ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MÁSTER UNIVERSITARIO

EN BIG DATA

TRABAJO FIN DE MÁSTER

OPTIMIZACIÓN DE ARQUITECTURA DE DATOS PARA MOVILIDAD URBANA EN ENTORNOS BIG DATA: INTEGRACIÓN DE LAKEHOUSE Y ANÁLISIS DE RENDIMIENTO DE FORMATO DE ALMACENAMIENTO

Autor: José Arturo Huchim Vela

Director: Mary Liliana Falcón López

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Optimización de arquitectura de datos para movilidad urbana en entornos big data: integración de lakehouse y análisis de rendimiento de formato de almacenamiento

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2024/2025 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: José Arturo Huchim Vela Fecha: ……/ ……/ ……

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Mary Liliana Falcón López Fecha: ……/ ……/ ……

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MÁSTER UNIVERSITARIO

EN BIG DATA

TRABAJO FIN DE MÁSTER

OPTIMIZACIÓN DE ARQUITECTURA DE DATOS PARA MOVILIDAD URBANA EN ENTORNOS BIG DATA: INTEGRACIÓN DE LAKEHOUSE Y ANÁLISIS DE RENDIMIENTO DE FORMATO DE ALMACENAMIENTO

Autor: José Arturo Huchim Vela

Director: Mary Liliana Falcón López

Madrid

Agradecimientos

A mi directora Mary Liliana Falcón López por su apoyo y soporte durante la realización de este proyecto.
A Carlos por la oportunidad de realizar este trabajo.
A Ana y mis padres por su soporte esencial durante este tiempo de elaboración del proyecto
A la EMT y AEMET, por proporcionar los datos necesarios para llevar a cabo este proyecto.

OPTIMIZACIÓN DE ARQUITECTURA DE DATOS PARA MOVILIDAD URBANA EN ENTORNOS BIG DATA: INTEGRACIÓN DE LAKEHOUSE Y ANÁLISIS DE RENDIMIENTO DE FORMATO DE ALMACENAMIENTO.

Autor: Huchim Vela, José Arturo.
Director: Falcón López, Mary Liliana.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas)

RESUMEN DEL PROYECTO

Este proyecto desarrolla una arquitectura de datos tipo Lakehouse en Cloud para integrar información abierta sobre los transportes urbanos superficiales en Madrid. Se automatizan procesos de ingesta y transformación hasta dejar los datos listos para su consumo. Finalmente, se evalúa el impacto de formatos de almancenamiento.

Palabras clave: Arquitectura Data Lakehouse, Cloud, Big Data, EMT, BiciMAD, Sistema de Transporte Público.

1. Introducción

El crecimiento de las ciudades y la generación de altos volúmenes de datos ha impulsado el desarrollo de soluciones analíticas para su integración. Y precisamente de los sistemas de transporte urbanos superficiales, para entender y mejorar la movilidad urbana. Sin embargo, se cuenta con un amplio número de integraciones de fuentes de datos y no todas presentan datos listos para utilizar. Este proyecto responde a esa necesidad mediante el diseño de una arquitectura moderna de tipo Data Lakehouse integrando tanto los beneficios del Data Lake como del Data Warehouse.

2. Definición del proyecto

Construir una arquitectura tipo Data Lakehouse en la nube, capaz de integrar, transformar, almacenar datos abiertos relacionados con el sistema de transporte superficial urbano (bicicletas públicas del sistema BiciMAD y autobuses de la EMT).Asi mismo, se considera agregar para más dinamismo datos de condiciones climáticas provenientes de la AEMET. Finalmente, se evaluará los formatos de almacenamiento con el fin de analizar su impacto en rendimiento, coste y compresión.

3. Descripción del modelo/sistema/herramienta

La arquitectura diseñada está pensada en Google Cloud Platform, un servicio de nube especializado en soluciones de Datos. La arquitectura está compuesta por lo siguiente:

- Almacenamiento por capas:
  - Landing: Para almacenamiento de datos originales
  - Capa Raw y Curated: Para almacenamiento estructurado y consumo analítico.
- Integraciones y Pipelines: Automatizadas para el almacenamiento de los datos en las diferentes capas de Datos y de manera batch para históricos de datos.

- Gobierno de Datos: Con un esquema básico de gobierno, incluyendo etiquetado, control de accesos de usuarios y control de secretos.

Figura 1. Arquitectura técnica del sistema desarrollado. Fuente: Elaboración propia

4. Resultados

- El sistema desarrollado integró con éxito diversas fuentes de datos abiertos sobre el transporte público urbano como lo son los viajes y estatus de estaciones (por lotes) y estatus de estaciones por hora perteneciente a BiciMAD, También, los datos operativos de autobuses EMT y condiciones climáticas de AEMET. Estas fuentes fueron unificadas mediante una arquitectura en capas (landing, raw, curated) implementada en Cloud, con automatización de procesos EL y ELT mediante extracciones con imágenes de Docker, servicios de Cloud para la ingesta y procedimientos SQL.
- Se aplicaron con éxito estrategia de gobierno basada en catálogo de datos, gestión de usuarios y secretos.
- Se evaluó con éxito el rendimiento de formatos de almacenamiento como CSV, Parquet y Avro
- Esto nos dio como resultado una arquitectura moderna y escalable capaz de poder aplicarse a distintos Casos de Uso. Por otra parte, se concluye que Parquet ofrece mayor eficiencia en compresión, coste y tiempo de procesamiento, consolidándose como el formato más adecuado para este tipo de entornos.

5. Conclusiones

El proyecto ha cumplido satisfactoriamente sus objetivos constituyendo esta arquitectura una base sólida y acorde para aplicación en otros escenarios y para futuras extensiones y/o actualizaciones.

DATA ARCHITECTURE OPTIMIZATION FOR URBAN MOBILITY IN BIG DATA ENVIRONMENTS: LAKEHOUSE INTEGRATION AND STORAGE FORMAT PERFORMANCE ANALYSIS.

Author: Huchim Vela, José Arturo.
Supervisor: Falcón López, Mary Liliana.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This project develops a Data Lakehouse style data architecture in Cloud to integrate open data on urban surface transportation in Madrid. Ingestion and transformation processes are automated until the data is ready for consumption. Finally, the impact of storage formats is evaluated.

Keywords: Data Lakehouse Architecture, Cloud, Big Data, EMT, BiciMAD, Public Transportation System.

1. Introducción

The growth of cities and the generation of high volumes of data has driven the development of analytical solutions for their integration, and specifically for this project, for urban surface transportation systems, to understand and improve urban mobility. However, there are a large number of data source integrations, and not all of them offer data ready to use. This project responds to this need by designing a modern Data Lakehouse architecture that integrates both the benefits of a Data Lake and a Data Warehouse

2. Project Definition

Construir una arquitectura tipo Data Lakehouse en la nube, capaz de integrar, transformar, almacenar datos abiertos relacionados con el sistema de transporte superficial urbano (bicicletas públicas del sistema BiciMAD y autobuses de la EMT). Asi mismo, se considera agregar para más dinamismo datos de condiciones climáticas provenientes de la AEMET. Finalmente, se evaluará los formatos de almacenamiento con el fin de analizar su impacto en rendimiento, coste y compresión.

Build a Data Lakehouse architecture in cloud, capable of integrating, transforming, and storing open data related to the urban surface transportation system (BiciMAD public bicycles and EMT buses). Additionally, the addition of weather data from the AEMET. Finally, storage formats will be evaluated to analyze their impact on performance, cost, and compression.

3. Description about model/system/tool

The designed architecture is built on Google Cloud Platform, a cloud service specialized in data solutions. The architecture is composed of the following:

- Layered Storage:
    - Landing: For storing original data

- Raw and Curated Layers: For structured storage and analytical consumption.

- Integrations and Pipelines: Automated for storing data in the different data layers and in batches for historical data.
- Data Governance: With a basic governance framework, including tagging, user access control, and secret control.

Figura 2. Technical architecture of the developed system. Source: Own elaboration

4. Results

- The developed system successfully integrated many open data sources on urban public transport, such as trips and station status (batch) and hourly station status from BiciMAD, as well as operational data from EMT buses and weather conditions from AEMET. These sources were unified using a layered architecture (landing, raw, curated) implemented in the cloud, with automation of ELT and ELT processes using Docker image extractions, cloud services for ingestion, and SQL procedures.
- A governance strategy based on a data catalog, user management, and secrets was successfully applied.
- The performance of storage formats such as CSV, Parquet, and Avro was successfully evaluated.
- This resulted in a modern and scalable architecture capable of being applied to different use cases. Furthermore, it was concluded that Parquet offers greater efficiency in compression, cost, and processing time, consolidating itself as the most suitable format for this type of environment.

5. Conclusions

The project has successfully met its objectives, providing this architecture with a solid and suitable foundation for application in other scenarios and for future extensions and/or updates.

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

Actualmente, cada día, las ciudades aplican más tecnología llegando a un contexto de ciudades inteligentes donde la eficiente movilidad urbana se ha convertido en un reto clave para instituciones públicas, operadores de transporte y ciudadanos. La disponibilidad de datos abiertos provenientes de sistemas de transporte superficial público urbano como BiciMAD, la Empresa Municipal de Transportes (EMT) y otros datos como condiciones climáticas, ofrece una oportunidad para mejorar la planificación y optimización de los servicios de movilidad. Sin embargo, toda esta cantidad de datos que al final, se convertirán en información que será debidamente activada, plantea desafíos técnicos en cuanto a integración, almacenamiento, procesamiento, gobierno y análisis en diversos entornos distribuidos o cloud. Lo anterior excede la capacidad de procesamiento de una computadora tradicional, lo que hace mirar hacia otro tipo de tecnologías con enfoque en arquitecturas en Big Data.

Este trabajo tiene como objetivo diseñar e implementar una arquitectura Lakehouse para integrar y analizar datos abiertos de movilidad urbana en la ciudad de Madrid, desplegada sobre una infraestructura moderna en la nube. La solución contempla la ingesta automatizada, transformación y gobierno de datos, siguiendo principios de ingeniería de datos y buenas prácticas del proveedor. Además, se realiza un análisis del impacto que tienen distintos formatos de almacenamiento tales como CSV, Avro y Parquet debido a que las diversas fuentes de datos, proveen los mismos en diversos formatos con el fin de identificar configuraciones óptimas que equilibren velocidad, compresión y costes.

El proyecto busca aportar una propuesta técnica y analítica que pueda darse en sistemas urbanos y con esto, hacerlos más inteligentes y sostenibles con el fin de obtener mejores resultados de movilidad vial.

1.1 MOTIVACIÓN DEL PROYECTO

El crecimiento de las ciudades y la era digital por las ciudades inteligentes o "Smart cities" ha hecho que la movilidad urbana sea una de las prioridades para las ciudades con el fin de proveer sitios eficientes y pensando siempre en el ciudadano, pero con estrategia. Bajo este contexto, los datos abiertos provenientes de sistemas de transporte superficial público como BiciMAD, la Empresa Municipal de Transportes de Madrid (EMT), y fuentes complementarias como datos meteorológicos representa una solución para optimizar la gestión del transporte urbano.

Sin embargo, el tener todos estos datos y, por consiguiente, riqueza de información, traen consigo desafíos tecnológicos. Hoy en día, la generación de datos crece a pasos agigantados por lo que debemos contar son soluciones de almacenamiento escalables y flexibles, pero no por eso, con alto coste. Por otro lado, tenemos diversos tipos de fuentes como API REST, Web, Scraping ente otros y, por consiguiente, diversos formatos de datos como archivos en texto plano como CSV/JSON o archivos comprimidos como parquet o avro y finalmente en otra línea, se encuentra el procesamiento de datos que puede ser batch o streaming, siendo este último un reto adicional al poder trabajar con datos en tiempo real, que sin embargo, queda de momento fuera del alcance de este proyecto.

Un sistema tradicional resulta hoy en día insuficiente ante la creciente demanda lo que ha impulsado la adopción de nuevas propuestas como el paradigma Lakehouse, que combina las ventajas del Data Lake y del Data Warehouse para integrar almacenamiento y analítica en un solo entorno.

La motivación de este proyecto se encuentra en abordar el reto desde la perspectiva de ingeniería de datos, diseñando una solución técnica, robusta y eficiente que permita gestionar volúmenes grandes de datos y hacer un consumo final de ellos y que, con esto, pueda reflejarse en la contribución de la mejora de la movilidad urbana y la calidad de vida en las ciudades.

1.2 SISTEMA DE TRANSPORTE PÚBLICO DE MADRID

El sistema de transporte público en Madrid está conformado por entidades públicas y privadas que da lugar a diversas formas de transporte, las cuales han ido evolucionando e incrementándose con el paso del tiempo [2].

Actualmente, se disponen los siguientes servicios [1]:

- Metro (**¡Error! No se encuentra el origen de la referencia.**)
- Metro Ligero y Tranvía
- Autobuses urbanos de Madrid EMT, de la Comunidad e interurbanos
- Cercanías
- Metro Ligero y tranvía
- Transporte privado: Incluye taxis y servicios gestionados mediante plataformas digitales.
- BiciMad

Figura 1. Metro de Madrid [4]

Como se mencionó anteriormente en el punto 1.1, el siguiente trabajo estará enfocado en el sistema de transporte superficial público que abarca los autobuses urbanos de la EMT y el sistema de bicicletas BiciMAD.

1.2.1 AUTOBUSES URBANOS

La Empresa Municipal de Transportes de Madrid (EMT) es una entidad pública propiedad del Ayuntamiento. Su función principal es la gestión y operación de todas las líneas de

autobuses urbanos (Figura 2) de la ciudad [2]. Actualmente, cuenta con una flota de 1.903 vehículos que tiene una edad media de 6,6 años. Además, 791 autobuses funcionan con gas natural comprimido (GNC) y 20 son eléctricos. Los restantes 1.092 funcionan con biodiesel [3]

Figura 2. Autobús de la EMT [5]

1.2.2 BICIMAD

BiciMAD es el sistema de bicicleta pública (Figura 3) o compartida de la ciudad de Madrid. Este sistema pone a disposición del usuario realizar trayectos o desplazamientos de un punto a otro mediante un servicio de bicicletas eléctricas [2].

Figura 3. Bicicletas pertenecientes al sistema público BiciMAD [6]

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

A continuación, se describe en primera instancia los diferentes tipos de arquitecturas y sus diferencias entre cada una. Posteriormente, una breve descripción de los formatos de almacenamiento a evaluar.

2.1 ARQUITECTURAS DE DATOS

2.1.1 DATA LAKE

Es un repositorio centralizado diseñado para almacenar, procesar y proteger grandes cantidades de datos estructurados, semiestructurados o no estructurados en su formato nativo. Los datos se procesan, limpian y transforman durante el análisis con el propósito de permitir velocidades de carga más rápidas. Sin embargo, requieren de experiencia en ciencia de datos, y, si no se mantienen de forma apropiada, su calidad puede deteriorarse con el tiempo. Además, los data lakes dificultan la obtención de consultas en tiempo real, ya que los datos no se procesan, por lo que aún deben limpiarse, procesarse, transferirse e integrarse para poder usarlos [23].

2.1.2 DATA WAREHOUSE

También llamados almacenes de datos, son una plataforma que proporcionan acceso rápido a los datos y generan informes y estadísticas para tomar decisiones. Todos los datos deben pasar por la fase ETL. Esto trae consigo formatos y/o esquemas determinados sobre datos estructurados y semiestructurados de varias fuentes de datos. Incluyen también una base de datos analítica, admiten análisis ad-hoc e informes personalizados. Sin embargo, se limita la flexibilidad de acceso a todos los datos y trae consigo costes adicionales por el tratamiento de los mismos [24].

2.1.3 LAKEHOUSE

Es una arquitectura de datos moderna que combina un Data Lake y un Data Warehouse en una plataforma única con los beneficios de ambas partes: Data Lakes (grandes repositorios de datos sin procesar en su forma original) y Data Warehouses (conjuntos organizados de datos estructurados). Los Data Lakehouses capturan todos los datos estructurados, no estructurados y semiestructurados y los almacena en un almacenamiento de bajo coste, a la vez que ofrece la capacidad de todos los usuarios de organizar y explorar los datos según sus necesidades [25].

2.2 FORMATOS DE ALMACENAMIENTO

2.2.1 AVRO

Es un sistema de serialización de datos, que ofrece estructuras de datos, formato binario compacto y rápido con integración sencilla con lenguajes dinámicos [27].

2.2.2 PARQUET

un formato de archivo de datos de código abierto, orientado a columnas, diseñado para el almacenamiento y la recuperación eficientes de datos. Ofrece esquemas de compresión y codificación de alto rendimiento para gestionar datos complejos en masa. Es compatible con múltiples lenguajes de programación [26].

2.2.3 CSV

Es un formato simple basado en texto donde los valores están separados por comas.

2.3 LIBRERÍAS

2.3.1 REQUESTS

Es una biblioteca HTTP para Python que permite realizar peticiones HTTP de manera sencilla. Facilita el consumo de APIs RESTful y así como la implementación de encabezados

(headers), parámetros y envío de información (body). Es una herramienta fundamental a la hora de integrar fuentes externas provenientes de APIs dentro de canalizaciones de datos [7]

2.3.2 PANDAS

Es una librería de Python esencial para el análisis y manipulación de datos estructurados. El manejo de los datos, se efectúa en objetos DataFrame que permiten trabajar de forma sencilla y eficiente datos tabulares. Se utiliza ampliamente para limpieza, transformación y/o análisis de datos dentro de las soluciones de Big Data [8]

2.4 GOOGLE CLOUD PLATFORM

También llamado GCP, es la plataforma de servicios en la nube (cloud) de Google que ofrece diversos servicios de infraestructura, análisis, inteligencia artificial, machine learning, bases de datos, redes, streaming [16], entre otros divididos en servicios *serverless* (si servidor, donde no hay que gestionar, configurar, actualizar o disponibilidad infraestructura) o servicios *fully manage*d (importa cómo y dónde se ejecuta mi servicio porque tengo un tipo específico de servicio que requiere más control sobre los recursos) [18].

GCP permite desarrollar soluciones modernas de Análisis de Datos, Machine Learning, Ingeniería de Datos, Almacenamiento y Automatización con seguridad integrada. Entre alguna de las populares herramientas, se encuentran BigQuery, Cloud Storage, Vertex AI [16]

2.4.1 BIGQUERY

Herramienta estrella de GCP y Data Warehouse serverless que permite consultar grandes volúmenes de datos en segundos utilizando SQL estándar. Es altamente escalable. Admite integración con diversas herramientas de BI, además, permite la integración sencilla de fuentes de datos de terceros o del ambiente Google mediante servicios de transferencias de datos definidos. Por otro lado, permite la construcción de modelos de aprendizaje automático utilizando SQL tradicional, calendarización de consultas SQL entre otros servicios [10].

2.4.2 CLOUD STORAGE

También llamado GCS, es un servicio de almacenamiento de objetos que permite almacenar archivos grandes y de cualquier formato. Permite una fácil integración con otros servicios de GCP [11].

2.4.3 DATAFLOW

Servicio gestionado para el procesamiento de datos por lotes y en tiempo real. Utiliza Apache Beam por debajo. Es ideal para tareas de ETL, transformaciones de datos y flujos de datos en tiempo real [28].

2.4.4 CLOUD RUN

Permite desplegar aplicaciones y microservicios en contenedores sobre una infraestructura sin servidor. Escala automáticamente según la demanda y es compatible con cualquier lenguaje de programación. Es útil para ejecutar trabajos por demanda, exponer servicios web o realizar procesamiento de datos que requiere lógica de negocio personalizada [13].

2.4.5 SECRET MANAGER

Servicio seguro para almacenar y gestionar secretos que pueden ser contraseñas, tokens, entre otros. Limita el acceso lo cual permite garantizar la seguridad y gobernanza las soluciones implementadas. Requiere siempre de una autenticación para su acceso [12].

2.4.6 ARTIFACT REGISTRY

Servicio que gestiona artefactos dentro de Google Cloud que van desde imágenes Docker, bibliotecas Maven, entre otros. Se puede llamar como un repositorio de artefactos que implementa versionados. Se relaciona con Cloud Run en el despliegue de contenedores para su ejecución [14].

2.4.7 CLOUD IAM

Servicio que mantiene el gobierno total del acceso a los recursos de un proyecto en Google Cloud mediante la asignación de roles y políticas de permisos. Garantiza que los principals tengan acceso únicamente con los privilegios mínimos lo cual es una de las buenas prácticas de la plataforma para la seguridad en relación a accesos sobre el proyecto [15].

2.4.7.1 Cuentas de Servicio

Es un tipo especial de cuenta en Google Cloud que pertenece a una aplicación, servicio o recurso (no a un usuario humano) y que se utiliza para autenticarse y autorizar acciones dentro de GCP. Es esencial cuando un componente como un job de Cloud Run, necesita acceder a otros servicios como BigQuery, Cloud Storage o Secret Manager. Cada cuenta de servicio puede tener uno o varios roles IAM asignados, lo que le otorga permisos específicos dentro del proyecto [36].

2.4.8 DATA CATALOG

Es un servicio de metadatos que permite descubrir, administrar y buscar datos en Google Cloud. Ofrece un catálogo centralizado para etiquetar, clasificar y organizar datasets, tablas, vistas [29].

2.5 DOCKER

Plataforma que nos permite empaquetar y ejecutar aplicaciones dentro de contenedores el cual contiene todo lo necesario para que una aplicación se ejecute correctamente, desde el código requerido, librerías, versiones, dependencias y configuraciones. En GCP mediante Cloud Run, facilita despliegues escalables [17]. Para la implementación de este proyecto, se requiere tener instalado docker en el ordenador, tal como se muestra en la Figura 4.

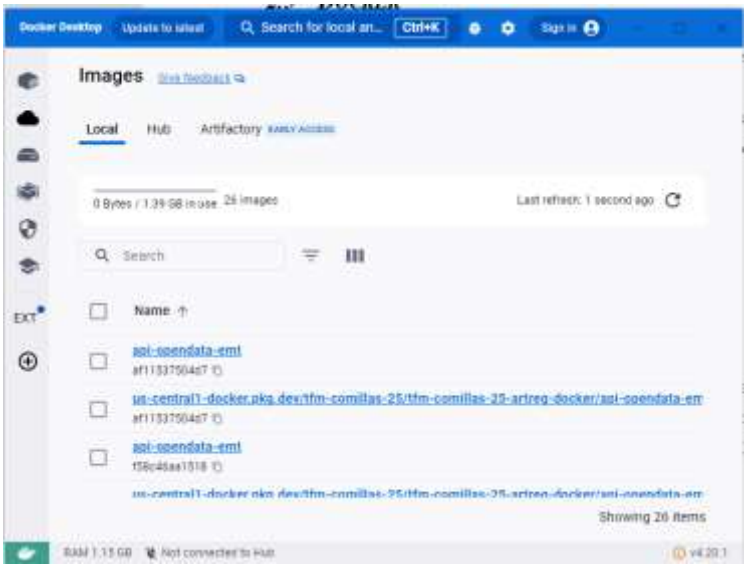


Figura 4. Terminal de Docker en escritorio. Fuente: Elaboración propia

2.6 CAPAS DE DATOS

2.6.1 RAW (BRUTA O CRUDA)

Es la primera capa donde se almacenan los datos tal como llegan desde las fuentes. Generalmente, no incluyen transformaciones ni limpieza. Tiene como propósito preservar los datos originales para auditoría o validaciones en el futuro. Se incluyen todos los formatos de datos.

2.6.2 CURATED (CURADA O REFINADA)

Es la capa que contiene los datos limpios, transformados y validados que ya se encuentran listos para su consumo. Por lo general, adoptan una estructura tabular.

Capítulo 3. ESTADO DE LA CUESTIÓN

En los últimos años, se ha impulsado soluciones comerciales que aprovechan datos urbanos para mejorar la movilidad, la planificación del transporte y jornada de los usuarios. Estas plataformas utilizan datos generados por servicios de transporte, sensores urbanos o aplicaciones móviles, y ofrecen funcionalidades como monitoreo, análisis de patrones de tráfico o visualización de rutas. Sin embargo, si bien estas iniciativas han incluido la explotación de datos urbanos, no representan una solución integral de diversas fuentes de datos abiertas.

A nivel comercial, se ha investigado de una plataforma que toma datos urbanos abiertos que es Transport for London (TfL), que proporciona acceso a datos en tiempo real o históricos, sin embargo, esta solución está centrada en la planificación de viajes solamente tomando en cuenta diversos medios de transporte [19].

También está Strava Metro que pone a disposición datos sobre rutas utilizadas por ciclistas y peatones que usan la app de Strava. Esta plataforma evalúa infraestructura ciclista, rediseñar cruces y planificar zonas peatonales. Su funcionamiento depende de sus usuarios por medio de su aplicación [20].

Otra plataforma es Keolis, una compañía de global de transporte público, especializado en tranvías y metros automáticos que ha desarrollado su propia plataforma de analítica de movilidad para optimizar rutas, horarios, pero es de uso interno [21].

Aunque existen similares productos en el mercado, no se ha identificado alguna solución completa que integre múltiples fuentes de datos abiertos urbanos en una arquitectura de datos optimizada y moderna a bajo coste en plataformas cloud modernas. Por esta razón se justifica y se da valor a la propuesta de este proyecto. Como punto final, la mayoría de las soluciones se centran en la visualización, el acceso en tiempo real o el análisis agregado de una sola fuente.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

En función del análisis realizado en el capítulo anterior, si bien existen diversas soluciones que explotan datos de movilidad urbana, por ejemplo: Transport for London, Strava Metro, están centradas en una única fuente de datos y no ofrecen una solución integral de diversas fuentes.

Hasta la elaboración de este proyecto, no se ha identificado en el mercado una propuesta que implemente una arquitectura integral de tipo Lakehouse orientado a datos abiertos de movilidad urbana.

Revisando el mercado, la creciente ola de ciudades inteligentes son una oportunidad a revisar puesto que toda la industria relacionado a este fin, pueden obtener arquitectura flexible que integre diversas fuentes y realice una trazabilidad de los datos. Este proyecto busca cubrir esa parte con dicha solución integral.

4.2 ALCANCE

A lo largo de este, proyecto se han identificado ciertas acotaciones que definen su alcance.

Sobre los datos a utilizar, serán solamente los provenientes de los medios de transporte públicos superficiales: Sistema de bicicletas públicas de Madrid, BiciMAD y el sistema de autobuses de la EMT.

Sobre el proyecto en Google Cloud:

- Se elaborará la arquitectura en un proyecto sin organización, a partir de un usuario declarado como *owner*.

Sobre las fuentes de datos, los datos que se trabajarán correspondientes a:

- BiciMAD:
    - Los viajes en bicicletas hechos por los usuarios y los datos van de enero 2020 a febrero 2023 disponibles en archivos en batch o por lotes.
    - Los estatus de las estaciones de bicicletas con datos que van de 2020 a 2022 disponibles en archivos en batch o por lotes.
    - La disponibilidad de estaciones del día en curso. Consumo de API.
- EMT, donde el consumo de datos es vía API:
    - La Información de líneas de autobuses por día
    - Las Rutas de líneas
    - La Información de las paradas
    - Los Viajes efectuados por día
- AEMET, donde el consumo de datos es vía API:
    - Los Datos de predicción diaria
    - El Tiempo actual
    - Las Estaciones

Sobre el Data Lakehouse, se incluirán los siguientes servicios:

- Almacenamiento:
    - Storage
    - BigQuery
- Gobierno
    - Cloud IAM
    - Data Catalog
- Procesamiento:
    - Cloud Run
    - Dataflow
- Operaciones
    - Logging
    - Secret Manager

Sobre las capas de datos, se implementarán:

- Cruda (raw)
- Curated (curada)

Sobre las cuentas de IAM:

- Se dispondrá de una cuenta de servicio en específico para la gestión de los Jobs de Cloud Run.
- Se dispondrá de una cuenta de servicio en específico para desarrolladores que cuente con los roles de acceso mínimo para los servicios planteados en esta arquitectura.
- Un solo usuario como owner.

4.3 OBJETIVOS

Diseñar y construir una arquitectura de datos tipo Lakehouse para la integración, procesamiento y análisis de datos abiertos relacionados con la movilidad urbana en Madrid, implementada en cloud. Por otro lado, se evaluará el impacto de distintos formatos de almacenamiento (CSV, Avro y Parquet) sobre el rendimiento de consultas analíticas y que, con el uso de buenas prácticas, se logre eficientar las mismas equilibrando coste y velocidad.

4.3.1 OBJETIVOS ESPECÍFICOS / ENTREGABLES

- Identificar las fuentes de datos abiertos de movilidad urbana que sean relevantes.
- Diseñar e implementar una arquitectura de almacenamiento por capas (raw, curated)
- Implementar canalizaciones de datos automáticas aplicando transformaciones.
- Establecer un gobierno de datos.
- Evaluar el rendimiento, coste y eficiencia de distintos formatos de almacenamiento (CSV, Avro y Parquet)

4.4 METODOLOGÍA

Scrum es un marco ágil y flexible para trabajar en entornos inestables o entornos con una alta variabilidad como suelen ser los proyectos de TI. Se aplica en escenarios donde el cliente conoce la visión de su producto, pero no puede detallar cómo será el producto final. Bajo Scrum se va iterando y dirigiendo el proyecto por aquellas funcionalidades donde el valor de negocio es máximo en cada momento. Lo que se fija en un proyecto bajo Scrum son los costes, y estos a su vez, fijan el tiempo, ya que un sprint es valorable. En proyectos el cliente lo que contrata es un número de sprints [22].

La Figura 5 es una imagen del triángulo de hierro, que muestra metodología tradicional (cascada) versus ágil y que nos ayuda a comprender mejor lo anterior.

Figura 5. El triángulo de hierro [22]

En las metodologías tradicionales lo único que es fijo es el alcance, y aunque se establezcan el tiempo y el precio en el contrato, el tiempo siempre es estimado y por tanto los costes también lo son [22].

En los proyectos ágiles tanto el tiempo como los costes son fijos y lo que es variable es el alcance. Esto puede dar a entender que puede que no se cumpla con toda la funcionalidad, pero si se prioriza correctamente, las funcionalidades más importantes que estarán completas y funcionando correctamente, se garantiza la construcción del mejor producto para un presupuesto dado [22].

Bajo este contexto, se implementará una metodología ágil basada en sprints quincenales con entregables al término del mismo. El trabajo a realizar en cada sprint, corresponde a cada uno de los objetivos específicos. El conjunto de entregables finales, nos conducirá a lograr el objetivo pensado en el punto 4.3.

Cada semana, se realizará una reunión de trabajo entre el autor y el director que tomarán los roles de desarrollador y cliente final con el propósito de obtener un feedback del trabajo ejecutado durante la semana y cuando sea finalización del sprint, del entregable obtenido.

Se requerirán realizar actividades de seguimiento al proyecto de forma periódica entre el autor y director en las diferentes etapas del proyecto para obtener una retroalimentación semanal del trabajo efectuado y quincenal respecto a los entregables pactados y con esto, dar lugar al objetivo del proyecto. A continuación, se ilustra el flujo de actividades resumido en la Figura 6.

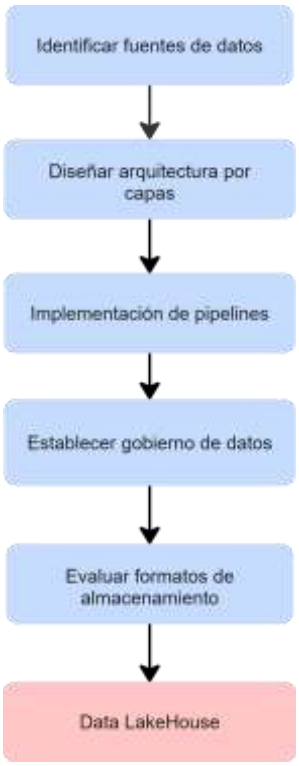


Figura 6. Metodología a implementar. Fuente: Elaboración propia

4.5 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA

El trabajo está propuesto a realizarse en un plazo de 12 semanas basado en jornadas diarias de 3 horas, lo equivalente a 180 horas de esfuerzo. El backlog de actividades será definido en conjunto en el inicio de cada sprint. Es por ello que las horas propuestas pueden llegar a cambiar de mutuo acuerdo en la ejecución del proyecto.

El cronograma de actividades se encuentra representado en la Figura 7 donde se hace un desglose de los entregables.

4.5.1 PLANIFICACIÓN

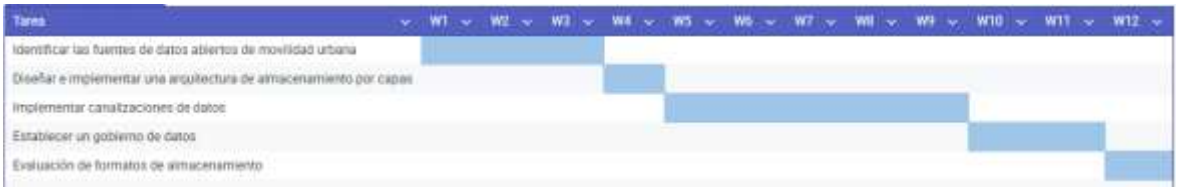


Figura 7. Planificación a implementar. Fuente: Elaboración propia

4.5.2 ESTIMACIÓN ECONÓMICA

De forma general y de acuerdo al objetivo planteado, la estimación de tiempo de ejecución del proyecto será de 12 semanas sumando un total de horas de esfuerzo por 180 horas con un total de 3 horas por día. El detalle de la estimación en base a horas esfuerzo, se desglosa en la **¡Error! No se encuentra el origen de la referencia.**.

Descripción	*Precio por hora (euros)*	*Número de Horas*	*Importe*
Horas esfuerzo en el desarrollo del proyecto	50	180	$9.000,00
		Sub-total	$9.000,00

Tabla 1. Presupuesto de horas esfuerzo

Del lado de la infraestructura, los costes por el uso de los servicios en la nube de Google Cloud, se desglosa de la siguiente forma en la Tabla 2 [38].

Servicio	*Descripción*	*Importe (mensual)*
Secret Manager	Access operations: 10000 Active secret versions: 10	$0,24
Cloud Run	CPU amount per instance: 1 vCPU Memory amount per instance: 1 GiB Region: us-central1 (Iowa) Resource Type: Job Number of executions per month (thousand): 2 Execution time per task: 2 Minutes	$4,8
Dataflow Classic	Number of job run hours: 28 Hours Region: Iowa (us-central1) Job Type: Batch Number of worker nodes: 1 Machine type: n1-standard-4, vCPUs: 4, RAM: 15 GB	$8,12
On-Demand (BigQuery)	Amount of data processing: 200 GiB Active storage: 30 GiB	$0,46

Cloud Logging	Logging storage amount: 20 GB Logging retention duration: 6	$1,20
Cloud Storage	Total amount of storage: 30 GiB Storage class: Standard Storage Sub-total	$0,5 $15,32~

Tabla 2. Presupuesto de nube

En la Tabla 3 se describe el coste estimado del proyecto durante los primeros 3 meses.

Descripción	*Importe (euros)*
Horas esfuerzo en el desarrollo del proyecto (mano de obra)	$12.000,00
I.V.A. (21%) de mano de obra	$2.520,00
Servicios de GCP por 3 meses	$15,32
Total	$14.535,32

Tabla 3. Presupuesto del proyecto.

Capítulo 5. SISTEMA/MODELO DESARROLLADO

En este capítulo, se llevará a cabo el desarrollo de toda la arquitectura planteada en la Figura 8 que se conformará por los primeros 4 puntos de los objetivos específicos descritos en el punto 4.3.1

5.1 DISEÑO DE LA ARQUITECTURA LAKEHOUSE

Bajo el Alcance establecido en el punto 4.2, a continuación, se presenta el diagrama de la Arquitectura del Data Lakehouse en la Figura 8 donde se puede visualizar el proceso end-to-end desde donde los datos son ingeridos hasta donde son almacenados y listos para su consumo.

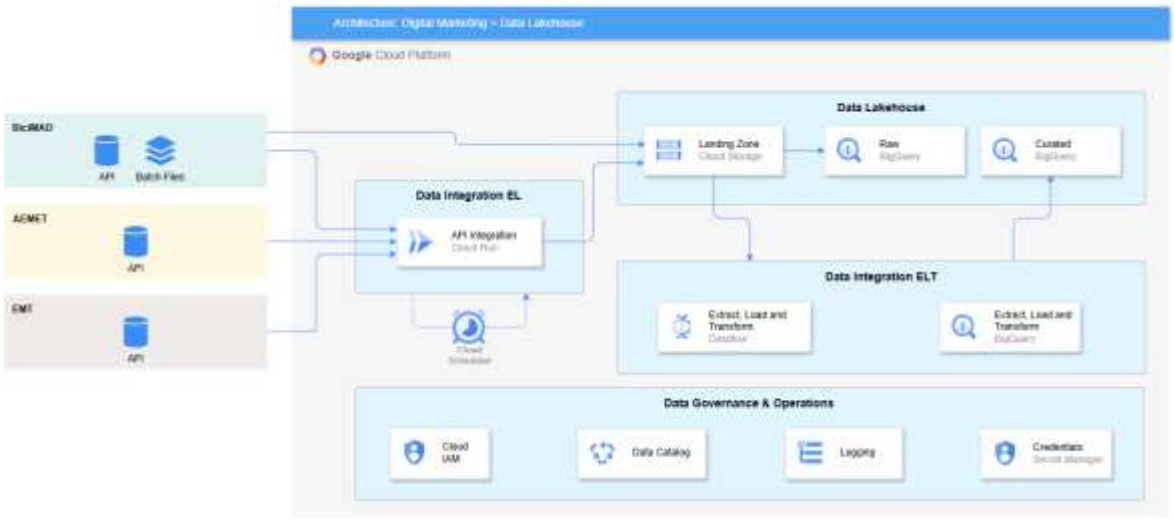


Figura 8. Arquitectura de Data Lakehouse. Fuente: Elaboración propia

5.2 IMPLEMENTACIÓN DE FUENTES DE DATOS.

Con base en lo mencionado en el punto 1 de los objetivos específicos (4.3.1), a continuación, se muestra la Figura 9 que nos ayuda a visualizar el número, tipo y origen de fuentes de datos que se estarán implementando al proyecto.

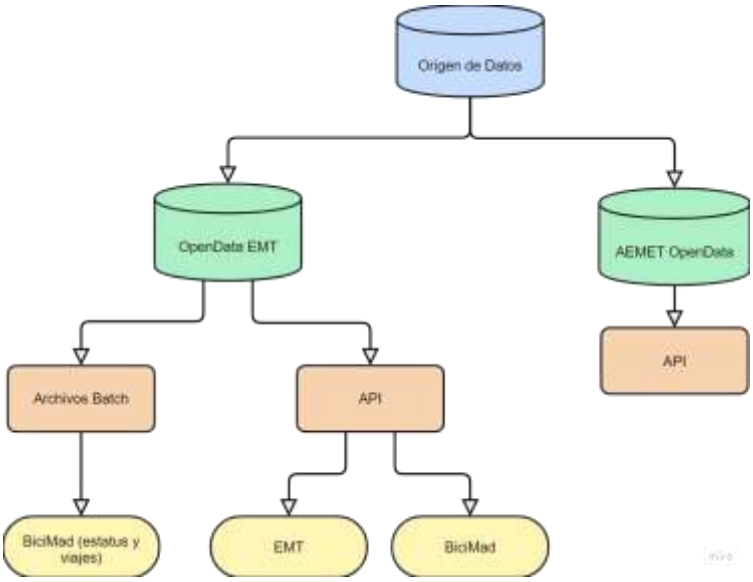


Figura 9. Origen de fuentes de Datos. Fuente: Elaboración propia

5.2.1 CLOUD RUN

Para la lectura y extracción de estos datos correspondientes a la API (punto 5.2.6.2), se contará con *Jobs* en Cloud Run, los cuales se definen mediante un contenedor de Docker.

5.2.2 AUTENTICACIÓN DE CUENTA GOOGLE

Previo a realizar la autentación, se debe contar con lo siguiente:

1. Cuenta de Google creada.
2. Descarga e instalación del SDK de Google Cloud.
3. Inicio de sesión de la cuenta de Google en el navegador.

De esta manera, será más ágil trabajar directamente con la cuenta de Cloud sincronizada.

Mediante el comando:

```
gcloud auth login
```

Se apertura el inicio de sesión para realizar la autenticación. Una vez efectuado y seleccionado el proyecto, deberá verse de esta forma marcado en la Figura 10.

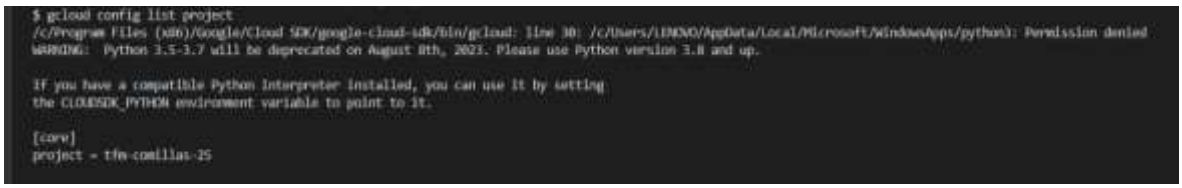


Figura 10. SDK autenticado. Fuente: Elaboración propia

Antes de subir o bajar imágenes, se debe configurar el cliente docker para autenticarse con el servicio de Artifact Registry. Para configurar la autenticación, ejecutar el siguiente comando:

```
gcloud auth configure-docker us-central1-docker.pkg.dev
```

y el resultado se mostrará cómo se muestra a continuación en la Figura 11.

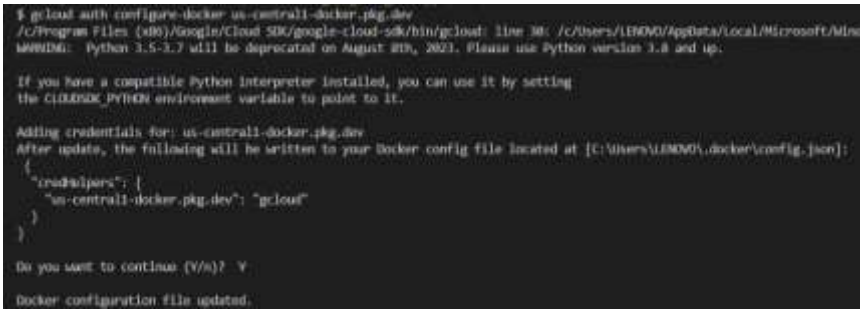


Figura 11. Docker sincronizado con Artifact Registry. Fuente: Elaboración propia

5.2.3 CONFIGURACIÓN DE ARTIFACT REGISTRY

Con este servicio alojaremos imágenes de Docker para posteriormente ser ejecutadas por Jobs de Cloud Run. A continuación, se enlistan los comandos implementados.

Habilitación de la API:

```
gcloud services enable artifactregistry.googleapis.com
```

Creación del repositorio via commando SDK:

```
gcloud artifacts repositories create $REPO_NAME --repository-format=docker \
--location=$REGION --description="Docker repository" --project $PROJECT_ID
```

Con el Proyecto actual:

```
gcloud artifacts repositories create tfm-comillas-25-artreg-docker --repository-
format=docker \
--location=us-central1 --description=" " --project tfm-comillas-25
```

Una vez creado, se verá en la consola como se muestra en la Figura 12.

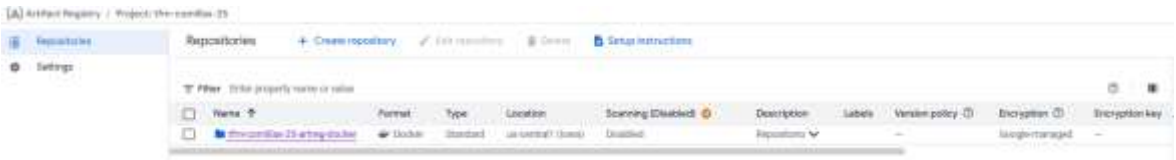


Figura 12. Repositorio de contenedores de Docker creado. Fuente [32]

5.2.4 CONFIGURACIÓN DE SECRETS MANAGER

Con Secrets manager, llevaremos a cabo el gobierno de secretos los cuales estarán encriptados bajo la administración de Google. Cabe mencionar que también puede implementarse una propia llave de encriptación, sin embargo, esto queda de momento fuera del alcance del trabajo.

Para crear un secreto:

```
gcloud secrets create {secret-id} --replication-policy="automatic"
```

Agregar una versión:

```
echo -n "text here" | \
    gcloud secrets versions add {secret-id} --data-file=-
```

Acceder a la última versión de un secreto

```
gcloud secrets versions access latest --secret="{secret-id}"
```

O también puede crearse desde la consola como a continuación de muestra en la Figura 13 y finalmente, estar disponible en la consola como se ilustra en la Figura 14.

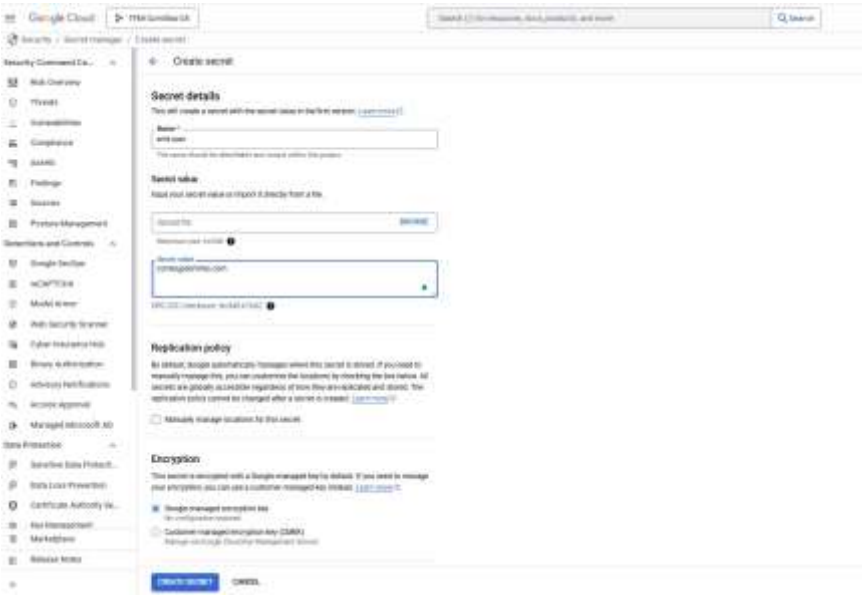


Figura 13. Creación del Secret en la Consola de Google. Fuente [32]

Figura 14. Secret creado y disponible en la Consola de Google. Fuente [32]

Para habilitar la API para el consumo de secretos posteriormente, en la Consola de Google buscamos Secret Manager API y click en Habilitar. Una vez efectuado, aparecerá como se muestra en la Habilitación de la API de Secret Manager. Fuente [32]Figura 15.

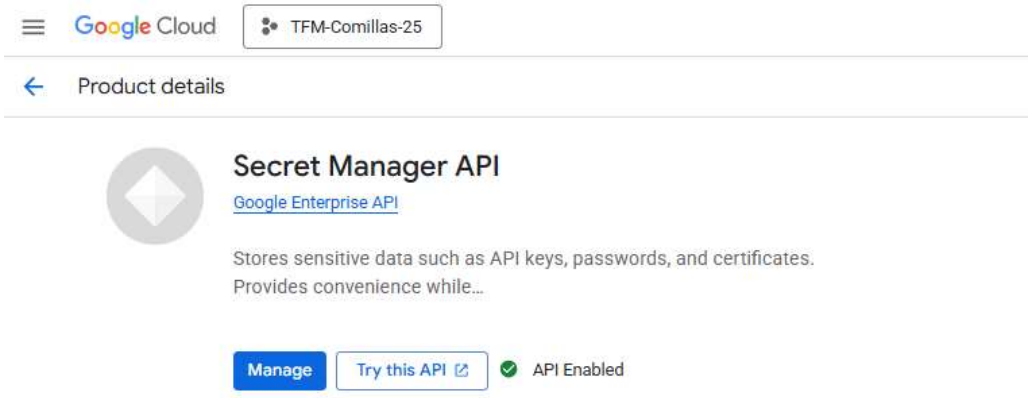


Figura 15. Habilitación de la API de Secret Manager. Fuente [32]

El diccionario de secretos utilizado en este proyecto, puede encontrarse en la Tabla 21.

5.2.5 CONFIGURACIÓN DE LOS JOBS DE CLOUD RUN

La disponibilidad de los *Jobs* de Cloud Run, está bajo la definición de un contenedor de Docker. En la Figura 16 se muestra la jerarquía de archivos de la aplicación.

Figura 16. Estructura de carpetas del repositorio de la imagen de docker para la extracción de datos desde la API de EMT y AEMET. Fuente: Elaboración propia

Archivo de configuración de Docker

```
FROM python:3.8-slim

RUN apt-get update && apt-get install -y build-essential

COPY requirements.txt /opt/requirements.txt
RUN pip install --upgrade pip && pip install -r /opt/requirements.txt

ENV BUCKET_NAME=${BUCKET_NAME}

ARG APP_DIR=/app
RUN mkdir $APP_DIR
COPY . $APP_DIR/

ENV PYTHONPATH $APP_DIR/src
WORKDIR $APP_DIR/src

# Run the web service on container startup.
CMD ["python3", "main.py"]
```

Los scripts empleados para obtener esta información se encuentran disponibles en el Anexo A (punto 1, carpeta *app-el*). Cada script está diseñado para consumir un endpoint específico de la API correspondiente. La decisión de crear un script por endpoint responde a la necesidad de mantener un control individual sobre cada descarga, lo que permite gestionarlas de forma independiente. Además, cada script está asociado a un Job de Cloud Run, facilitando su ejecución y administración.

Una vez definido la aplicación, debe empaquetarse y subirse hacia el Artifact Registry en Cloud. Una vez alojado, podrá crearse los *Jobs* de Cloud Run. Las variables requeridas para elaborar esto, se definen a continuación:

```
export PROJECT_ID=tfm-comillas-25
export REPO_NAME=tfm-comillas-25-artreg-docker
export REGION=us-central1
export IMAGE_NAME=api-opendata-emt
export SVC_JOB=crunjob-integration
export VERSION=12
export BUCKET_NAME='tfm-comillas-25-rawdata'
export USERNAME=emt-user
export PASSWORD=emt-password
export AEMET_TOKEN=aemet-token
export ID_CD=28079
export URL_EMT="https://openapi.emtmadrid.es/"
export URL_AEMET="https://opendata.aemet.es/"
```

El diccionario de las variables, se muestra a continuación en la Tabla 4:

Secreto	*Descripción*
PROJECT_ID	ID del proyecto dentro de Google Cloud Platform
REPO_NAME	Nombre del repositorio dentro de Artifact Registry
REGION	Región donde está alojado el artefacto
IMAGE_NAME	Nombre de la imagen de Docker en Artifact Registry
SVC_JOB	Nombre de la cuenta de servicio con los roles correspondientes para ejecutar de forma automática los jobs
VERSION	Versión del artefacto
BUCKET_NAME	Nombre del bucket en Cloud Storage de la capa cruda
USERNAME	Nombre del Secret correspondiente al correo electrónico para inicio de sesión en la API de EMT
PASSWORD	Nombre del Secret correspondiente a la constraseña para inicio de sesión en la API de EMT
AEMET_TOKEN	Nombre del Secret correspondiente al token para inicio de sesión en la API de AEMET
ID_CD	Variable de ambiente que contiene la clave la Ciudad a descargar datos en el Job de Cloud Run
URL_EMT	Variable de ambiente que contiene la URL de la API de EMT
URL_AEMET	Variable de ambiente que contiene la URL de la API de AEMT

Tabla 4. Diccionario de Variables de Cloud Run.

Habilitación de la API:

```
gcloud services enable cloudrun.googleapis.com
```

Compilar y empaquetar la imagen (el proceso se ilustra en la Figura 17):

```
docker build -t $IMAGE_NAME:$VERSION .
```

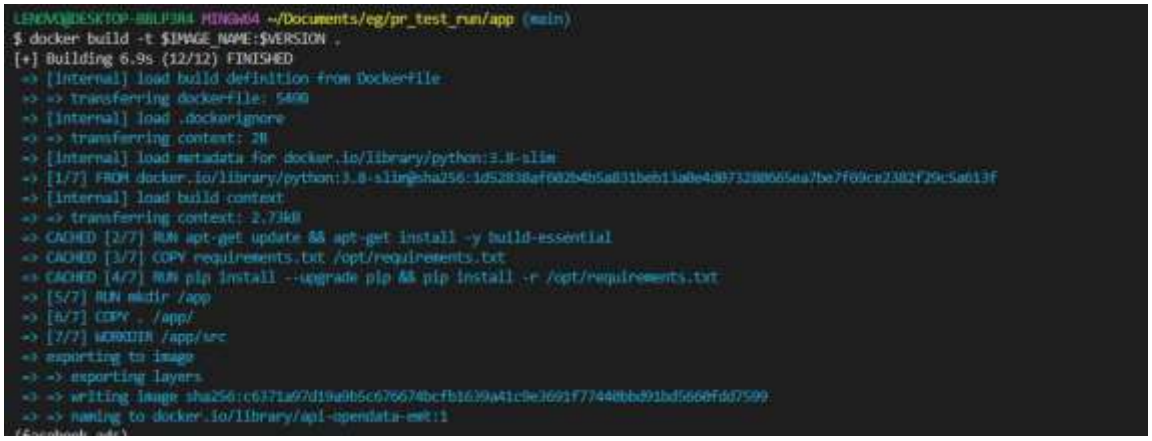


Figura 17. Compilación y empaquetamiento de la imagen. Fuente: Elaboración propia.

Agregar etiqueta a la imagen como referencia dentro de docker:

```
docker tag $IMAGE_NAME:$VERSION $REGION-
docker.pkg.dev/$PROJECT_ID/$REPO_NAME/$IMAGE_NAME:$VERSION
```

Subir imagen a Artifact Registry (el proceso de ilustra en la Figura 18):

```
docker push $REGION-docker.pkg.dev/$PROJECT_ID/$REPO_NAME/$IMAGE_NAME:$VERSION
```

Figura 18. Subida de contenedor hacia Artifact Registry. Fuente: Elaboración propia.

Una vez alojado la imagen, se reflejará en el Artifact Registry como se ilustra en la Figura 19.

Figura 19. Imagen creada en Artifact Registry. Fuente [32]

Conforme se vaya realizando diversas actualizaciones sobre la imagen, se requiere ir cambiando la versión o *tag* de la imagen para su identificación como se ilustra en la Figura 20. Por otro lado, al utilizar un repositorio de *git*, este genera un número de *commit* por cada actualización efectuada que tambien vale para agregarse como valor de *tag*.

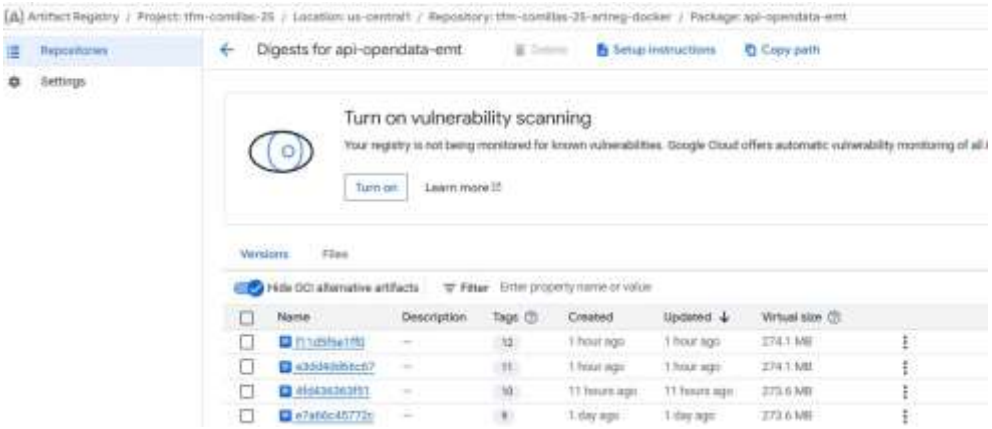


Figura 20. Versionamiento de la Imagen creada en Artifact Registry. Fuente [32]

Crear Job de Cloud Run:

```
export CLOUD_RUN_JOB_NAME=tfm-comillas-25-aemet-clima-actual-dly
gcloud beta run jobs create $CLOUD_RUN_JOB_NAME \
  --image $REGION-docker.pkg.dev/$PROJECT_ID/$REPO_NAME/$IMAGE_NAME:$VERSION \
  --set-env-vars "BUCKET_NAME=$BUCKET_NAME" \
  --service-account $SVC_JOB@$PROJECT_ID.iam.gserviceaccount.com \
  --region $REGION \
  --max-retries=0 \
  --set-secrets TOKEN=$AEMET_TOKEN:latest \
```

```
--set-secrets USERNAME=$USERNAME:latest \
--set-secrets PASSWORD=$PASSWORD:latest \
--set-env-vars "ID_CD=$ID_CD" \
--set-env-vars "URL_EMT=$URL_EMT" \
--set-env-vars "URL_AEMET=$URL_AEMET" \
--project $PROJECT_ID \
--command="python" \
--args="aemet_clima_actual.py" \
--task-timeout=2m
```

Se crea un Job de Cloud Run bajo el nombre definido en la variable $CLOUD_RUN_JOB_NAME, que, para ejemplificar este caso, es *tfm-comillas-25-aemet-clima-actual-dly*. Este job está diseñado para ejecutar el script `aemet_clima_actual.py`, contenido en la imagen almacenada en Artifact Registry.

Secretos

Se inyectan tres secretos directamente desde Secret Manager, lo que garantiza la seguridad de la información sensible. El parámetro *latest*, indica que se lee la última versión del secreto. La definición de estos secretos se encuentra en la Tabla 21. Diccionario de Secretos.

- TOKEN=$AEMET_TOKEN:latest
- USERNAME=$USERNAME:latest
- PASSWORD=$PASSWORD:latest

Variables de entorno

Estas variables son necesarias para la ejecución del script Python las cuales se describen dentro de la Tabla 4:

- BUCKET_NAME=$BUCKET_NAME
- ID_CD=$ID_CD
- URL_EMT=$URL_EMT
- URL_AEMET=$URL_AEMET

Cuenta de servicio

- --service-account $SVC_JOB@$PROJECT_ID.iam.gserviceaccount.com:

Se define la cuenta de servicio con los permisos adecuados para acceder a Cloud Storage, Secret Manager, y otras APIs necesarias durante la ejecución. En el capítulo 5.5.1.4, se describirá en detalle la cuenta de servicio.

Configuración de ejecución

- --max-retries=0: No se permiten reintentos en caso de fallo; el Job se ejecuta solo una vez.
- --region $REGION: Región donde se ejecuta el Job, alineada con donde está almacenada la imagen.
- --task-timeout=2m: Tiempo máximo de ejecución de 2 minutos. Si se excede, el Job se cancela automáticamente para evitar sobrecostos innecesarios, dado que en pruebas previas se ha verificado que la ejecución toma menos de un minuto.

Comando ejecutado

- --command="python" y --args="aemet_clima_actual.py": Se indica que el contenedor debe ejecutar el intérprete de Python con el script principal `aemet_clima_actual.py`.

La ejecución del comando de creación de un job en Cloud Run, provee una salida como se muestra en la Figura 21.

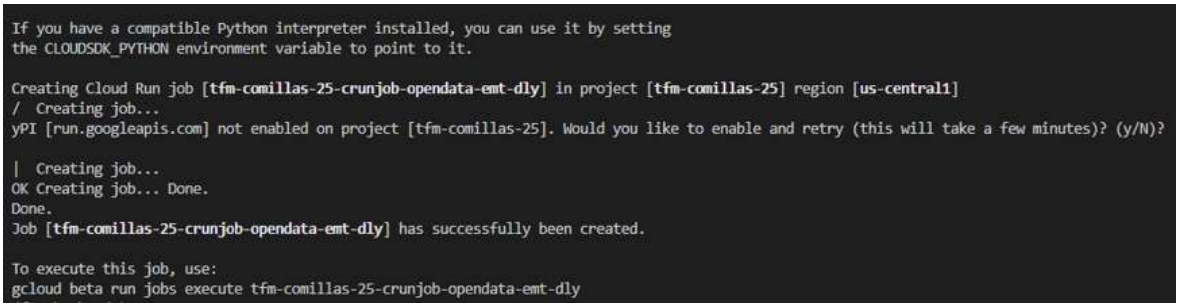


Figura 21. Job creado en Cloud Run. Fuente: Elaboración propia.

Finamente, en la Figura 22, se muestra el conjunto de Jobs en Cloud Run una vez desplegados.

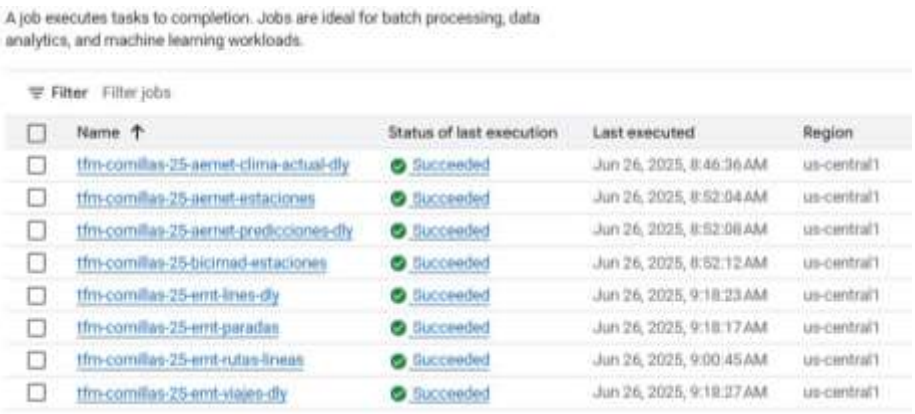


Figura 22. Listado de Job creados en Cloud Run. Fuente [32]

5.2.5.1 Calendarización

Cloud Run incluye una herramienta que permite configurar un *trigger*, el cual permitirá calendarizar un job. Utiliza la notación *unix-cron* para agregar la programación.

En la Figura 23, se selecciona la cuenta de servicio *crunjob-integration* la cual ya contiene el rol acorde para la automatización. En la Figura 24, se agrega la hora de programación con formato unix-cron y de acuerdo a la zona horaria UTC. Una vez establecido el trigger, se refleja dentro del job como se ilustra en la Figura 25.

Todos los Jobs de Cloud Run que estén con nomenclatura *_dly*, se calendarizarán de forma diaria.

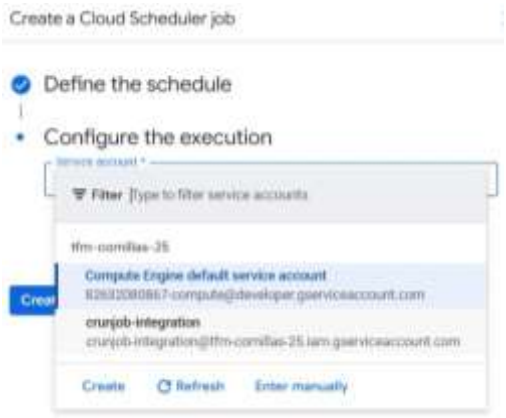


Figura 23. Selección de la Cuenta de Servicio para la ejecución automatizada del Job. Fuente [32]

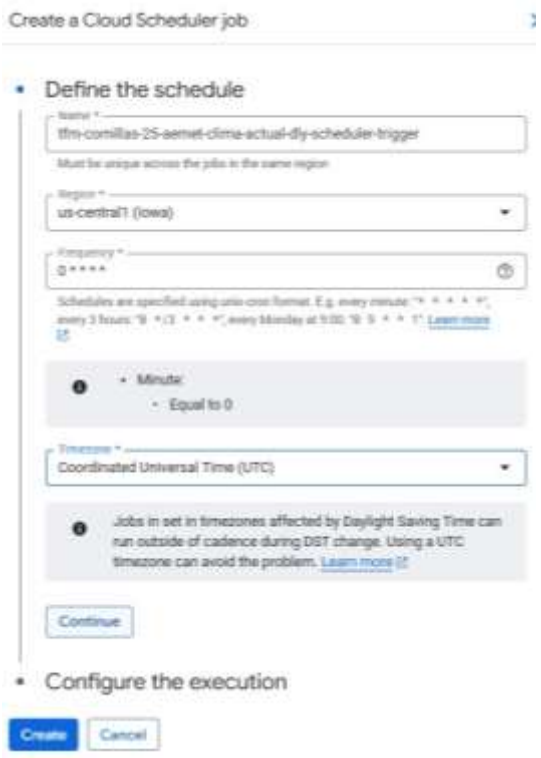


Figura 24. Definición del formato unix-cron. Fuente [32]

Figura 25. Trigger creado. Fuente [32]

5.2.5.2 Proceso

Cada Job de Cloud Run, tiene como objetivo escribir en su folder correspondiente en el bucket de storage los archivos descargados. En las tablas de los puntos 5.2.6.2 y 5.2.7.1 se describen el folder correspondiente donde se escribe los datos.

5.2.6 OPEN DATA EMT

La Empresa Municipal de Transportes (EMT) dispone de un portal de datos abiertos [30], donde es posible consultar información relacionada con los medios de transporte públicos superficiales. Se estará utilizando los archivos por lotes para datos históricos relacionados a BiciMAD y se estará utilizando la API para las extracciones diarias.

5.2.6.1 BiciMAD – Archivos por lotes

El sitio de Open Data, ofrece una sección específica que contiene los datos históricos de BiciMAD desde el año 2017 hasta el año 2023 (solo enero y febrero para los datos de viajes hechos por el usuario), aunque como se mencionó en el Alcance (4.2), se implementarán los datos desde el año 2020. Cada archivo está comprimido por año y al efectuar la extracción de los archivos, contiene por año archivos en formato *json* o *csv*, sin embargo, no mantiene una nomenclatura uniforme por entre los tipos de formato y renombramientos. Se ha desarrollado un script que se encarga de unificar todos los datos de viajes y de estatus de estaciones por año en un solo dataset con extensión *csv* que serán los archivos batch o por lotes almacenados en la capa cruda en Cloud Storage y que se aplicará un ETL de los datos

para posteriormente ser depositados a BigQuery utilizando Dataflow. El script de unificación, puede encontrarse en el Anexo A (punto 4, carpeta *script-unificacion*).

A continuación, el listado de archivos unificados.

- df_status_stations_2020.csv
- df_status_stations_2021.csv
- df_status_stations_2022.csv
- df_trips_2020.csv
- df_trips_2021.csv
- df_trips_2022.csv
- df_trips_2023.csv

Como mejora a considerar para trabajos futuros, sería conveniente establecer y mantener una nomenclatura uniforme para los archivos, lo que facilitaría un proceso de extracción y lectura y, por consiguiente, sea una unificación homogénea.

Por otro lado, el script de unificación está diseñado para integrarse en un proceso ETL y ejecutarse de forma diaria, con el objetivo de alimentar continuamente un histórico general de datos. En este sentido, este sería otra mejora a considerar en tener a disposición la información de manera frecuente. En la Tabla 5 y Tabla 6, se detalla los sizing de cada archivo CSV según sea de viajes por el usuario o estatus de estaciones respectivamente.

Datos de viajes hechos por usuarios

Dataset	*Tamaño*	*Dimensiones*
df_trips_2020.csv	543.09 MB	Filas: 3379801 - Columnas: 12
df_trips_2021.csv	894.45 MB	Filas: 5204373- Columnas: 30
df_trips_2022.csv	1479.26 MB	Filas: 8288268 - Columnas: 19

df_trips_2023.csv	123.00 MB	Filas: 687014 - Filas: 19

Tabla 5. Dimensiones de datasets de viajes hechos por usuarios.

Se revisó que los esquemas entre los datasets son distintos. La información de los esquemas puede encontrarse en el Anexo A (punto 2b, carpeta *bq-raw-schemas*). De acuerdo al alcance de este trabajo, no se profundizará en un análisis exhaustivo de la información. En el punto 5.4.1.1, se revisará un conjunto de transformaciones con el fin de enriquecer esta información.

Datos de los estatus de las estaciones

Dataset	*Tamaño*	*Dimensiones*
df_status_stations_2020.csv	498.02 MB	Filas: 8740 - Columnas: 3
df_status_stations_2021.csv	610.39 MB	Filas: 8716- Columnas: 3
df_status_stations_2022.csv	612.35 MB	Filas: 8738 - Columnas: 3

Tabla 6. Dimensiones de datasets de los estatus de las estaciones.

Se revisó que los esquemas entre los datasets son iguales.

Todos los archivos csv finales, se suben vía local desde el ordenador hasta el folder correspondiente en el bucket de Cloud Storage.

La definición del bucket, se desarrolla en el punto 5.3.1.1.

5.2.6.2 API

Se encuentra divida en diversos bloques y endpoints. Para fines de este proyecto y de acuerdo a los puntos de información a obtener descritos en el Alcance (4.2), se estará obteniendo datos de los endpoints situados en los bloques 3 (Transport BUSEMTMAD) y 4 (Block 4

Transport BICIMAD). Cada uno de los endpoints, contará con un respectivo job de Cloud Run siguiendo los pasos descritos en el capítulo 5.2.1.

5.2.6.2.1 EMT

De este bloque, consumiremos 4 endpoints considerado los más relevantes a fines de este proyecto para obtener. En la tabla Tabla 7, Tabla 8, Tabla 9, Tabla 10, se detallan las configuraciones finales por cada Job de Cloud Run.

La Información de líneas de autobuses por día

Endpoint	*/v2/transport/busemtmad/lines/info/{dateref}/*
Nombre del Job en Cloud Run	tfm-comillas-25-emt-lines-dly
Script de python	emt_lineas_diario.py
Frecuencia de extracción	Diaria
Hora de extracción	40 0 * * *

Tabla 7. Tabla de información del job de extracción de líneas de autobuses por día.

Parámetros:

- *dateref*: Fecha en formato YYYYMMDD

Las Rutas de líneas

Endpoint	*/v1/transport/busemtmad/lines/{line_id}/route/*
Nombre del Job en Cloud Run	tfm-comillas-25-emt-rutas-lineas
Script de python	emt_rutas_lineas.py
Frecuencia de extracción	Semanal

Hora de extracción	50 0 * * 1

Tabla 8. Tabla de información de las rutas de líneas.

Parámetros:

- *line_id*: Identificador de la línea de autobús.

La Información de las paradas

Endpoint	*/v1/transport/busemtmad/stops/{stop_id}/detail/*
Nombre del Job en Cloud Run	tfm-comillas-25-emt-paradas
Script de python	emt_paradas.py
Frecuencia de extracción	Semanal
Hora de extracción	45 0 * * 1

Tabla 9. Tabla de información de las paradas.

Parámetros:

- stop_id: Identificador de la parada de autobús

Los Viajes efectuados por día

Endpoint	*/v1/transport/busemtmad/lines/{line_id}/trips/{dataref}/*
Nombre del Job en Cloud Run	tfm-comillas-25-emt-viajes-dly
Script de python	emt_viajes_diario.py
Frecuencia de extracción	Diaria
Hora de extracción	45 0 * * *

Tabla 10. Tabla de información de viajes efectuados por día.

Parámetros:

- *line_id*: Identificador de la línea de autobús.
- *dateref*: Fecha en formato YYYYMMDD

5.2.6.2.2 BiciMAD

De este bloque, consumiremos 1 endpoint correspondiente al detalle de las estaciones de BiciMAD. La Tabla 11 contiene el detalle del job.

Endpoint	*/v1/transport/bicimad/stations/*
Nombre del Job en Cloud Run	tfm-comillas-25-bicimad-estaciones
Script de python	bicimad_estaciones.py
Frecuencia de extracción	Cada hora
Hora de extracción	15 * * * *

Tabla 11. Tabla de información de las rutas de líneas.

5.2.7 OPEN DATA AEMET

AEMET OpenData es un API REST a través del cual se pueden descargar gratuitamente los datos explicitados en el Anexo II de la resolución de 30 de diciembre de 2015 de AEMET, por la que se establecen los precios públicos que han de regir la prestación de servicios meteorológicos y climatológicos [34].

5.2.7.1 API

Se encuentra divida en diversos endpoints. Para fines de este proyecto y de acuerdo a los puntos de información a obtener descritos en el Alcance (4.2), se estará obteniendo los siguientes datos descritos en la Tabla 12, Tabla 13, Tabla 14.

Los Datos de predicción diaria

Endpoint	*/opendata/api/prediccion/especifica/municipio/diaria/{id_cd}/*
Nombre del Job en Cloud Run	tfm-comillas-25-aemet-predicciones-dly
Script de python	aemet_predicciones.py
Frecuencia de extracción	Diaria
Hora de extracción	35 0 * * *

Tabla 12. Tabla de información de la predicción diaria de AEMET.

El Tiempo actual

Endpoint	*/opendata/api/observacion/convencional/todas*
Nombre del Job en Cloud Run	tfm-comillas-25-aemet-clima-actual-dly
Script de python	aemet_clima_actual.py
Frecuencia de extracción	Diaria
Hora de extracción	30 0 * * *

Tabla 13. Tabla de información del tiempo actual.

Las Estaciones

Endpoint	*/opendata/api/valores/climatologicos/inventarioestaciones/todasestaciones/*
Nombre del Job en Cloud Run	tfm-comillas-25-aemet-estaciones

Script de python	aemet_estaciones.py
Frecuencia de extracción	Semanal
Hora de extracción	35 0 * * 1

Tabla 14. Tabla de información de las estaciones de climatología.

Cada uno de los endpoints, contará con un respectivo Job de Cloud Run siguiendo los pasos descritos en el capítulo 5.2.1.

5.3 IMPLEMENTACIÓN DE ALMACENAMIENTO POR CAPAS

El modelo de arquitectura pensado para este proyecto sigue el enfoque Lakehouse, que combina los principios del Data Lake (flexibilidad, escalabilidad) y del Data Warehouse (estructura, gobernanza y analítica). Esta arquitectura se apoya en un modelo por capas de almacenamiento que permite separar las distintas fases del ciclo de vida de los datos: desde la ingesta hasta la transformación y explotación.

Para la implementación de la arquitectura, se utiliza Google Cloud Storage como repositorio intermedio y BigQuery como motor de almacenamiento estructurado y análisis. Se han implementado tres capas principales:

- Landing: Almacenamiento inicial del dato tal como llega, sin transformación.
- Raw (BigQuery): Carga estructurada de los datos sin transformaciones significativas.
- Curated (BigQuery): Datos transformados, estandarizados y enriquecidos.

5.3.1 CONFIGURACIÓN DE LANDING ZONE EN CLOUD STORAGE

La capa Landing actúa como la zona de entrada del Data Lake. Aquí se almacenan los archivos tal como se extraen desde su fuente permitiendo una copia sin alteraciones. Esta capa sirve como respaldo y punto de partida para reprocesamientos.

5.3.1.1 Configuración de Cloud Storage

Previo a ingresar datos a esta capa, en la Figura 26, se muestra el proceso de creación de un bucket en Cloud Storage. A continuación, se describen los parámetros implementados:

- Bucket: `tfm-comillas-25-rawdata` (el nombre es único a nivel global en todo GCP)
- Locación: Región > us-central1 (Iowa). Desde mi ubicación en México, implica menor latencia y menor costo. Se puede contar con alta disponibilidad al habilitar multiregional pero para fines de este proyecto, no se considera como tal.
- Clase de almacenamiento: Estándar. Al acceder frecuentemente a los datos como se describirá en el capítulo 5.3.2, es la opción indicada debido a su bajo coste al tener altas consultas
- Acceso a los objetos: Impedir el acceso público está activado. Por temas de seguridad para evitar que se acceda a los datos de manera libre.
- Control de acceso: Uniforme. Permite gestionar permisos a nivel de bucket en lugar de cada objeto individual. Más simple y seguro para la mayoría de casos.

Figura 26. Procedimiento de creación de un bucket en Cloud Storage vía Consola. Fuente [32]

5.3.1.1.1 Estructura de almacenamiento

Los archivos se almacenan en Google Cloud Storage (GCS) bajo una estructura jerárquica por fuente y fecha de extracción. Los que son de extracción diaria, dentro de la nomenclatura se agrega *_daily*.

- AEMET_CLIMA_ACTUAL/
  - aemet__daily_*.csv
- AEMET_ESTACIONES/
  - estaciones__daily_*.csv
- AEMET_PREDICCIONES/
  - prediccion_aemet__daily_*.csv
- BICIMAD_ESTACIONES/
  - bicimad_stations_*.csv
- EMT_LINEAS_DIARIO/
  - horarios_lineas_daily_*.csv
- EMT_PARADAS/
  - paradas_detalle_*.csv
- EMT_RUTAS_LINEAS/
  - ruta_lineas_*.csv
- EMT_VIAJES_DIARIO/
  - viajes_lineas_daily_*.csv
- bicimad_2020/
  - df_status_stations.csv
  - df_trips.csv
  - schema_bicimad_2020_trips.json
  - schema_bicimad_stations.json
- bicimad_2021/
  - df_status_stations.csv
  - df_trips.csv

  - schema_bicimad_2021_trips.json
  - schema_bicimad_stations.json
- bicimad_2022/
  - df_status_stations.csv
  - df_trips.csv
  - schema_bicimad_2022_trips.json
  - schema_bicimad_stations.json
- bicimad_2023/
  - df_trips.csv
  - schema_bicimad_2023_trips.json
  - schema_bicimad_stations.json

La anotación *, representa que los archivos tienen bajo su nombramiento el valor delta de la fecha en la que se extrajo la información.

5.3.2 CONFIGURACIÓN DE CAPA DE DATOS RAW EN BIGQUERY

La capa raw, tiene como objetivo almacenar los datos en su forma original o con mínimas transformaciones (por ejemplo, conversión de tipos o fechas), permitiendo así conservar una copia fiel de las fuentes externas para control de calidad o reprocesamiento posterior.

Fuentes integradas

- BiciMAD – viajes históricos (2020–2023)
- BiciMAD – estaciones
- EMT – líneas, paradas, rutas de líneas y viajes diarios
- AEMET – clima de hoy, estaciones de meteorología, predicciones climatológicas

Estructura

Los datos están organizados en *tablas* y *datasets*. Los datasets se encargan de almacenar las tablas. La nomenclatura de nombramiento de datasets es: `raw_<origen>`

Comando para crear un dataset:

```
bq --location=REGION mk --dataset PROYECT_ID:NOMBRE_DAYASET
```

Por ejemplo:

```
bq --location=us-central1 mk --dataset tfm-comillas-25:raw_aemet_api
```

Formato de carga

Archivos originales en CSVs.

Proceso de carga

Los datos se cargan en BigQuery como tablas externas, referenciando directamente los archivos almacenados en Cloud Storage. En la Figura 27, se detalla el proceso de creación vía Consola.

Una vez identificado el dataset donde se ubicará la tabla, se inicia su creación. El primer paso consiste en seleccionar la fuente de datos, que debe ser Cloud Storage, lo cual permite elegir un archivo o carpeta específica dentro del bucket.

A continuación, se debe establecer el tipo de tabla como **Externa** y, por último, configurar el esquema como **Auto-detect** para que BigQuery infiera automáticamente la estructura de los datos a partir del contenido del archivo. En la Figura 28, se puede ver el detalle de la configuración de la tabla externa donde resalta la ruta hacia el objeto dentro del bucket.

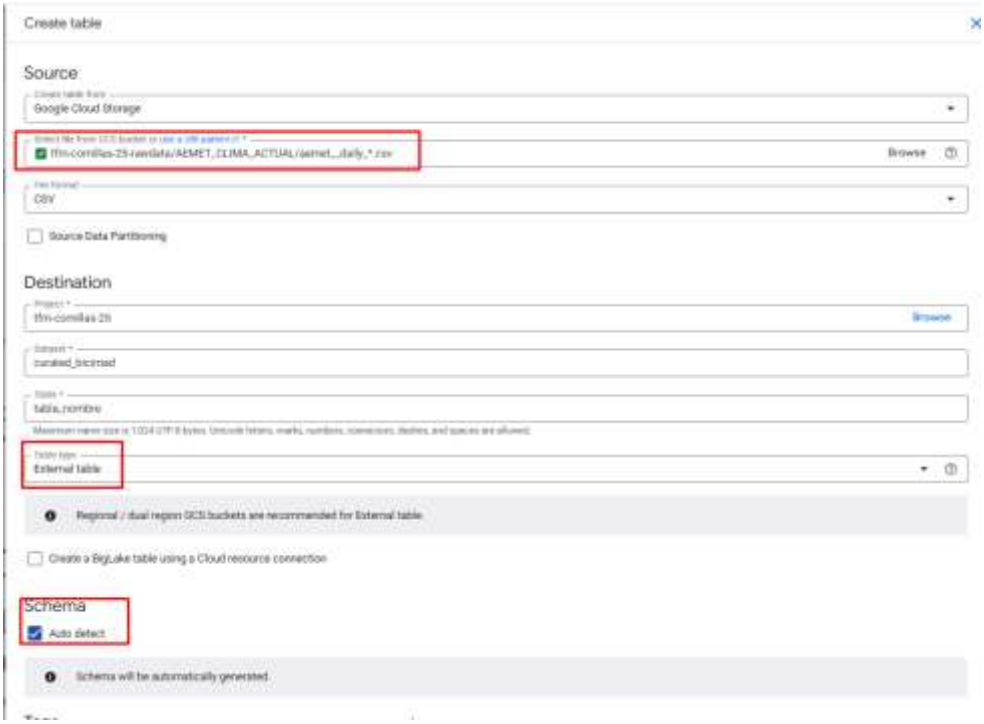


Figura 27. Creación de tabla externa con BigQuery. Fuente [32]

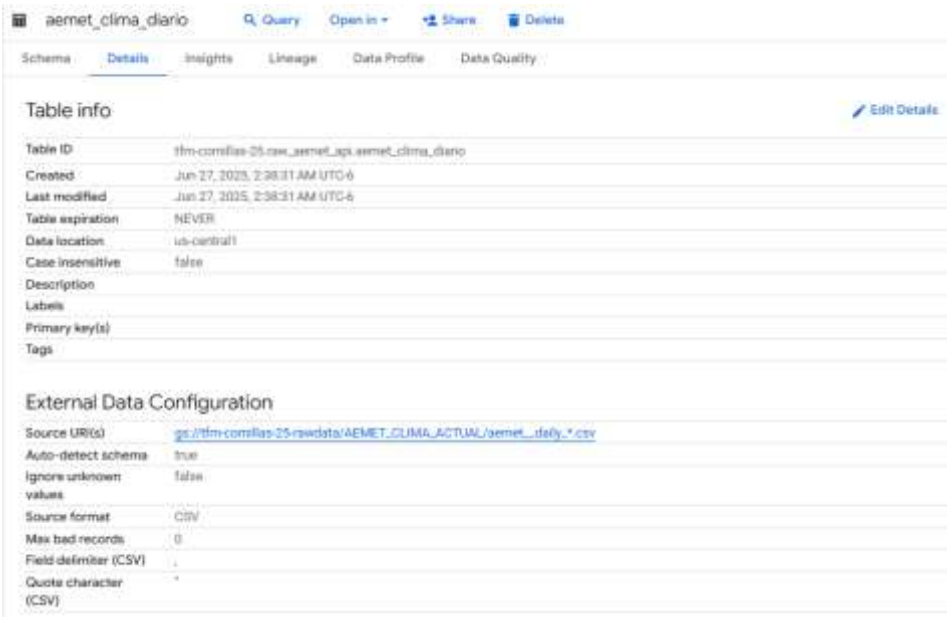


Figura 28. Creación de tabla externa con BigQuery. Fuente [32]

En la Figura 29, con SQL se puede consultar los datos utilizando el motor de BigQuery sin tener los datos almacenados físicamente y manteniendo el formato original como CSV.

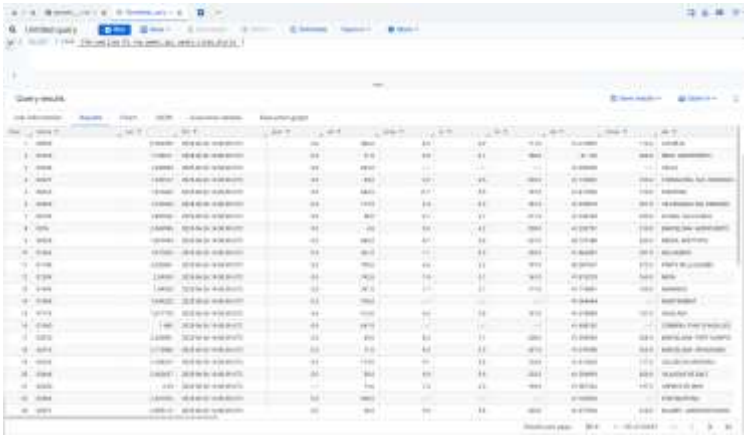


Figura 29. Creación de tabla externa con BigQuery. Fuente [32]

Jerarquía

En la Figura 30 se desglosa la jerarquía de los datasets y tablas dentro de BigQuery y en la Tabla 15 el catálogo por cada tabla.

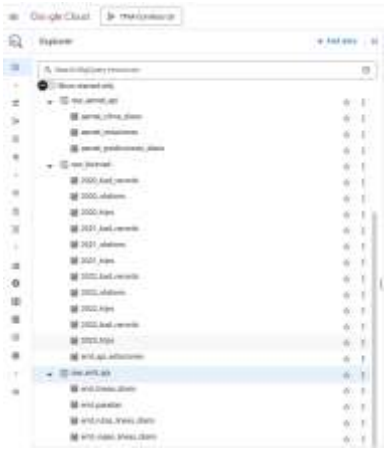


Figura 30. Jerarquía de tablas y datasets en BigQuery. Fuente [32]

Tabla}	*Datasets*	*Descripción*
aemet_clima_diario	raw_aemet_api	Contiene la información diaria del estado de clima
aemet_estaciones	raw_aemet_api	Contiene la información detallada de las estaciones de meteorología
aemet_predicciones_diario	raw_aemet_api	Contiene la información de las predicciones en una ventana de tiempo de los siguientes 3 días
2023_trips	raw_bicimad	Contiene data histórica de viajes efectuados en el año 2023 para los meses de enero y febrero
2022_trips	raw_bicimad	Contiene data histórica de viajes efectuados en el año 2022
2022_stations	raw_bicimad	Contiene información del estados de las estaciones durante el año 2022
2021_trips	raw_bicimad	Contiene data histórica de viajes efectuados en el año 2021
2021_stations	raw_bicimad	Contiene información del estados de las estaciones durante el año 2021
2020_trips	raw_bicimad	Contiene data histórica de viajes efectuados en el año 2020
2020_stations	raw_bicimad	Contiene información del estados de las estaciones durante el año 2020

emt_api_estaciones	raw_bicimad	Contiene información por hora del estatus de las estaciones de BiciMAD
emt_viajes_lineas_diario	raw_emt_api	Contiene información diaria de los viajes efectuados durante el día por cada línea de autobús
emt_rutas_lineas_diario	raw_emt_api	Contiene información diaria de la ruta que opera cada línea
emt_paradas	raw_emt_api	Contiene información semanal de la actualización de las paradas de autobuses
emt_lineas_diario	raw_emt_api	Contiene información diaria de la información en detalle de cada línea de autobús

Tabla 15. Catálogo de tablas crudas.

5.3.3 CONFIGURACIÓN DE CAPA DE DATOS CURATED EN BIGQUERY

La capa curated representa una versión procesada, transformada y estandarizada de los datos, lista para ser consumida por analistas o algún software de visualización. En esta capa se aplican las reglas de negocio, transformaciones, filtros entre otros cambios.

Estructura

Los datos están organizados en vistas y datasets. Los datasets se encargan de almacenar las vistas. La nomenclatura de nombramiento de los datasets es: `curated_<origen>`.

Proceso de Carga

Vía SQL aplicando un ETL de los datasets crudos hacia los datasets curados. A continuación, se encuentra en la Tabla 16 el catálogo de las tablas curadas. En el Anexo A (punto 2a,

carpeta *bq-curated*), puede encontrarse las consultas SQL implementadas con las transformaciones y limpieza para la obtención de datos curados.

Tabla	*Dataset*	*Descripción*
aemet_clima_hoy	curated_aemet_api	Contiene información transformada de las condiciones climatológicas del día en curso y enriquecido con la datos de cada estación
aemet_predicciones	curated_aemet_api	Contiene información transformada de las predicciones climatológicas en una ventana de tiempo de los siguientes 3 días y enriquecido con la datos de cada estación
viajes	curated_bicimad	Contiene información curada del histórico de viajes efectuados desde enero 2021 hasta febrero 2021
viajes_hourly	curated_bicimad	Contiene información curada con periodicidad de una hora del estatus de las estaciones de bicicletas
emt_lineas	curated_emt_api	Contiene información transformada de la unión de las 4 tablas en la capa cruda: emt_viajes_lineas_diario, emt_rutas_lineas_diario, emt_paradas, emt_lineas_diario,

Tabla 16. Catálogo de tablas curadas.

5.4 IMPLEMENTACIÓN DE CANALIZACIONES DE DATOS

En este proyecto se implementaron canalizaciones modulares y escalables que permiten automatizar el flujo de datos desde la capa de entrada hasta la capa analítica, utilizando herramientas del ecosistema de Google Cloud.

5.4.1 CLOUD RUN

5.4.1.1 Transformación y limpieza

Tal como se describe en la sección 5.2.5, se ha empleado un contenedor de Docker alojado en Artifact Registry que permite el despliegue de 8 Jobs de Cloud Run que permiten descargar la información de la API de la AEMET (con 3 jobs) y la API de la EMT (con 5 jobs donde 1 es para BiciMAD y el resto para la información de los autobuses).

Una vez descargados los datos desde sus fuentes, se encuentran en formato JSON y se aplica la conversión a un DataFrame de Pandas que, a su vez, nos permite guardar los datos en Cloud Storage en texto plano con formato *csv*, lo que posteriormente nos permite cargar hacia BigQuery.

- Limpieza:
  - N/A
- Transformaciones:
  - En el archivo `bicimad_estaciones.py` que puede consultarse en el Anexo A (punto 1d, carpeta *app-el/src*) de este documento, se ha agregado la siguiente transformación que nos permite anexar en una columna adicional con un valor timestamp para tener el registro de fecha y hora de extracción dado que solamente dicho script es bajo una frecuencia de cada hora.

```
ahora = datetime.now()
df['timestamp_'] = ahora.strftime("%Y-%m-%d %H:%M:%S")
```

5.4.2 DATAFLOW

Para canalizaciones de procesamiento por lotes de gran volumen, se utilizó Apache Beam como motor de procesamiento distribuido, ejecutado sobre el servicio Dataflow de Google Cloud.

Para el uso de Dataflow, se utilizó una plantilla predefinida de Apache Beam [35] la cual permite llevar archivos batch de gran volumen almacenados en Cloud Storage en texto plano con formato CSV hacia tablas vacías disponibles en BigQuery. Al utilizar esta herramienta, nos aseguramos de obtener una escalabilidad automática, ejecución paralela y la integración entre ambos servicios es directa.

La configuración de un job de Dataflow puede mirarse en la Figura 31.

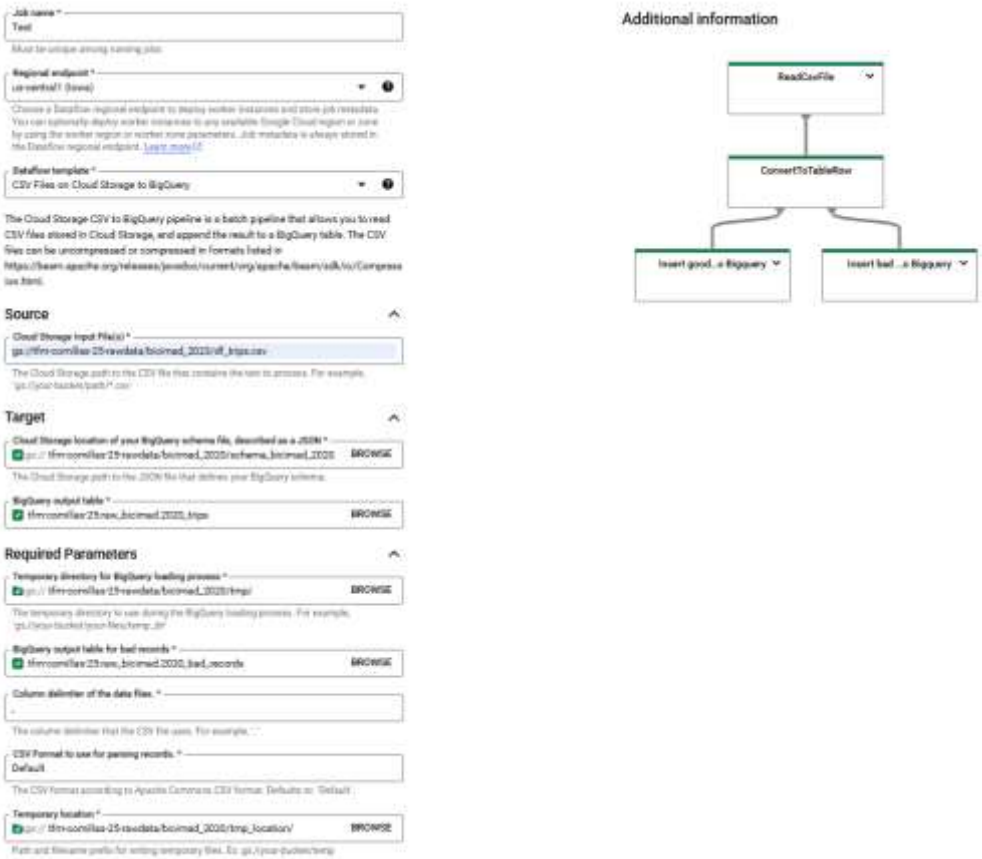


Figura 31. Configuración de Job en Dataflow. Fuente [32]

Puntos importantes a considerar:

- **Dataflow template:** CSV Files on Cloud Storage to BigQuery.
- **Source:** Directorio del objeto CSV en Cloud Storage a ingestar en BigQuery.
- **Target:** Archivo JSON alojado en Cloud Storage que contenga el esquema de la tabla. En el Anexo A (punto 2d, carpeta *elt/dataflow-schemas*) se podrá encontrar los implementados en este proyecto.
- **BigQuery Output table:** Tabla de BigQuery creada previamente y vacía donde se estará ingestando la información.
- **Temporaly directory:** Folder temporal en Cloud Storage para registrar logs de ejecución del job.
- **BigQuery output table:** Tabla para registrar posibles incidentes al momento de ejecutar el Job de Dataflow. Estas tablas no forman parte de la arquitectura y estructura de datos a consumo.
- **Cursor delimiter of the data files:** Para este Proyecto, coma (,) como separador de columnas en los archivos CSV.
- **CSV format to use for parsing records:** Valor por defecto (Default).
- **Temporaly location:** Folder en Cloud Storage para registrar datos de la ejecución del pipeline.

En la Figura 32 se puede mirar como es la ejecución de un Job de Dataflow desde la lectura de los datos en Cloud Storage hasta escribirlos en BigQuery. Por otro lado, en la Figura 33, se encuentra el listado de todos los Jobs de Dataflow elaborados para este proyecto con el fin de ingestar los datos batch o históricos a las tablas correspondientes de BigQuery.

Hay un Job de Dataflow por cada tipo de dataset: viajes (trips) y estaciones (stations) del 2020 al 2023 con excepción de las estaciones que no se encuentra datos del año 2023 hasta el momento de desarrollo de este proyecto.

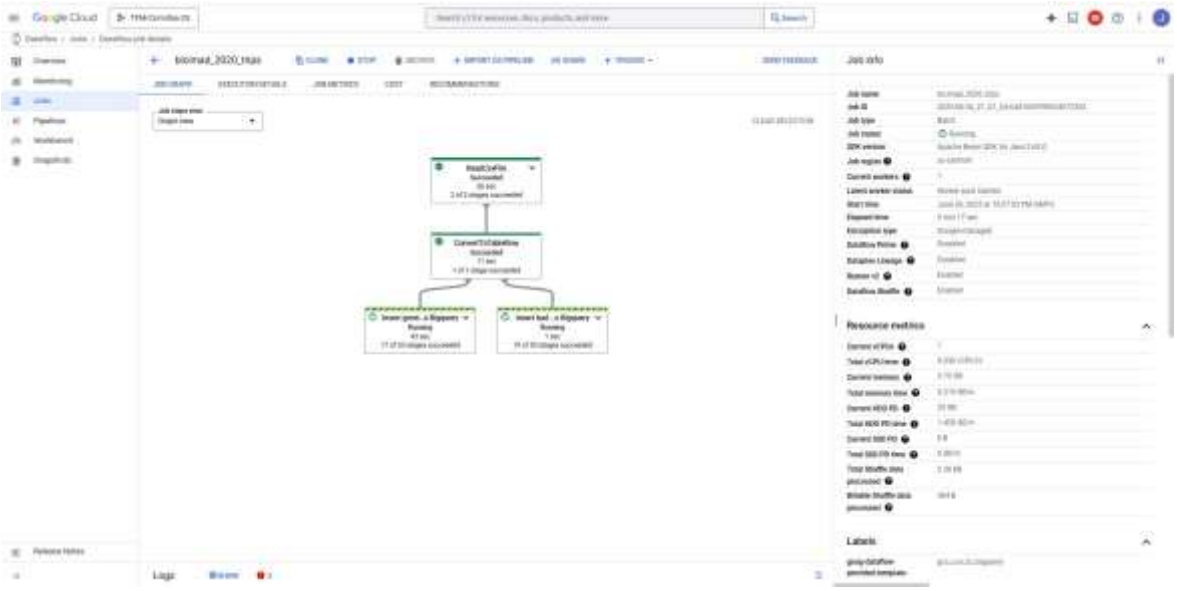


Figura 32. Job de Dataflow en ejecución. Fuente [32]

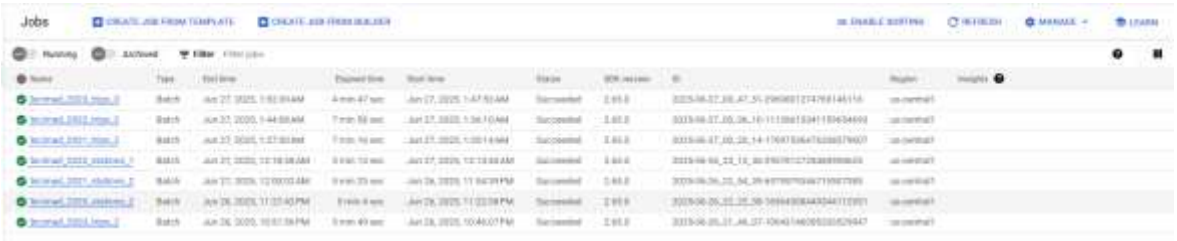


Figura 33. Listado de Jobs de Dataflow ejecutados para el proyecto. Fuente [32]

5.4.2.1 Transformación y limpieza

Durante la ejecución de los Jobs de Dataflow, no se implementaron transformaciones o limpieza adicionales sobre los datos.

5.4.3 BIGQUERY

La información puede llegar a BigQuery desde dos fuentes principales: archivos en Google Cloud Storage (GCS) o a través de pipelines como Dataflow.

- En el caso de Dataflow, los datos se cargan en BigQuery de forma nativa o física.

- En cambio, los archivos CSV provenientes de GCS se integran mediante tablas externas, es decir, los datos permanecen en el almacenamiento y no se cargan físicamente en BigQuery.

5.4.3.1 Tablas Curadas

Mediante BigQuery, se ha implementado **vistas** (definido mediante consultas SQL) para llevar a cabo la transformación y limpieza de los datos con el objetivo de presentar datos listos para su consumo. A continuación se enlistan las transformaciones y limpiezas efectuadas por cada tabla dentro de los datasets curados (Referencia en la Tabla 16). En el Anexo B se puede encontrar los modelos de cada tabla en la Tabla B. 23, Tabla B. 24, Tabla B. 25, Tabla B. 26 y Tabla B. 27.

aemet_clima_hoy

- Limpieza:
  - Conversión segura de `latitud` y `longitud` (Función `SAFE_CAST`).
  - Manejo de nulos en `precipitación` y `visibilidad`.
- Transformaciones:
  - Sustitución de valores nulos en `velocidad` y `dirección del viento` (Función `COALESCE`).
  - Extracción de fecha y hora desde `fint`.
  - Cálculo del rango térmico (`tamax - tamin`).
  - Clasificación de precipitación en `categorías` (sin lluvia, ligera, etc.).
  - Indicador de `visibilidad reducida` (< 1000 m).
  - Enriquecimiento con datos de estación mediante LEFT JOIN.

aemet_predicciones

- Limpieza:

- Conversió'n segura (`SAFE_CAST`) de múltiples campos numéricos (`temp_max`, `temp_min`, `hum_max`, `hum_min`, `sens_max`, `sens_min`, `precipitacion`) para asegurar su tipo
- Manejo de valores nulos en `precipitación`, `UV` y `viento` para permitir su clasificación.

- Transformaciones:
  - Extracción de fecha y hora a partir del campo fecha (tipo `timestamp`).
  - Cálculo de rango de temperatura (`temp_max - temp_min`).
  - Clasificación de la condición de lluvia esperada (`precipitacion`) en categorías: 'seco', 'lluvia ligera', 'moderada', 'lluvioso', 'sin datos'.
  - Categorización del riesgo `UV` en niveles desde 'bajo' hasta 'extremo'.
  - Clasificación de la `intensidad del viento` en niveles: 'desconocido', 'suave', 'moderado', 'fuerte', 'muy fuerte').
  - Cálculo de un índice compuesto de `riesgo climático` combinando valores altos de radiación UV y precipitaciones.
  - Enriquecimiento con datos (geográficos y altitud) de la tabla de estaciones de cada estación meteorológica. La unión es con base al nombre de provincia aplicando la transformación `LOWER()` para evitar datos perdidos

Viajes

- Limpieza:
  - Filtro de registros con `idBike` nulo para garantizar solo viajes válidos.
  - Conversión segura de fechas (`SAFE.PARSE_DATE`) y timestamps (`SAFE.PARSE_TIMESTAMP`) para evitar errores por formatos inválidos.
  - Unificación del tipo de dato `dock_unlock` para 2022 (CAST AS FLOAT64) para mantener consistencia.
  - Estandarización de campos categóricos (`locktype, unlocktype`) a mayúsculas (`UPPER()`).
- Transformaciones:

- Unión de datasets históricos (2021, 2022, 2023) mediante UNION ALL para formar un histórico consolidado. Si bien el año 2020 está definido en el alcance de este proyecto, los datos de dicho año contienen un esquema distinto a los años 2021, 2022, 2023. Dado que esto es desde la fuente original, no puede modificarse o acoplarse al esquema de la mayoría de años disponibles por lo que, de momento, se tratará futuramente de otra manera.
- Conversión de fechas y horas de desbloqueo y bloqueo a formato DATETIME en zona horaria 'Europe/Madrid'.
- Cálculo de duración real del viaje (`duracion_minutos_real`) mediante la diferencia de timestamps.
- Clasificación de viajes como 'válido' o 'inválido' según reglas de duración mínima y consistencia entre fechas.
- Clasificación del tipo de viaje por duración (muy corto, corto, medio, largo).
- Extracción de latitud y longitud desde campos JSON anidados (`geolocation_unlock`, `geolocation_lock`) con la función `JSON_EXTRACT_SCALAR`.

viajes_hourly

- Limpieza:
  - Filtrado de registros para incluir solo estaciones activas (1)
- Transformaciones:
  - N/A

emt_lineas

- Limpieza:
  - Conversión de tipos para homogeneizar campos numéricos (`CAST` a `INT64` y `FLOAT64` en paradas, viajes y rutas).

  - Normalización de texto a mayúsculas (`UPPER`) para campos como `nameA`, `nameB`, `dayType`, `idDayType`, `nameSectionA`, `nameSectionB`.
  - Filtro de consistencia en las uniones mediante condiciones que comparan encabezados (`headerA/headerB`) con nombres de línea (`nameA/nameB`) ignorando mayúsculas/minúsculas.
- Transformaciones:
  - Mapeo de campos de dirección de tipo string a enteros (`A/B`, `Direction1/Direccion2`) para facilitar operaciones de comparación y análisis.
  - Conversión de campos de fecha y hora (`TIME()` y `CAST`) para estandarizar formatos.
  - Unión lógica basada en múltiples condiciones clave como `línea`, `dirección`, `dayType` y encabezados para enriquecer la información de cada línea con sus respectivas paradas, viajes y segmentos de ruta.

5.4.4 AUTOMATIZACIÓN

Para automatizar el proceso, se implementaron *triggers* de Cloud Run que inician automáticamente la ingesta de datos mediante una calendarización basada en cron o llamadas programadas. Al configurar los triggers, indirectamente se utilizan *Jobs* de Cloud Scheduler.

Estos triggers gestionan la ejecución de las canalizaciones a través de contenedores que se activan bajo demanda, optimizando recursos y tiempos de respuesta.

Cuando se activan los Jobs de Cloud Scheduler, activan los triggers de Cloud Run en intervalos regulares. Así, se garantiza una ingesta programada y continua, acorde con la frecuencia de generación de los datos.

En las tablas de la sección 5.2.6.2 y 5.2.7.1, están descritos los horarios de extracción de cada Job de Cloud Run. En la Figura 34 se ve el listado de Jobs de Cloud Scheduler, luego de configurar los triggers para cada Job de Cloud Run.

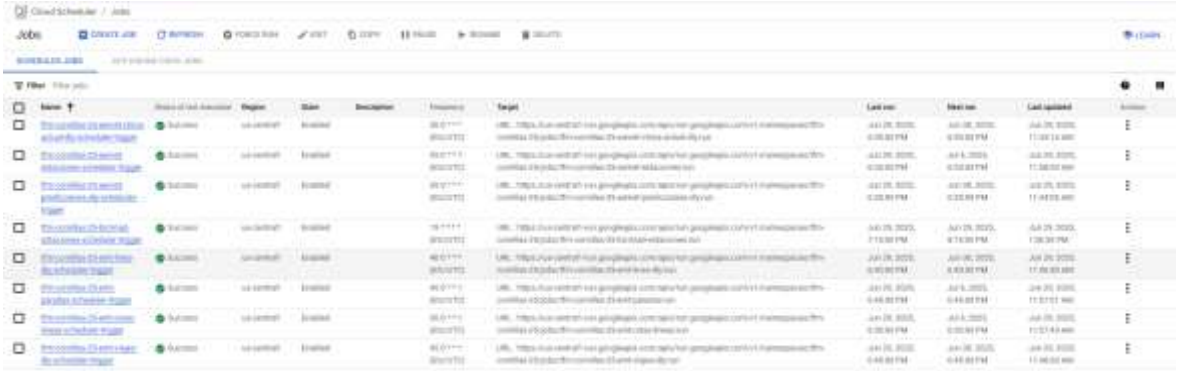


Figura 34. Jobs de Cloud Scheduler correspondientes a cada Job de Cloud Run. Fuente [32]

5.4.4.1 Configuración

Al configurar configurar un trigger para el Job de Cloud Run, basta con ir a la configuración de cada Job y agregar un *scheduler trigger* como se muestra en la Figura 35.

Figura 35. Definición del trigger del Job de Cloud Run. Fuente [32]

Esta configuración fue realizada vía consola o UI por lo que en la Figura 37 se muestran los parámetros de configuración. Lo principal es la expresión cron que nos definirá cuando se iniciará cada Job. En la Figura 36, se selecciona la cuenta de servicio que se encargará de llevar a cabo la ejecución del Job en nuestro nombre. Para encontrar más detalles de la cuenta de servicio, se puede leer en el apartado 5.5.1.

Una vez establecido, se mostrará adjunto al Job de Cloud Run el Job de Cloud Scheduler como se muestra en la Figura 38.

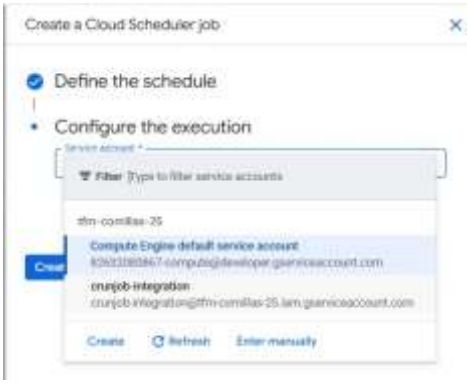


Figura 36. Configuración de la cuenta de servicio para la ejecución del Job de Cloud Run. Fuente [32]

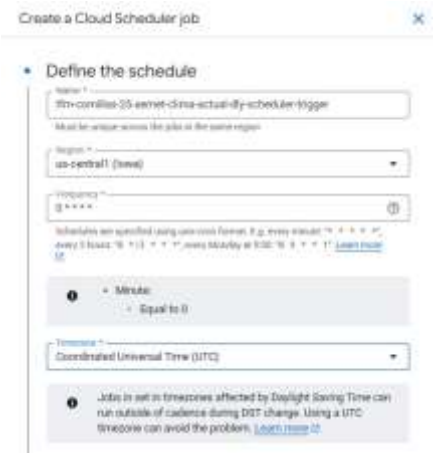


Figura 37. Configuración de trigger del Job de Cloud Run. Fuente [32]

Figura 38. Configuración final del trigger del Job de Cloud Run. Fuente [32]

Finalmente, en la Figura 39, se muestra un ejemplo de ejecución automatizada de Jobs cada hora de acuerdo al programa de calendarización del Job: *bicimad-estaciones*. En la se puede mirar los objetos creados en el bucket (ver Figura 40) como resultado final de cada ejecución y alimentando la tabla cruda *emt_api_estaciones* de BigQuery.

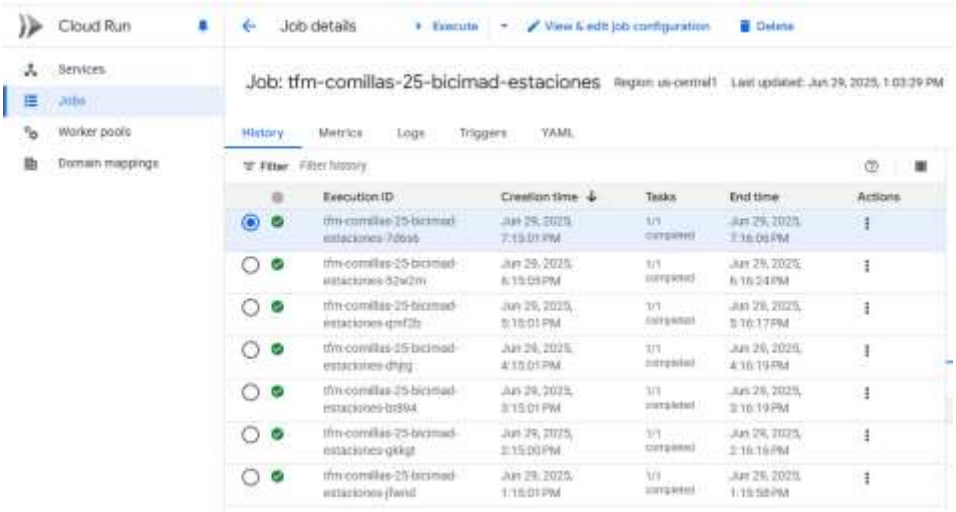


Figura 39. Ejecuciones calendarizadas para el Job de bicimad-estaciones. Fuente [32]

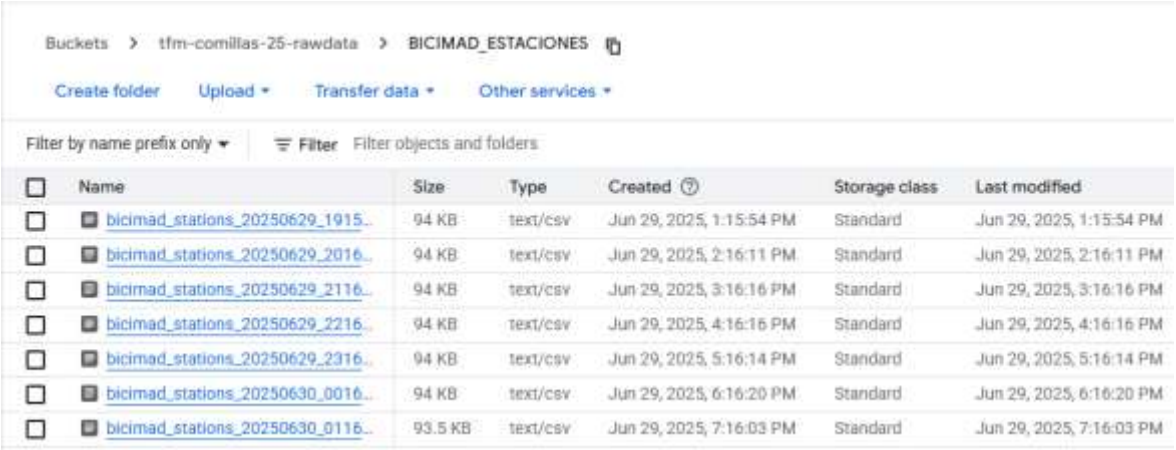


Figura 40. Objetos CSV como resultado de la ejecución automatizada del Job bicimad-estaciones. Fuente [32]

5.5 IMPLEMENTACIÓN DE GOBIERNO

El gobierno del dato es una parte fundamental dentro de cualquier arquitectura moderna de datos. En un entorno como el que plantea este proyecto, basado en la integración de múltiples fuentes abiertas, automatizaciones, es ideal asegurar que los datos estén bien documentados, protegidos y accesibles solo a los roles adecuados.

La estrategia de gobierno del dato implementada en esta solución se apoya en tres pilares principales:

- Gestión de identidades y accesos (IAM)
- Catalogación de activos de datos (Data Catalog)
- Gestión segura de credenciales y claves (Secret Manager)

5.5.1 IAM

Se ha implementado IAM en el proyecto de forma que podamos preguntar quién puede acceder y a qué recursos dentro del proyecto, en función de roles predefinidos o personalizados. Esta configuración garantiza la seguridad del sistema y permite aplicar el principio de mínimos privilegios.

5.5.1.1 Prácticas aplicadas

- IAM configurado a nivel de dataset en BigQuery y bucket en GCS.
- Se evitaron roles de editor a nivel de proyecto para reducir exposición.
- Si bien se recomienda el uso de **grupos** de acuerdo a un rol definido, y relacionar cada usuario a un respectivo grupo, las creaciones de estos quedan fuera del alcance de este proyecto puesto que las configuraciones de grupos son exclusivas de cuentas empresariales u organizaciones dentro de Google Cloud (de acuerdo al primer punto de la sección 4.2)

5.5.1.2 Principals

Un principal es aquel identificador que con roles y permisos establecidos, puede hacer uso de GCP. Entre los principals comunes, se encuentran:

- Usuario (relacionado a una cuenta Google)
- Cuenta de Servicio
- Grupos

A continuación, en la Tabla 17, se describen los principals definidos para este proyecto excluyendo los principals por defecto que son generados de manera automática para la comunicación entre los diferentes servicios.

Principal	*Tipo de Principal*	*Función en el proyecto*
crunjob-integration	Cuenta de Servicio	Cuenta de servicio encargada de ejecutar los Jobs automatizados para la descarga de datos
desarrollador	Cuenta de Servicio	Cuenta de servicio configurada para interactuar como desarrollador solamente con los servicios implementados en la arquitectura.
jhuchim	Usuario	Principal de usuario de cuenta Google declarado como owner o propietario del proyecto

Tabla 17. Principals definidos para el proyecto

5.5.1.3 Almacenamiento

Los buckets de Google Cloud Storage (GCS) y los datasets de BigQuery pueden gestionarse de forma independiente mediante políticas de acceso basadas en principals, que incluyen usuarios individuales (cuentas de Google), grupos o cuentas de servicio. En el contexto de este proyecto, los permisos sobre los buckets de GCS (ver Figura 41) y los datasets de

BigQuery (ver Figura 42) se han asignado exclusivamente a las cuentas de servicio específicas utilizadas en la arquitectura, al usuario propietario del proyecto y a las cuentas de servicio asociadas a cada componente implementado (este último de forma automática al activar las APIS de cada servicio y entablar relación entre cada uno) garantizando así un control de acceso basado en el principio de mínimo privilegio.

Figura 41. Lista de principals con interacción en el bucket de GCS. Fuente [32]

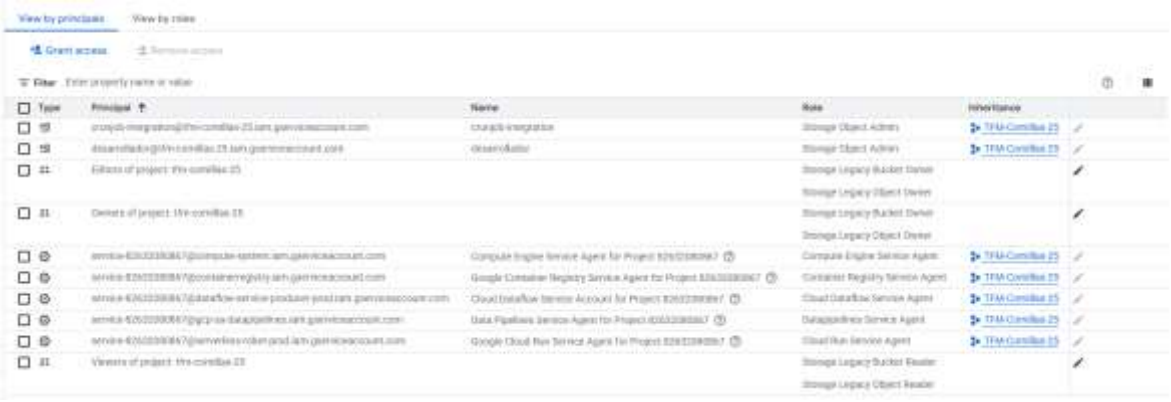


Figura 42. Lista de principals con interacción en cada dataset de BigQuery. Fuente [32]

5.5.1.4 Cuentas de servicio

Para fines de este proyecto, se han generado dos Cuentas de servicio. El detalle se muestra a continuación en la Tabla 18.

Cuenta de Servicio	*Roles*
crunjob-integration	Cloud Run Invoker Cloud Run Viewer Logs Writer Secret Manager Secret Accessor Secret Manager Secret Version Adder Secret Manager Viewer Storage Object Admin
desarrollador	Artifact Registry Administrator Artifact Registry Writer BigQuery Data Editor BigQuery Data Owner BigQuery Job User Cloud Run Admin Dataflow Developer Eventarc Admin Secret Manager Secret Accessor Secret Manager Viewer

	Storage Object Admin

Tabla 18. Cuentas de servicio.

5.5.1.5 Roles

En la Tabla 19, se describe brevemente la funcionalidad de los Roles dentro de GCP [37].

Rol	*Nombre*	*Función*
roles/artifactregistry.admin	Artifact Registry Admin	Acceso de administrador para crear y administrar repositorios.
roles/artifactregistry.writer	Artifact Registry Writer	Acceso para leer y escribir elementos del repositorio.
roles/bigquery.dataEditor	BigQuery Data Editor	Leer, insertar, actualizar y borrar datos en tablas.
roles/bigquery.dataOwner	BigQuery Data Owner	Crear, modificar o borrar datasets; gestionar permisos de
roles/bigquery.jobUser	BigQuery Job User	Proporciona permisos para ejecutar trabajos, incluidas consultas, dentro del proyecto.
roles/run.admin	Cloud Run Admin	Control total sobre todos los recursos de Cloud Run.
roles/run.invoker	Cloud Run Invoker	Puede invocar servicios de Cloud Run y ejecutar trabajos de Cloud Run.

roles/run.viewer	Cloud Run Viewer	Puede ver el estado de todos los recursos de Cloud Run, incluidas las políticas de IAM.
roles/dataflow.developer	Dataflow Developer	Proporciona los permisos necesarios para ejecutar y manipular trabajos de Dataflow.
roles/eventarc.admin	Eventarc Admin	Control total sobre todos los recursos de Eventarc.
roles/secretmanager.secretAccessor	Secret Manager Secret Accessor	Permite acceder a la carga útil de secretos.
roles/secretmanager.secretVersionAdder	Secret Manager Secret Version Adder	Permite agregar versiones a secretos existentes.
roles/secretmanager.viewer	Secret Manager Viewer	Permite visualizar metadatos de todos los recursos de Secret Manager
roles/logging.logWriter	Logs Writer	Proporciona los permisos para escribir entradas de registro.
roles/storage.objectAdmin	Storage Object Admin	Otorga control total sobre los objetos, incluyendo enumerarlos, crearlos, visualizarlos y eliminarlos.

Tabla 19. Descripción de Roles

5.5.2 DATA CATALOG

Tal como se describió en el punto 2.4.8, es una herramienta clave para implementar documentación, linaje y trazabilidad sobre los conjuntos de datos.

5.5.2.1 Metadada

- Se aplicaron etiquetas y descripciones a las tablas en curated_mobility.
- Se ha reservado un catálogo orientado a Gobierno de Datos

En la Tabla 20, se describen los campos implementados en el catálogo creado.

Nombre	*Tipo*	*Descripción*	*Valor por defecto*	*Presencia*
Business Owner	Text	Nombre del propietario		Requerido
Data Lifecycle	Enum	Etapa del ciclo de vida	TEST, DEV, QA, PRODUCTION, OTHER, and DEPRECATED	Requerido
Is Encrypted	Boo	Especifica si el dataset está cifrado		Requerido
Has PII	Boll	Especifica si el dataset contiene datos sensibles		Requerido

Tabla 20. Metadata del catálogo de Gobierno de Datos elaborado

5.5.2.2 Procedimiento

Creación del Catálogo de Datos usando plantilla de Gobierno de Datos

1. Acceder a la interfaz de administración de Data Catalog.
2. Crear un nuevo catálogo de datos seleccionando la opción de plantilla

3. En el selector de plantillas, elegir la plantilla denominada *Data Governance*, como se muestra en la Figura 43.
4. Revisar y definir las etiquetas a agregar de las sugeridas por la plantilla. En el caso de este proyecto, se han seleccionado los campos descritos en la tabla Tabla 20. En la Figura 44, se puede revisar los cambios establecidos. No olvidar especificar el tipo de dato para cada etiqueta (cadena, número, booleano, etc.).
5. Una vez establecido las configuraciones, se confirma que el catálogo creado esté identificado como un catálogo de tipo Gobierno de Datos. Esto puede verse en la Figura 45.

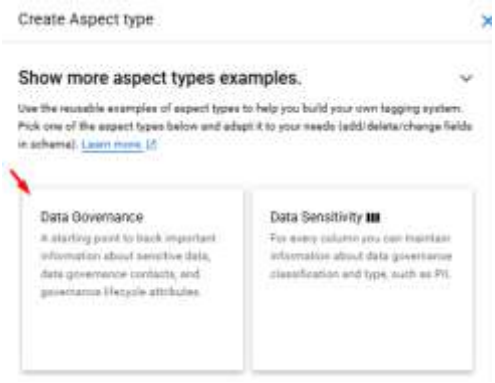
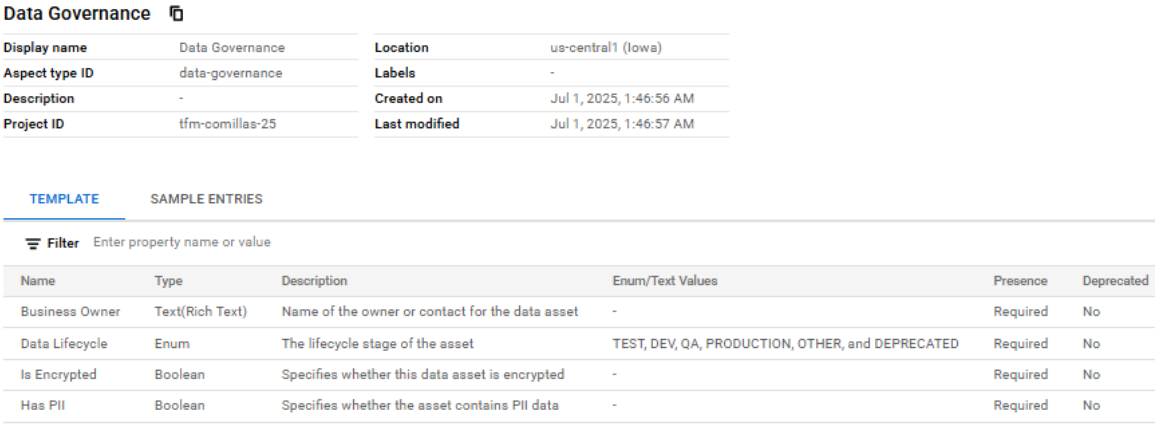


Figura 43. Configuración de Catálogo de Datos mediante template de Data Governance. Fuente [32]

Figura 44. Configuración de etiquetas. Fuente [32]



Data Governance

Display name	Data Governance	**Location**	us-central1 (Iowa)
Aspect type ID	data-governance	**Labels**	-
Description	-	**Created on**	Jul 1, 2025, 1:46:56 AM
Project ID	tfm-comillas-25	**Last modified**	Jul 1, 2025, 1:46:57 AM

TEMPLATE | SAMPLE ENTRIES

Filter: Enter property name or value

Name	Type	Description	Enum/Text Values	Presence	Deprecated
Business Owner	Text(Rich Text)	Name of the owner or contact for the data asset	-	Required	No
Data Lifecycle	Enum	The lifecycle stage of the asset	TEST, DEV, QA, PRODUCTION, OTHER, and DEPRECATED	Required	No
Is Encrypted	Boolean	Specifies whether this data asset is encrypted	-	Required	No
Has PII	Boolean	Specifies whether the asset contains PII data	-	Required	No

Figura 45. Configuración de Catálogo de Datos de tipo Gobierno de Datos. Fuente [32]

Asignación del catálogo a la vista de datos

1. Desde Data Catalog, en la sección de buscador, podemos encontrar el activo de datos a aplicar al modelo. Para fines de este proyecto, aplicaremos el catálogo a la vista de *viajes_hourly* y seleccionar la opción para asignar catálogo de datos y se elige el previamente configurado (tipo Gobierno de Datos), y confirmar la asignación, como se puede ver en la Figura 46.
2. Con el catálogo ya asignado, se procede a asignar valores específicos a cada una de las etiquetas configuradas para la vista viajes_hourly, tal como se muestra en la Figura 47.
3. Con los cambios establecidos, puede verificarse la relación entre Vista y Catálogo puesto que estará relacionado la vista en la metadata del catálogo como se muestra en la Figura 48.

Esta relación permitirá gestionar la información conforme a los lineamientos de gobernanza establecidos, facilitando su trazabilidad, clasificación y control de acceso.

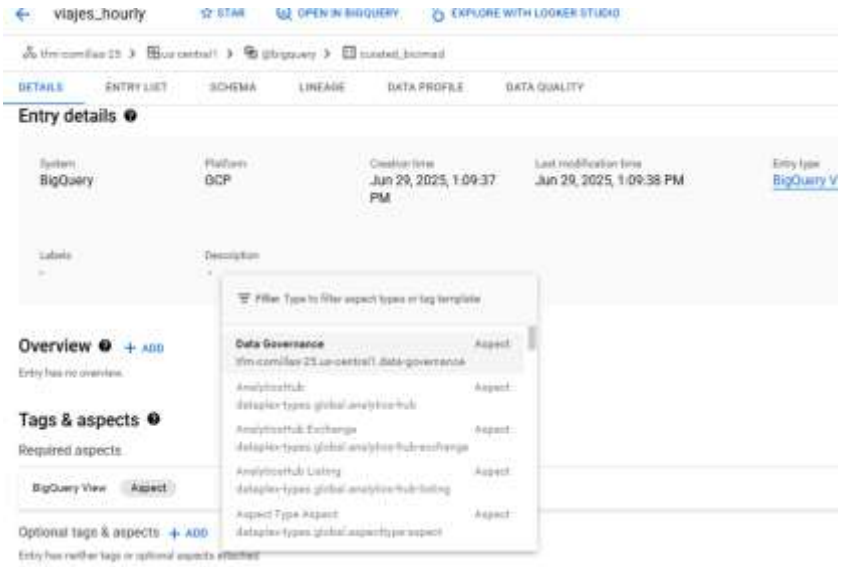


Figura 46. Asignación de Catálogo de Datos a la vista: viajes_hourly. Fuente [32]

Figura 47. Asignación de valores a las etiquetas de la vista. Fuente [32]

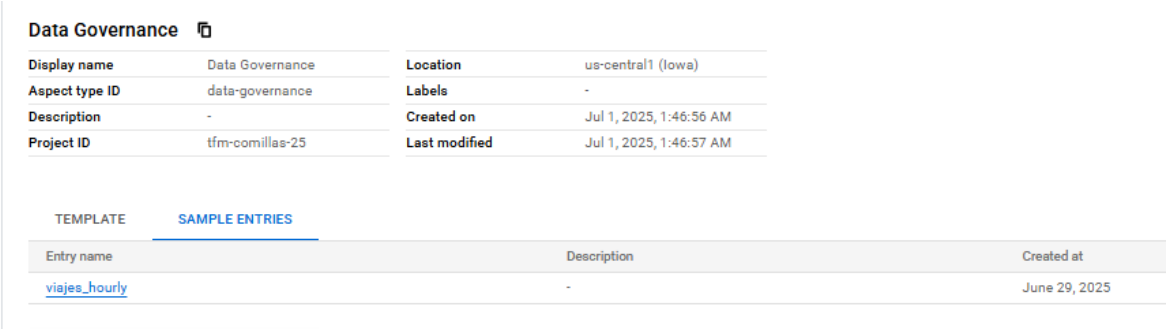


Figura 48.Vista viajes_hourly relacionada con el catálogo Data Governance. Fuente [32]

5.5.3 SECRET MANAGER

Es utilizado para almacenar de forma segura y gobernar las credenciales sensibles, como claves de API o tokens de autenticación necesarios para acceder a fuentes externas como EMT o AEMET.

5.5.3.1 Prácticas aplicadas

Los secretos fueron almacenados en versiones para permitir rotación segura gestionados y encriptados bajo la administración de Google.

Para el proyecto, se estará utilizando 3 secretos descritos en la Tabla 21:

Secreto	*Descripción*
aemet_token	Token de uso de la API de AEMET
emt_password	Contraseña de inicio de sesión y autenticación en la API de la EMT
emt_user	Correo electrónico de inicio de sesión y autenticación en la API de la EMT

Tabla 21. Diccionario de Secretos.

Capítulo 6. EVALUACIÓN DE FORMATOS

El siguiente punto del proyecto es evaluar el impacto de distintos formatos de almacenamiento (CSV, Avro, Parquet) en el rendimiento de consultas analíticas sobre grandes volúmenes de datos urbanos, almacenados en GCS y procesados en BigQuery mediante referencias externas. Esta evaluación tiene como fin identificar el formato más eficiente en cuanto a velocidad, compresión y coste computacional, con base en escenarios realistas obtenidos a partir de los datos de movilidad superficial urbana integrados en la arquitectura.

6.1 PRUEBAS EJECUTADAS EN DIFERENTES ESCENARIOS

Para evaluar el rendimiento de los formatos CSV, Avro y Parquet, se ejecutaron una serie de consultas con un dataset de un tamaño de 2.47GB de peso. Las pruebas se realizaron en BigQuery. La consulta efectuada, es la definida en la Tabla: Viajes. En el Anexo A (punto 2c, carpeta *elt/bq-rendimiento-formatos*) se encuentra la consulta y para más detalles de la tabla, puede consultarse el capítulo 5.4.2.1.

Escenario base:

- Datos de viajes históricos de BiciMAD (2021- 2022).
- Datos exportados en los tres formatos: CSV, Avro y Parquet.
- Volumen de prueba: 2.47GB.
- Carga en GCS y posterior consulta en BigQuery.

Variables evaluadas:

- Tiempo de consulta: Medido con EXPLAIN en BigQuery.
- Tamaño en disco: Comparado directamente en GCS por archivo.
- Coste de escaneo: Reportado por BigQuery en bytes leídos.

- Soporte de esquema: Según el archivo.
- Lectura por columnas: Según el archivo.
- Soporte para compresión: Según el archivo.

6.2 *COMPARACIÓN DE RESULTADOS Y ANÁLISIS*

Los resultados obtenidos muestran diferencias relevantes entre los formatos evaluados. Se muestran a continuación en la Tabla 22. En el Anexo C, se encuentran la Figura C. 49, Figura C. 50 y Figura C. 51 con las ejecuciones efectuadas.

Endpoint	*CSV*	*Parquet*	*Avro*
Tiempo de consulta (seg)	23	32	19
Tamaño en disco (Bytes shuffled)	1.92 GB	2.19 GB	2.47 GB
Slot time consumed	5 min 25 seg	3 min 24 seg	9 min 57 seg
Soporte de esquema	NO	SI	SI
Lectura por columnas	NO	SI	NO
Soporte para compresión	NO	SI	SI

Tabla 22. Tabla de información de las paradas.

6.2.1 ANÁLISIS

6.2.1.1 Tiempo de consulta

Avro es el más rápido en tiempo de ejecución (19 segundos), seguido por CSV (23s), y luego Parquet (32s). Sin embargo, el tiempo por sí solo no representa eficiencia si el slot time y el coste son más altos.

6.2.1.2 Consumo de recursos

- Parquet destaca con el menor slot time (3 min 24s), lo cual es deseable.
- CSV intermedio (5 min 25s).
- Avro es el más costoso en slots (9 min 57s), posiblemente por el mayor volumen de datos y falta de lectura por columnas.

6.2.1.3 Volumen de datos procesado (bytes shuffled)

- CSV genera el menor volumen (1.92 GB).
- Parquet queda en medio (2.19 GB).
- Avro el mayor (2.47 GB), lo que aumenta su coste potencial.

6.2.1.4 Compatibilidad técnica

- Parquet es el más completo: compresión, lectura por columnas y esquema.
- Avro soporta esquema y compresión, pero no lectura por columnas.
- CSV es el menos flexible, lo que puede provocar errores o necesidad de validaciones adicionales.

6.3 RECOMENDACIONES

A partir de los resultados obtenidos, se proponen las siguientes recomendaciones técnicas para arquitecturas Lakehouse independientemente que utilicen datos de movilidad urbana:

- Utilizar Parquet como formato, especialmente en donde se realizarán consultas frecuentes en BigQuery. Utilizar Parquet como formato si se realizan consultas más frecuentes. Si bien su tiempo de consulta no es el menor, la lectura por, menor uso de slots y soporte de compresión lo hacen ideal para consultas complejas y almacenamiento eficiente.
- Conservar los datos en CSV solo en la capa Raw ya que es la fuente original y disponible en texto plano.

- Evitar CSV en producción: Aunque es soportado y legible, no soporta compresión, esquemas, lectura por columnas, y puede generar errores de formato o encoding.

Capítulo 7. ANÁLISIS DE RESULTADOS

Luego del desarrollo de este proyecto, se identifican los siguientes puntos:

- El enfoque por capas: Raw - Curated dentro de una arquitectura de tipo Lakehouse, permite estructurar el flujo de datos de manera uniforme para completar el ciclo de ingeniería desde la extracción desde la fuente hasta la información lista para su consumo en el alojamiento en la capa curada.
- La integración de diversas fuentes de datos, han habilitado un enriquecimiento de los mismos permitiendo un descubrimiento mayor de lo estimado y esto puede dar pie a disponer de un mayor número de casos de uso.
- El formato Parquet demostró ser el más adecuado para cargas analíticas en BigQuery. Al mantener una estructura columnar y compresión, reducen el coste y el tiempo de ejecución de las consultas, especialmente en datasets grandes como los de movilidad urbana.
- Las herramientas sin servidor de Google Cloud Platform como Cloud IAM, Data Catalog y Google Secrets Manager, permiten disponer de una configuración de alto nivel para temas de gobierno de datos lo que hace que sea una configuración sencilla, flexible y que su implementación es limpia.
- De acuerdo a la configuración de uno de los endpoints de la EMT respecto a BiciMAD, la frecuencia de actualización es de cada hora lo que acerca a una actualización lo más cercano a tiempo real por lo que en el establecimiento de casos de uso con esta fuente, proveerá un nutrido contenido de datos.

Dentro del éxito encontrado en el proyecto, se encontraron ciertas limitaciones que se describen a continuación:

- En las APIs de EMT o AEMET, tienen políticas de uso limitado por token y no exponen históricos completos, lo cual limita la disponibilidad y el tener un estado de datos reciente.

- Se encontró un año de información de viajes de BiciMAD con un esquema diferente al resto de años descargados lo que redujo el sizing establecido.

Capítulo 8. CONCLUSIONES Y TRABAJOS FUTURO

Este trabajo ha demostrado la viabilidad y el valor técnico de diseñar e implementar una arquitectura moderna tipo Lakehouse para integrar y analizar datos abiertos de movilidad superficial urbana en un entorno cloud.

A lo largo del desarrollo, se lograron los siguientes objetivos:

- Se diseñó e implementó una arquitectura escalable, modular y gobernada, basada en capas lógicas (landing, raw, curated)
- Se han integrado múltiples fuentes de datos abiertas como datos históricos de viajes y estatus de estaciones (BiciMAD), información operativa de autobuses EMT y condiciones meteorológicas (AEMET), demostrando la capacidad del sistema para combinar múltiples orígenes con distintas estructuras y frecuencias.
- Se aplicaron buenas prácticas de ingeniería de datos respecto a la orquestación de extracciones e implementación de canalizaciones.
- Se ha aplicado una estrategia básica de gobierno del dato, con uso de etiquetas, catálogo de datos y control de accesos IAM, que permite un etiquetado normalizado y una gestión acorde de las credenciales.
- Se evaluó el uso de distintos formatos de almacenamiento (CSV, Avro y Parquet) sobre el rendimiento de las consultas analíticas, concluyendo que Parquet es el formato más adecuado en términos de compresión, coste y eficiencia para entornos analíticos distribuidos.

8.1 APORTACIONES

- Se aporta una arquitectura replicable y escalable que puede aplicable a otros ambientes y/o casos de uso y no precisamente de entornos urbanos con datos abiertos.
- Se sientan las bases para una futura plataforma de análisis urbano

8.2 TRABAJOS FUTUROS:

Procesamiento en tiempo real: incluir una capa de streaming que permitiría escalar la arquitectura para disponibilidad datos de manera inmediata, es decir, en tiempo real.

Datos: Agregar datos demográficos y/o topografía para enriquecer y complementar la información proveniente de los medios de transporte.

Formatos de almacenamiento: Incluir en la arquitectura una lógica automatizada de conversión que transforme los archivos CSV de la capa raw a formatos más eficientes como Parquet y formar parte de la capa raw de BigQuery además de integrar almacenamiento físico y no externo mediante consultas federadas.

Agregar y/o cambiar otras herramientas de procesamiento: Incluir otras herramientas de procesamiento y gestión de pipelines a una solución más visual y declarativa como Cloud Data Fusion, lo cual facilitaría la gestión y mantenimiento para equipos menos técnicos, y permitiría construir canalizaciones con menor esfuerzo de desarrollo.

Redes: Incorporar en la arquitectura un módulo de red que ubique el Data Lake dentro de una VPC. A través de reglas de firewall y configuración de routers, se deberá permitir el acceso controlado a Internet y restringir la apertura de puertos no autorizados.

IaS: Implementación de Terraform para un despliegue de la arquitectura de forma automatizada.

Capítulo 9. BIBLIOGRAFÍA

[1] Comunidad de Madrid. "Muévete en transporte público". Comunidad de Madrid. https://www.comunidad.madrid/servicios/transporte/muevete-transporte-publico

[2] Huchim Vela J. *Análisis de patrones del sistema de transporte público BiciMAD*. Universidad Pontificia Comillas. Madrid. Junio 2020

[3] Consorcio Regional de Transportes de Madrid. "Autobuses EMT". Consorcio Regional de Transportes de Madrid. s.f. https://www.crtm.es/tu-transporte-publico/autobuses-emt.aspx

[4] Guisasola, C. "Madrid amplía su red eléctrica con la línea 75 de autobús". El Mundo. Octubre, 2024. https://www.elmundo.es/madrid/2024/10/21/67169cc7e9cf4a24278b4594.html

[5] EMT Madrid. "La línea 75 de autobús se suma a la red 100% eléctrica". EMT Madrid. Febrero, 2022. https://www.emtmadrid.es/Noticias/La-linea-75-de-autobus-se-suma-a-la-red-100-elec.aspx

[6] Bonilla, A. "Adiós a BiciMAD gratis: el truco para seguir usando el servicio sin coste". El Español. Febrero, 2024. https://www.elespanol.com/madrid/sociedad/20240201/adios-bicimad-gratis-truco-puedas-seguir-usando-servicio-sin-coste/829417183_0.html

[7] Reitz, K. "Requests: HTTP for Humans". Python Requests Documentation. s.f https://requests.readthedocs.io/en/latest/

[8] Pandas. "Pandas Documentation". pandas.pydata.org. s.f. https://pandas.pydata.org

[9] Google Cloud. "Data Fusion". Google Cloud. s.f. https://cloud.google.com/data-fusion

[10] Google Cloud. "BigQuery". Google Cloud. s.f. https://cloud.google.com/bigquery

[11] Google Cloud. "Cloud Storage". Google Cloud. s.f. https://cloud.google.com/storage

[12] Google Cloud. "Secret Manager". Google Cloud. s.f. https://cloud.google.com/secret-manager

[13] Google Cloud. "Cloud Run". Google Cloud. s.f. https://cloud.google.com/run

[14] Google Cloud. "Artifact Registry". Google Cloud. s.f. https://cloud.google.com/artifact-registry

[15] Google Cloud. "Identity and Access Management (IAM)". Google Cloud. s.f. https://cloud.google.com/iam

[16] Google Cloud. "Google Cloud Platform". Google Cloud. s.f. https://cloud.google.com/

[17] Docker. "Empowering App Development for Developers". Docker. s.f. https://www.docker.com/

[18] Vergadia, P. "Serverless vs Fully Managed: What's the Difference". Google Cloud Blog. s.f. https://cloud.google.com/blog/topics/developers-practitioners/serverless-vs-fully-managed-whats-difference

[19] Transport for London. "Keeping London Moving". TfL. s.f. https://tfl.gov.uk/

[20] Strava Metro. "Explore Metro Data". Strava. s.f. https://metro.strava.com/ea

[21] Keolis. "Sustainable Mobility Solutions". Keolis. s.f. https://www.keolis.com/en/

[22] Menzinsky, S. "¿Se pueden calcular las horas-hombre de un proyecto Scrum?". Scrum Menzinsky. Julio, 2014. https://scrum.menzinsky.com/2014/07/se-pueden-calcular-las-horashombre-de.html

[23] Google Cloud. "¿Qué es un Data Lake?". Google Cloud Learn. s.f. https://cloud.google.com/learn/what-is-a-data-lake?hl=es-419

[24] Google Cloud. "¿Qué es un Data Warehouse?". Google Cloud Learn. s.f. https://cloud.google.com/learn/what-is-a-data-warehouse?hl=es

[25] Google Cloud. "¿Qué es un Data Lakehouse?". Google Cloud Learn. s.f. https://cloud.google.com/discover/what-is-a-data-lakehouse?hl=es-419

[26] Apache Software Foundation. "Apache Parquet". Apache Parquet. s.f. https://parquet.apache.org/

[27] Apache Software Foundation. "Apache Avro Documentation". Apache Avro. s.f. https://avro.apache.org/docs/

[28] Google Cloud. "Dataflow: Overview". Google Cloud Documentation. s.f. https://cloud.google.com/dataflow/docs/overview?hl=es-419

[29] Google Cloud. "Data Catalog: Overview". Google Cloud Documentation. s.f. https://cloud.google.com/data-catalog/docs/concepts/overview?hl=es-419

[30] EMT Madrid. "Portal de Datos Abiertos". datos.emtmadrid.es. s.f. https://datos.emtmadrid.es/

[31] EMT Madrid. "Históricos de BiciMAD (2017–2023)". datos.emtmadrid.es. s.f. https://datos.emtmadrid.es/dataset/historicos-de-bicimad-2017_2023

[32] Google Cloud. "Consola". Google Cloud Console. s.f. https://console.cloud.google.com

[33] AEMET. "Centro de Descargas". AEMET Open Data. s.f. https://opendata.aemet.es/centrodedescargas/productosAEMET?

[34] AEMET. "Portal de Datos Abiertos". AEMET. s.f. https://www.aemet.es/es/datos_abiertos/AEMET_OpenData

[35] Google Cloud. "CSV to BigQuery Template". GitHub - GoogleCloudPlatform. s.f. https://github.com/GoogleCloudPlatform/DataflowTemplates/blob/main/v1/README_GCS_CSV_to_BigQuery.md

[36] Google Cloud. "Cuentas de servicio". Google Cloud Documentation. s.f. https://cloud.google.com/iam/docs/service-account-overview?hl=es-419

[37] Google Cloud. "Roles y permisos". Google Cloud Documentation. s.f. https://cloud.google.com/iam/docs/roles-permissions

[38] Google Cloud. "Estimador de Costes". Google Cloud Products Calculator. s.f. https://cloud.google.com/products/calculator/estimate-preview/CiRkOTkwMDZhMy02ZGIwLTRiZGQtODY0OS02NTIzM2I2YjhmMDYQAQ==?hl=es_419

ANEXO A

El repositorio que engloba todos los scripts implementados, pude encontrarse en este enlace: https://github.com/ArturoHuchim/tfm_comillas_25. Los scripts se han organizado de la siguiente forma:

1. app-el
    a. Dockerfile
    b. empaquetamiento.sh
    c. requirements.txt
    d. src
        i. aemet_clima_actual.py
        ii. aemet_estaciones.py
        iii. aemet_predicciones.py
        iv. bicimad_estaciones.py
        v. el.py
        vi. emt_lineas_diario.py
        vii. emt_paradas.py
        viii. emt_rutas_lineas.py
        ix. emt_viajes_diario.py
        x. main.py
        xi. push.py
    e. variables.sh
2. elt
    a. bq-curated
        i. curated_aemet_api.aemet_clima_hoy.sql
        ii. curated_aemet_api.aemet_predicciones.sql
        iii. curated_bicimad.viajes.sql
        iv. curated_bicimad.viajes_hourly.sql

v. curated_emt_api.emt_lineas.sql

b. bq-raw-schemas
   i. 2020_trips.json
   ii. 2021_trips.json
   iii. 2022_trips.json
   iv. 2023_trips.json
   v. stations.json

c. bq-rendimiento-formatos
   i. avro.sql
   ii. csv.sql
   iii. parquet.sql

d. dataflow-schemas
   i. error_dataflow.json
   ii. schema_bicimad_2020_trips.json
   iii. schema_bicimad_2021_trips.json
   iv. schema_bicimad_2022_trips.json
   v. schema_bicimad_2023_trips.json
   vi. schema_bicimad_stations.json

3. infra-servicios
   a. artifact_registry.sh
   b. bigquery.sh
   c. cloud_run.sh
   d. iam.sh
   e. scheduler.sh
   f. secrets_manager.sh

4. script-unificacion
   a. bicimad.py

ANEXO B

En los siguientes apartados se describen los modelos de los conjuntos de datos curados trabajados durante el proyecto:

Campo	*Tipo*	*Descripción*
fecha	DATE	Fecha de la medición
hora	TIME	Hora de la medición
estacion_id	STRING	Identificador de la estación meteorológica
estacion_nombre	STRING	Ubicación de la estación
provincia	STRING	Provincia donde reside la estación
latitud	FLOAT	Latitud de la estación
longitud	FLOAT	Longitud de la estación
altitud	INTEGER	Altitud de la estación
temp_min_horaria	FLOAT	Temperatura mínima
temp_max_horaria	FLOAT	Temperatura máxima
temp_actual	FLOAT	Temperatura Actual
humedad_relativa	FLOAT	Humedad relativa
viento_velocidad	FLOAT	Velocidad del viento

viento_direccion	FLOAT	Dirección del viento (N/Norte, NE/Nordeste, E/Este, SE/Sudeste, S/Sur, SO/Suroeste, O / Oeste, NO / Noroeste, C / Calma
rango_termico	FLOAT	Diferencia entre temperatura máxima y mínima
precipitacion_horaria	FLOAT	Valor de la precipitación
categoria_precipitacion	STRING	Categorización de precipitación (Desconocido, Sin lluvia, Ligero, Moderado, Intenso)
visibilidad_reducida	BOOLEAN	Indicador booleano de visibilidad reducida por menos de 1000m

Tabla B. 23.Tabla de información del modelo de datos de la tabla: curated_aemet_api.aemet_clima_hoy

Campo	*Tipo*	*Descripción*
estacion_nombre	STRING	Ubicación de la estación
provincia	STRING	Provincia donde reside la estación
ciudad	STRING	Nombre de la ciudad donde reside la estación
altitud	INTEGER	Altitud de la estación
fecha	DATE	Fecha de la estimación
hora	TIME	Hora de la estimación
temperatura_maxima	FLOAT	Temperatura máxima prevista
temperatura_minima	FLOAT	Temperatura Mínima prevista

sensasion_maxima	FLOAT	Sensación térmica máxima prevista
sensacion_minima	STRING	Sensación térmica mínima prevista
humedad_maxima	FLOAT	Humedad relativa máxima prevista
humedad_minima	STRING	Humedad relativa mínima prevista
latitud	STRING	Latitud de la estación
longitud	STRING	Longitud de la estación
rango_temperatura	FLOAT	Diferencia de temperaturas previstas
condicion_lluvia	STRING	Clasificación simple según precipitación esperada (Sin Dato, Seco, Lluvia ligera, Lluvia moderada, lluvioso)
riesgo_uv	STRING	Categorización del riesgo UV (Desconocido, Bajo, Moderado, Alto, Muy Alto, Extremo)
intensidad_viento	STRING	Clasificación de intensidad del viento (Desconocido, Suave, Moderado, Fuerte, Muy Fuerte)
riesgo_climatico	STRING	Clasificación de riesgo climático (Riesgo alto, Riesgo medio, Riesgo bajo)
estado_cielo	STRING	Periodo de validez para el estado del cielo (00-06, 06-12, 12-18, 18-24, 00-12, 00-24, 12-24
viento_direccion	STRING	Dirección del viento (N/Norte, NE/Nordeste, E/Este, SE/Sudeste, S/Sur, SO/Suroeste, O / Oeste, NO / Noroeste, C / Calma

Tabla B. 24. Tabla de información del modelo de datos de la tabla: curated_aemet_api.aemet_predicciones

Campo	*Tipo*	*Descripción*
fecha	DATE	
id_bici	INTEGER	Identificador de la estación de BiciMAD
id_flotilla	INTEGER	N/A
unlock_date	DATETIME	Timestamp que marca el desenganche de la bicicleta
lock_date	DATETIME	Timestamp que marca el enganche de la bicicleta
duracion_minutos_real	INTEGER	Diferencia entre unlock_date y lock_date para saber el tiempo exacto entre el desenganche y el enganche de la bicicleta.
trip_minutes	FLOAT	Tiempo total en minutos, entre el desenganche y el enganche de la bicicleta.
lon_unlock	FLOAT	Longitud de coordenada geográfica del desenganche de la bicicleta
lat_unlock	FLOAT	Latitud de coordenada geográfica del desenganche de la bicicleta
lon_lock	FLOAT	Longitud de coordenada geográfica del enganche de la bicicleta
lat_lock	FLOAT	Latitud de coordenada geográfica del enganche de la bicicleta

locktype	STRING	N/A
unlocktype	STRING	N/A
estatus_viaje	STRING	N/A
tipo_viaje	STRING	N/A

Tabla B. 25. Tabla de información del modelo de datos de la tabla: curated_bicimad.viajes

Campo	*Tipo*	*Descripción*
dock_bikes	INTEGER	Bicicletas enganchadas
free_bases	INTEGER	Bases disponibles
id	INTEGER	Identificador de la estación
light	INTEGER	Nivel de ocupación (0-bajo, 2-medio, 1-alto, 3-No disponible)
name	STRING	Nombre de la estación
total_bases	INTEGER	Número de bases de bicicleta de la estación
longitude	FLOAT	Longitud de coordenada geográfica de la cabecera
latitude	FLOAT	Latitud de coordenada geográfica de la cabecera
timestamp_	TIMESTAMP	Fecha y hora del evento

Tabla B. 26. Tabla de información del modelo de datos de la tabla: curated_bicimad.viajes_hourly

Campo	*Tipo*	*Descripción*
line	INTEGER	Número de la linea de autobús
nameA	STRING	Nombre de la cabecera de origen
nameB	STRING	Nombre de la cabecera de destino
dateRef	DATE	Fecha de referencia
idDayType	STRING	Tipo de día en función de la fecha. (LA, día de trabajo - lunes a viernes; SA, sábado; FE, festivo)
line_direction	INTEGER	B significa de A a B, A significa de B a A.
line_startTime	STRING	Tiempo de inicio de la línea
line_stopTime	STRING	Tiempo de finalización de la línea
line_minFreq	INTEGER	Frecuencia mínima de línea
line_maxFreq	INTEGER	Frecuencia máxima de línea
freqText	STRING	N/A
stop_id	INTEGER	Identificador de la parada
stop_name	STRING	Nombre de la parada
postal_address	STRING	Dirección
stop_longitude	FLOAT	Longitud de la parada
stop_latitude	FLOAT	Latitud de la parada
stop_startTime	STRING	Tiempo de inicio en la parada

stop_stopTime	STRING	Tiempo de finalización en la parada
stop_minFreq	INTEGER	Frecuencia mínima de la parada
stop_maxFreq	INTEGER	Frecuencia máxima de la parada
tripNum	INTEGER	Número de viaje
startTimeTrip	TIME	Viaje de inicio teórico
endTimeTrip	TIME	Viaje final teórico
logicBus	INTEGER	Identificación interna única en servicio simultáneo
trip_direction	INTEGER	Dirección de la ruta hecha en bicicleta. Valores: “AtoB” y “BtoA”
trip_dayType	STRING	Tipo de día en función de la fecha. (LA, día de trabajo - lunes a viernes; SA, sábado; FE, festivo)
segment_id	INTEGER	identificador
segment_name	STRING	Dirección
segment_distance	INTEGER	Distancia entre paradas
segment_longitude	FLOAT	Longitud
segment_latitude	FLOAT	Latitud

Tabla B. 27. Tabla de información del modelo de datos de la tabla: curated_emt_api.emt_lineas

ANEXO C

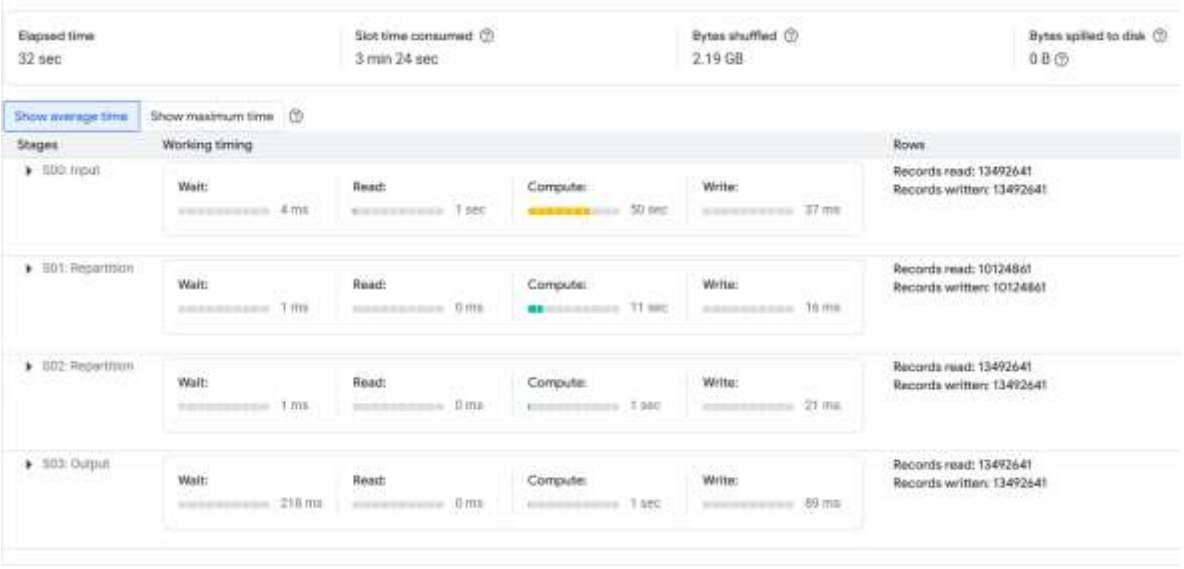


Figura C. 49. Detalle de ejecución de prueba de formato de datos: Parquet. Fuente: [32]

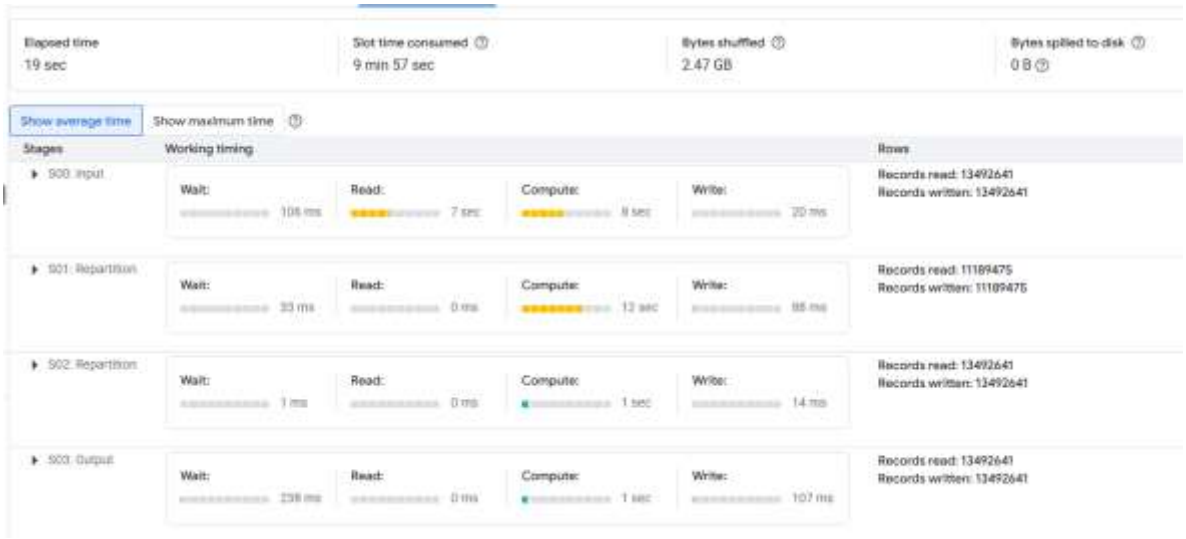


Figura C. 50. Detalle de ejecución de prueba de formato de datos: Avro. Fuente: [32]

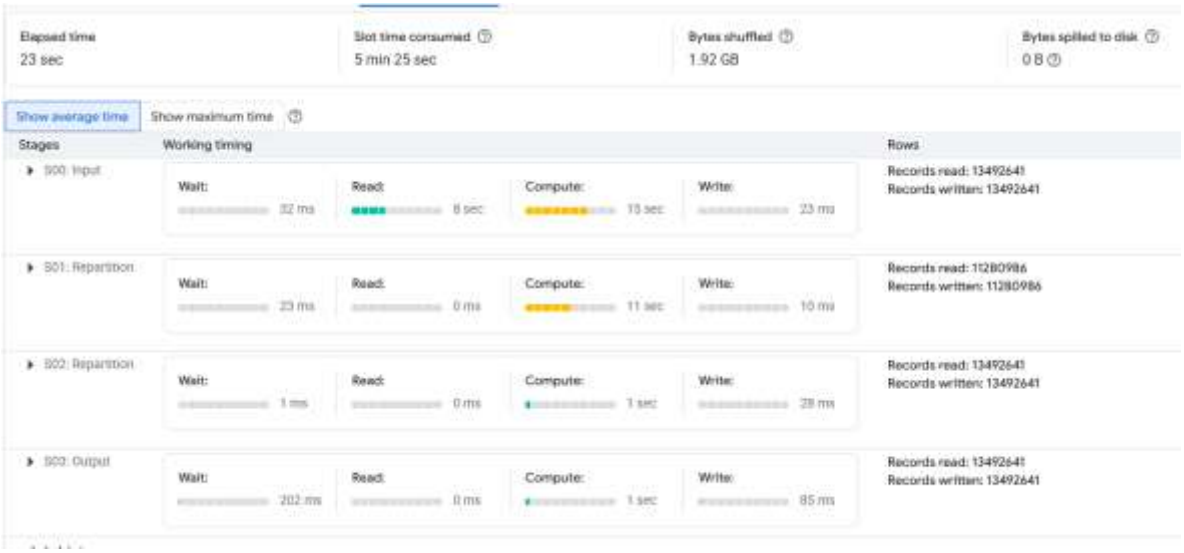


Figura C. 51. Detalle de ejecución de prueba de formato de datos: CSV. Fuente: [32]