DOCUMENTACIÓN: Trabajo de Fin de Máster

TÍTULO: Extracción y Orquestación de Datos en Azure aplicado al negocio (NMBS)

MODULE / COURSE : Máster Data Science, Big Data & Business Analytics 2024-2025

ALUMNO: Judith Castillo Martínez

DNI: 48656517-V

Índice

1.Resumen Ejecutivo

El presente trabajo desarrolla una plataforma de datos en la nube de Microsoft Azure, orientada a la ingesta, procesamiento y explotación de información operativa ferroviaria NBMS *(De Nationale Maatschappij der Belgische Spoorwegen (NMBS) — en francés Société Nationale des Chemins de fer Belges (SNCB)* — es la Sociedad Nacional de Ferrocarriles Belgas. La motivación principal surge de la necesidad de mejorar la puntualidad de los servicios, optimizar los recursos de operación y reforzar la transparencia en la comunicación con el viajero y con los distintos actores internos de la organización.

Para alcanzar estos objetivos se ha diseñado un pipeline de datos gobernado que aprovecha la capacidad de cómputo distribuido de Azure Databricks (Apache Spark), el almacenamiento escalable y seguro de Azure Data Lake Storage Gen2, y la orquestación mediante Azure Data Factory. La explotación final de la información se apoya en Azure Cosmos DB con API MongoDB como base de datos documental NoSQL de baja latencia, lo que facilita la consulta rápida de los datos procesados y abre la puerta a su integración en aplicaciones internas y tableros ejecutivos.

2. Contexto y Objetivos

2.1 Justificación del proyecto

En el ámbito ferroviario, la puntualidad constituye uno de los indicadores más críticos de la calidad del servicio. Las empresas operadoras necesitan monitorizar en tiempo casi real tanto las incidencias como el uso de la capacidad de la red. Contar con información precisa y actualizada resulta esencial no solo para mejorar la planificación y apoyar la toma de decisiones estratégicas, sino también para garantizar un servicio más confiable y seguro al viajero.

En este contexto, disponer de una plataforma de datos moderna en la nube permite centralizar, procesar y servir información de manera ágil, reduciendo los tiempos de respuesta ante incidencias y mejorando la comunicación hacia el usuario final y los equipos internos.

2.2 Objetivos específicos

El objetivo general de este proyecto es diseñar e implementar un pipeline de datos gobernado en Azure, capaz de transformar datos crudos y heterogéneos en información accesible y de calidad. Para ello se plantean los siguientes objetivos específicos:

- Construir un flujo de ingesta y procesamiento basado en Azure Databricks (Spark), que garantice la calidad y consistencia de los datos.
- Implementar una arquitectura en capas (RAW, CLEAN, Serving) que asegure trazabilidad, escalabilidad y reutilización de los datos.
- Exponer los datos en un servicio Azure Cosmos DB (API para MongoDB), sustituyendo soluciones locales y eliminando dependencias de túneles o configuraciones manuales.
- Facilitar la explotación de la información a través de consultas rápidas, aplicaciones internas y futuras integraciones con herramientas de visualización (Power BI, Tableau).
- Automatizar el pipeline para permitir cargas periódicas, garantizando que la información esté siempre actualizada.

Estos objetivos se apoyan en las competencias adquiridas en el máster, en particular:

- Procesamiento distribuido con Spark (Databricks).
- Modelado y consulta en bases de datos NoSQL.

- Desarrollo en Python para la implementación modular y la automatización de notebooks.
- Uso de tecnologías de Big Data en la nube para el aprovisionamiento de recursos y la orquestación de procesos.

2.3 Stakeholders y áreas impactadas

Los principales grupos de interés que estarían beneficiados por este proyecto son:

- Dirección de Operaciones, que requiere indicadores fiables para la gestión de la puntualidad y la optimización de servicios.
- Equipos de Planificación y Seguridad Ferroviaria, que utilizan los datos para detectar cuellos de botella y prevenir riesgos.
- Área de Atención al Cliente, que mejora la comunicación con el viajero mediante información validada y accesible en tiempo casi real.
- Unidades de Tecnología y Business Intelligence, que encuentran en el pipeline una base de datos consistente sobre la cual construir informes, dashboards y modelos analíticos.

2.4 Nota sobre los datos y NDA con NMBS

Los datos empleados en este trabajo corresponden al estándar GTFS (General Transit Feed Specification), proporcionados por NMBS/SNCB (National Railway Company of Belgium). El acceso a dichos datasets se ha realizado bajo un acuerdo de confidencialidad (NDA, Non-Disclosure Agreement), que restringe su uso exclusivamente al desarrollo académico del presente proyecto.

El propósito de utilizar estos datos es estrictamente didáctico y experimental, orientado a demostrar la viabilidad de un pipeline de datos moderno en la nube. No se contempla su explotación comercial ni su difusión pública más allá del marco académico del Trabajo Fin de Máster.

En un eventual escenario de producción, la obtención de los datos se realizaría a partir de portales abiertos, APIs públicas o acuerdos específicos de suministro de información, garantizando en todo momento el cumplimiento normativo, contractual y de seguridad de los datos.

3. Alcance y Entregables

El alcance del proyecto ha abarcado la implementación completa del ciclo de vida de un pipeline de datos moderno, siguiendo las buenas prácticas de la arquitectura medallion (Bronze, Silver, Gold).

En la capa RAW (Bronze) se ha definido la ingesta de los ficheros GTFS proporcionados por NMBS/SNCB. Estos ficheros, en formato texto (.txt), se descargan de forma automatizada desde el portal de datos y se almacenan en Azure Data Lake Storage Gen2, transformados a tablas Delta Lake. Este enfoque garantiza la persistencia, el versionado de la información y la capacidad de rehacer procesos en caso de ser necesario.

En la capa CLEAN (Silver) se han desarrollado distintos notebooks en Azure Databricks, cada uno orientado a una tabla GTFS específica. En esta fase se han aplicado reglas de calidad esenciales: eliminación de nulos en campos clave (stop_id, trip_id, route_id), eliminación de duplicados, validación de formatos horarios en notación HH:MM:SS, normalización de coordenadas geográficas, estandarización de texto en campos descriptivos y conversión de tipos de datos. Como resultado, los datos quedan depurados, coherentes y listos para un análisis fiable.

Finalmente, en la capa GOLD (Serving) se ha implementado un proceso de carga hacia una base de datos documental gestionada en la nube, concretamente Azure Cosmos DB con API para MongoDB. En esta etapa, los datos limpios se exportan desde Databricks a colecciones en Cosmos DB, garantizando la idempotencia mediante la generación de identificadores deterministas (_id) a partir de claves naturales. Además, se han adaptado los nombres de columnas para asegurar compatibilidad con MongoDB (evitando caracteres como “.” o “$”). Este cambio elimina la dependencia de túneles externos (ngrok) y de despliegues locales en Docker, y aporta mayor robustez, seguridad y escalabilidad al pipeline.

Los entregables principales de este trabajo incluyen:

- Un notebook de auto-ingesta encargado de la descarga diaria desde la fuente web de NMBS/SNCB y almacenamiento en RAW.
- Un notebook de ingesta general, responsable de convertir los ficheros a Delta Lake y organizarlos en la zona RAW.
- Los notebooks de limpieza específicos por tabla (12 en total), orientados a la depuración y transformación de cada dataset GTFS.
- Un notebook de carga hacia Cosmos DB (03_load_to_cosmos), que expone las tablas limpias como colecciones documentales listas para consulta.
- Un Job de Databricks, configurado para orquestar todas las fases de

forma secuencial y trazable (Ingesta → Limpieza → Carga).

- La documentación técnica, que detalla las reglas de transformación aplicadas, el diseño de la arquitectura y la configuración necesaria para la ejecución del pipeline, incluyendo aspectos de parametrización y seguridad.

4. Resultados esperados

La plataforma desarrollada permite disponer de un pipeline ETL orquestado y reproducible en Azure Databricks, capaz de transformar datos crudos en información depurada y servida en tiempo casi real.

- Los resultados esperados se concretan en tres dimensiones:
- Calidad de los datos: al aplicar procesos de limpieza y normalización en la capa Silver, se eliminan inconsistencias, nulos y duplicados, y se valida el cumplimiento de formatos y rangos. Esto garantiza que los datos utilizados en análisis y aplicaciones son fiables y trazables.
- Flexibilidad y escalabilidad: el uso de widgets y configuraciones parametrizables en los notebooks permite ajustar rutas de almacenamiento, bases de datos y colecciones destino, facilitando la reutilización y la ampliación del pipeline a nuevas fuentes GTFS, XML o APIs. Además, la integración con Cosmos DB (API MongoDB) elimina la dependencia de soluciones locales como ngrok, permitiendo una conexión nativa en la nube con escalabilidad gestionada.
- Explotación de la información: la publicación en Cosmos DB habilita el consumo por parte de aplicaciones internas, APIs (FastAPI) y tableros ejecutivos en herramientas como Tableau o Power BI. Con ello, la organización gana visibilidad sobre el estado de los servicios, mejora sus procesos de planificación y ofrece una comunicación más transparente al viajero.

A largo plazo, la arquitectura planteada ofrece la posibilidad de integrar más fuentes en tiempo real, automatizar controles de calidad más avanzados y habilitar un ecosistema de analítica en el que convivan consultas operativas, informes ejecutivos y modelos predictivos de la puntualidad.

5. Arquitectura de referencia en Azure

La arquitectura diseñada en este proyecto se ha basado en un enfoque de plataforma de datos moderna en la nube, siguiendo las recomendaciones de Microsoft para arquitecturas analíticas en Azure y adaptándolas al caso de uso concreto del transporte ferroviario.

El diseño responde a la necesidad de disponer de un flujo ETL gobernado, escalable y seguro, que permita pasar de datos crudos a información útil para la operación y la toma de decisiones, cumpliendo además con buenas prácticas de seguridad y normativas ISO/IEC 27001 y 27017 para entornos cloud.

5.1. Fuentes de datos

Las principales fuentes de datos integradas corresponden al estándar GTFS (General Transit Feed Specification), el cual describe información relacionada con horarios, rutas, paradas y viajes en ficheros planos (.txt). Estos datasets provienen de portales abiertos (ej. NMBS/SNCB) y constituyen la base operativa del pipeline.
Además de GTFS, la arquitectura queda preparada para admitir fuentes adicionales como:

- Ficheros XML con estructuras jerárquicas, útiles para enriquecer la información con incidencias o condiciones de servicio.
- APIs REST/JSON, que permiten capturar datos en tiempo casi real, como condiciones meteorológicas o estado de tráfico ferroviario.

Todas estas fuentes se incorporan inicialmente en bruto, sin transformación, dentro del repositorio de almacenamiento en la nube.

5.2. Zonas de datos (Bronze, Silver, Gold)

Se ha adoptado el patrón de medallion architecture, que organiza los datos en tres capas y facilita la gobernanza:

- Bronze (RAW): contiene los datos tal como llegan desde las fuentes, almacenados en formato Delta en Azure Data Lake Storage Gen2. Esta capa asegura trazabilidad y versionado.
- Silver (CLEAN): alberga los datos tras los procesos de limpieza y normalización, eliminando duplicados, validando formatos y aplicando reglas de calidad.
- Gold (Serving/Applications): corresponde a la capa de explotación. En este proyecto, los datos Silver se cargan en colecciones documentales de Azure Cosmos DB (API MongoDB), lo que permite alimentar aplicaciones internas, APIs (FastAPI) y dashboards en herramientas como Power BI o Tableau.
- Este diseño garantiza que los procesos de transformación no alteran los datos

originales y habilita diferentes niveles de consumo según las necesidades.

5.3. Servicios de Azure y roles

La arquitectura se apoya en varios servicios clave del ecosistema Azure:

- Azure Data Lake Storage Gen2 (ADLS): repositorio central de datos, organizado en las zonas RAW y CLEAN.
- Azure Databricks: motor de procesamiento distribuido, donde se desarrollan los notebooks de ingesta, limpieza y carga, así como la orquestación mediante Jobs.
- Azure Data Factory (ADF): en un despliegue extendido, puede encargarse de la orquestación a nivel de pipeline, integrando Databricks con otras fuentes o sistemas.
- Azure Cosmos DB (API MongoDB): servicio gestionado en la nube que sustituye la dependencia de Mongo local y ngrok, proporcionando una capa Serving robusta, escalable y segura.
- Azure Key Vault: gestor de credenciales y secretos, imprescindible en entornos productivos para proteger URIs de conexión y contraseñas.

Los roles de cada servicio quedan claramente diferenciados: ADLS como almacenamiento, Databricks como procesamiento, ADF como orquestación y Cosmos DB como base de datos documental gestionada.

5.4. Diseño de seguridad y cumplimiento

La seguridad es un aspecto transversal en toda la arquitectura.

- Accesos a ADLS: regulados mediante Azure RBAC (Role-Based Access Control) y permisos de contenedor.
- Conexiones a Cosmos DB: cifradas en tránsito (SSL/TLS) y protegidas con autenticación de usuario y claves gestionadas en Key Vault.
- Databricks: autenticación mediante Service Principals en lugar de credenciales personales.

Este diseño sigue prácticas compatibles con ISO/IEC 27001 (seguridad de la información) y ISO/IEC 27017 (controles de seguridad para servicios cloud): cifrado en tránsito y reposo, separación de entornos, auditoría de accesos y uso de secretos gestionados.
En cuanto a cumplimiento, aunque los datos GTFS son públicos, la arquitectura está preparada para gestionar datos sensibles si se incorporaran nuevas fuentes, aplicando anonimización, mascarado y auditoría de accesos con Azure Purview.

5.5. Observabilidad y costes

La observabilidad del pipeline se consigue con los mecanismos nativos de Databricks (logs de ejecución, métricas de proceso, errores), integrables en Azure Monitor y Log Analytics

para consolidar indicadores de rendimiento y calidad.
En cuanto a costes, los principales recursos consumidos son:

- ADLS Gen2: bajo coste de almacenamiento.
- Databricks: recurso más costoso por uso de clústeres Spark, mitigado con políticas de auto-terminación y escalado automático.
- Cosmos DB (API MongoDB): coste asociado al plan elegido (serverless o provisioned throughput). Para este proyecto académico, el modo serverless asegura costes reducidos y pago por consumo real.
- Posibles costes del despliegue de la API en un futuro de cara al crecimiento, si aplica.

Se han aplicado un serie de medidas que permiten obtener y optimizar con un mayor impacto el gasto de mantenimiento de los servicios, sin incurrir en un coste desmesurado para la envergadura de este proyecto.

Se podrían resaltar las siguientes medidas:

- El uso de clústeres con un tamaño que se ajusta a las necesidades actuales del proyecto
- Una consolidación de los datos contenidos en los ficheros mediante el uso de COALESCE(1).
- La no utilización indefinida de los recursos una vez realizado el proceso de ejecución correspondiente para evitar costes elevados y no tener recursos que se utilicen de manera ineficiente

6. Diseño del pipeline (ETL)

En este proyecto se ha optado explícitamente por un enfoque ETL (Extract, Transform, Load) y no ELT. La razón principal es que las transformaciones y validaciones de calidad se llevan a cabo en la capa de procesamiento distribuido de Databricks (Apache Spark), antes de cargar los datos en la base documental. De dicha forma, se puede asegurar y garantizar que los datos tratados, contenidos en los ficheros anteriormente procesados, que se van a utilizar para su explotación en fases posteriores se encuentran depurados y consistentes en la información que contienen tras el proceso de limpieza.

Esto permitirá no delegar ciertas funcionalidades o responsabilidades a la base de datos NoSQL (Cosmos DB), cuya función en la arquitectura se limita a actuar como repositorio de consulta y consumo de baja latencia.
Este enfoque permite:

- Asegurar la trazabilidad y calidad de extremo a extremo.
- Aprovechar la potencia de Spark para aplicar transformaciones complejas y reglas de negocio de forma eficiente.
- Mantener la base documental ligera y orientada exclusivamente al consumo, evitando sobrecargarla con tareas de procesamiento que no forman parte de su propósito.

De este modo, el pipeline cumple la lógica de extraer los datos de las fuentes (Extract), transformarlos en la zona CLEAN (Transform) y, finalmente, cargarlos en MongoDB/Cosmos DB (Load) para su explotación en aplicaciones y dashboards.

6.1. Ingesta (zona RAW)

La primera etapa del pipeline corresponde a la capa **RAW**, cuya función principal es la ingesta y persistencia de los datos en su formato más cercano al original. En este caso, los ficheros de entrada pertenecen al estándar **GTFS (General Transit Feed Specification)** proporcionado por NMBS/SNCB, en formato texto (.txt). Entre los ficheros incluidos se encuentran:

- *agency.txt*, *calendar.txt*, *calendar_dates.txt*, *feed_info.txt*, *routes.txt*, *stop_times.txt*, *stop_time_overrides.txt*, *stops.txt*, *trips.txt*, *transfers.txt* y *translations.txt*.

La ingesta se implementa en Databricks mediante PySpark, aprovechando sus capacidades de inferencia de esquema y validación automática de cabeceras. Los ficheros se convierten

y almacenan en formato Delta Lake dentro de Azure Data Lake Storage Gen2, lo que permite disponer de un almacenamiento histórico versionado y optimizado para consultas posteriores.

Desde el punto de vista organizativo, la ingesta se ha desarrollado en un único notebook (01_ingesta) que centraliza la carga y escritura de los datos en la capa RAW. Esta decisión responde al principio de *divide y vencerás*: se evita mezclar transformaciones posteriores en esta fase y se mantiene el flujo más claro y sencillo de depurar. Una vez almacenados en RAW, los datos quedan disponibles para ser utilizados en las siguientes etapas de limpieza y transformación.

6.2. Limpieza y transformaciones (zona CLEAN)

La segunda etapa del pipeline corresponde a la capa CLEAN, cuyo propósito es depurar los datos almacenados en RAW, normalizarlos y asegurar su consistencia. Esta fase es fundamental, ya que garantiza que los datos expuestos a las aplicaciones y análisis posteriores cumplen con reglas básicas de calidad y coherencia.

Las transformaciones se realizan en PySpark (DataFrames) y se han dividido en diferentes notebooks, uno por cada tabla GTFS, lo que facilita la modularidad y la detección de errores. Entre las principales reglas de limpieza aplicadas destacan:

- Eliminación de valores nulos en campos que actúan como claves primarias (*stop_id*, *trip_id*, *route_id*).
- Eliminación de registros duplicados según identificadores únicos.
- Normalización de cadenas de texto, por ejemplo estandarizando los nombres de paradas.
- Conversión de tipos de datos, como el casteo de coordenadas (*stop_lat*, *stop_lon*) a tipo double o de campos categóricos (*direction_id*, *transfer_type*) a int.
- Validación de formatos horarios en los campos *arrival_time* y *departure_time* bajo el patrón HH:MM:SS.
- Validación de rangos geográficos para las coordenadas de latitud y longitud.

Como salida de este proceso se generan tablas Delta limpias almacenadas en la ruta CLEAN del Data Lake, organizadas por tabla. Cada notebook lee directamente de la zona RAW y escribe en la zona CLEAN, lo que asegura una clara separación de responsabilidades y facilita el mantenimiento del pipeline. Esta estructura modular (un notebook por cada fichero GTFS) permite una orquestación más eficiente y precisa dentro del Job global.

6.3. Carga en base documental (Azure Cosmos DB – API MongoDB)

Una vez depurados los datos, se lleva a cabo la fase de carga hacia la capa de servicio o Serving Layer, implementada en este proyecto mediante Azure Cosmos DB con API para

MongoDB.

El objetivo de esta etapa es poner a disposición los datos limpios en un formato accesible y de baja latencia, apto para consultas operativas, APIs y aplicaciones de consumo.

Para este fin se ha utilizado el conector oficial mongo-spark-connector, que permite escribir directamente desde Spark en colecciones Mongo. Cada tabla Delta de la capa CLEAN se transforma en una colección independiente dentro de la base de datos Cosmos DB.

Durante el proceso de carga:

- Se generan identificadores deterministas (_id) aplicando funciones hash (sha2) sobre combinaciones de columnas clave, garantizando idempotencia en recargas.
- Se adaptan los nombres de las columnas para cumplir con las restricciones de MongoDB (evitando caracteres como “.” o “$”).
- Se controla el nivel de paralelismo (coalesce(1)) para optimizar la escritura en Cosmos, dado que este servicio penaliza excesivas conexiones concurrentes.

Todo este flujo se ha centralizado en un único notebook (03_load_to_cosmos), dependiente de la finalización correcta de la etapa de limpieza.

El uso de Cosmos DB gestionado sustituye la necesidad previa de desplegar MongoDB en local y abrir túneles con ngrok, aportando seguridad, escalabilidad y un modelo de costes ajustado al consumo real.

7. Orquestación y operación

La orquestación del pipeline completo se ha realizado mediante un *Job* en Databricks , diseñado para garantizar la ejecución secuencial y trazable de todas las fases. El flujo contempla las siguientes etapas :

- Etapa cero (descarga automática): obtención diaria de los ficheros GTFS desde la web de NMBS/SNCB y carga directa en la zona RAW del Data Lake.
- Ingesta (RAW): ejecución del notebook 01_ingesta, que almacena los datos en formato Delta en la capa RAW.
- Limpieza (CLEAN): ejecución, de forma paralela o secuencial, de los notebooks 02_clean_**, cada uno dedicado a una de las tablas GTFS.
- Carga en Cosmos DB (Serving): ejecución del notebook 03_load_to_cosmos, dependiente de la finalización correcta de los procesos de limpieza.
- Explotación mediante FastAPI: despliegue de un servicio ligero que consulta los datos en Cosmos DB y los expone a aplicaciones internas o dashboards.

De esta manera se asegura que los datos siguen un flujo lógico desde la ingesta inicial hasta la exposición final, con dependencias explícitas que previenen inconsistencias o ejecuciones incompletas.

7.1 Ejecución secuencial en Databricks

El Job encadena: Etapa cero → Ingesta → Limpieza → Publicación → Explotación (FastAPI). Las dependencias aseguran orden y recuperación controlada en caso de fallo.

7.2 Conexión con Cosmos DB (API Mongo)

En esta versión del pipeline se ha sustituido la solución provisional inicial basada en MongoDB local + ngrok por un despliegue en Azure Cosmos DB con API para MongoDB.
Esto aporta ventajas notables:

- Eliminación de túneles externos y puertos expuestos.
- Seguridad reforzada y cumplimiento normativo, al integrarse nativamente en Azure.
- Escalabilidad y monitorización centralizada, facilitando un futuro paso a producción.

El notebook 03_load_to_cosmos utiliza el conector oficial mongo-spark-connector, configurado con la URI proporcionada por Cosmos, autenticación PLAIN y parámetros de optimización para cargas distribuidas.

7.3 Programación periódica del pipeline

El Job de Databricks se ha programado para ejecutarse diariamente a las 2:00 de la madrugada mediante el scheduler nativo. Con esta configuración se consigue:

- Actualizar automáticamente los datos en Cosmos DB.
- Reducir la dependencia de intervención humana.
- Asegurar consistencia y trazabilidad en la actualización diaria.

7.4 FastAPI**: Explotación de los datos.**

Con el objetivo de habilitar un acceso sencillo y estandarizado a los datos procesados en la capa GOLD, se ha desarrollado una API REST utilizando el framework FastAPI en Python.
Esta API expone de forma controlada la información almacenada en la base documental (Azure Cosmos DB con API para MongoDB), lo que permite a aplicaciones internas y dashboards consumir los datos sin necesidad de acceder directamente al sistema de almacenamiento.
Arquitectura de la API:

- Conexión a la base de datos: se gestiona mediante variables de entorno (MONGO_URI, DB_NAME), garantizando buenas prácticas de seguridad y facilitando despliegues en distintos entornos.
- Modelos de datos (Pydantic): definen la estructura de las respuestas de la API, validando automáticamente los documentos obtenidos de MongoDB.
- Endpoints REST: permiten consultar paradas, rutas, viajes, horarios y transferencias de forma sencilla.

Modelos principales :

- Route: información de rutas (id, agencia, nombres, colores).
- Trip: definición de un viaje asociado a una ruta y un servicio.
- Stop: datos de paradas con nombre, coordenadas y tipo de ubicación.
- StopTime: horarios de llegada y salida en cada parada para un viaje.
- Transfer: información de transferencias entre paradas.

Ejemplos de endpoints implementados:

- /routes → devuelve la lista de rutas.
- /stops → lista de paradas con coordenadas.
- /trips/{route_id} → viajes asociados a una ruta.
- /trips/{trip_id}/stops → paradas ordenadas de un viaje.
- /search/stops?name=X → búsqueda de paradas por nombre.

- /transfers/{stop_id} → transferencias desde una parada.

Beneficios:

- Facilita el consumo externo de los datos (aplicaciones móviles, dashboards, integraciones).
- Provee documentación automática en formato OpenAPI/Swagger, útil para equipos de desarrollo.
- Permite escalar hacia una arquitectura API-first, en la que diferentes aplicaciones se apoyen en un backend común de datos.

8. Conclusiones

El desarrollo de este proyecto ha permitido construir un pipeline de datos moderno en Azure, con una arquitectura modular basada en las zonas RAW, CLEAN y Serving. A lo largo del trabajo se ha validado tanto la viabilidad técnica de la solución como su alineación con las necesidades del sector ferroviario en cuanto a puntualidad, calidad de datos y capacidad de explotación.

Uno de los principales logros del proyecto ha sido la construcción de un flujo orquestado y reproducible, que garantiza la trazabilidad de extremo a extremo: desde la ingesta de datos crudos hasta su exposición en una base documental y su explotación mediante una API. Esta aproximación no solo asegura consistencia, sino que también facilita la escalabilidad, ya que cada etapa es autónoma y parametrizable.

Durante la discusión técnica se han identificado fortalezas y limitaciones. Entre las fortalezas destacan:

- El uso de **Delta Lake** como formato de almacenamiento, que aporta versionado y capacidades de *time travel*, muy útiles en entornos de datos históricos.
- La división modular de los notebooks, que facilita la depuración y la orquestación en Jobs de Databricks.
- La implementación de reglas de calidad explícitas (eliminación de nulos, validación de coordenadas y horas), que elevan la fiabilidad de los datos procesados.
- La integración de los datos con **Azure Cosmos DB (API MongoDB)**, lo que elimina la necesidad de túneles externos y garantiza mayor seguridad y escalabilidad.
- El despliegue de una **FastAPI** para exponer los datos como servicios REST, lo que habilita su consumo directo en aplicaciones, integraciones o dashboards.

Sin embargo, también se han detectado áreas de mejora. Si bien Cosmos DB resuelve los

problemas de conexión de la primera versión, resulta conveniente reforzar la automatización de comprobaciones de calidad. Una evolución natural sería integrar un **framework de *data quality*** más avanzado (como *Deequ* o reglas en Azure Data Factory), con alertas y validaciones antes de publicar los datos en la capa de servicio.

Desde el punto de vista de negocio, el pipeline desarrollado sienta las bases para mejorar la monitorización de la puntualidad, la detección de incidencias y la planificación de recursos. La exposición de los datos a través de FastAPI incrementa la accesibilidad y permite construir aplicaciones internas o tableros en tiempo real, lo que mejora la transparencia y la capacidad de respuesta de la organización.

En términos de conclusiones generales, el proyecto ha demostrado que es posible construir, con recursos relativamente accesibles, un pipeline de datos escalable y gobernado en la nube. La arquitectura propuesta es flexible y fácilmente ampliable a nuevas fuentes, y puede servir como base para futuras iniciativas de analítica avanzada, incluyendo modelos predictivos de retrasos o análisis de demanda.

Finalmente, este trabajo abre varias líneas de investigación futura:

- Automatización y estandarización de reglas de calidad de datos.
- Integración con sistemas de visualización y BI corporativos (Power BI, Tableau).
- Extensión hacia fuentes de datos en tiempo real (APIs de tráfico, IoT ferroviario).
- Evolución hacia un enfoque **API-first**, donde FastAPI sea la capa central de acceso a los datos.

En definitiva, el pipeline desarrollado constituye un paso importante hacia la construcción de una plataforma de datos moderna en el sector ferroviario, y sienta las bases tanto para mejorar la operación diaria como para habilitar una estrategia de analítica más ambiciosa en el futuro.

9. Bibliografía

9.1 Documentación oficial de tecnologías usadas

- Databricks. (s.f.). *Delta Lake Documentation*. Recuperado de https://docs.databricks.com/delta
- Microsoft Azure. (s.f.). *Azure Data Lake Storage Gen2 Documentation*. Recuperado de https://learn.microsoft.com/azure/storage/blobs/data-lake-storage-introduction
- MongoDB. (s.f.). *MongoDB Manual*. Recuperado de https://www.mongodb.com/docs/
- Azure Cosmos DB. (s.f.). *Cosmos DB API for MongoDB Documentation*. Recuperado de https://learn.microsoft.com/azure/cosmos-db/mongodb
- Apache Spark. (s.f.). *Spark SQL, DataFrames and Datasets Guide*. Recuperado de https://spark.apache.org/docs/latest/sql-programming-guide.html
- FastAPI. (s.f.). *FastAPI Documentation*. Recuperado de https://fastapi.tiangolo.com/

9.2 Otra documentación proporcionada en los contenidos del máster.

- Universidad Complutense de Madrid (2024-2025). *Material docente de la asignatura "Spark"*. Máster Data Science, Big Data & Business Analytics 2024-2025 UCM
- Universidad Complutense de Madrid. (2024-2025). Material docente de la asignatura ''Tecnologias del Big Data'' Máster Data Science, Big Data & Business Analytics 2024-2025 UCM
- Universidad Complutense de Madrid. (2024-2025). Material docente de la asignatura “Python, módulos 1 y 2”
- Universidad Complutense de Madrid. (2024-2025). Material docente de la asignatura “NoSQL”

Los principios aplicados en el diseño del pipeline responden a los contenidos impartidos en el Máster Oficial en Big Data & Business Analytics, particularmente en las asignaturas de Tecnologías del Big Data para procesamiento de datos en la nube y Procesamiento distribuido con Spark. Se han seguido las metodologías y prácticas recomendadas en el material docente impartido, lo cual ha permitido garantizar tanto la validez técnica y alineamiento con estándares académicos y profesionales.

10. Anexos

Anexo 1. Arquitectura en Azure

Diagrama de la arquitectura general (Data Lake, Databricks, Cosmos DB, FastAPI). Representa las tres capas del pipeline (RAW, CLEAN, GOLD).

- **Fuentes de datos (GTFS, APIs, XML)**

  ↓

- **Azure Data Lake (Bronze)**

  ↓

- **Azure Databricks**

  - Ingesta (RAW)
  - Limpieza (Silver)
  - Publicación (Gold)

  ↓

- **Cosmos DB (API Mongo)**

  ↓

- **FastAPI en Azure App Service**

  ↓

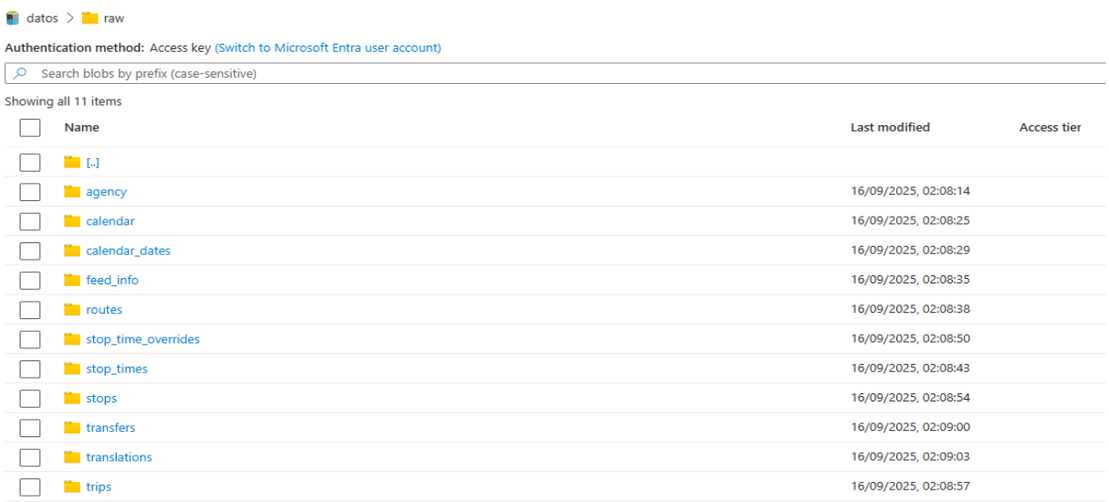
- **Usuarios (apps, consultas, BI)**

Anexo 2. Ingesta y almacenamiento

- Captura del notebook 00_autoloader_ingesta_gtfs (con el código que descarga los GTFS).

```python
# Databricks notebook: 00_fetch_gtfs_to_raw

import requests, os, zipfile, io
from datetime import datetime

# Widgets
dbutils.widgets.text("RAW_PATH", "abfss://datos@masterjcmtz002sta.dfs.core.windows.net/raw/", "RAW_PATH")
dbutils.widgets.text("GTFS_URL", "https://sncb-opendata.hafas.de/gtfs/static/c21ac6758dd25af84cca5b707f3cb3de", "GTFS_URL")
RAW_PATH = dbutils.widgets.get("RAW_PATH").strip()
GTFS_URL = dbutils.widgets.get("GTFS_URL").strip()

# 1) Descarga a memoria
print(f"📥 Descargando GTFS desde {GTFS_URL} ...")
resp = requests.get(GTFS_URL, timeout=60)
resp.raise_for_status()

# 2) Descomprime en /dbfs/tmp y luego copia a ABFSS (RAW)
ts = datetime.utcnow().strftime("%Y%m%d_%H%M%S")
local_dir = f"/dbfs/tmp/gtfs_{ts}"
os.makedirs(local_dir, exist_ok=True)

with zipfile.ZipFile(io.BytesIO(resp.content)) as zf:
    zf.extractall(local_dir)

print(f"🗂️  Descomprimido en {local_dir}")

# 3) Sube los .txt a RAW (sobrescribe)
#    Estructura: RAW_PATH/<nombre>.txt
for fname in os.listdir(local_dir):
    if fname.lower().endswith(".txt"):
        src = f"file:{local_dir}/{fname}"
        dst = f"{RAW_PATH}{fname}"
        # si quieres mantener una carpeta por “drop”, usa: f"{RAW_PATH}{ts}/{fname}"
        dbutils.fs.cp(src, dst, True)
        print(f"✅ Copiado {fname} → {dst}")

print("GTFS actualizado en RAW")
```

- Ejemplo de los ficheros GTFS en Azure Data Lake (contenedor RAW).

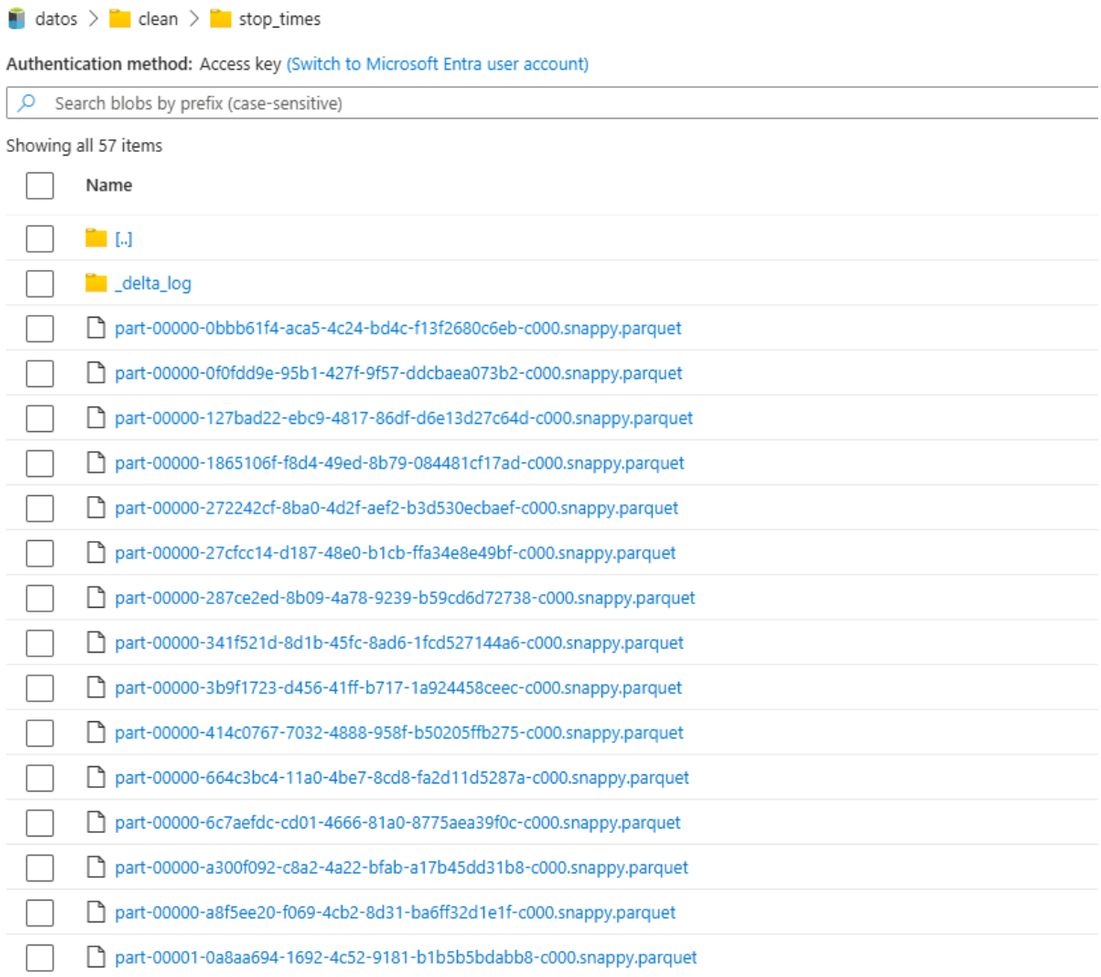


Anexo 3. Limpieza y transformación

- Captura de un notebook de limpieza stop_times como ejemplo.

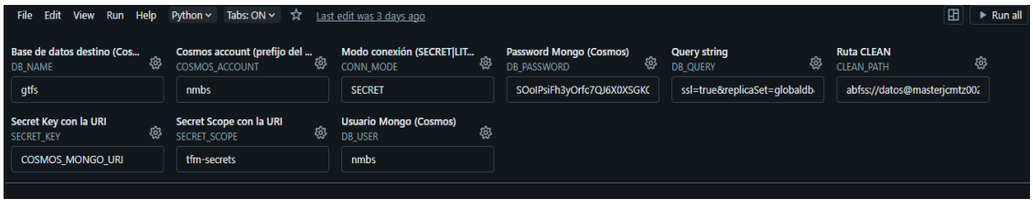
```
#Celda 1
#----------------------

from pyspark.sql import functions as F

# Widgets
dbutils.widgets.text("RAW_PATH","abfss://datos@masterjcmtz002sta.dfs.core.windows.net/raw/")
dbutils.widgets.text("CLEAN_PATH","abfss://datos@masterjcmtz002sta.dfs.core.windows.net/clean/")
RAW_PATH=dbutils.widgets.get("RAW_PATH"); CLEAN_PATH=dbutils.widgets.get("CLEAN_PATH")

# Leer stop_times desde RAW

df = spark.read.format("delta").load(f"{RAW_PATH}stop_times")
print("stop_times cargado ✅")

# Limpieza
df_clean = (
    df.filter(F.col("trip_id").isNotNull())
      .filter(F.col("stop_id").isNotNull())
      .filter(F.col("stop_sequence").isNotNull())
      .dropDuplicates(["trip_id","stop_sequence"])
      .withColumn("stop_sequence", F.col("stop_sequence").cast("int"))
)

regex_hms = r"^([0-9]{1,2}|[0-9]{2}):([0-5][0-9]):([0-5][0-9])$"  # acepta >24h
df_clean = df_clean.filter(
    (F.col("arrival_time").isNull()   | F.col("arrival_time").rlike(regex_hms)) &
    (F.col("departure_time").isNull() | F.col("departure_time").rlike(regex_hms))
)
# Guardado en zona CLEAN
df_clean.write.format("delta").mode("overwrite").save(f"{CLEAN_PATH}stop_times")
print("stop_times → CLEAN ✅")
dbutils.notebook.exit("OK")
(4) Spark Jobs
```

- Ejemplo de tabla Delta en Databricks después de la limpieza.


datos > clean > stop_times
Authentication method: Access key (Switch to Microsoft Entra user account)
Search blobs by prefix (case-sensitive)
Showing all 57 items
Name
[..]
_delta_log
part-00000-0bbb61f4-aca5-4c24-bd4c-f13f2680c6eb-c000.snappy.parquet
part-00000-0f0fdd9e-95b1-427f-9f57-ddcbaea073b2-c000.snappy.parquet
part-00000-127bad22-ebc9-4817-86df-d6e13d27c64d-c000.snappy.parquet
part-00000-1865106f-f8d4-49ed-8b79-084481cf17ad-c000.snappy.parquet
part-00000-272242cf-8ba0-4d2f-aef2-b3d530ecbaef-c000.snappy.parquet
part-00000-27cfcc14-d187-48e0-b1cb-ffa34e8e49bf-c000.snappy.parquet
part-00000-287ce2ed-8b09-4a78-9239-b59cd6d72738-c000.snappy.parquet
part-00000-341f521d-8d1b-45fc-8ad6-1fcd527144a6-c000.snappy.parquet
part-00000-3b9f1723-d456-41ff-b717-1a924458ceec-c000.snappy.parquet
part-00000-414c0767-7032-4888-958f-b50205ffb275-c000.snappy.parquet
part-00000-664c3bc4-11a0-4be7-8cd8-fa2d11d5287a-c000.snappy.parquet
part-00000-6c7aefdc-cd01-4666-81a0-8775aea39f0c-c000.snappy.parquet
part-00000-a300f092-c8a2-4a22-bfab-a17b45dd31b8-c000.snappy.parquet
part-00000-a8f5ee20-f069-4cb2-8d31-ba6ff32d1e1f-c000.snappy.parquet
part-00001-0a8aa694-1692-4c52-9181-b1b5b5bdabb8-c000.snappy.parquet

Anexo 4. Publicación en Cosmos DB

- Captura del código de escritura desde Spark hacia Cosmos DB.




```python
# 03_load_to_cosmos — Carga de tablas CLEAN a Azure Cosmos DB (API MongoDB)
# Requiere librería de cluster: org.mongodb.spark:mongo-spark-connector_2.12:10.3.0

from pyspark.sql.functions import sha2, concat_ws, col

#  Configuración fija
CLEAN_PATH = "abfss://datos@masterjcmtz002sta.dfs.core.windows.net/clean/"
DB_NAME    = "gtfs"

COSMOS_URI = (
    "mongodb://nmbs:"
    "SOoIPsiFh3yOrfc7QJ6X0XSGKCWmTwyyKFIiPZAEch96EXxjI0fGc8nFKbCoziE6VFE2xHEb85ulACDbX8YbXg=="
    "@nmbs.mongo.cosmos.azure.com:10255/"
    "?ssl=true&replicaSet=globaldb&retrywrites=false&maxIdleTimeMS=120000&appName=@nmbs@"
)

print("→ Conexión a Cosmos configurada (URI incluida).")
```

```python
# Claves por tabla para _id determinista
KEYS = {
    "routes": ["route_id"],
    "trips": ["trip_id"],
    "stop_times": ["trip_id","stop_id","stop_sequence"],
    "calendar": ["service_id"],
    "calendar_dates": ["service_id","date"],
    "transfers": ["from_stop_id","to_stop_id"],
    "agency": ["agency_id"],
    "stops": ["stop_id"],
    "feed_info": ["feed_publisher_name","feed_version"],
    "translations": ["table_name","field_name","language","record_id"],
    "stop_time_overrides": ["trip_id","stop_id","stop_sequence"],
}

def add_id(df, table):
    keys = KEYS.get(table, [])
    if keys and all(k in df.columns for k in keys):
        return df.withColumn("_id", sha2(concat_ws(":", *[col(k).cast("string") for k in keys]), 256))
    return df.withColumn("_id", sha2(concat_ws(":", *[col(c).cast("string") for c in df.columns]), 256))
```

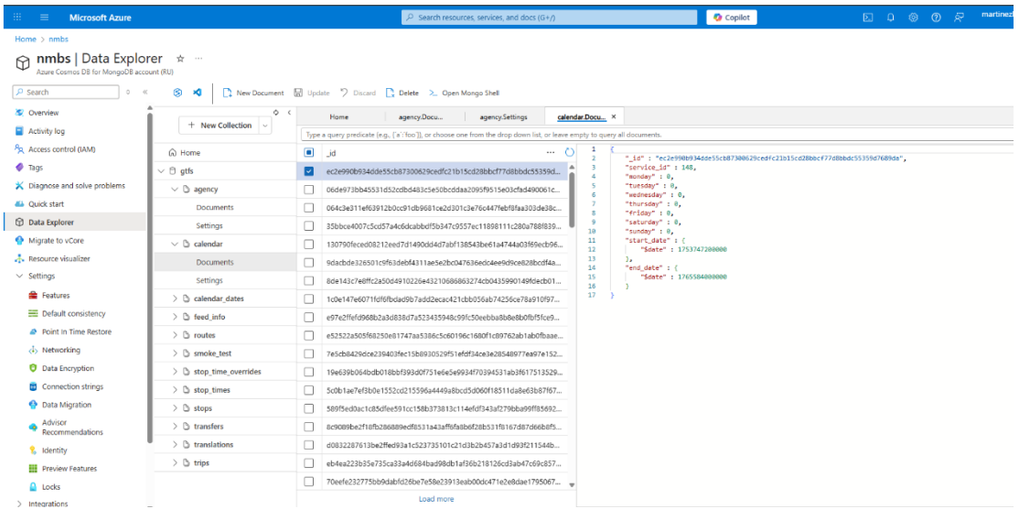
```
# Prueba mínima (smoke test)
test_df = spark.createDataFrame([(1, "ok"), (2, "cosmos")], ["_id", "msg"]).coalesce(1)

(test_df.write
   .format("mongodb")
   .mode("overwrite")
   .option("connection.uri", COSMOS_URI)
   .option("database", DB_NAME)
   .option("collection", "smoke_test")
   .option("replaceDocument", "true")
   .option("maxBatchSize", "100")
   .save())
print(f"✅ Smoke test escrito en Cosmos ({DB_NAME}.smoke_test)")

#  Tablas CLEAN para publicar
TABLES = [
    "agency","calendar","calendar_dates","feed_info","routes",
    "stop_times","stop_time_overrides","stops","trips","transfers","translations"
]
```

```
#  Escritura por tabla
for t in TABLES:
    try:
        print(f"→ Procesando {t} ...")
        df = spark.read.format("delta").load(f"{CLEAN_PATH}{t}")

        # Limpieza de nombres de columnas
        df = df.toDF(*[c.replace(".", "_").lstrip("$") for c in df.columns])

        # Añadir _id determinista
        df = add_id(df, t)

        (df.coalesce(1)
           .write
           .format("mongodb")
           .mode("overwrite")
           .option("connection.uri", COSMOS_URI)
           .option("database", DB_NAME)
           .option("collection", t)
           .option("replaceDocument", "true")
           .option("maxBatchSize", "100")
           .save())

        print(f"✅ {t} → Cosmos ({DB_NAME}.{t})")
```

```
        print(f"✅ {t} → Cosmos ({DB_NAME}.{t})")

    except Exception as e:
        print(f"❌ {t}: error → {e}")

print(" Publicación completa en Azure Cosmos DB (API MongoDB)")
dbutils.notebook.exit("OK")
```

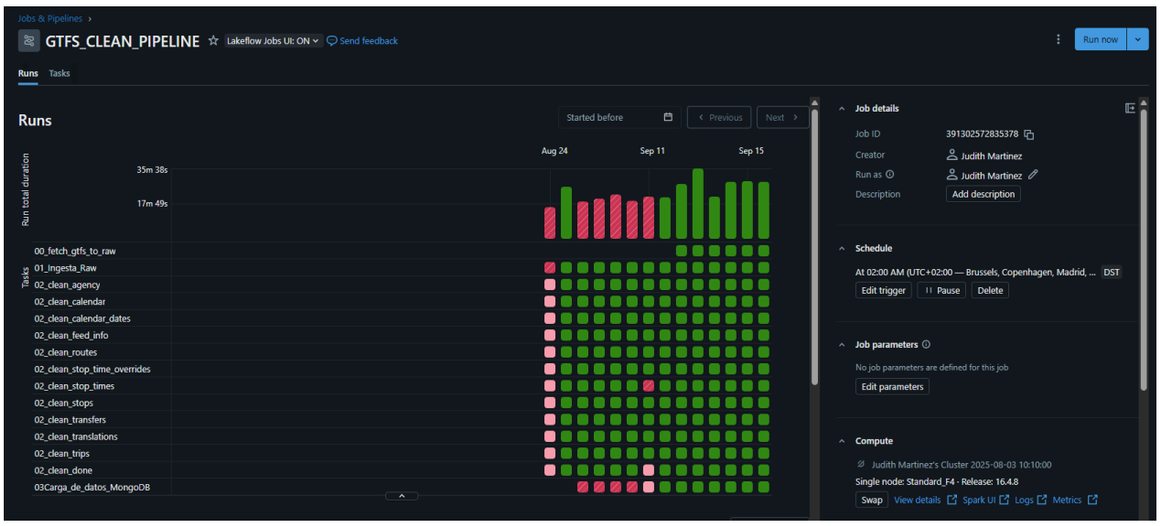
```
(23) Spark Jobs
tebook exited: OK
```

- Imagen de la base de datos en **Azure Cosmos DB Explorer** mostrando las colecciones creadas.

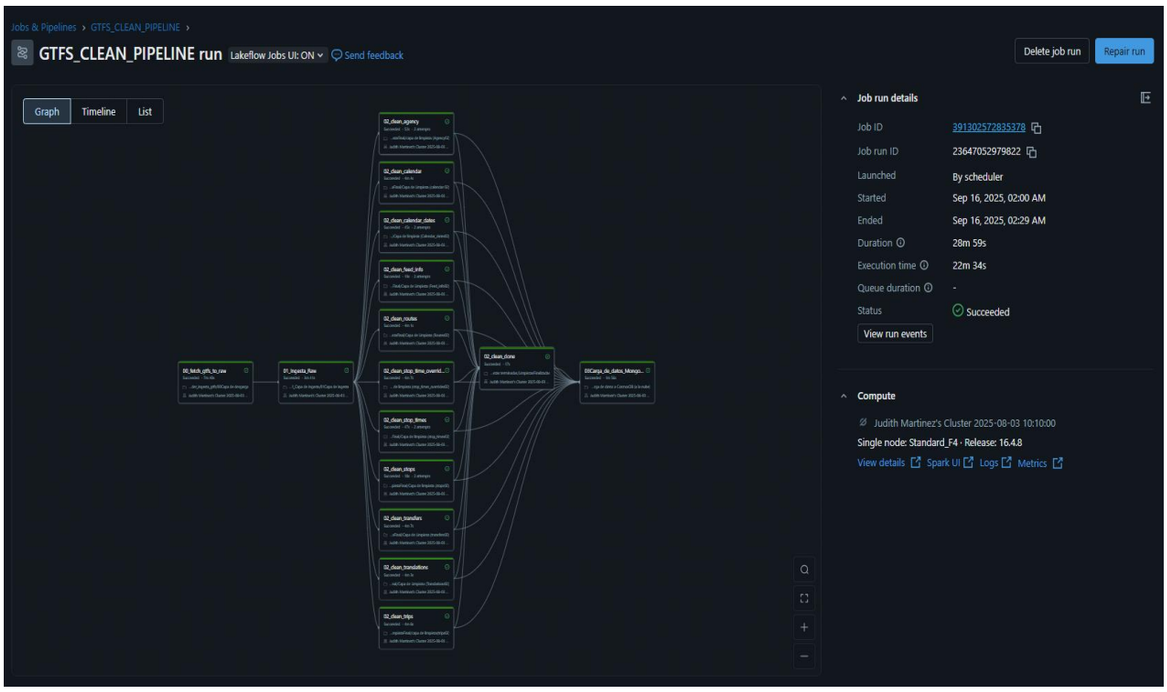


Anexo 5. Orquestación

- Captura de la configuración del **Job en Databricks** que encadena los notebooks.



- Ejemplo del log de una ejecución exitosa.



Anexo 6. FastAPI

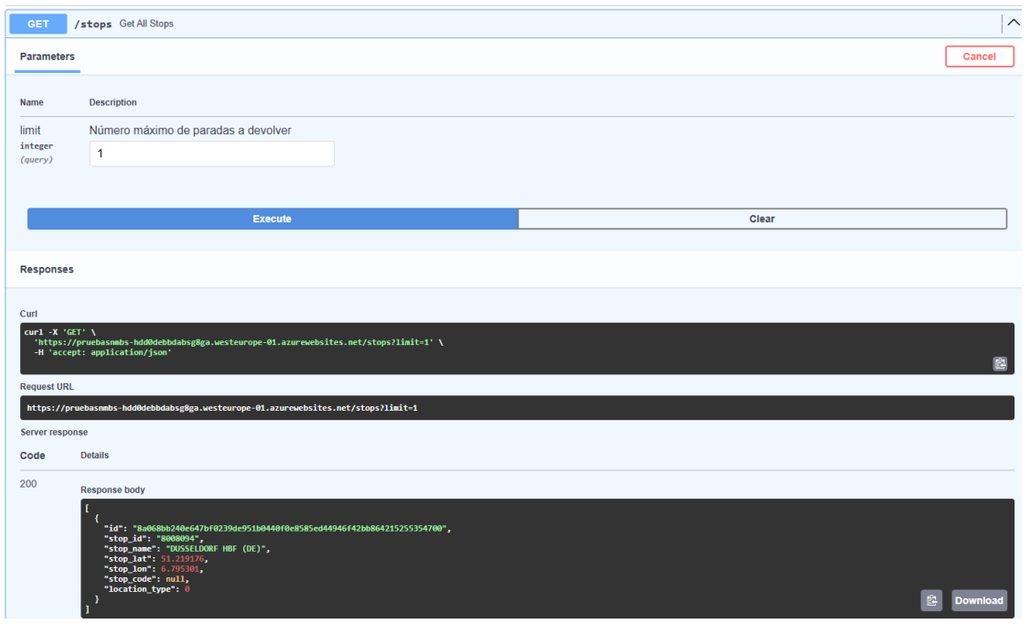
- Captura del código de la API (ejemplo: endpoint `/stops`).

```python
class Stop(BaseModel):  5 usages
    id: str
    stop_id: str
    stop_name: str
    stop_lat: float
    stop_lon: float
    stop_code: Optional[str] = None
    location_type: Optional[int] = None
```

```python
/trips/{trip_id}/stops
@app.get( path: "/trips/{trip_id}/stops", response_model=list[Stop])
def get_stops_for_trip(trip_id: str):
    times = list(db.stop_times.find({"trip_id": trip_id}))

    times.sort(key=lambda x: int(x["stop_sequence"]))

    stop_ids = [t["stop_id"] for t in times]

    stops = db.stops.find({"stop_id": {"$in": stop_ids}})
    stops_dict = {s["stop_id"]: serialize_doc(s) for s in stops}

    return [stops_dict[stop_id] for stop_id in stop_ids if stop_id in stops_dict]
```

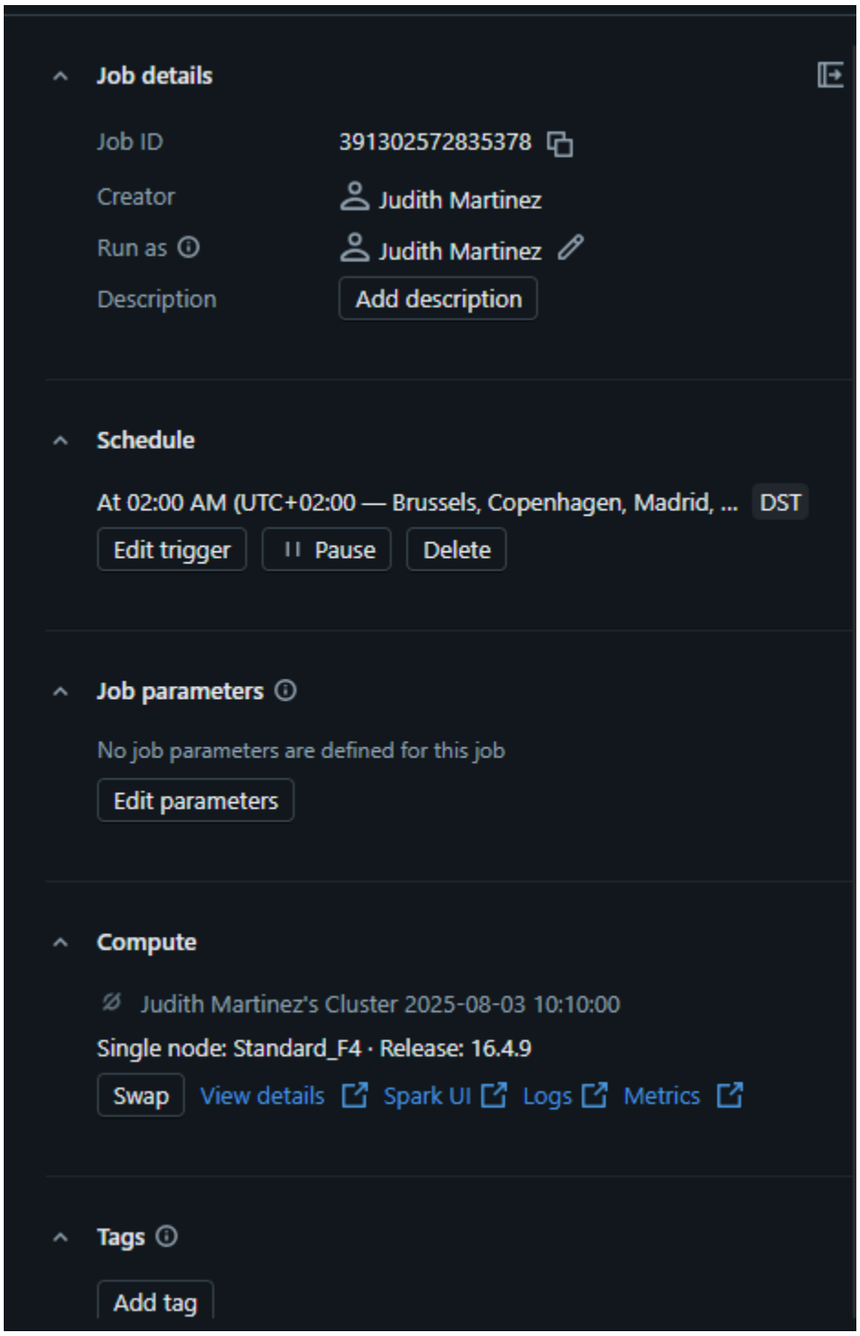
Ruta:
https://pruebasnmbs-hdd0debbdabsg8ga.westeurope-01.azurewebsites.net/docs

- Prueba en **Swagger UI** (documentación interactiva de FastAPI) mostrando un endpoint funcionando.



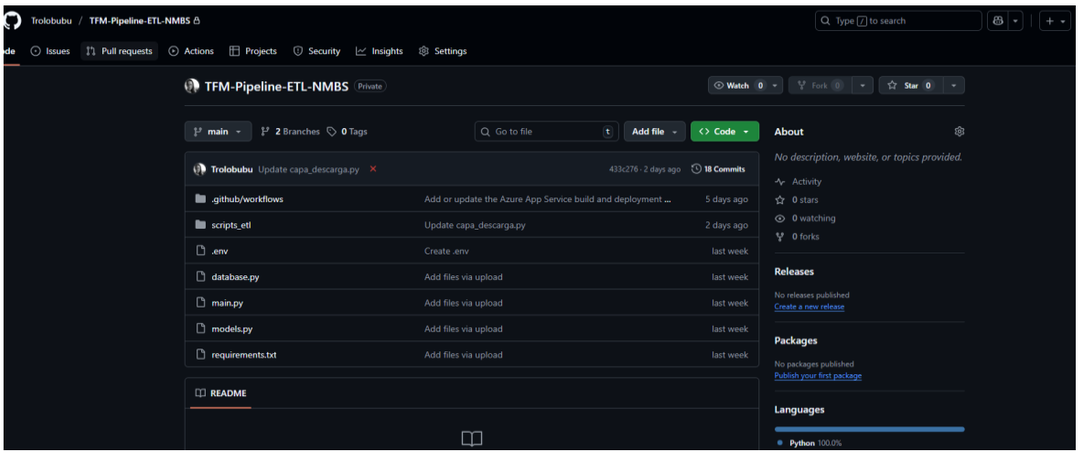
Anexo 7. Automatización

- Imagen del **programador de Jobs en Databricks** con la tarea configurada a las 2 de la mañana.



Anexo 8. Repositorio GitHub

Imagen del repositorio con la estructura de carpetas: `01_ingesta`, `02_clean`, `03_load_to_cosmos`, `fastapi/`.



Rutas para el repositorio:

- El repositorio es privado y se ha dado acceso a Santiago Mota y a Carlos Ortega en los siguientes emails: **smota.clases@gmail.com** y **cof@qualityexcellence.es**
- **Link: https://github.com/Trolobubu/TFM-Pipeline-ETL-NMBS**

Video:

https://drive.google.com/file/d/135acukN-sFDi6IlypIjdtcztF6ELjse-/view?usp=sharing