Trabajo Fin de Máster

Máster en Ciencia de Datos e Ingeniería de Datos en la Nube

Procesamiento y análisis de una base de datos masiva de libros

Autor: Juan Jesús Fernández Fernández

Tutor: Luis de la Ossa

Junio, 2023

Resumen

En el presente trabajo se ha llevado a cabo el procesamiento de datos de obras (unos 30 millones de obras y 11 millones de autores) disponibles en la plataforma Open Library, que ofrece archivos de texto descargables con todos los datos disponibles hasta la fecha además de una API adicional para volúmenes bajos de consulta que puede complementar los archivos originales.

El procesamiento de datos se ha hecho siguiendo una arquitectura de medallón (Medallion o MultiHop), la cual sigue unos pasos de procesamiento comenzando con la carga de los datos en crudo en la capa Bronce (Bronze layer), tras la cual se realiza una primera limpieza y procesamiento en la capa Plata (Silver layer) y por último se hace una última transformación en la capa Oro (Gold layer) en la cual se presentan los datos preparados para ser utilizados en reportes por el usuario final.

El objetivo del proyecto es el de procesar y analizar los 30 millones de obras y sus datos adicionales para proporcionar un conjunto de datos final con los 1000 libros con más reseñas para nuestro departamento de Marketing, el cual necesita los datos solicitados para identificar los libros más populares para sus publicaciones con el objetivo de llegar al máximo número de usuarios en sus cuentas en redes sociales relacionadas con la lectura.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Introducción

En la actualidad, cada vez es más común encontrarse con grandes volúmenes de datos que se escapan de las capacidades de procesamiento de ordenadores personales. Si hace pocos años un ordenador con Excel permitía analizar los datos de la mayoría de pequeñas y medianas empresas, los conjuntos de datos actuales necesitan cada vez más de mayores capacidades de procesamiento, más allá de las capacidades del hardware disponible en las empresas.

Es por esto que los servicios en la nube ponen a nuestro alcance, a un precio asequible, la capacidad de procesamiento necesaria bajo demanda, reduciendo la necesidad de adquirir hardware con dicho propósito. El pago por uso y el serverless se encuentran disponibles en la mayoría de proveedores de Public Cloud, como AWS, Microsoft Azure, GCP (Google Cloud) o IBM Cloud. En todas estas plataformas hay disponibles servicios de almacenamiento y procesamiento de datos. Además, existen otras plataformas que utilizan la capacidad de procesamiento de dichos proveedores y que han construido sus servicios en la nube utilizando los procesadores en su arquitectura. En nuestro caso en particular, utilizaremos una de estas plataformas: Databricks.

1.2 Objetivos

El objetivo del proyecto es el de procesar los archivos disponibles obtenidos de la plataforma Open Library, la cual permite descargar todos los datos disponibles hasta el momento en formato de texto comprimido txt.gz.

En este supuesto práctico, se quiere simular el trabajo para resolver una petición de nuestro departamento de Marketing para obtener un archivo que se pueda manejar en Excel y será necesario proporcionar un dataset final con los 1000 libros con más reseñas y algunos datos adicionales. Nuestros compañeros de Marketing necesitan los datos para elegir los libros más populares para sus publicaciones con el objetivo de llegar al máximo número de usuarios en las cuentas de redes sociales relacionadas con la lectura.

Los archivos iniciales contienen 30 millones de libros, 11 millones de autores y 350 mil reseñas. Es un volumen lo suficientemente grande como para no permitir el procesamiento en local utilizando la herramienta más utilizada en las oficinas: Excel.

Además, haremos uso de la API disponible para obtener algunos datos más que nos serán de utilidad para alcanzar el objetivo de identificar los libros más populares entre nuestro público, como el número de usuarios que han guardado un libro en su lista y las valoraciones otorgadas.

Los archivos iniciales están disponibles tanto en Azure Data Lake Storage gen.2 (Almacenamiento en la nube de Microsoft, equivalente a AWS S3) como en S3.
Estos archivos serán leídos desde Databricks, para ser procesados, analizados y exportados a un dataset reducido que será almacenado en S3 y que permitirá un procesamiento en local para enriquecerlo con más datos asociados a cada obra llamando a la API. Debido a que la API tiene un cupo muy limitado de solicitudes, es necesario seleccionar de antemano que obras van a incluirse en el archivo final. El archivo final será de nuevo puesto a su disposición cargándolo en S3.

1.3 Estructura del proyecto

Primera parte, Databricks: La primera parte del proyecto ha sido desarrollada en un notebook de Databricks (.dbc), el cual sigue una estructura de procesamiento en tres capas, bronze, silver y gold para cada uno de los archivos disponibles convertidos en tablas, seguido de un análisis preliminar para identificar las obras que tienen disponibles valoraciones y que son candidatas a pasar al dataset final.

Tecnologías relevantes: Databricks, Spark (PySpark, Spark SQL), Azure ADLS2, AWS S3

Segunda parte, Jupyter notebook: Una vez hecho el procesamiento inicial en la nube y se ha reducido el dataset a las obras con reviews (unas 300.000), se continúa en local (se ha considerado esta opción para reducir costes ya que el volumen de datos ya se considera lo suficientemente pequeño para ser procesarlo en un ordenador personal), donde se continúa trabajando en un notebook de Jupyter (.ipynb) en el cual se seleccionan los 1000 libros con más valoraciones y se hacen las llamadas a la API para solicitar más datos, como parte del enriquecimiento del dataset, y se calculan algunas métricas que podrían ser de utilidad para el equipo de marketing.

Tecnologías relevantes: Python, API calls, AWS S3

Capítulo 2

Estado del Arte

2.1 Introducción

Actualmente, con el crecimiento en popularidad de los datalakes y el Data Lakehouse (como evolución del clásico Data Warehouse construido directamente sobre un Data Lake), es común encontrar empresas que utilicen una arquitectura medallion, así como un modelado de datos basado en Data Vault en lugar del clásico Star Schema (Kimball). En la primera parte del proyecto, se hace uso de la capacidad de procesamiento de Spark en la plataforma Databricks, la cual ofrece una versión gestionada que facilita el uso de Spark y abstrae gran parte de la complejidad de gestionar su arquitectura.

2.2 Data Lakes

El Data Lake es un almacén de datos que puede albergar prácticamente cualquier formato de archivo. Con el auge en bases de datos no relacionales y tecnologías basadas en datos semiestructurados y no estructurados, el Data Lake se ha convertido en el punto de encuentro de múltiples fuentes de datos, pudiendo almacenar no solo datos tabulares como en las bases de datos basadas en SQL, sino imágenes, documentos y archivos binarios.

2.3 Data Lakehouse

El concepto de Data Lakehouse es relativamente reciente y la idea principal que lo define es la sinergia conseguida con la unión de un Data Lake y un Data Warehouse en una plataforma que combina lo mejor de cada una de las partes.

Sin embargo, estos beneficios no vienen solos, ya que también añade cierta complejidad y requiere cierta experiencia en el trabajo con datos. Por ejemplo, desde el punto de vista de un analista de datos que centra su trabajo en obtener información usando SQL, un Data Lakehouse no trae muchos beneficios con respecto a una base de datos columnar como Amazon Redshift. Por otro lado, para un ingeniero de datos facilita el flujo de trabajo, ya que desde una misma plataforma se tiene acceso a los datos desde su fase en crudo hasta los datos ya procesados.

Otra desventaja es que el Data Lakehouse también puede convertirse en un pozo de datos, en los que se van almacenando todos los datos en crudo hasta llegar a ser ingobernable. Además, por el momento, las plataformas como Databricks no ofrecen un gobierno del dato tan avanzado como otras bases de datos SQL, plataformas como Synapse Analytics o sistemas BI como Power BI, en referencia al control de la exposición de filas y columnas.

2.4 Databricks

El propósito de Databricks es el de aprovechar y unificar las ventajas del Data Lake, donde se pueden almacenar grandes volúmenes de datos en archivos, con las de un Data Warehouse, donde se puede acceder a tablas que están disponibles para su utilización en analítica, reportes y/o machine learning, y la capacidad de procesamiento de Spark.

Spark permite leer archivos directamente desde el Data Lake (como S3, ADLS2) y almacenar el resultado después de su procesado de vuelta al Data Lake como, por ejemplo, en archivos parquet. Existen alternativas como AWS EMR, que al igual que Databricks, ofrece Spark runtimes. La diferencia es que Databricks también permite crear schemas y tablas como tendríamos en un Data Warehouse tradicional basado en SQL.

Además, también incluye soluciones de Machine Learning, como MLflow y AutoML, y permite la construcción de Dashboards para análisis de datos.

Databricks utiliza la capacidad de procesamiento y almacenamiento disponible en los proveedores de Public Cloud y puede estar basado en AWS, Azure o GCP (Google Cloud), (véase Figura 2.1). En el caso de Azure, el procesamiento de Spark se hace VMs; Virtual Machines de Azure y en el caso de AWS, en Instancias EC2. El almacenamiento, igualmente, está basado en S3 o en ADLS2. Databricks también dispone de su propio sistema de almacenamiento de archivos distribuido, Databricks File System (DBFS), y que es una abstracción de la capa de almacenamiento en la nube.

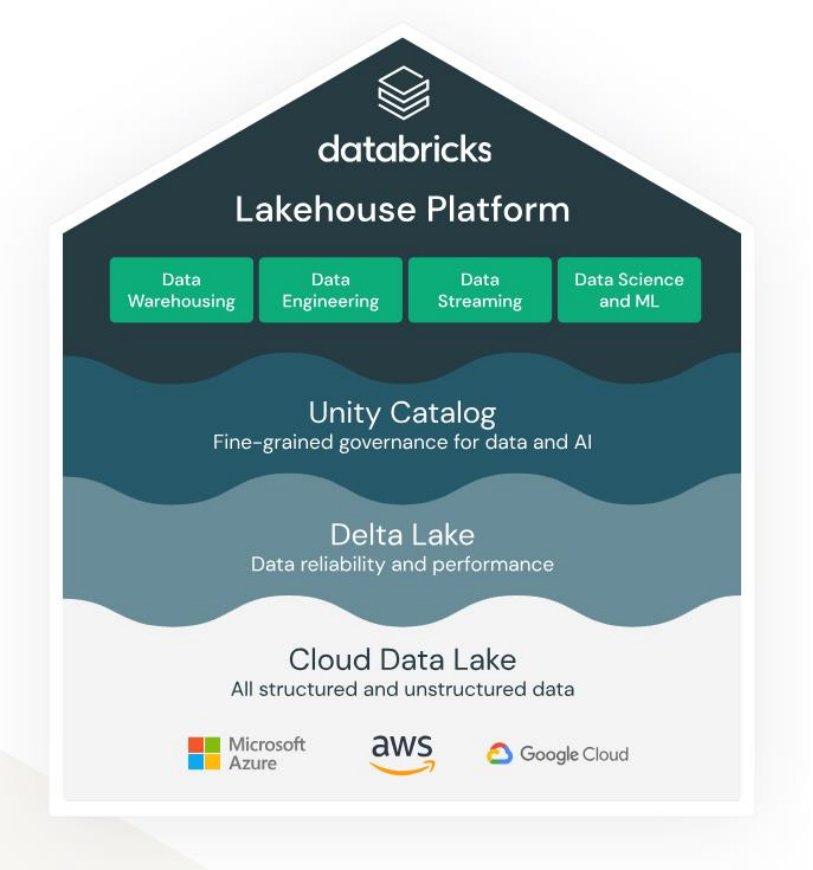

Figura 2.1. Referencia a la plataforma Lakehouse de Databricks

Si atendemos a grandes rasgos cuando usar y cuando no una plataforma como Databricks, podríamos resumirlo en los siguientes puntos:

Cuando es Databricks una buena opción:

Tenemos un gran cantidad de datos a procesar, principalmente almacenados en archivos en un Data Lake como S3 o similar, y trabajar en local o en plataformas no pensadas para trabajar con grandes volúmenes de datos es inviable.

El propósito del data warehouse es la analítica, el machine learning y la ingeniería de datos para exponer tablas en un sistema OLAP (Online Analytical Processing).

Cuando NO es Databricks una buena opción:

- La cantidad de datos es relativamente pequeña y una base de datos relacional con índices (indexing) es suficiente para el caso de uso. Se debe tener en cuenta que Spark requiere la puesta en marcha de clústeres y que los beneficios son visibles cuando se procesan grandes cantidades de datos. Para volúmenes de datos pequeños, el tiempo que tarda una consulta a la base de datos puede llegar a ser mucho menor que el tiempo de puesta en marcha del clúster.
- Se va a trabajar con datos transaccionales en un sistema OLTP (Online Transaction Processing). Como hemos comentado, Spark está pensado para tablas analíticas. Es posible ingerir datos transaccionales para transformarlos en tablas analíticas, pero no se debe usar como base de datos transaccional con tablas normalizadas.

2.5 Arquitectura de Medallón

La arquitectura recomendada para trabajar en Databricks es la conocida como Arquitectura de Medallón (Medallion Architecture), la cual describe una serie de capas que definen la calidad de los datos en cada una de ellas.

- La primera capa es la Bronce, donde se almacenan los datos en crudo. Son datos no validados que son cargados directamente en batches o streaming. Por lo general, es posible cargar datos provenientes de batch y stream en la misma tabla. Los datos son adjuntados (appended) al final de la tabla y se espera que en el siguiente paso se lleven a cabo procesos de deduplicado y de calidad del dato, evitando en la medida de lo posible transformar los datos en este paso.

- La segunda capa es la Plata, donde se hace la limpieza de los datos como el deduplicado, transformación, comprobación o casteo de los tipos de datos, etc. Las tablas Plata ya pueden ser utilizadas en distintos procesos que requieran de datos limpios no agregados.
- La tercera capa es la Oro, donde se presentan los datos enriquecidos y/o agregados, listos para ser usados por otros sistemas (BI, reportes, machine learning). Son tablas normalmente desnormalizadas pensadas para ser usadas en analítica de datos. La capa Oro suele presentarse según un modelo de estrella, a diferencia de las capas Bronce y Plata, que suelen tener un modelado en Data Vault.

La arquitectura de medallón no excluye y es independiente a otros tipos de modelado que se lleven a cabo en cada una de las capas. Es decir, las tablas de cada una de las capas pueden a su vez formar parte de un modelo de estrella (Star Schema), un modelo de Data Vault o cualquier otra arquitectura de modelado de datos.

2.6 Tablas Delta

Otro concepto importante cuando se trabaja con Databricks es el de las tablas Delta. Las tablas Delta de Databricks son una adaptación del Delta Lake Open Source, el cual permite almacenar el histórico de cambios en la table, permitiendo el viaje en el tiempo (time travel) para volver o simplemente consultar los datos tal y como estaban en un cierto momento en el pasado, pudiendo consultarse especificando la versión de la tabla o una marca de tiempo (timestamp).

2.7 Modelado de datos Data Vault

Por completar la lista de conceptos generales y a pesar de que no se ha utilizado en el presente proyecto, se debe hacer mención al modelado de datos tipo Data Vault por su popularidad en el uso junto a la arquitectura de medallón. Este modelo se compone de 3 tipos de entidades, Hubs, Links and Satellites. Las tablas Hub contienen business keys, es decir, contienen solo la referencia y los campos necesarios para ser conectada a otras tablas que contienen el resto de información sobre ella. Las tablas satélite (Satellites)

contienen la información sobre las business keys contenidas en los Hubs. Las tablas de enlace (Links) sirven para unir distintas tablas Hub, como podemos ver en la figura 2.2.

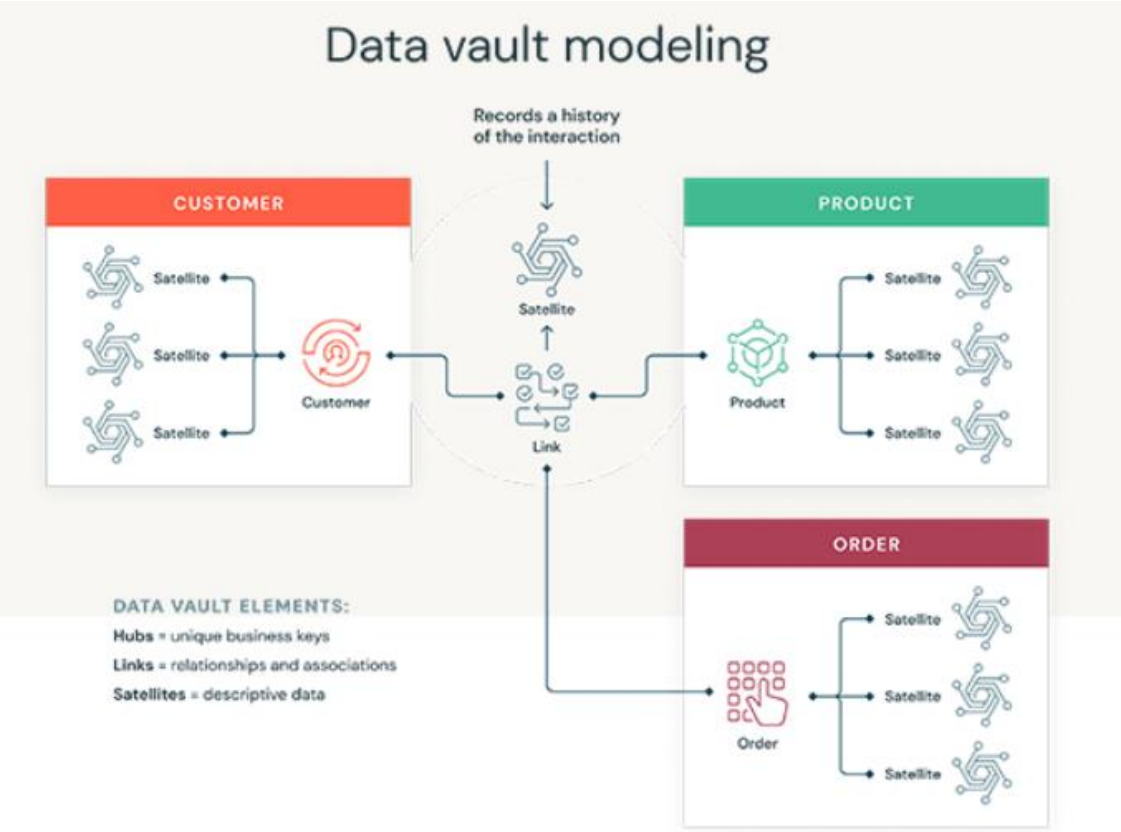

Figura 2.2. Modelado de datos según la arquitectura Data Vault

Este tipo de modelado permite mayor escalabilidad y es utilizado en sistemas Big Data. Además, las Keys suelen estar hasheadas (e.g. mediante funciones tipo xxhash64()) para optimizar las búsquedas a la hora de filtrar o hacer uniones entre tablas. El hash es un algoritmo que mapea los campos a un numero entero.

2.8 Conclusiones

Databricks es una plataforma que permite el acceso a los datos a grupos con distintos propósitos, como los ingenieros de datos que trabajan en ETLs, analistas de datos que preparan cuadros de mando y visualizaciones e ingenieros de machine learning que trabajan en modelos de predicción.

La plataforma está construida sobre el Data Lake y los procesadores en la nube proporcionados por los proveedores de servicios en la nube como AWS, Azure y GCP.

Cuando se utiliza como Data Lakehouse, las arquitecturas más utilizadas son la arquitectura de medallón junto a modelos de datos según arquitecturas de Data Vault y Kimball. Asimismo, se suele tener acceso a las tablas que contienen tanto los datos en crudo hasta los datos ya procesados y listos para su uso en otras funciones de analítica.

Capítulo 3

Metodología y Desarrollo

3.1 Introducción

La metodología seguida se ha basado en un caso práctico simulado. Se ha planteado una situación en la que un equipo de trabajo solicita un archivo con los libros más populares según ciertas métricas, con el problema inicial de que los datos disponibles se encuentran en archivos de texto demasiado grandes para ser gestionados en local por herramientas de uso común como el Excel. Por ello, será necesario hacer uso de herramientas de procesamiento de datos más potentes (en este caso Databricks) para procesar y reducir el dataset a un tamaño manejable por los solicitantes con los datos necesarios. Seguiremos los principios de una arquitectura de medallón en la que crearemos las capas bronce, plata y oro de los conjuntos de datos de obras disponibles en Open Library.

3.2 Capa Bronce

-Disponemos inicialmente de cuatro archivos de texto relativamente grandes que contienen información sobre obras escritas, sus autores e información básica sobre sus reseñas. El primer paso ha sido el de cargar los 4 ficheros al Data Lake (S3 / ADLS2). Inicialmente, se ha trabajado en Azure Databricks y posteriormente se ha modificado la conexión del Data Lake ADLS2 de Azure a Amazon S3. El único cambio necesario es el

proceso de mount del Data Lake y la modificación de las URI de los archivos. Esta es otra de las ventajas de Databricks y su compatibilidad con múltiples cloud vendors.

Tabla 3.2. Información sobre los archivos con los datos iniciales

Nombre del archivo	Tamaño	Número de filas
ol_dump_authors_2023-01-10.txt.gz	505.0 MB	11,316,199
ol_dump_ratings_2023-01-10.txt.gz	3.1 MB	321,435
ol_dump_reading-log_2023-01-10.txt.gz	46.0 MB	4,882,308
ol_dump_works_2023-01-10.txt.gz	2.7 GB	30,852,119

Para cargar los datos a una tabla Bronce, el proceso ha sido el siguiente:

-Se ha cargado el archivo con *spark.read.csv()* en un dataframe de Spark infiriendo el esquema.

-Se han renombrado las columnas y añadido los nombres de las columnas de referencia, ya que los datos en el archivo no los incluye.

-Se han añadido dos columnas de control con la fecha y hora de la carga y el origen del archivo. Al cargar los datos en la table de Bronce, se pueden añadir columnas que faciliten su identificación posterior, como, por ejemplo:

```
df.withColumn("ctl_created_timestamp",
current_timestamp().cast("string"))
      >> En PySpark, añade una columna con el timestamp.

SELECT …. , input_file_name() AS ctl_file_name FROM ….
      >> En Spark SQL, añade una columna con el nombre del archivo de
origen.
```

Figura 3.1. Ejemplo de columnas de control

-Se ha creado una vista temporal para añadir la información necesaria (control columns) para luego crear una tabla con un CTAS (CREATE TABLE name AS)

3.3 Capa Plata

En la segunda capa de procesamiento, se hacen las transformaciones necesarias para limpiar y organizar los datos. En las tablas autores (authors) y obras (works), la mayor

parte de la información que necesitamos excepto la referencia de la obra viene en una sola columna como JSON y ha permanecido así en la capa Bronce, tal y como se ve en el siguiente ejemplo:

```
{"title": "JC de Castelbajac",
"created": {"type": "/type/datetime", "value": "2009-12-
11T01:57:19.964652"}, "covers": [3140687],
"last_modified": {"type": "/type/datetime", "value": "2010-04-
28T06:54:19.472104"},
"latest_revision": 3,
"key": "/works/OL10000387W",
"authors": [{"type": "/type/author_role",
"author": {"key": "/authors/OL3965419A"}}],
"type": {"key": "/type/work"},
"revision": 3}
```

Figura 3.2. Muestra del contenido de una columna en formato JSON

Es posible extraer la información de una columna en JSON definiendo el esquema de las columnas que se van a extraer:

```
schema = StructType(
    [
        StructField('name', StringType(), True),
        StructField('personal_name', StringType(), True)
    ]
)
```

Figura 3.3. Definición del esquema de las columnas a extraer de la columna JSON

y mapeando cada campo en su nueva columna:

```
    df_authors = df_authors.withColumn("json_col", from_json("json_col",
    schema))\
        .select(col('reference').alias('author_ref'), \
                col('json_col.name').alias('author_name'), \ …
```

Figura 3.4. Extracción de la información de una columna JSON

También en la capa plata, se han agrupado las valoraciones por obra, año y mes como preparación a la unión posterior de los datos en la capa Oro.

3.4 Capa Oro

En la tercera capa de procesamiento, se han hecho las uniones entre las tablas de autores, obras y reseñas para crear tablas agregadas que ya sean válidas para consultas de analítica. Además, se ha añadido una nueva columna de control que nos será muy útil a la hora de particionar las tablas y exportarlas, por ejemplo, en ficheros .parquet.

La columna de control, como referencia, extrae el año y el mes del timestamp de creación y los concatena, y en el caso de no existir dicho timestamp, lo establece en 190001 (equivalente a enero de 1900) para asegurar que es incluida en una de las particiones. Se ha hecho de la siguiente manera:

```
                COALESCE(
                CONCAT(      YEAR(w.created_timestamp),
                MONTH(w.created_timestamp)),
                "190001")
                AS ctl_year_month
```

Figura 3.5. Columna de control basada en fecha de creación utilizada en partición

Se han creado varias tablas en la capa Oro como ejemplo de tablas analíticas con el propósito de realizar varias consultas analíticas, aunque a nosotros nos centraremos de aquí en adelante en la tabla que incluye las obras que han recibido alguna reseña, ya que la petición sugiere seleccionar los libros más populares. La tabla Oro mencionada contiene y 268,995 filas y se ha construido de la siguiente manera:

```
CREATE OR REPLACE TABLE gold_openlibrary.works_with_reviews_summary AS
SELECT r.works_reference,
       r.review_year,
       r.review_month,
       r.rating_count,
       r.rating_avg,
       b.title,
       b.author_name,
       b.author_personal_name,
       b.subjects,
       b.description_text,
       b.ctl_year_month
FROM silver_openlibrary.ol_ratings r
INNER JOIN gold_openlibrary.ol_books b
   ON r.works_reference = b.works_reference
```

Figura 3.6. DDL para crear la tabla de obras con reseñas

3.5 Exportación de los datos de la tabla Oro a S3

Una vez tenemos los datos necesarios, los datos son exportados a S3 en archivos parquet, particionados según el año y mes de creación de la entrada. Para ello, se han creado unas funciones que cargan la información de la tabla Delta y escriben un fichero dentro de su key para cada año-mes.

En el Anexo I se ha añadido más información sobre este paso.

3.6 Lectura de los archivos desde S3

El siguiente paso es procesar la selección de libros para enriquecer el dataset con algunos datos disponibles a través de la API desde un Jupyter notebook.

Como el dataset actual es pequeño (unas 300.000 filas) se puede gestionar en local sin necesidad de tener clústers funcionando en la nube.

Los archivos parquet son leídos, almacenados en un dataframe y guardados en un archivo csv que nos permitirá trabajar sin necesidad de recargar los datos de nuevo desde S3. Si se vuelven a ejecutar las funciones de extracción y el fichero ya existe, los datos son cargados desde el fichero y no desde S3.

Tabla 3.6. Dataset con los datos agregados

	works_reference	author_name	title	rating_count	rating_average
148364	/works/OL82563W	J. K. Rowling	Harry Potter and the Philosopher's Stone	55	4.30
95120	/works/OL262758W	J.R.R. Tolkien	The Hobbit	54	4.20
131277	/works/OL5720023W	Stephenie Meyer	Twilight	52	4.07
148326	/works/OL82536W	J. K. Rowling	Harry Potter and the Prisoner of Azkaban	51	4.34
147435	/works/OL81613W	Stephen King	It	50	4.05

3.7 Transformación y agregación de los datos

Los datos importados, los cuales incluyen las reseñas agregadas por año-mes-obra se agregarán de nuevo para obtener los totales para cada obra y filtrar los 1000 libros con más reseñas.

Una vez agrupados y filtrados, y en base a la referencia de la obra, disponemos de un conjunto de datos con los 1000 libros con más reseñas. Para completar dicho dataset con la información necesaria para calcular ciertas métricas, se harán llamadas a la API de Open Library (Developer Center / APIs | Open Library).

Los resultados devueltos por la API son almacenados en un dataframe y guardados también en un fichero. Al igual que con los datos descargados desde S3, si se vuelven a ejecutar las funciones de extracción de datos que llaman a la API y los datos ya están disponibles en local, los datos son cargados desde el archivo csv y no se hacen llamadas a la API.

Tabla 3.7. Dataset con los datos extraídos a través de la API

	work_reference	bs_want_to_read	bs_currently_reading	bs_already_read	rating_average	rating_count	rating_1	rating_2	rating_3	rating_4	rating_5
0	/works/OL3759085W	43	1	40	4.250000	16	1	0	1	6	8
1	/works/OL261196W	53	4	44	4.000000	22	0	1	7	5	9
2	/works/OL262460W	288	14	45	4.043478	23	0	0	8	6	9
3	/works/OL453743W	45	2	38	4.388889	18	0	0	2	7	9
4	/works/OL17043626W	180	5	32	3.750000	24	1	2	6	8	7

3.8 Unión de ambos datasets

El siguiente paso ha sido el de unir ambos datasets mediante su refencia. Como se han extraído únicamente los datos que necesitábamos, la unión es 1:1.

3.9 Cálculo de la métrica de NPS

El NPS, Net Promoter Score, es una métrica muy utilizada que da una idea de la actitud de los clientes ante un producto o servicio. La media de las valoraciones puede dar una idea general, aunque no ofrece información sobre la distribución de las valoraciones.
La métrica del NPS divide las valoraciones de los clientes en Promotores, Detractores y Neutrales de la siguiente manera:

Tabla 3.9. Grupos del Net Promoter Score

Escala	Detractores	Neutrales	Promotores
1-5	1-3	4	5
1-10	1-6	7-8	0-10

Una vez se tienen las valoraciones, la manera de calcularlo es sencilla:

```
NPS(%) = (Promotores - Detractores) / Total de valoraciones
```

Figura 3.7. Fórmula de cálculo de la métrica NPS

El resultado es un porcentaje que puede ir desde el 100% (muy favorable) hasta el -100% (muy desfavorable).

Para un producto con las siguientes valoraciones: 5, 5, 3, 1, el NPS sería:

```
(2 promotores – 1 detractor) / 4 valoraciones totales = 25%
```

Figura 3.8. Ejemplo de cálculo de la métrica NPS

Para los libros, se ha añadido tanto el valor de NPS como el ranking según esta métrica. Como podemos observar, los libros más populares (con mayor número de valoraciones) no son los mejor considerados por los lectores. Sin embargo, hay libros no tan conocidos con un número considerable de reseñas y un valor de NPS mucho mayor. Los libros con un mayor NPS podrían ser ideales para ser recomendados a nuestros lectores, ya que se trata de libros quizá menos conocidos, pero muy bien valorados por sus lectores.

3.10 Conclusiones

En este trabajo se ha intentado presentar un caso de uso de los servicios en la nube junto a lenguajes de programación como Python y PySpark y de consulta a bases de datos como SQL siguiendo todos los pasos para completar una petición que de por sí podría ser bastante frecuente en ciertas empresas que requieren del soporte de los departamentos de tecnología para llevar a cabo ciertas tareas que se escapan de los conocimientos de otras áreas de la empresa.

También se ha demostrado la versatilidad de un Data Lakehouse como es Databricks para llevar a cabo tareas de procesamiento de datos utilizando el lenguaje más adecuado para cada uno de los procesos, pudiendo obtener lo mejor de cada uno de manera sencilla.

Capítulo 4

Conclusiones y Trabajo Futuro

4.1 Conclusiones

En este trabajo, hemos discutido las ventajas y desventajas de utilizar plataformas de procesamiento en la nube como Azure, AWS y Databricks. Hemos visto que cada plataforma tiene sus propias características y beneficios, y que la elección de una plataforma sobre otra dependerá de las necesidades específicas de cada empresa. Por ejemplo, Azure ofrece una amplia gama de servicios y herramientas para el desarrollo de aplicaciones en la nube, mientras que AWS es conocido por su escalabilidad y flexibilidad. Databricks, por otro lado, se destaca por su capacidad para procesar grandes cantidades de datos de manera eficiente.

4.2 Trabajo futuro

Como ampliación del proyecto, se propone la optimización y automatización del proceso de la descarga de datos mensual de los archivos. Se podría investigar la manera más sencilla de descargar los archivos y cargarlos a S3, posiblemente con bash o Python de manera orquestada junto al procesamiento posterior de los datos.

Para reducir los costes y el tiempo de procesamiento, se podría trabajar solamente con los archivos de obras (works) y reseñas (ratings) para extraer solamente la lista de libros y el número de reseñas, ya sea totales o para los meses más recientes.

Una vez recalculada la lista de libros con más reseñas, se extraerían los datos adicionales mediante llamadas a la API como se ha hecho en la segunda parte del proyecto.

A la hora de exportar, no sería necesario cargar los archivos para reseñas de meses pasados, ya que solo se tendrían que considerar las entradas añadidas desde la última carga (> max creation date timestamp).

Con los datos finales, se podría crear y compartir un cuadro de mando que pueda ser consultado por el equipo de Marketing usando tecnologías como Plotly Dash, Quicksight o PowerBI, con la posibilidad de exportar los datos en csv o Excel.

Bibliografía

Data Lakehouse Platform by Databricks

What is the Databricks File System (DBFS)? | Databricks on AWS

Row-level security - Amazon Redshift

Column-level security for dedicated SQL pool - Azure Synapse Analytics

What is a Medallion Architecture?

What is a Data Vault?

Developer Center / APIs | Open Library

Anexo I. Funciones de exportación

I.1 Exportación de los datos desde Databricks (Delta Table) a .parquet

Como se ha mencionado en el capítulo 3.4, en la capa Oro se ha añadido una columna de control basada en año y fecha de creación de la entrada para utilizarla en la partición durante la exportación.

Las funciones de exportación ayudan a exportar cualquiera de las tablas Gold de manera sencilla.

I.2 Widgets desplegables

Para facilitar la interacción con el notebook sin necesidad de modificar el código, se puede hacer uso de los widgets. En este caso, se puede utilizar para seleccionar la tabla a exportar. Se han añadido solamente las tablas de Gold.

```
dbutils.widgets.dropdown("table", "gold_openlibrary.ol_books",
             ['gold_openlibrary.ol_books',
              'gold_openlibrary.ol_reviews',
              'gold_openlibrary.works_with_reviews_summary',
              'gold_openlibrary.ol_works_summary'])
```

Figura Anexo.I.1. Widgets de selección en Databricks

I.3 Funciones de exportación de ficheros particionados

El proceso que se ha implementado para exportar las tablas a S3 en ficheros parquet particionados según la columna de control año-mes es el siguiente:

```
def load_files_to_s3(tbl: str, where_clause: str):

    try:
        file_name = tbl.replace(".","_")+".csv"
        s3_filepath = file_name.split(".")[0]
        query = f"SELECT * FROM {tbl} WHERE ctl_year_month = {where_clause}"
        df = spark.sql(query)
        pd_df = df.toPandas()
        print("LOADING FILES TO S3...")
        pq.write_to_dataset(
            Table.from_pandas(pd_df),
            s3_filepath,
            filesystem=fs,
            use_dictionary=True,
            partition_cols=['ctl_year_month'],
            compression="snappy",
            version="2.4",
)
    except ClientError as e:
        print(e)
    finally:
        print("ALL FILES UPLOADED TO S3")
#load selected table to S3
query_part = f"SELECT DISTINCT ctl_year_month FROM {tbl}"
partitions = list(spark.sql(query_part).toPandas())
for where_clause in partitions:
    load_files_to_s3(tbl,where_clause)
```

Figura Anexo.I.2. Función de exportación de tabla Delta en archivos parquet.

Se ha hecho de esta forma utilizando las librerías boto3, s3fs y pyarrow como parte del caso práctico y como alternativa a df.write.parquet, ya que esta función nos serviría también para exportar archivos desde un dataframe de Pandas usando un Jupyter notebook en local con Python. Además, garantiza que el notebook sea compatible tanto si la cuenta de Databricks está asociada a una cuenta de Azure como una de AWS. De esta forma, no haría falta montar el bucket como almacenamiento.

I.4 Enlaces a GitHub

Parte 1: Exportación en Jupyter notebook .ipynb del notebook de Databricks .DBC

https://github.com/fernandezj-cjro/open-library/blob/main/Cidaen-OpenLibrary%20(AWS).ipynb

Parte 2: Jupyter notebook .ipynb

https://github.com/fernandezj-cjro/open-library/blob/main/tfm_openlibrary_part2.ipynb