferencia.

```
import pandas as pd
import json
df_cirrus = pd.read_csv("CirrusLink_10000.csv")
df_cedalo = pd.read_csv("Cedalo_10000.csv")
def extract_submetrics_cirrus(row):
    try:
        msg = json.loads(row)
        submetrics = []
        for metric in msg.get("metrics", []):
            for m in metric.get("value", {}).get("metrics", []):
                submetrics.append((m["name"], m["timestamp"], m["value"]))
        return submetrics
    except Exception as e:
        print("Error en fila Cirrus:", e)
        return []
def extract_submetrics_cedalo(row):
    try:
        payload = json.loads(row)
        submetrics = []
        for metric in payload:
            for m in metric.get("value", {}).get("metrics", []):
                submetrics.append((m["name"], m["timestamp"], m["value"]))
        return submetrics
    except Exception as e:
        print("Error en fila Cedalo:", e)
        return []
# Extraer todas las submtricas de todas las filas (lista de listas)
cirrus_submetrics_lists = df_cirrus["MSG"].apply(extract_submetrics_cirrus)
cedalo_submetrics_lists = df_cedalo["PAYLOAD_JSON"].apply(
    extract_submetrics_cedalo)
# Aplanar todas las listas en un solo iterable y convertir a set para unicidad
all_cirrus_submetrics = set([item for sublist in cirrus_submetrics_lists for item
    in sublist])
all_cedalo_submetrics = set([item for sublist in cedalo_submetrics_lists for item
    in sublist])
print(f"Total submetricas unicas en Cirrus: {len(all_cirrus_submetrics)}")
print(f"Total submtricas nicas en Cedalo: {len(all_cedalo_submetrics)}")
missing_submetrics = all_cirrus_submetrics - all_cedalo_submetrics
print(f"Submetricas en Cirrus que NO estan en Cedalo: {len(missing_submetrics)}")
if missing_submetrics:
    print("Ejemplos de submetricas faltantes:")
    for i, sm in enumerate(missing_submetrics):
        print(sm)
        if i >= 1000:
            break
extra_in_cedalo = all_cedalo_submetrics - all_cirrus_submetrics
print(f"Submetricas en Cedalo que NO estan en Cirrus: {len(extra_in_cedalo)}")
if extra_in_cedalo:
    print("Ejemplos de submetricas adicionales en Cedalo:")
    for i, sm in enumerate(extra_in_cedalo):
        print(sm)
        if i >= 9:
            break
```

Apéndice M
Configuraciones en Pub/Sub e Ignition para prueba de conexión a GCP

Creación del tópico desde la consola de Google Pub/Sub, en este se muestra el ID del tema que será usado por el plugin, además de habilitar la ingesta automática en BigQuery

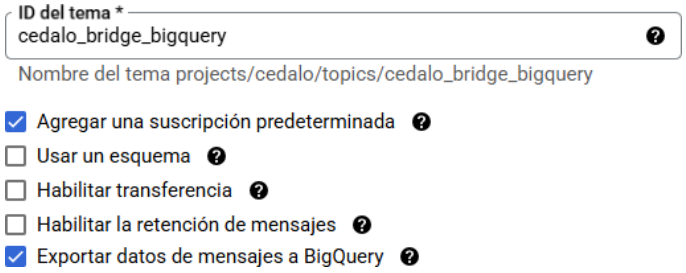


Figura 12
Creación de tópico en Pub/Sub

Estructura de tags que serán enviados al broker desde Ignition, se usó una plantilla de ejemplo que simula la generación de datos en cada tag de manera periódica.

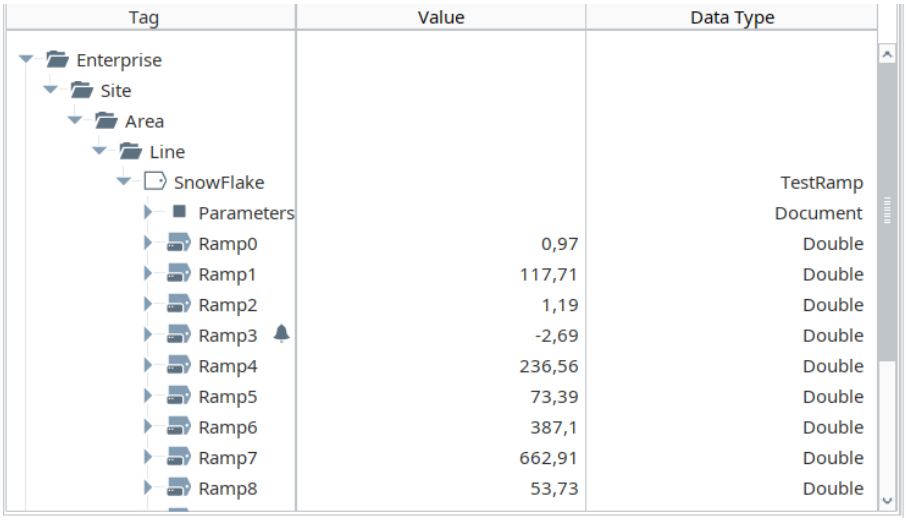


Figura 13
Tags de prueba

Apéndice N
Código Python para testeo de Google Pub/Sub

```python

from google.cloud import pubsub_v1
import os

# === Configuracin ===
ruta_credenciales = "C:/Users/rugel/Downloads/mosquitto-3.0-platform-3.0-docker-sn (1)/mosquitto-3.0-platform-3.0-docker-sn/mosquitto/data/pubsub-key.json"
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = ruta_credenciales

proyecto_id = "cedalo"
nombre_tema = "cedalo_bridge_bigquery"
mensaje = "Hola desde Python con cuenta de servicio"
# === Crear el cliente de publicacin ===
publisher = pubsub_v1.PublisherClient()
ruta_tema = publisher.topic_path(proyecto_id, nombre_tema)
# === Publicar el mensaje ===
mensaje_codificado = mensaje.encode("utf-8")
future = publisher.publish(ruta_tema, mensaje_codificado)
print(f"Mensaje publicado con ID: {future.result()}")
```

Código 3: Código Cliente Python usando cuenta de servicio

```python

from google.cloud import pubsub_v1
import json
import datetime

# Configura tu proyecto y tema
project_id = "cedalo"
topic_id = "cedalo_bridge_bigquery"

# Crea el publicador
publisher = pubsub_v1.PublisherClient()
topic_path = publisher.topic_path(project_id, topic_id)

mensaje = {
    "tag_id": "PT100-01",
    "timestamp": datetime.datetime.utcnow().isoformat(),
    "valor": 85.5,
    "unidad": "C"
}

# Publicar como JSON
mensaje_bytes = json.dumps(mensaje).encode("utf-8")
future = publisher.publish(topic_path, mensaje_bytes)
print(f"Mensaje enviado: {future.result()}")
```

Código 4: Código Cliente Python con mensaje estructurado según esquema en Big Query

Apéndice Ñ
pubsub-config.json

Archivo de configuración .json usado para la prueba de conexión con Google Pub/Sub

```
{
    "connections": [
        {
            "name": "publish",
            "topicMappings": [
                {
                    "mqttTopics": ["Prueba_MQTT"],
                    "targets": ["cedalo_bridge_bigquery"]
                }
            ],
            "options": {
                "maxQueuedMessages": 5,
                "retryDelayMs": 4000,
                "maxRetries": 5
            }
        }
    ]
}
```