F25 - EA4. Documentación de la Arquitectura y Modelo de Datos

Sergio Andres Rios Gómez
Edwin Alexander Ibarra Ortiz

IU Digital de Antioquia
Ingeniería de Software y Datos
2025

IU Digital de Antioquia

Contenido

Introducción

Este proyecto se enfoca en la construcción de una infraestructura y arquitectura para el procesamiento de Big Data, demostrando un flujo de trabajo completo que abarca desde la ingestión de datos desde una API hasta el enriquecimiento y la preparación para análisis posteriores. El objetivo principal es garantizar la calidad y confiabilidad de los datos mediante etapas de preprocesamiento y limpieza.

El proyecto utiliza GitHub Actions para automatizar el proceso de ingestión de datos, como se define en el workflow bigdata.yml. Este proceso incluye la clonación del repositorio, la creación y activación de un entorno virtual, la instalación de dependencias y la generación de un identificador único para los archivos. Además, se configuran los directorios necesarios para almacenar la base de datos (db) y los archivos de evidencia (Excel en xlsx y auditoría en auditoria).

Posteriormente, el proyecto aborda el preprocesamiento y la limpieza de los datos para asegurar su calidad. Este proceso incluye la eliminación de valores nulos y atípicos, así como la corrección de tipos de datos, lo que resultó en una reducción del tamaño del conjunto de datos al eliminar registros inconsistentes o incompletos. Se genera un archivo de datos limpios en formato CSV y un archivo de auditoría que documenta las operaciones realizadas.

Finalmente, se lleva a cabo el enriquecimiento de los datos mediante la integración de información adicional de múltiples fuentes y formatos, lo que aumenta la profundidad y relevancia del dataset. Se aplican técnicas de transformación y limpieza para garantizar la consistencia y normalización de los datos, y se genera un registro de auditoría de estas operaciones. Los datos enriquecidos se almacenan de manera eficiente en SQLite, optimizando su organización para futuras consultas.

En resumen, este proyecto demuestra un proceso integral para la gestión de datos en una plataforma de Big Data, utilizando herramientas como GitHub Actions, Python y Jupyter Notebook, con el fin de obtener un conjunto de datos limpio, enriquecido y listo para su análisis. Los archivos de la base de datos y de evidencia se generan y actualizan automáticamente con cada ejecución del workflow.

Visión global

Una visión global de este proyecto se puede definir como la creación de una infraestructura integral para el procesamiento de Big Data, que abarca todo el ciclo de vida de los datos, desde su ingestión inicial hasta su preparación final para el análisis. El objetivo principal es establecer un flujo de trabajo automatizado y robusto que garantice la calidad, confiabilidad y utilidad de los datos.

Esta visión global se articula a través de las siguientes etapas clave, respaldadas por las fuentes:

- Ingestión de Datos: El proyecto comienza con la extracción de datos desde una API. Este proceso está automatizado mediante GitHub Actions, utilizando un workflow definido en bigdata.yml. La automatización incluye la configuración del entorno, la instalación de dependencias y la generación de archivos necesarios. Se definen URLs de APIs y rutas de archivos en el .env.
- Preprocesamiento y Limpieza de Datos: Una etapa crucial es asegurar la calidad de los datos mediante su limpieza y preprocesamiento. Esto implica la eliminación de valores nulos y atípicos, así como la corrección de tipos de datos. Este proceso resulta en un conjunto de datos más reducido, pero de mayor calidad. Se genera un archivo de datos limpios (CSV) y un archivo de auditoría que documenta las operaciones realizadas.
- Enriquecimiento de Datos: El proyecto busca aumentar el valor de los datos mediante su enriquecimiento con información adicional de diversas fuentes y formatos. Esto incluye la integración y normalización de datos, lo que mejora la profundidad y relevancia del dataset para análisis posteriores. Se genera un registro de auditoría de este proceso.
- Almacenamiento Optimizado: Los datos enriquecidos se almacenan de manera eficiente en una base de datos SQLite, con una estructura que optimiza la organización y facilita futuras consultas.

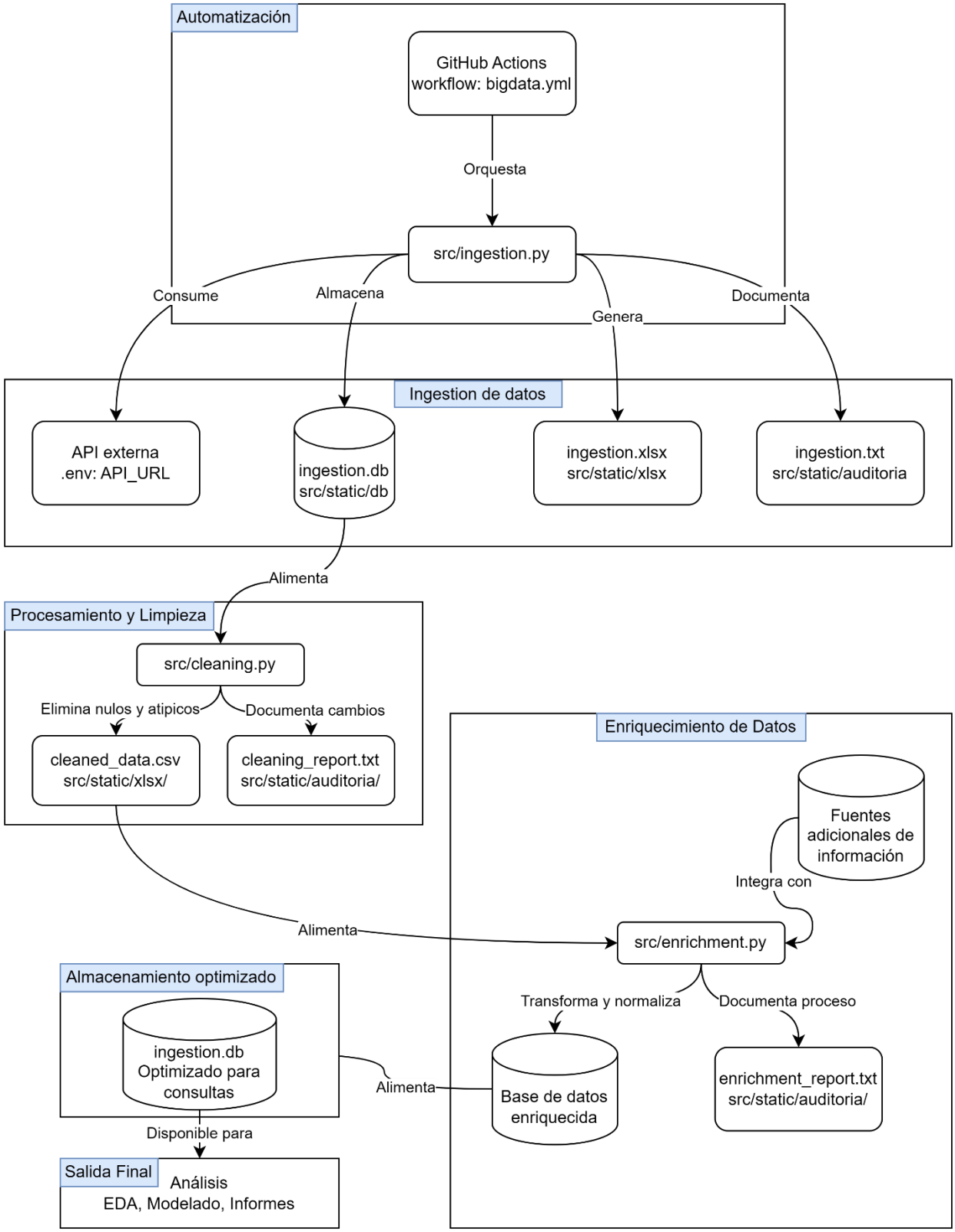
Componentes principales

- Ingestión de Datos: Este componente se encarga de la extracción inicial de datos desde una API. Se utilizan GitHub Actions para automatizar este proceso mediante un workflow definido en el archivo bigdata.yml. Este workflow incluye tareas como clonar el repositorio, configurar un entorno virtual, instalar dependencias y ejecutar el script de ingestión (src/ingestion.py). Se definen las URLs de las APIs que se van a consumir, como API_URL.
- Preprocesamiento y Limpieza de Datos: Una vez que los datos son ingeridos, este componente se enfoca en mejorar su calidad y confiabilidad. Esto implica la eliminación de valores nulos y atípicos, así como la corrección de los tipos de datos. Como resultado de este proceso, el tamaño del conjunto de datos puede reducirse significativamente, como se observó al pasar de 5,478 a 2,163 registros. Se genera un archivo de datos limpios (en formato CSV, cleaned_data.csv o cleaning_.csv) y un archivo de auditoría (cleaning_report.txt) que documenta las operaciones realizadas. El script principal para esta etapa es src/cleaning.py.
- Enriquecimiento de Datos: Este componente tiene como objetivo aumentar la profundidad y relevancia del dataset mediante la integración de información adicional de múltiples fuentes y formatos (JSON, XLSX, CSV, XML, HTML, TXT). Se aplican técnicas de transformación y limpieza para asegurar la consistencia y normalización de los datos. Se genera un registro de auditoría (enrichment_report.txt) que documenta las operaciones de enriquecimiento. El script encargado de esta fase es src/enrichment.py.
- Almacenamiento de Datos: Los datos procesados y enriquecidos se almacenan en una base de datos SQLite (ingestion.db). La estructura de almacenamiento se optimiza para facilitar futuras consultas, incluso simulando particiones.
- Automatización con GitHub Actions: GitHub Actions es un componente central que automatiza el flujo de trabajo de ingestión de datos. El workflow definido en bigdata.yml orquesta la ejecución de las diferentes etapas, desde la clonación del repositorio hasta el commit y push de los archivos generados.
- Archivos de Evidencia y Auditoría: A lo largo del proyecto, se generan varios archivos para documentar y verificar el proceso. Estos incluyen:
- Archivos de base de datos (db).
- Archivos Excel (xlsx).
- Archivos de auditoría (auditoria, cleaning_report.txt, enrichment_report.txt) que registran las operaciones realizadas y el estado de los datos.
- Archivos de datos limpios (CSV).

- Scripts en Python: El proyecto utiliza principalmente scripts en Python (ingestion.py, cleaning.py, enrichment.py) para llevar a cabo las tareas de ingestión, preprocesamiento, limpieza y enriquecimiento de datos.
- Archivos de Configuración: Se utiliza un archivo .env para manejar variables de entorno como las URLs de las APIs y las rutas de los archivos.

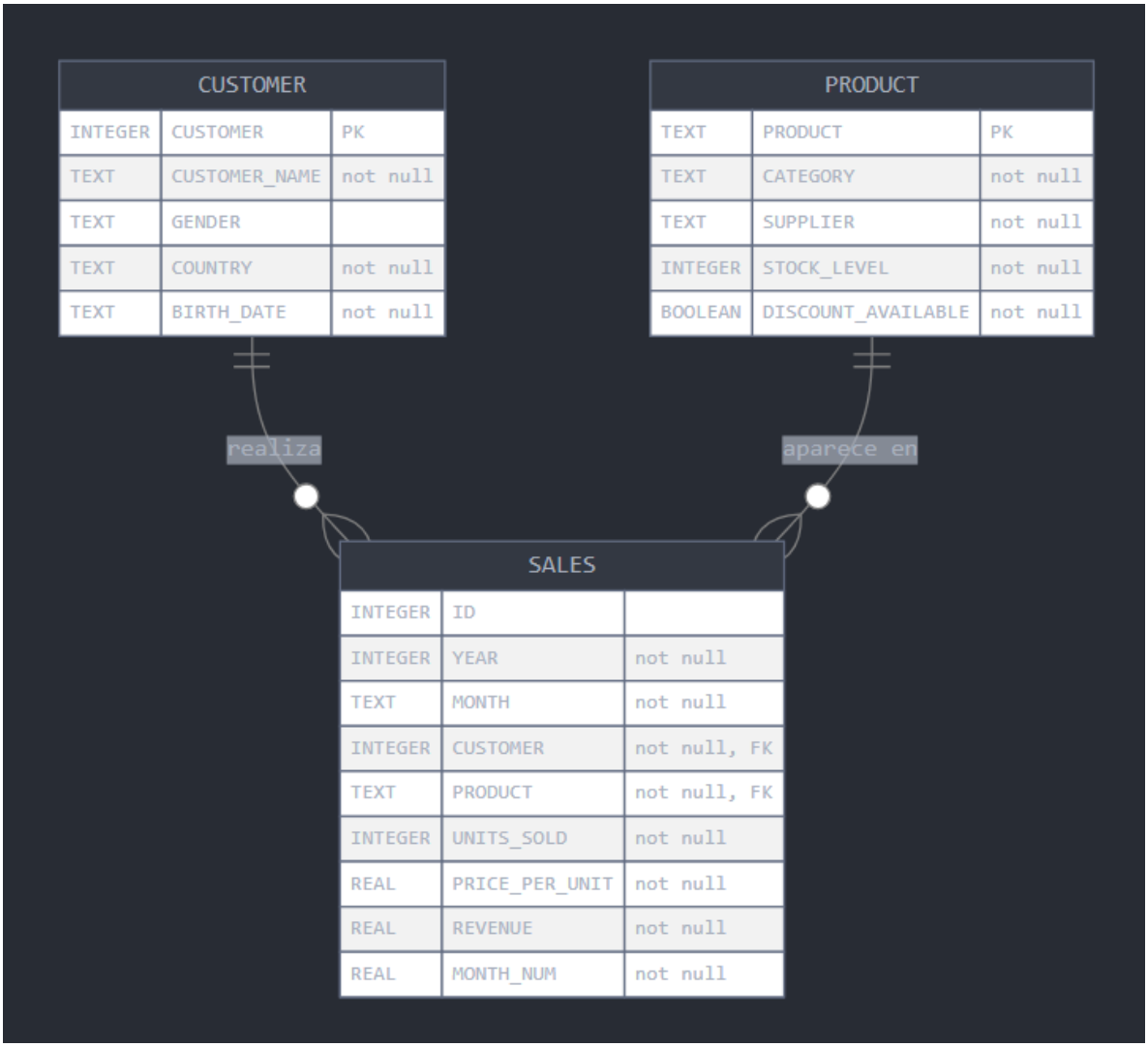
Estos componentes trabajan de manera coordinada para construir una infraestructura robusta para el procesamiento de Big Data, desde la obtención inicial de los datos hasta su preparación para análisis posteriores.

Diagrama de arquitectura



Modelado de datos

Diagrama



Definición del esquema

- Tabla SALES:

Esta tabla contiene los detalles de cada transacción de venta individual.

Las columnas ID, YEAR, MONTH, CUSTOMER, PRODUCT, UNITS_SOLD, PRICE_PER_UNIT, REVENUE y MONTH_NUM que se incluyen aquí son consistentes

con las columnas que identificamos anteriormente como provenientes de la ingestión inicial de datos.

La designación de CUSTOMER y PRODUCT como claves foráneas (FK) que referencian a las tablas CUSTOMER y PRODUCT respectivamente, es lógica ya que cada venta está asociada a un cliente y a un producto específico.

Las restricciones not_null en YEAR, MONTH, CUSTOMER, PRODUCT, UNITS_SOLD, PRICE_PER_UNIT, REVENUE y MONTH_NUM sugieren que, después del proceso de limpieza de datos donde se eliminaron valores nulos, estas columnas deberían tener valores para cada registro de venta.

- Tabla CUSTOMER:

Esta tabla almacena información sobre los clientes.

La columna CUSTOMER se define como la clave primaria (PK), lo que permite identificar de manera única a cada cliente. Esto coincide con su uso como clave foránea en la tabla SALES.

Las columnas CUSTOMER_NAME, GENDER, COUNTRY y BIRTH_DATE son columnas que identificamos previamente como muy probablemente añadidas durante la fase de enriquecimiento de datos, donde se integró información adicional sobre los clientes. La restricción not_null en CUSTOMER_NAME, COUNTRY y BIRTH_DATE implica que esta información está disponible para todos los clientes en el modelo final, lo cual podría ser un resultado del proceso de enriquecimiento.

- Tabla PRODUCT:

Esta tabla contiene información sobre los productos.

La columna PRODUCT se define como la clave primaria (PK), permitiendo identificar de forma única cada producto. Esto también coincide con su uso como clave foránea en la tabla SALES.

Las columnas CATEGORY, SUPPLIER, STOCK_LEVEL y DISCOUNT_AVAILABLE también se identificaron anteriormente como resultado del enriquecimiento de datos, proporcionando detalles adicionales sobre los productos. La restricción not_null en estas columnas sugiere que esta información se obtuvo para todos los productos.

Relaciones: Las relaciones definidas en el diagrama son coherentes con un modelo de datos de ventas:

SALES ||--o{ CUSTOMER : "tiene": Esto indica una relación de "muchos a uno" desde SALES hacia CUSTOMER (muchas ventas pueden pertenecer a un mismo cliente).

SALES ||--o{ PRODUCT : "incluye"}: Esto indica una relación de "muchos a uno" desde SALES hacia PRODUCT (muchas ventas pueden incluir el mismo producto).

Justificación

El modelo de datos, representado en el diagrama Entidad-Relación (ER) proporcionado, se diseñó de esta manera para organizar de forma eficiente y estructurada los datos de ventas después de que han pasado por las etapas de ingestión, limpieza y enriquecimiento. La estructura del modelo busca minimizar la redundancia de datos y facilitar la integración y el análisis de la información en las etapas posteriores del proyecto.

Justificación de tecnologías y herramientas:

Elección de herramientas

El uso de herramientas y tecnologías utilizadas para este proyecto se justifica en base a su funcionalidad, su adecuación para las tareas específicas del flujo de trabajo de datos (ingestión, limpieza, enriquecimiento) y los beneficios que aportan al proyecto en términos de automatización, gestión de datos y colaboración.

A continuación, se presenta la justificación de las principales herramientas y tecnologías utilizadas:

GitHub: La selección de GitHub como plataforma central del proyecto se justifica por varias razones:

- Control de versiones: Permite la gestión del código fuente de manera eficiente, facilitando el seguimiento de cambios, la colaboración entre los integrantes del equipo y la posibilidad de revertir a versiones anteriores si es necesario.
- Colaboración: Proporciona herramientas para la colaboración a través de "issues", "pull requests" y "discussions", lo que facilita la planificación, revisión de código y comunicación entre los miembros del equipo.
- Automatización (GitHub Actions): Ofrece la capacidad de automatizar flujos de trabajo, como el proceso de ingestión de datos, lo cual es crucial para la eficiencia y la consistencia del proyecto.

GitHub Actions: La implementación de GitHub Actions para automatizar el proceso de ingestión de datos se justifica por los siguientes beneficios:

- Automatización del flujo de trabajo: Permite ejecutar las tareas de ingestión de datos de forma automática y programada (aunque no se especifica una programación en las fuentes), eliminando la necesidad de intervención manual repetitiva.
- Consistencia y confiabilidad: Asegura que el proceso de ingestión se realice de manera consistente en cada ejecución, reduciendo el riesgo de errores humanos.
- Eficiencia: Acelera el proceso de obtención y almacenamiento de datos, permitiendo dedicar más tiempo a las etapas posteriores de análisis.
- Generación automática de evidencia: El workflow incluye la creación de archivos de evidencia y la actualización del repositorio, lo que facilita el seguimiento y la verificación del proceso.

Python: La elección de Python como lenguaje de programación principal se fundamenta en su:

- Amplias bibliotecas: Python cuenta con una gran cantidad de bibliotecas poderosas para el manejo de datos (como pandas, aunque no se mencione explícitamente, es una suposición razonable dada las tareas de limpieza y enriquecimiento), la interacción con APIs y bases de datos (como SQLite).
- Facilidad de uso y aprendizaje: Su sintaxis clara y legible facilita el desarrollo y el mantenimiento del código.
- Versatilidad: Es un lenguaje adecuado para diversas tareas dentro del flujo de trabajo de datos, desde la extracción y transformación hasta el análisis y la generación de informes.

SQLite: La elección de SQLite como base de datos para almacenar los datos ingeridos se puede justificar por:

- Simplicidad: Es una base de datos ligera y fácil de configurar, ya que no requiere un servidor independiente.
- Portabilidad: Los datos se almacenan en un único archivo, lo que facilita su transporte y gestión.
- Adecuado para prototipos y proyectos de menor escala: Para las etapas iniciales del proyecto y el volumen de datos que se maneja en la descripción, SQLite puede ser una solución adecuada antes de escalar a sistemas de bases de datos más robustos si fuera necesario. La mención de optimización del almacenamiento en SQLite también sugiere una consideración de su eficiencia para las consultas.

Jupyter Notebook: La presencia de un archivo .ipynb (validate.ipynb) sugiere el uso de Jupyter Notebooks, lo cual se justifica por:

- Exploración y análisis interactivo: Permite combinar código ejecutable, visualizaciones y texto explicativo en un mismo documento, lo que facilita la exploración de datos, la experimentación y la documentación del proceso.•

.md (Markdown): Se utiliza para generar informes de manera legible y estructurada.

Simulación del entorno Cloud

la simulación del entorno en la nube se basa en la utilización de herramientas y tecnologías locales (SQLite para la base de datos, scripts de Python para el procesamiento, el sistema de archivos local para el almacenamiento) orquestadas mediante GitHub Actions, para replicar la lógica y el flujo de trabajo que se encontrarían en una plataforma de big data en la nube. La optimización del almacenamiento en SQLite simulando particiones es un ejemplo específico de cómo se intenta conceptualmente imitar aspectos de la infraestructura de nube a nivel local.

Flujo de Datos y Automatización

El flujo de datos a lo largo de las distintas etapas del proyecto, desde la ingesta hasta la construcción del modelo de datos, se puede detallar de la siguiente manera, resaltando la automatización implementada mediante GitHub Actions

1. Ingesta de Datos: La primera etapa del proyecto consiste en la ingesta de datos desde una API, se define una API_URL (dataset de Kaggle).

Un script de Python, presumiblemente src/ingestion.py, es el encargado de extraer los datos de estas APIs.

GitHub Actions automatiza este proceso, ejecutando el script de ingestión como parte de su flujo de trabajo definido en bigdata.yml. Los pasos incluyen clonar el repositorio, configurar un entorno virtual, instalar dependencias y finalmente ejecutar el script de ingestión.

Durante esta etapa, se genera un UUID único para identificar los archivos.

Los datos extraídos del API se almacenan en una base de datos SQLite local ubicada en la ruta definida por DB_PATH ('src/static/db/ingestion.db'). Además, se genera un archivo Excel (SAMPLE_FILE_PATH) y un archivo de auditoría (AUDIT_FILE_PATH) como evidencia del proceso de ingestión. Estos archivos se almacenan en los directorios /static/db/, /static/xlsx/ y /static/auditoria/ creados para este propósito.

Finalmente, GitHub Actions realiza un commit y push de los archivos generados al repositorio, manteniendo un registro de los datos ingeridos.

2. Preprocesamiento y Limpieza de Datos: La segunda etapa se centra en garantizar la calidad y confiabilidad de los datos.

El script src/cleaning.py se encarga de leer los datos (presumiblemente de la base de datos SQLite o del archivo Excel generado en la etapa anterior) y realizar operaciones de limpieza

Las operaciones realizadas incluyen la eliminación de valores nulos, el manejo de outliers, la corrección de tipos de datos y la eliminación de registros duplicados (aunque no se encontraron en este caso)

Se genera un archivo CSV con los datos limpios (CLEANED_FILE_PATH definido como 'src/static/xlsx/cleaning_.csv') y un archivo de auditoría (cleaning_report.txt y cleaning_report_.md) que documenta las operaciones realizadas y compara el estado de los datos antes y después de la limpieza

GitHub Actions también automatiza la ejecución de este script, asegurando que el proceso de limpieza se realice de manera consistente.

3. Enriquecimiento de Datos: La tercera etapa busca aumentar la profundidad y relevancia del dataset integrando información adicional de múltiples fuentes y formatos.

El script src/enrichment.py es responsable de esta tarea JSON, XLSX, CSV, XML, HTML y TXT .

Se aplican técnicas de transformación y limpieza para garantizar la consistencia y normalización de los datos enriquecidos, incluyendo la estandarización de nombres de columnas y la conversión de tipos de datos.

Se genera un archivo de auditoría (enrichment_report.txt) que documenta las operaciones de enriquecimiento, la cantidad de registros coincidentes, las transformaciones aplicadas y posibles discrepancias detectadas.

El dataset enriquecido se almacena nuevamente en la base de datos SQLite, optimizando la organización de la información y facilitando futuras consultas. GitHub Actions también automatiza la ejecución de este script.

Conclusiones y recomendaciones

Beneficios Principales de la Arquitectura y el Modelo de Datos Propuestos:

- Automatización del Flujo de Datos: El uso de GitHub Actions para automatizar el proceso de ingesta, limpieza y enriquecimiento de datos es un beneficio significativo. Permite una ejecución consistente y programada de las diferentes etapas del proyecto, simulando la automatización de pipelines de datos en la nube.
- Claridad en las Etapas del Procesamiento de Datos: El proyecto define claramente las etapas de ingesta, preprocesamiento y limpieza, y enriquecimiento de datos. Cada etapa tiene un script de Python dedicado (ingestion.py, cleaning.py, enrichment.py) y genera archivos de evidencia y auditoría, lo que facilita la comprensión y el seguimiento del flujo de datos.
- Mejora de la Calidad y el Valor de los Datos: Los procesos de limpieza y enriquecimiento han demostrado una mejora en la calidad de los datos al eliminar valores nulos, manejar outliers y corregir tipos de datos. El enriquecimiento busca aumentar la profundidad y relevancia del dataset al integrar información de múltiples fuentes.
- Modelo de Datos Estructurado (SQLite): El uso de SQLite como base de datos local proporciona una forma estructurada de almacenar los datos ingeridos, limpios y enriquecidos. La simulación de particiones sugiere una consideración de la organización de los datos para futuras consultas.

Posibles Limitaciones de la Arquitectura y del Modelo de Datos Propuestos:

- Simulación Local vs. Entorno Real en la Nube: La arquitectura actual simula un entorno de nube utilizando herramientas locales. Esto implica que no se están utilizando servicios nativos de la nube, lo que podría generar limitaciones en términos de escalabilidad, procesamiento distribuido y la gama completa de servicios disponibles en plataformas de nube reales (AWS, Azure, Google Cloud)
- Limitaciones de SQLite: Aunque SQLite es útil para el desarrollo y la simulación, puede no ser la opción más adecuada para manejar grandes volúmenes de datos o cargas de trabajo concurrentes en un entorno de producción en la nube.

Las bases de datos nativas de la nube ofrecen mayor escalabilidad, disponibilidad y funcionalidades avanzadas y para ello podríamos hacer uso de bases de catos como cosmoDB o cassandra.

Recomendaciones para Futuras Mejoras o para la Implementación en un Entorno Real de Nube:

Migración a Servicios Nativos de la Nube: Para una implementación real, se recomienda migrar los componentes de la arquitectura a servicios equivalentes en una plataforma de nube (por ejemplo, AWS, Azure o Google Cloud). Esto podría incluir:

- Almacenamiento de Datos: Utilizar servicios de almacenamiento escalables como AWS S3, Azure Blob Storage o Google Cloud Storage para los archivos de datos y evidencia.
- Bases de Datos: Emplear bases de datos gestionadas en la nube como AWS RDS/DynamoDB, Azure SQL/Cosmos DB o Google Cloud SQL/Spanner/Bigtable.
- Procesamiento de Datos: Utilizar servicios de procesamiento de datos en la nube como AWS Lambda/Glue, Azure Functions/Data Factory o Google Cloud Functions/Dataflow para ejecutar los scripts de ingestión, limpieza y enriquecimiento de manera escalable y serverless.
- Orquestación de Flujos de Trabajo: Emplear servicios de orquestación de flujos de trabajo en la nube como AWS Step Functions, Azure Logic Apps o Google Cloud Workflows para definir y gestionar los pipelines de datos, en lugar de depender únicamente de GitHub Actions para la automatización en un entorno de producción.
- Implementación de un Framework de Procesamiento Distribuido: Para manejar grandes volúmenes de datos, considerar la integración con frameworks de procesamiento distribuido como Apache Spark o Apache Flink, que pueden ejecutarse en servicios de nube como AWS EMR, Azure HDInsight o Google Cloud Dataproc.
- Establecimiento de Monitoreo y Logging: Implementar sistemas de monitoreo y logging para rastrear el rendimiento de los pipelines de datos, identificar posibles errores y garantizar la disponibilidad y confiabilidad del sistema en la nube.