UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Análisis de herramientas de ETL y reporting en entornos Big Data

AUTORES: Jerónimo Marchetti

DIRECTOR/A: Waldo Hasperué

CODIRECTOR/A:

ASESOR/A PROFESIONAL:

CARRERA: Licenciatura en Sistemas

Resumen

Las organizaciones enfrentan el desafío de integrar datos heterogéneos de diversas fuentes. Para abordarlo, se pueden utilizar diversas herramientas bajo el paradigma ETL. En un contexto de Big Data, la gestión efectiva de datos se vuelve crucial para respaldar la toma de decisiones estratégicas. El proyecto se centra en estudiar el modelo ETL, empleando múltiples herramientas y comparando enfoques para resolver problemas utilizando diversas tecnologías.

Palabras Clave

ETL, Big Data, Reporting, Data Warehouse, Apache NiFi, Apache Spark, PowerBI, Python, Notebooks, SQL, NoSQL, RDD, Business Intelligence, Organizaciones, Cloud.

Trabajos Realizados

Se llevó a cabo una comparación entre herramientas de tipo Drag and Drop y aquellas basadas en código, focalizándonos específicamente en las fases clave del proceso ETL. Se evaluaron las capacidades de ambas aproximaciones para determinar sus ventajas y limitaciones en diferentes contextos, especialmente el de Big Data. Luego se ejecutaron pruebas y análisis comparativos sobre los informes generados por estas herramientas, analizando además costos computacionales, económicos y de recursos humanos.

Conclusiones

En las fases iniciales del desarrollo, la agilidad proporcionada por herramientas gráficas es beneficiosa durante la prototipación; sin embargo, a medida que el proyecto progresa, la eficiencia y control inherentes al código se vuelven ventajosos. La dicotomía entre herramientas de código y drag and drop no es estricta, sino una oportunidad para combinar lo mejor de ambos enfoques según las necesidades del proceso. En el ámbito del Big Data, la elección entre SQL y NoSQL refleja la dicotomía entre herramientas visuales y desarrollo basado en código, con matices específicos que impactan la gestión de grandes volúmenes de datos.

Trabajos Futuros

El principal trabajo a futuro es implementar un sistema de ETL real time para evaluar las ventajas y desventajas de las herramientas en este contexto.
Por otro lado, implementar un ejemplo del modelo de ETL explicado en la sección de cloud perteneciente al paquete 'advanced analytics on big data' esquematizado en la sección de costos.

Fecha de la presentación: 12 2023

Análisis de herramientas de ETL y reporting en entornos Big Data

Jerónimo Marchetti
Director: Waldo Hasperué

23 de agosto de 2023

Resumen

Un desafío intrínseco que atraviesan las organizaciones contemporáneas radica en la cuestión de amalgamar datos derivados de diversas fuentes, caracterizados por su heterogeneidad en términos de origen y formato, para posteriormente integrarlos en uno o múltiples almacenes de datos y, consiguientemente, dirigirlos hacia diversos puntos de destino. Un componente adicional de esta problemática conlleva la necesidad recurrente de depurar y uniformizar los datos, dada su propensión a divergir en función de su procedencia.

Para abordar este escenario complejo, se han desarrollado diversas herramientas que, en conjunto, ofrecen soluciones a estos desafíos, adoptando el enfoque del paradigma ETL (Extracción, Transformación y Carga, por sus siglas en inglés). Estas herramientas permiten extraer datos de múltiples fuentes, transformarlos para que adopten utilidad, y cargarlos de manera eficiente en almacenes de datos centralizados. Esta integración y procesamiento de datos se convierte en un componente crítico para la toma de decisiones informadas dentro de las organizaciones.

A medida que el fenómeno del Big Data continúa creciendo, la gestión efectiva de los datos se vuelve aún más crucial. Esto se debe a que la información de alta velocidad y volumen generada por diversas fuentes debe ser administrada de manera eficiente en aras de respaldar el proceso de Inteligencia Empresarial (Business Intelligence). La Inteligencia Empresarial implica la recopilación, análisis y presentación de datos para obtener información valiosa que respalde la toma de decisiones estratégicas en las empresas.

En resumen, la gestión de datos se ha convertido en un factor clave para el éxito de las organizaciones en un entorno empresarial cada vez más complejo y orientado a los datos. La capacidad de reunir, transformar y utilizar eficazmente la información diversa es fundamental para impulsar la toma de decisiones basadas en datos y, en última instancia, la sustentabilidad y viabilidad organizacional.

La sección de desarrollo de este proyecto va a seguir el modelo ETL en secuencia. Se emplearán múltiples sistemas simultáneamente para resolver un problema, utilizando diferentes tecnologías, y luego comparar los enfoques usados.

El problema de ejemplo estudiado en esta tesina, consiste en observar la variación de precios en diversas categorías de productos desde mayo de 2023 hasta hoy. Para lograrlo, vamos a obtener datos de precios desde dos fuentes clave. Una es la API del Banco Central de la República Argentina (BCRA), la otra implica extraer datos de diferentes categorías de ítems en el portal de Mercado Libre.

La forma en que tratemos con los datos obtenidos desde estas fuentes dependerá del enfoque que elijamos, ya sea usando una herramienta drag n’

drop o construyendo un flujo sólo con módulos de programación. Durante la transformación, prepararemos los datos realizando las transformaciones necesarias para adaptarlos al destino final.

Posteriormente, evaluaremos la efectividad de la carga de datos en dos arquitecturas diferentes. Por un lado, los almacenaremos en una base de datos SQL y por otro, los guardaremos en una base de datos NoSQL.

Después, emplearemos estos datos para generar informes y reportes. Usaremos herramientas convencionales de arrastrar y soltar como Microsoft PowerBI, además de crear reportes mediante notebooks en Python. Estos informes nos servirán para analizar y presentar los resultados obtenidos al analizar la variación de precios en las categorías de productos estudiadas.

Agradecimientos

Quiero expresar mi agradecimiento a la Universidad Pública, una institución que ha sido la piedra angular de mi formación académica y el escenario donde he llevado a cabo tanto mi carrera universitaria como la elaboración de esta tesina de grado. A su vez quiero reconocer el papel fundamental de los profesores que, con su dedicación y conocimiento, han guiado y moldeado mi desarrollo académico, además de haber sido una parte fundamental para establecer los cimientos y principios sobre los cuales estoy construyendo mi trayectoria profesional.

A su vez, deseo extender mi gratitud a mis leales compañeros de vida, mis adoradas mascotas Teo, Justin Bieber y Budín. Su afecto incondicional y su presencia tranquilizadora han iluminado las arduas jornadas de estudio y clases virtuales durante la pandemia. En este punto, también quiero expresar mi profundo agradecimiento a mis padres, cuyo apoyo constante y aliento han sido fundamentales en mi trayectoria.

A todos aquellos que, de una u otra manera, han contribuido a que en este momento se culmine mi proceso, les agradezco sinceramente por formar parte de este capítulo tan significativo de mi vida académica y profesional. Cada uno de ustedes ha dejado una marca imborrable en este recorrido, y este logro no habría sido posible sin la valiosa colaboración y apoyo de quienes me rodean. Estoy profundamente agradecido por la comunidad académica y el entorno que me ha permitido crecer y aprender de manera integral.

Índice

1. Introducción

1.1 Objetivo del trabajo

El objetivo de este trabajo es llevar a cabo un análisis exhaustivo y una evaluación crítica de la pertinencia y el alcance de la adopción de tecnologías modernas en los procesos de Extracción, Transformación y Carga (ETL) dentro de las organizaciones. El propósito principal es proporcionar recomendaciones sólidas y fundamentadas en la evidencia, basadas tanto en implementaciones reales de organizaciones como en implementaciones modelo, para lograr mejoras sustanciales en la eficiencia operativa, la calidad de los datos y la capacidad de toma de decisiones en el contexto de la gestión de datos organizacionales.

Para alcanzar este objetivo se examinará cómo se puede abordar la incorporación de tecnologías ETL en procesos de ejecución periódica y por lotes (procesos batch). Esto incluirá un análisis detallado de los desafíos y beneficios experimentados, así como la medición del impacto en la eficiencia y precisión de los procesos de gestión de datos.

Además, se llevará a cabo la creación de implementaciones modelo, simulando entornos empresariales específicos, con el objetivo de explorar cómo las tecnologías ETL modernas podrían aplicarse teóricamente y cómo podrían contribuir a la optimización de los procesos de datos en dichos contextos simulados. Estos escenarios ficticios permitirán una evaluación más precisa de las posibles ventajas y desafíos asociados con la adopción de tecnologías ETL avanzadas.

El análisis y la evaluación se llevarán a cabo considerando múltiples dimensiones, como la eficiencia en la extracción y transformación de datos, la calidad y consistencia de los datos resultantes, la escalabilidad de las soluciones propuestas y, lo que es más importante, cómo estas mejoras pueden respaldar y potenciar la toma de decisiones en las organizaciones.

Al finalizar este trabajo, se espera proporcionar una guía sólida y práctica

que permita tomar decisiones informadas sobre la adopción de tecnologías ETL, identificando cómo se pueden lograr mejoras significativas en diferentes tipos de procesos relacionados con la gestión de datos para así, en última instancia, mejorar la capacidad para tomar decisiones estratégicas fundamentadas en datos sólidos y confiables.

1.2 Motivación

En el corazón de toda toma de decisiones informadas se encuentran los datos precisos y relevantes. Las organizaciones modernas se esfuerzan por aprovechar al máximo los recursos disponibles, y los procesos de extracción, transformación y carga son fundamentales para garantizar que los datos sean limpios, coherentes y estén listos para el análisis. En este contexto, surge un dilema crucial: ¿Cuál es la mejor manera de implementar y gestionar estos procesos?

La motivación detrás de esta Tesina radica en la necesidad de explorar y comprender a fondo las ventajas y desafíos de dos enfoques aparentemente divergentes para llevar a cabo procesos de ETL. Por un lado, están las herramientas basadas en el código, que ofrecen un alto nivel de control y personalización, pero a menudo requieren un conocimiento técnico profundo. Por otro lado, tenemos las herramientas de interfaz gráfica tipo "drag and drop", que buscan democratizar el proceso al hacerlo más accesible, aunque a veces a expensas de cierto grado de flexibilidad.

Este contraste entre herramientas de distinta especie plantea preguntas apasionantes sobre la eficiencia, la velocidad de implementación, la escalabilidad y la calidad de los resultados finales. Al abordar estas cuestiones, no solo se contribuye al conocimiento en el campo de la gestión de datos, sino que también se brinda a las organizaciones una base sólida para tomar decisiones informadas sobre la elección de herramientas para sus procesos de ETL. En última instancia, esta investigación tiene el potencial de optimizar la capacidad de las organizaciones para transformar datos en conocimientos, lo que les permitirá mantenerse ágiles y competitivas en un entorno empresarial en constante evolución.

La motivación subyacente a este trabajo de investigación aplicada se origina en el reconocimiento de que los datos precisos y relevantes son el pilar

fundamental de cualquier proceso de toma de decisiones informadas en el entorno empresarial actual. Las organizaciones modernas se esfuerzan por optimizar al máximo la utilización de sus recursos, y es en este contexto donde los procesos de ETL desempeñan un papel crítico al asegurar que los datos sean limpios, coherentes y estén preparados para el análisis.

Sin embargo, este contexto también plantea un dilema esencial: ¿Cuál es la estrategia más adecuada para implementar y gestionar estos procesos de ETL de manera eficaz y eficiente? Es aquí donde radica la esencia de la motivación detrás de esta tesina.

En el ámbito de la gestión de datos, se contrastan dos enfoques aparentemente opuestos. Por un lado, existen las herramientas basadas en código, que ofrecen un alto grado de control y personalización, permitiendo a las organizaciones adaptar los procesos de ETL a sus necesidades específicas. Sin embargo, el uso de estas herramientas a menudo requiere un profundo conocimiento técnico, lo que limita su accesibilidad a un grupo más reducido de profesionales.

Por otro lado, están las herramientas de interfaz gráfica tipo "drag and drop", diseñadas para democratizar el proceso de ETL, haciendo que sea más accesible incluso para aquellos sin experiencia técnica profunda. Sin embargo, esta accesibilidad puede venir acompañada de limitaciones en cuanto a flexibilidad y personalización.

Este contraste plantea cuestiones apasionantes relacionadas con la eficiencia, la rapidez de implementación, la escalabilidad y la calidad de los resultados finales en la gestión de datos. Al abordar y explorar estas cuestiones, esta tesina no solo contribuye al avance del conocimiento en el campo de la gestión de datos, sino que también proporciona a las organizaciones una base sólida para tomar decisiones informadas sobre la elección de herramientas y enfoques para sus procesos de ETL.

En última instancia, la investigación realizada aquí tiene el potencial de optimizar significativamente la capacidad de las organizaciones para transformar datos en conocimiento, lo que, a su vez, les permitirá mantenerse ágiles y competitivas en un entorno empresarial en constante evolución. En un mundo donde la toma de decisiones basadas en datos se ha convertido en un activo estratégico crítico, esta tesis busca arrojar luz sobre cómo las organizaciones pueden abordar de manera óptima esta faceta esencial de la gestión de datos.

1.3 Aportes de la Tesina

Los aportes de esta Tesina son:

- Proporcionar una visión completa de conveniencia sobre las opciones al momento de construir un proceso de ETL, tomando en cuenta los factores internos de las organizaciones a la hora de elegir su stack tecnológico para construir o actualizar sus procesos.
- Analizar desafíos comunes que las organizaciones se encuentran al construir o actualizar procesos de ETL, para poder proporcionar soluciones simples para la integración de diversas tecnologías y productos entre sí.
- Sentar una propuesta de realización para la implementación de una tecnología ETL en organizaciones ficticias que simulen ejemplos del mundo real.
- Evaluación de las propuestas tecnológicas actuales, comentando sus ventajas, desventajas, casos de uso, escalabilidad y costo.

1.4 Estructura de la Tesina

Esta Tesina se estructura de la siguiente forma. En el capítulo 2 se describe el Marco Teórico de la tesis, definiendo los campos técnicos y teóricos en donde posará el desarrollo de los sistemas desarrollados, los cuales se detallan en el capítulo 4. Anteriormente se dará una introducción a las herramientas utilizadas en el capítulo 3.

Durante el capítulo 5 se realizarán las comparativas entre los sistemas, dependiendo sus costos tanto en recursos de computación , humanos y económicos. Por último, en el capítulo 6 se encuentran las conclusiones finales del trabajo.

En la sección 8 se encuentra la bibliografía por la cual basamos este trabajo y finalmente se encuentra un apéndice que detalla cómo configurar los diversos entornos de desarrollo para aquel que desee probar los sistemas construidos.

2. Marco Teórico

2.2 ETL

El proceso y paradigma de ETL ganó notoriedad en la década de 1970 cuando las empresas comenzaron a utilizar múltiples bases de datos para almacenar una amplia gama de información empresarial. La creciente necesidad de unificar datos dispersos en estas bases de datos condujo al rápido ascenso del ETL como el método convencional para extraer datos de diversas fuentes y transformarlos antes de cargarlos en un destino específico.

En las postrimerías de la década de 1980 y los primeros años de la década de 1990, aparecieron los almacenes de datos, que ofrecían un acceso integrado a datos procedentes de una variedad de sistemas, como mainframes, minicomputadoras, computadoras personales y hojas de cálculo. Sin embargo, los diferentes departamentos solían optar por diferentes herramientas ETL para utilizar con almacenes de datos distintos, lo que resultaba en una falta de integración. Además, las fusiones y adquisiciones a menudo dejaban a las organizaciones con múltiples soluciones ETL que no estaban interconectadas.

A lo largo del tiempo, la diversidad de formatos, fuentes y sistemas de datos ha experimentado un crecimiento significativo. El proceso de Extracción, Transformación y Carga (ETL) ahora se considera uno de varios métodos que las organizaciones emplean para reunir, importar y procesar datos.

Este proceso se desglosa en tres etapas sucesivas [1]: Extracción (E) , Transformación (T) y Carga (L, por ‘load’). En conjunto desempeñan un papel crítico en la gestión de datos empresariales, ya que asegura que los datos sean adquiridos, preparados y entregados de manera eficiente y confiable a los sistemas y aplicaciones que los utilizan para la toma de decisiones informadas y el análisis estratégico.

Cabe aclarar que si bien E, T y L representan etapas separadas, son interdependientes en la práctica. La extracción no solo implica tomar datos de una fuente, sino también realizar transformaciones ligeras, como la conversión de formatos o la limpieza de datos. La transformación a menudo implica la extracción de datos de múltiples fuentes antes de aplicar transformaciones más complejas. La carga a su vez puede requerir transformaciones finales antes de insertar datos en el destino, como se abordará posteriormente.

Así, vemos que ETL no es un proceso del todo secuencial, sino que en cada paso podemos volver a iterar sobre los anteriores.

2.2.1 Extracción

En esta fase inicial, se adquieren los datos desde sus fuentes primarias Algunos ejemplos de estas son: bases de datos, APIs, aplicaciones web y

programas de planificación.

La fase de extracción comprende el proceso de recolección de información a través de un framework específico, seguido de su concentración en una plataforma, dirigiéndola con el fin de facilitar su subsiguiente tratamiento y análisis. La esencia de este procedimiento reside en la recuperación exhaustiva de datos desde las distintas fuentes, de la forma más eficiente posible para minimizar la utilización de recursos: debe ser meticulosamente diseñada para evitar cualquier impacto negativo en el rendimiento del framework utilizado, asegurando que no se vea afectado en términos de velocidad de procesamiento (sobre todo en procesos real time) o la ocurrencia de bloqueos inoportunos, garantizando al mismo tiempo la integridad y disponibilidad del sistema.

Además, esta etapa puede clasificarse según la frecuencia y forma de llegada de datos, pudiendo ser procesamiento por lotes (batch processing) o stream processing [2] (procesamiento real time):

Extracción por lotes: las unidades de información vienen empaquetadas en grupos y se extraen a partir de la ocurrencia de un evento en particular, como por ejemplo, una actualización en una base de datos, o la señal de un temporizador.

Este método es eficiente para tareas que no requieren respuestas instantáneas y donde la latencia no es crítica. El procesamiento por lotes se utiliza comúnmente en tareas como el procesamiento de informes periódicos , el análisis de grandes conjuntos de datos históricos y la generación de informes programados, como lo es el objeto de desarrollo de este trabajo.

Extracción real time o stream processing: los datos una vez generados en sus fuentes (por ejemplo, la publicación de tweets, o la creación de un ticket de soporte), van a parar a una cola de mensajes, como pueden ser las tecnologías Kafka o IBM MQ [3]. De ahí, son consumidos por el proceso de ETL constantemente. Esto genera que, a diferencia del procesamiento por lotes en donde se generan actividades cada cierto tiempo determinado, el proceso esté continuamente recibiendo solicitudes de procesamiento. Ejemplos de aplicaciones en tiempo real incluyen sistemas de monitoreo en línea, procesamiento de transacciones financieras, sistemas de recomendación en el momento y seguimiento de eventos en redes sociales.

El procesamiento en tiempo real es más exigente en términos de recursos, ya que requiere una infraestructura que pueda manejar datos de manera instantánea y continua. Además, los sistemas en tiempo real a menudo tienen requisitos estrictos de latencia. Por otro lado, el procesamiento por lotes puede ser menos demandante en términos de recursos, ya que permite la consolidación

de datos antes de su procesamiento.

2.2.2 Transformación

La información extraída de las fuentes primarias se presenta de manera cruda y no es utilizable en sus estructuras únicas (es decir, la diferencia de estructuración de la información que pueden llegar a tener las diferentes fuentes). Por lo tanto, es necesario someterla a un proceso de purificación, planificación y modificación. En este contexto, el ciclo ETL agrega valor y transforma la información para que sea comprensible y precisa, permitiendo así la generación de informes de Business Intelligence (BI).

Durante este proceso, se aplican diversas funciones para la extracción de datos. Aquellos datos que no requieren ninguna modificación se denominan datos de transferencia directa, también conocidos como datos enriquecidos.

El proceso de transformación implica la limpieza, filtrado, validación y aplicación de reglas de negocio a los datos extraídos. El objetivo principal de esta etapa es cargar los datos extraídos en la base de datos de destino con un formato limpio y general, ya que se extraen datos de diversas fuentes, cada una con su propio formato.

El proceso de transformación se rige por una serie de reglas diseñadas para convertir los datos desde la fuente hasta el destino. Este proceso también implica la combinación de datos de múltiples fuentes, la creación de totales, la ordenación, la inferencia de nuevos valores derivados y la aplicación de reglas avanzadas de validación. Este enfoque garantiza que la información final en la base de datos de destino sea coherente, precisa y cumpla con los requisitos específicos del análisis de datos y manejo de la información dependiendo del rubro.

Por ejemplo, si hablamos del área de los procesos ETL en la salud, al tener información sensible sobre los pacientes, es necesario por ley (HIPAA, en el caso de los Estados Unidos) un proceso de desidentificación, en donde los datos de los pacientes son codificados para resguardar su privacidad durante todo el proceso, rompiendo el vínculo entre la persona a la cual representan los datos, y estos mismos datos, por ejemplo, para el uso de tecnologías con objetivo de prevención de enfermedades. Esto es parte de la transformación.

2.2.3 Carga

La última fase, correspondiente a la carga, transfiere los datos hacia los sistemas de destino preestablecidos. Se debe considerar la estructura de los

mismos. Estos sistemas pueden variar de acuerdo a la estructura de la organización, incluyendo bases de datos operativas, data warehouses, aplicaciones empresariales, o cualquier otro tipo de sistema que requiera acceso a los datos. Esto implica la creación o adaptación de tablas, esquemas, o estructuras de almacenamiento que sean coherentes con los requisitos y objetivos del sistema. Existen diversos tipos de carga en el paradigma:

Carga inicial: este procedimiento es esencial para establecer la infraestructura inicial de un Data Warehouse. Durante la Carga Inicial, todas las tablas del almacén de datos se llenan por primera vez con la información necesaria para respaldar las operaciones de análisis y generación de informes. Este paso proporciona la base sobre la cual se realizarán análisis posteriores y se generarán informes significativos. La selección y carga de datos durante esta fase inicial son cruciales para asegurar que la información base sea completa y precisa, sentando así las bases para un análisis robusto a posteriori. Este tipo de carga se suele usar en migraciones de información de un sistema a otro.

Carga incremental: este tipo de carga es el que usaremos en el desarrollo del sistema durante este trabajo. Mantiene actualizado el Data Warehouse con cambios periódicos en los datos, aplicando inserciones periódicas a las tablas del almacén de datos, garantizando así que la información sea siempre relevante y actualizada. Este enfoque es especialmente valioso en entornos de reportes periódicos, donde la información está en constante evolución. Se optimiza el proceso al actualizar solo la información que ha experimentado cambios, minimizando el tiempo y los recursos requeridos.

Recarga completa: la recarga completa implica la eliminación (TRUNCATE o DELETE en el caso de SQL) y recarga de una o más tablas del Data Warehouse. Este enfoque puede ser necesario cuando se requiere una actualización completa de los datos, cuando los actualmente cargados resultan obsoletos. Esta técnica es evidentemente más intensiva en cuanto la utilización de los recursos, ya que implica realizar periódicamente la técnica de ‘carga inicial’ con el agregado de la anterior eliminación de los registros.

A su vez, incluye la validación y control de calidad de los datos una vez estos llegan a su destino, verificando que se carguen correctamente, cumpliendo con las reglas de negocio y calidades requeridas según los procesos de DQ (Data Quality):

Es imperativo garantizar la integridad de los datos mediante la verificación de que la información en los campos clave no presente ausencias ni sea inválido. Este procedimiento es esencial para preservar la coherencia y confiabilidad de la base de datos, asegurando que los elementos identificativos

mantengan su integridad y que las relaciones entre diferentes entidades sean precisas.

A su vez, la evaluación de la validez de los datos se realiza a través de pruebas que dependen de las tablas objetivo. Este enfoque implica la comprobación de la conformidad de los datos en las tablas relevantes con criterios predefinidos. Por ejemplo, al analizar datos de rendimiento financiero, se verifica que los datos pertinentes en las tablas correspondientes se ajusten a estándares específicos, contribuyendo así a la precisión y confiabilidad de los informes resultantes. Por lo general existen documentos que detallan los porcentajes de probabilidad o la frecuencia que tiene cada valor en una columna específica. Por ejemplo, si vemos que en una columna los valores son iguales para todas las entradas, pero la documentación indica que los valores de la misma deberían ser variados, nos da pauta para revisar el proceso de transformación realizado a estos registros.

Además, se debe gestionar de manera precisa la evolución temporal de la información. La aplicación de esta verificación contribuye a la coherencia general de la base de datos y, por ende, a la calidad de los informes derivados.

2.3 Big Data

La integración de datos pertenecientes a entornos Big Data es crucial para numerosos ámbitos de aplicación, tanto en la ciencia como en empresas, ya que tiene el fin de posibilitar el análisis de grandes volúmenes de datos [16]. El concepto de Big Data abarca una serie de características fundamentales que definen conjuntos de datos únicos en su clase. Estas cualidades incluyen un gran volumen de datos, una amplia variedad de formatos y fuentes de origen, una velocidad de generación a menudo vertiginosa y la veracidad, que se refiere a la confiabilidad y precisión de los datos. Además, algunos expertos en el campo, como Joshua Chibuike Nwokeji y Richard Matovu, agregan una quinta dimensión, el valor [4]. Se ampliarán estas características [5]:

Volumen: La medición de datos en términos de volumen se sitúa comúnmente en terabytes pero se ha llegado a peta bytes, y se está mostrando una clara tendencia hacia exabytes. Este aumento exponencial en la cantidad de datos destaca la necesidad de infraestructuras de almacenamiento y procesamiento capaces de manejar estas magnitudes. Este fenómeno subraya la importancia de desarrollar y adoptar tecnologías que permitan gestionar eficientemente conjuntos de datos masivos. Es evidente que son volúmenes

complejos de analizar utilizando la infraestructura computacional tradicional y actual.

Velocidad: La producción de datos ocurre a tasas muy elevadas, y debido a este volumen significativo, algunas aplicaciones requieren procesamiento de datos en tiempo real para determinar la forma de procesamiento de las unidades de información. La velocidad de generación de datos plantea desafíos en términos de capacidad de procesamiento y almacenamiento, haciendo necesario implementar sistemas que puedan manejar la rapidez con la que se generan y modifican los datos.

Variedad: La naturaleza de los datos es heterogénea, pudiendo ser altamente estructurada, semiestructurada o completamente no estructurada. Esta diversidad en la forma y estructura de los datos implica la necesidad de enfoques flexibles y adaptables para su almacenamiento y análisis. La variedad en los datos destaca la importancia de las tecnologías y estrategias que pueden manejar la diversidad de formatos y tipos de datos de manera efectiva.

Veracidad: Debido a los procesos intermediarios, la diversidad entre las fuentes de datos y la evolución de los datos plantean preocupaciones sobre seguridad, privacidad, confianza y responsabilidad. Esto genera la necesidad de verificar el origen seguro de los datos (data provenance) para abordar estas preocupaciones. La veracidad de los datos se convierte en un aspecto crítico para garantizar la integridad y la confiabilidad de la información procesada.

Valor: El análisis de estos datos puede proporcionar percepciones contraintuitivas e inteligencia accionable. El valor inherente en estos conjuntos de datos masivos se encuentra en la capacidad de extraer conocimientos significativos que pueden informar decisiones estratégicas y operativas. Este enfoque en el valor destaca la importancia de aprovechar las capacidades analíticas avanzadas para obtener el máximo beneficio de la ingente cantidad de datos disponibles. Esta dimensión hace referencia al potencial de un conjunto de datos para proporcionar conocimiento valioso que se puede utilizar en la toma de decisiones estratégicas dentro de una organización.

En conjunto, estas cualidades hacen que el concepto de Big Data vaya más allá de simples conjuntos de datos; implica también los métodos y procesos necesarios para gestionar y procesar eficazmente esta información de gran envergadura y diversidad.

El ciclo de vida de los datos en el contexto de Big Data comienza con la recolección de datos de diversas fuentes. Aquí es donde entra en juego el desafío de la integración de datos, que busca unificar esta diversidad en un formato coherente y utilizable. Para abordar este desafío, existen varias

estrategias y métodos. Uno de ellos es el uso de sistemas de consulta mediados [6], que introducen una capa intermedia llamada "mediador" entre el usuario y los datos. Este mediador traduce las consultas del usuario en consultas a múltiples fuentes de datos, permitiendo así presentar resultados coherentes y unificados.

Otro enfoque es el uso de bases de datos federadas [7], que son independientes entre sí pero se presentan al usuario como si fueran parte de un único sistema federal. Este sistema distribuye las consultas entre las bases de datos subyacentes de manera eficiente.

Sin embargo, el paradigma más eficiente y popular para lograr la integración de datos en el contexto de Big Data es el ETL [8].

Esta metodología se ha convertido en una herramienta esencial para lidiar con la complejidad y el volumen de los datos en el mundo del Big Data, permitiendo a las organizaciones aprovechar al máximo su potencial y convertirlos en activos estratégicos para la toma de decisiones informadas. En resumen, el Big Data no se trata solo de conjuntos de datos masivos, sino también de cómo gestionar y procesar eficazmente esta abundancia de información diversa y valiosa.

2.3.1 Data Warehousing

El concepto de data warehousing hace referencia al procedimiento de recopilación y manipulación de datos provenientes de diversas fuentes con el propósito de obtener información valiosa para una empresa. Un Data Warehouse constituye una plataforma destinada a la recopilación y análisis de datos procedentes de múltiples fuentes [9]. Se sitúa en una posición central dentro de un sistema de Business Intelligence. El objetivo principal de los mismos es brindar accesibilidad de la forma más simple posible a los usuarios.

Por lo general, un data warehouse se encuentra separado de la base de datos operacional de la empresa. Esto permite a los usuarios basarse en datos históricos y actuales para tomar decisiones más informadas. Las bases operacionales de una empresa tienen un diseño OLTP (OnLine Transaction Processing): se diseñan para dar soporte al procesamiento diario de datos de las organizaciones, con una esperada gran cantidad de transacciones en cortos períodos de tiempo. Se enfatiza la maximización de la capacidad transaccional y la concurrencia ya que habrá muchos usuarios requiriendo transaccionar al mismo tiempo.

Los Data Warehouse usan OLAP (Online Analytical Processing). Son datos que, aunque no están disponibles en tiempo real, pueden ser analizados de forma rápida y masiva sin interrumpir los procesos del usuario. Poseen tablas de facto y tablas de dimensiones.

Una tabla de facto (o tabla de hecho) representa datos cuantitativos, estos son filtrados y agrupados mediante los datos presentes en las tablas de dimensiones, que representan datos cualitativos, como aspectos de interés mediante los cuales los usuarios deciden filtrar los datos cuantitativos. Estos dos tipos de tablas están relacionadas mediante claves foráneas, y es importante mencionar que están denormalizadas. La denormalización busca mejorar la eficiencia operativa de una base de datos al incorporar información redundante, con el objetivo de agilizar y optimizar las consultas y operaciones. Esta técnica implica la introducción de redundancias en los datos para facilitar el acceso y la manipulación de la información almacenada.

Las características que debe poseer una base de almacenamiento para ser considerada data warehouse son:

- Debe hacer que la información sea fácilmente accesible. El contenido del sistema DW debe ser comprensible. Los datos deben ser intuitivos y evidentes para el usuario en la organización, y no solo para el desarrollador. Las estructuras y etiquetas de los datos deben imitar los procesos y vocabulario del pensamiento de los usuarios empresariales. Estos usuarios desean separar y combinar datos analíticos de diversas maneras. Las herramientas y aplicaciones de inteligencia empresarial que acceden a los datos deben ser simples y fáciles de usar, devolviendo resultados de consultas al usuario con tiempos de espera mínimos.

- El sistema DW debe presentar información de manera consistente. Los datos en el sistema DW deben ser creíbles y cuidadosamente ensamblados de diversas fuentes, limpiados, asegurados en cuanto a calidad y liberados solo cuando estén listos para el consumo del usuario.

- Debe adaptarse al cambio. Las necesidades del usuario, las condiciones organizacionales, los datos y la tecnología están sujetos a cambios. El sistema DW debe diseñarse para manejar este cambio inevitable de modo que no invalide los datos o aplicaciones existentes. Los datos existentes no deben modificarse ni interrumpirse cuando la organización formula nuevas preguntas o se agrega nueva información al almacén.

- El sistema debe procurar la protección de los activos de información. Las joyas de la corona informativa de una organización se almacenan en data warehouses. El almacén probablemente contenga información sensible, potencialmente perjudicial en manos equivocadas. El sistema DW debe controlar eficazmente el acceso a la información confidencial de la organización.

- El almacén de datos debe tener los datos correctos para respaldar la toma

de decisiones. Las salidas más importantes de un sistema DW son las decisiones que se toman basadas en la evidencia analítica presentada; estas decisiones proporcionan el impacto empresarial y el valor atribuible al sistema.

2.4 Reporting y BI

Business Intelligence es un término por primera vez nombrado en el artículo ‘A Business Intelligence System’, escrito por el investigador de IBM Hans Peter Luhn en 1958 [10]. En el mismo, establece que la comunicación eficiente desempeña un papel fundamental en el avance de todas las áreas de la actividad humana. Hans comenta que en esos años, los métodos de comunicación disponibles resultaban insuficientes para satisfacer las necesidades futuras. Esto se debe a que la generación y el uso de información se estaba acelerando constantemente debido al ritmo y el alcance creciente de las actividades humanas.

Simultáneamente, el crecimiento de las organizaciones junto con la especialización y división del trabajo han creado nuevos obstáculos para la circulación de la información. Además, plantea una creciente demanda de tomar decisiones más rápidas en niveles de responsabilidad mucho más bajos de lo que era común décadas anteriores debido a una ‘abrumadora’ cantidad de información que debía gestionarse.

Se plantea la automatización como la solución más eficiente para la recuperación y la difusión de esta información. Si bien se habían logrado avances en la aplicación de máquinas para los procesos de recuperación de información, se debía aún avanzar en la diseminación automática de la misma. Hans propone tres técnicas para lograr esto:

1. Resumir automáticamente documentos.
2. Codificar automáticamente documentos.
3. Crear y actualizar automáticamente perfiles de puntos de acción.

Es así como en 1958, se comenzó a esbozar y describir el funcionamiento actual de la repartición de la información de las organizaciones actuales.

El reporting se refiere al proceso de generar informes o reportes a partir de datos almacenados o disponibles en una organización. Estos informes suelen ser documentos o presentaciones visuales que resumen y presentan información relevante de manera comprensible para apoyar la toma de decisiones. Los informes pueden incluir gráficos, tablas, análisis de tendencias y métricas clave que ayudan al sector estratégico, encargado de la toma de decisiones, a entender la situación actual de la empresa. El reporting es esencial para comunicar datos e insights de manera efectiva a diferentes niveles de una organización.

Por otro lado, Business Intelligence (BI) se refiere a un conjunto de tecnologías, procesos y herramientas que permiten a las organizaciones recopilar, almacenar, analizar y presentar datos de manera que se conviertan en información útil para la toma de decisiones empresariales. La BI incluye no solo la generación de informes (reporting) sino también la creación de paneles de control interactivos, análisis de datos, minería de datos y la visualización de información para proporcionar una visión integral de la empresa y su desempeño.

Supongamos una organización multinacional siderúrgica con una red extensa de tiendas mayoristas que opera en diversos mercados geográficos y verticales de productos. Previamente a la implementación de una infraestructura de BI avanzada, dicha organización se apoyaba en un enfoque de toma de decisiones tradicional, basado en la experiencia y en datos retrospectivos. La recopilación y análisis de datos se llevaba a cabo de manera manual y resultaba limitada en su alcance y capacidad de proporcionar información en tiempo real.

Con el correr de los años y tras la implementación de un sistema de BI moderno, la organización experimentó un cambio sustancial en su proceso de toma de decisiones empresariales. Algunos ejemplos concretos de cómo esta transición afectó positivamente a sus operaciones son:

Optimización de inventario: A través del BI, la organización logró un seguimiento en tiempo real de las tendencias de pedidos en todas sus ubicaciones. Esto permitió una gestión de inventario y procesos de fabricación más precisa y oportuna. Por ejemplo, cuando se detectaba una demanda creciente de ciertos productos en una ubicación específica, se podía actuar proactivamente reabasteciendo el stock, evitando así la pérdida de ventas por agotamiento de inventario.

Segmentación de clientes: Mediante el análisis de datos demográficos corporacionales y patrones de demanda, la organización pudo segmentar a sus clientes de manera más precisa. Esto condujo a la creación de campañas de marketing altamente personalizadas y dirigidas a cada organización cliente en específico.

Predicción de tendencias: Gracias a las capacidades analíticas del BI, la organización pudo identificar tendencias emergentes en el mercado antes de que se hicieran evidentes. Por ejemplo, al identificar un incremento en la demanda de determinados tipos de acero, pudieron anticipar la necesidad y tomar medidas anticipadas para abastecer y promocionar eficazmente estos productos

Análisis de rentabilidad de fábricas: Utilizando el BI, la organización

pudo comparar el rendimiento de sus diversas fábricas y ubicaciones en tiempo real. Esto les permitió tomar decisiones informadas sobre la expansión o reubicación de fábricas en función de su rentabilidad y contribución a los resultados generales.

En resumen, la implementación del Business Intelligence permitió a la organización adoptar un enfoque de toma de decisiones más estratégico y basado en datos en lugar de depender exclusivamente de la intuición o de datos históricos desactualizados [15]. El valor del proceso es determinado por lo bien que puede servir a la organización. En este caso, el enfoque aplicado mejoró significativamente la eficiencia operativa, impulsó el aumento de las ventas y permitió una mejor alineación con las expectativas y necesidades cambiantes de su base de clientes, otorgándole una ventaja competitiva en su industria.

3. Herramientas utilizadas para el desarrollo

3.1 Python y entornos de construcción basada en código para procesos ETL y Big Data

Python es una opción popular para implementar procesos ETL debido a su flexibilidad, rica biblioteca de herramientas y capacidad para conectarse a una variedad de fuentes de datos.

Sus ventajas clave abarcan su facilidad de uso, debido a su sintaxis simple. Esto facilita la escritura y mantenimiento del código. Incluye bibliotecas populares las cuales se destinan para el tratamiento de datasets. Estas bibliotecas permiten realizar tareas de extracción, transformación y carga de datos de manera eficiente y sencilla.

Python es versátil en términos de conectividad con diferentes fuentes de datos y destinos. Incluye además librerías de visualización de datos para crear reportes, sin necesidad de integrar otro producto.

Si bien es adecuado para proyectos pequeños y medianos, también se puede utilizar en proyectos más grandes mediante el uso de técnicas de optimización y paralelización.

A su vez se integra bien con herramientas de Big Data como Apache Hadoop, Apache Spark y Apache Kafka. Es común utilizar Python para orquestar y controlar flujos de datos en entornos de Big Data.

En este trabajo haremos uso de Apache Spark, mediante pyspark y el uso de notebooks de código.

Una notebook de código es un entorno de desarrollo interactivo que permite escribir y ejecutar código de manera interactiva, integrando el código con elementos de texto enriquecido, gráficos y otros elementos visuales. Las notebooks de código son herramientas populares en ciencia de datos, aprendizaje automático, investigación científica y educación.

Estas incluyen los siguientes elementos:

- Celdas de código: el código se organiza en celdas, que pueden ejecutarse de forma independiente. Esto permite probar y depurar porciones de código de manera incremental.

- Celdas de texto: las notebooks permiten la inclusión de celdas de texto enriquecido, que pueden contener explicaciones, documentación, fórmulas, imágenes y otros elementos visuales.

- Ejecución interactiva: se pueden ejecutar celdas de código de manera interactiva y ver los resultados inmediatamente debajo de la celda. Esto facilita la exploración y comprensión del código.

- Visualización integrada: admiten la visualización de gráficos y otros resultados visuales directamente en el documento, lo que facilita la interpretación de los resultados. En este trabajo aplicamos la funcionalidad de visualización integrada en el reporting basado en código.

Ejemplos comunes de notebooks de código incluyen Jupyter Notebooks, Google Colab, R Markdown, y Microsoft Azure Notebooks. Estas herramientas son ampliamente utilizadas en el ámbito de la ciencia de datos y la investigación, ya que proporcionan un entorno interactivo y colaborativo para trabajar con código y datos. En el desarrollo de este trabajo utilizaremos Jupyter Notebooks, en un servidor local.

A su vez, en este trabajo nuevamente se emplea Python para realizar las tareas de web scraping pertenecientes a la fase de extracción. Definimos al web scraping como un proceso que implica extraer y combinar contenidos de páginas web de manera sistemática, donde un agente de software imita la experiencia humana de interacción con la navegación web [17]. Para realizar este procedimiento en este trabajo se utiliza la librería beautifulSoup para obtener y posteriormente organizar los datos a tratar durante las siguientes tareas del proceso.

Como extra, y no solo incluyendo al lenguaje citado, construir un propio entorno basado en código propio tiene diversas ventajas, las que se suelen

considerar a resaltar son las siguientes [12]:

- Los sistemas programados a base de código tienen a su alcance múltiples herramientas de testeo unitario, mientras que en el enfoque basado en herramientas con GUI no suelen estar. Por ejemplo, la biblioteca PyTest es una herramienta altamente valorada y bien respaldada para realizar pruebas unitarias en programas Python, que podríamos haber implementado en nuestro pipeline.
- Esto abre las puertas a poder automatizar los procesos de prueba, lo que mejora significativamente la eficiencia durante el control de calidad de los sistemas a desarrollar. Mientras que en la mayoría de las herramientas con interfaz gráfica, esto se debe realizar manualmente.
- Por otra parte, las técnicas de programación orientada a objetos ayudan a hacer que todas las transformaciones sean consistentes en cuanto a informes de errores, validación y actualizaciones de metadatos, si es que desarrollamos a parte módulos que confeccionen los mismos.
- Un sistema desarrollado en un lenguaje común y ampliamente conocido, como es en nuestro caso de ejemplo, permite una flexibilidad ilimitada, pudiendo realizar cualquier cosa que se requiera sin estar condicionado por las capacidades del proveedor de la herramienta.
- A su vez, es más probable que los recursos necesarios para construirlo ya estén disponibles en la organización y no necesiten capacitación antes de ponerse a producir el sistema.
- En nuestro ejemplo, el problema a resolver es sencillo y pudo resolverse con un script simple resuelto en una notebook a la cual podemos acceder en un servidor local (apéndice).

En resumen, sumado a las ventajas mencionadas, Python además es una elección popular para ETL debido a su facilidad de uso, su amplio ecosistema de bibliotecas y su versatilidad para conectarse a diversas fuentes y destinos de datos. Su capacidad para manejar tareas complejas de procesamiento y transformación de datos lo convierte en una herramienta valiosa para aquellos que desean implementar flujos de datos eficientes y flexibles.

3.2 Spark

Apache Spark es un motor multi-lenguaje de código abierto para el procesamiento de datos. Puede implementarse en Python mediante la librería pyspark, aunque también puede ser usado en Java, R, Scala y SQL.

Se utiliza especialmente en entornos Big Data ya que procesa de 10 a 100 veces más rápido los datos que las herramientas alternativas [14], ya que escala distribuyendo el procesamiento entre grupos de computadoras, además de que no lee ni escribe en disco los datos que va procesando, sino que los mantiene en

memoria para realizar los pasos posteriores.

Spark opera en un modelo de arquitectura maestro/esclavo, donde el nodo maestro coordina las tareas y los nodos esclavos la ejecutan de manera distribuida. Los datos se dividen en particiones y se distribuyen entre los nodos del clúster para el procesamiento paralelo. Estas particiones son llamadas RDD: Resilient Distributed Dataset o conjuntos de datos resilientes y distribuidos. Resilientes porque son tolerantes a errores, al estar presentes en varios nodos, si uno falla, la información se puede recuperar de los otros. Y distribuido porque cada RDD representa una colección distribuida de objetos. Pueden ser creados por datos de almacenamiento (los datos con los que se está trabajando) o por transformaciones. Una transformación es una operación que genera otro RDD a partir de uno existente, por ejemplo, un filtro de datos (filter). Las transformaciones son uno de los dos tipos de operación que proporciona Spark. El otro tipo de operación es la acción, que son agregaciones de datos que devuelven un resultado al usuario o los llevan a un medio externo, como por ejemplo un save o un count.

En el trabajo se utilizó pyspark que es una API de Python para utilizar Spark. Esto ya que posee manejo de datos estructurados mediante Spark SQL para transformar los datos de forma que queden aptos para la inserción en nuestra base Postgres.

3.3 Apache Nifi

Apache NiFi, en este trabajo referido simplemente como NiFi, es una herramienta de integración de datos de código abierto que proporciona una interfaz intuitiva para diseñar flujos de datos y automatizar el movimiento de datos entre varios sistemas. Fue desarrollado originalmente por la Agencia de Seguridad Nacional de los Estados Unidos (NSA) y posteriormente se convirtió en código abierto como parte de la Fundación Apache.

Las características clave y los componentes de Apache NiFi incluyen la programación basada en flujos, donde los flujos de datos se representan como procesadores interconectados. Un procesador es un módulo Java que realiza un determinado procesamiento de la data que se le ingresa. Es representado como una caja con nombre (definido por el usuario), tipo de procesador (determina que procesamiento realiza), y una configuración (representando parámetros de entrada del módulo Java, que condicionan cómo se ejecuta el mismo):

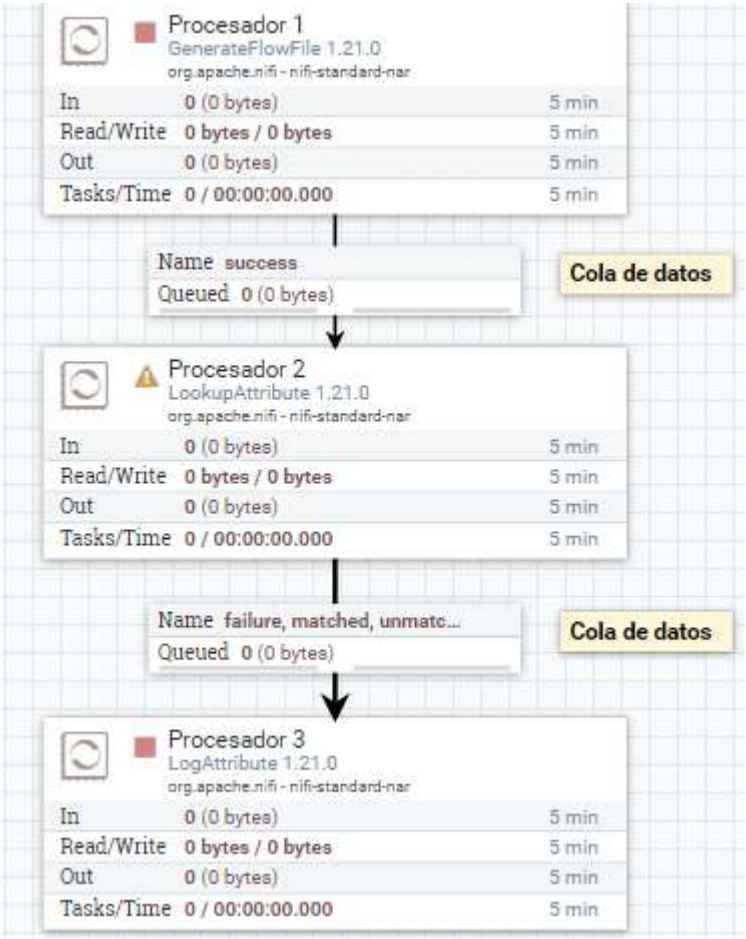

Figura 1. se muestra un ejemplo de procesadores encadenado formando un proceso

La unidad de información es el ‘flowfile’, que representa el payload del proceso y va siendo procesado de procesador a procesador, pasando por las colas intermedias entre estos. Cada procesador realiza una tarea específica, y pueden tener entradas y salidas de flowfiles. Encapsulan scripts de Java, abstraídos a una forma de caja o flechas de dirección. Estos están predefinidos, aunque tenemos la posibilidad de crear nuevos dependiendo de nuestras necesidades de uso.

Se utiliza comúnmente para la ingestión de datos desde diversas fuentes, incluyendo bases de datos, archivos, sensores, dispositivos IoT y APIs externas. Admite una amplia gama de formatos de datos, incluyendo JSON, XML, Avro, Parquet y más. Permite además la transformación de la data utilizando procesadores que pueden realizar tareas como enriquecimiento de datos, conversión, filtrado y validación. A su vez se pueden rutear los mismos, permitiendo crear flujos condicionales.

Sumado a esto, ofrece capacidades de monitoreo y gestión en tiempo real a través de su interfaz de usuario basada en web. NiFi mantiene un registro detallado del recorrido de los datos a través del sistema, lo que le permite rastrear la genealogía de los datos, auditar los cambios en los datos e investigar problemas. Esto constituye el sistema integrado de metadata.

Forma parte del amplio ecosistema de Apache, contando con una comunidad amplia y activa. Puede integrarse fácilmente con herramientas de la misma fundación para construir tuberías de datos completas.

En síntesis, NiFi se utiliza comúnmente en diversos escenarios de integración de datos y gestión de flujos de datos, incluyendo la ingestión de datos para data lakes, el procesamiento de datos en tiempo real, la migración de datos y el procesamiento de datos de IoT. Proporciona una interfaz fácil de usar para diseñar flujos de datos complejos sin necesidad de tener un amplio conocimiento de programación.

Un caso de uso real de un proceso de ETL desarrollado con Nifi es el siguiente:

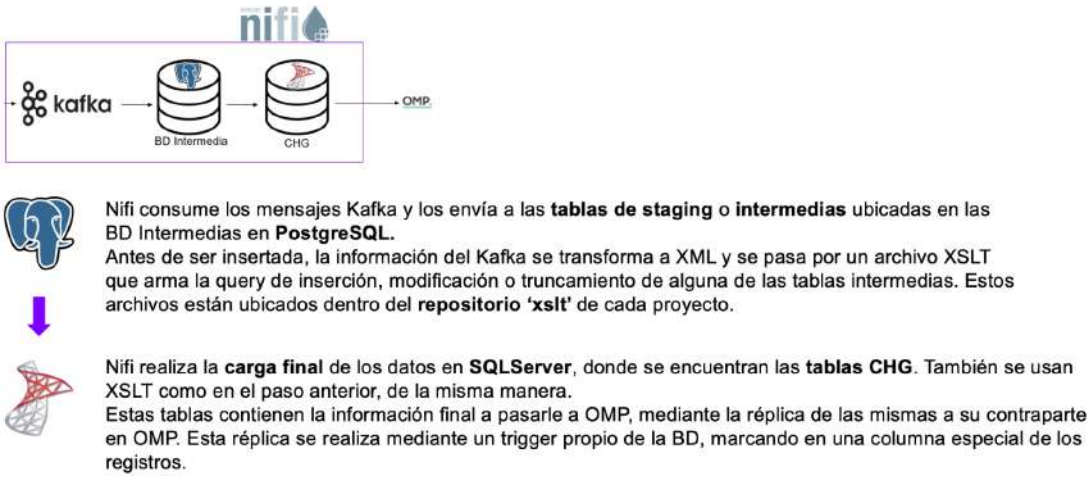

Figura 2. Se muestra la documentación del proceso descripto

Una empresa nacional metalúrgica dedicada a la fabricación de tubos de acero sin costura para la construcción de oleoductos y gasoductos posee fábricas en donde se trata el material en crudo para lograr estos productos. Necesitan construir un sistema de ETL para poder controlar las demandas de sus clientes, las producciones, el estado de las piezas y el stock disponible.

Para realizar esto, integraron a sus sistemas el uso de apache Nifi para la transmisión y recepción de mensajes en tiempo real.

Mediante la herramienta se realizaron interfaces near real time (NRT) para conectar registros de producción de tubos de acero, con una cola Apache Kafka (extracción), que a su vez alimenta una base de datos Postgres, y esta, a una base de datos SQL Server (transformación) a través también de Nifi. Luego se envían los datos a un software de planificación (carga). Por otra parte, los datos obtenidos divergen también a sistemas de análisis y reportes de datos.

Los procesos se encuentran activos en servidores, a la espera de mensajes que activen los flujos. En el caso explicado, constantemente se reciben flowfiles para procesar, ya que simula un proceso real time. En cambio, para el

procesamiento en lotes, existe la posibilidad de cronometrar su ejecución. Podemos establecer mediante el lenguaje de expresión temporal CRON la periodicidad de ejecución, o simplemente establecer en los procesadores un timer de ejecución deseado para que se activen al llegar los mismos a 0. A estos procesadores de inicio se les llaman Trigger o disparadores de proceso.

Desde una perspectiva integradora de los procesos en el seno de una organización la cual ya cuenta con sistemas desarrollados, una de las complicaciones que se pueden presentar al trabajar con esta herramienta, es que, como mencionamos anteriormente, Nifi está desarrollado en Java. Por lo tanto, a forma de ejemplo, si queremos ejecutar un script en Python directamente en Nifi, tenemos la opción de hacerlo permitiéndo que Nifi traduzca nuestro script a Jython. Jython es una implementación de Python que se ejecuta en la máquina virtual de Java (JVM), lo que permite escribir y ejecutar código Python en un entorno Java. El motor identificado como "python" en la lista de motores de script disponibles es, en realidad, Jython en lugar de CPython, la implementación estándar de Python. Es importante destacar que, al utilizar Jython, no es posible importar módulos exclusivos de CPython, como es el caso beautifulsoup, utilizado para el scraping. Esta y demás librerías que usamos para escribir el script no están disponibles en el lenguaje disponible, por lo tanto el script en su versión .py no es posible de ejecutar en esta herramienta, teniendo que tomar el trabajo extra de transformar el archivo a ejecutable cada vez que lo modifiquemos.

Aún así, cabe aclarar, que todas las herramientas de ETL tienen la capacidad de prescindir de su interfaz gráfica y escribir código ejecutable. Lo que exponen Ralph Kimball y Joe Caserta en The Data Warehouse ETL Toolkit [12] es que un enfoque basado en herramientas limita el sistema a las capacidades del proveedor de la herramienta y a su lenguaje de programación único. Sin embargo, se suele poder escapar de las limitaciones de la interfaz, y desarrollar módulos codificados en un lenguaje común y reconocido para posteriormente integrarlo al sistema desarrollado con la herramienta de ETL.

Hay que poner en la balanza, la facilidad de llegar a estos escapes de código (o a estas formas de prescindir de la GUI para personalizar el proceso), más la frecuencia que necesitaremos utilizar los mismos. O sea, si la herramienta que elegimos para nuestro proceso ya provee la mayoría de las funcionalidades que necesitamos para el mismo.

3.4 Herramientas de reporting con GUI y basadas en código

Existen dos principales alternativas para realizar un reporte de datos interactivo:

- Herramientas empresariales con GUI (PowerBI)
    - Power BI es una herramienta empresarial desarrollada por Microsoft que facilita la creación de informes interactivos y paneles de control. Permite la conexión a diversas fuentes de datos y ofrece una amplia gama de visualizaciones predefinidas. Se puede programar actualización de datos en forma automática y compartir informes en la nube. La interfaz de usuario de Power BI es intuitiva, lo que la hace accesible para usuarios no técnicos.
- Módulos de código (Python con librerías de visualización)
    - Python es ampliamente utilizado en análisis de datos y visualización. Se pueden utilizar bibliotecas como Matplotlib, Seaborn, Plotly u otras para crear visualizaciones personalizadas. Se ofrece un alto grado de flexibilidad y personalización, lo que es especialmente útil para escenarios en los que se requieren visualizaciones altamente personalizadas o análisis avanzados.

En este trabajo utilizaremos Microsoft Power BI, ya que ejemplifica las herramientas drag and drop, teniendo una interfaz simple de entender, a diferencia de otras herramientas similares de la competencia como Tableau.

Por otro lado, al igual que en la programación del flujo de ETL, utilizaremos Python con pandas, matplotlib y numpy para manipular los datos obtenidos y crear informes. Se hará también en un servidor local montando un entorno con Jupyter Notebooks.

4. Desarrollo

4.1 Esquema general del desarrollo

La sección correspondiente al desarrollo de este trabajo será presentada siguiendo el paradigma ETL (Extract, Transform, Load), en su propio orden. Se implementarán múltiples sistemas para abordar una misma problemática, empleando diversas tecnologías y herramientas, con la finalidad de efectuar una posterior comparación entre los enfoques empleados.

El problema ejemplo estudiado en esta tesina implica la evaluación de la variación de precios en distintas categorías de productos desde Julio de 2023 hasta Noviembre 2023. Para llevar a cabo esta tarea, fue recolectada información relativa a la escala de precios a través de dos fuentes primarias. En primer lugar, se accedió a datos proporcionados por la API del Banco Central de la República Argentina (BCRA) [10]. En segundo lugar, se obtendrán datos mediante la técnica de web-scraping de diversas categorías de ítems en el portal de Mercado Libre.

La metodología de ejecución empleada para estas dos fuentes de datos será adaptada según el enfoque seleccionado, es decir, si se utiliza una herramienta de ETL de tipo "drag-n-drop" como Apache Nifi, o si se opta por construir un pipeline exclusivamente mediante programación y el uso de librerías especializadas para el tratamiento y procesamiento de datos en Python como pyspark. Durante esta fase, se llevarán a cabo todas las transformaciones necesarias sobre los datos adquiridos con el fin de prepararlos adecuadamente para su posterior carga en los destinos específicos.

Se procederá a evaluar la eficacia de las operaciones de carga de datos en ambos sistemas seleccionados. Por un lado, se realizará la carga en un almacén de datos SQL empleando el sistema de gestión de bases de datos Postgres. Por otro lado, se llevará a cabo la carga en un sistema NoSQL utilizando MongoDB.

A partir de este punto, se procederá a la utilización de los datos obtenidos con el propósito de generar informes y reportes. Para tal fin, se emplearán herramientas de generación de reportes "drag-n-drop" estándar como Microsoft PowerBI, así como también se desarrollará un reporte ejecutable mediante notebooks de código, escritas en el lenguaje de programación Python. Estos informes servirán para analizar y presentar de manera efectiva los resultados derivados de la evaluación de la variación de precios en las categorías de productos consideradas.

4.2 Extracción

Para realizar la fase de extracción de los datos, primero definimos las fuentes. Queremos comparar los precios de los productos en tecnología, contra la inflación mensual dictada por el BCRA.

Para obtener los precios de los productos, utilizamos técnicas de web-scraping en Python para poder extraer de Mercado Libre los precios de artículos de tipo televisores, celulares y heladeras. Se escogió este sitio ya que concentra productos de diversas marcas y fabricantes. Además de que las etiquetas en su web referentes al listado de artículos no suele cambiar con frecuencia. Estos precios serán obtenidos a lo largo de los siguientes meses, empezando desde julio 2023, por lo que semanalmente el programa se ejecutará para recolectar los datos de la web.

Por otro lado, nos conectamos a la API del BCRA [11] que nos proporcionará datos sobre la inflación mes a mes. Al ser API REST podremos realizar directamente la extracción dentro de los contextos de las herramientas de reporting, ya que no existe la necesidad de persistir localmente estos datos.

Los métodos de puesta en ejecución de estos programas dependen de las herramientas que utilizaremos para tratar estos datos. Serán invocados con regularidad automáticamente, simulando un proceso batch de una organización real.

Dejamos de lado las interfaces del tipo real time o near real time para un futuro trabajo, ya que la data proveniente de este tipo de procesos suele tener diferente uso, propósito y forma de tratamiento, que pueden influir en las herramientas a elegir. Además, suelen requerir de una mayor capacidad de cómputo y requieren una disponibilidad constante de los recursos de cómputo.

4.2.1 Scraping

Se realizó un programa de script en Python que busca tres principales categorías de tecnología en Mercado Libre: teléfonos móviles, heladeras y televisores. Se eligieron estos artículos ya que su precio no varía por temporadas (por ejemplo, los aires acondicionados suelen estar más caros acercándose el verano, o los caloventores más caros en invierno), además de existir suficiente variedad de marcas de los mismos.

De cada artículo se toman 50 ejemplares semanalmente. De cada uno se almacena su categoría, su nombre, su vendedor y su precio. Las páginas que se utilizan para obtener los listados simplemente son la búsqueda del tipo de artículo en la plataforma. Por ejemplo, para obtener el listado de celulares la fuente es:

https://listado.mercadolibre.com.ar/**celulares**#D[A:**celulares**]

Cambiando la palabra ‘celulares’ por ‘heladeras’ obtenemos el listado de heladeras a realizar el scraping.

Se utilizó principalmente la librería beautiful soup, que permite analizar documentos HTML y XML, permitiendo extraer la información del precio, modelo y vendedor para cada artículo.

Luego de extraída la información, la almacenamos en forma de JSON en un directorio. Este almacena toda la información cruda que posteriormente transformaremos para insertar en nuestras bases de datos

4.2.1.1 Implementación del scraping en Nifi

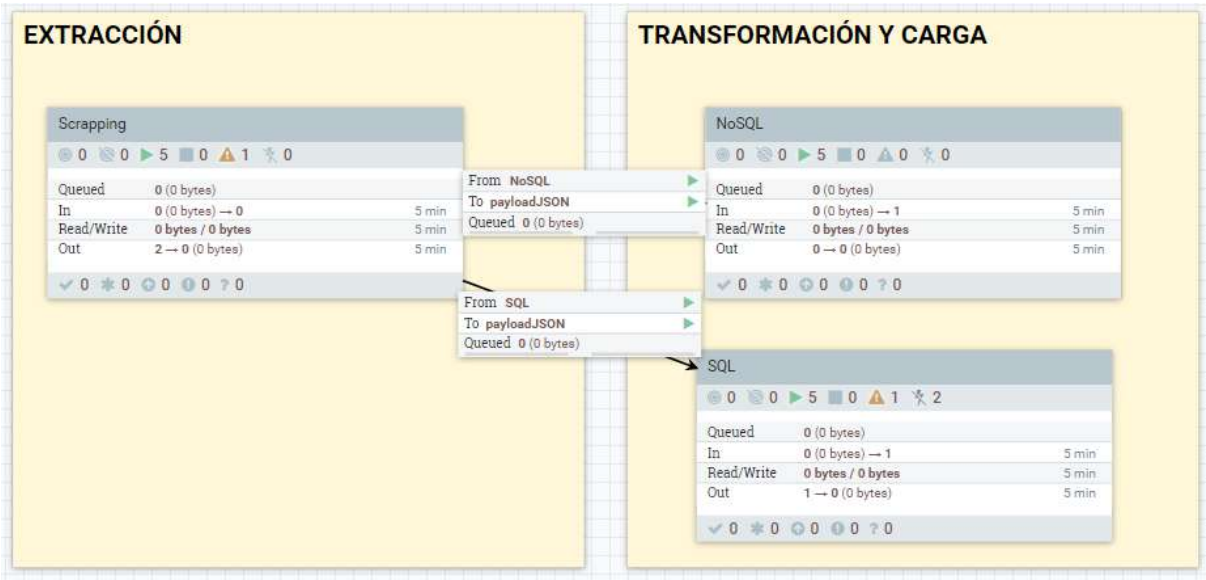

Figura 3: se muestra el proceso dividido en la sección de extracción y transformación/carga. La sección de extracción incluye el scraping

Tenemos tres subprocesos dentro del proceso de ETL en Nifi. Estos a su vez engloban más subprocesos en su interior, a los cuales podemos acceder para ver en más detalle sus secuencias de ejecución.

El primero de ellos corresponde al procedimiento de scraping (y consecuentemente al proceso dedicado a la etapa de extracción).

Procedemos adentrarnos al mismo para analizar su funcionamiento en detalle:

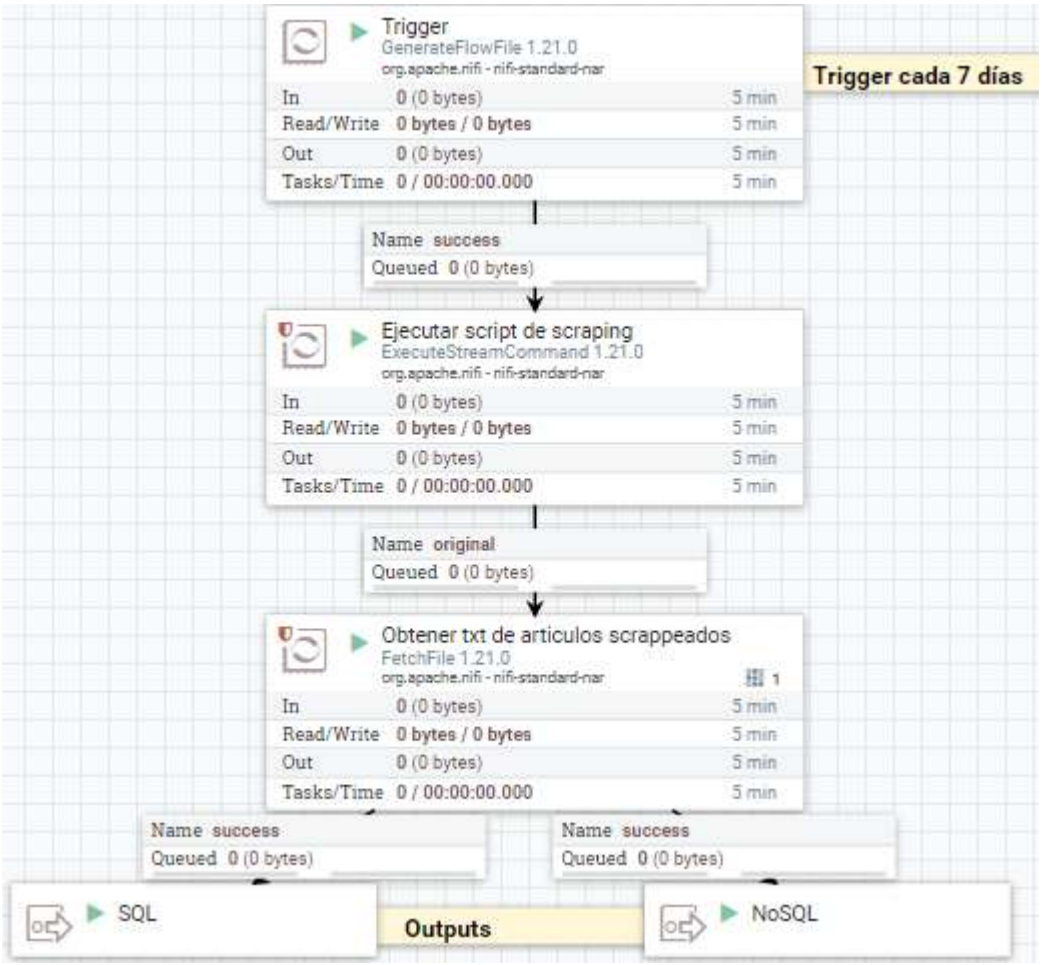

Figura 4: Se muestra el contenido del proceso ‘scraping’ dentro de la sección de extracción

Vemos que tenemos tres procesadores secuenciales más dos compuertas de output. Vamos a dividir la cadena por procesador:

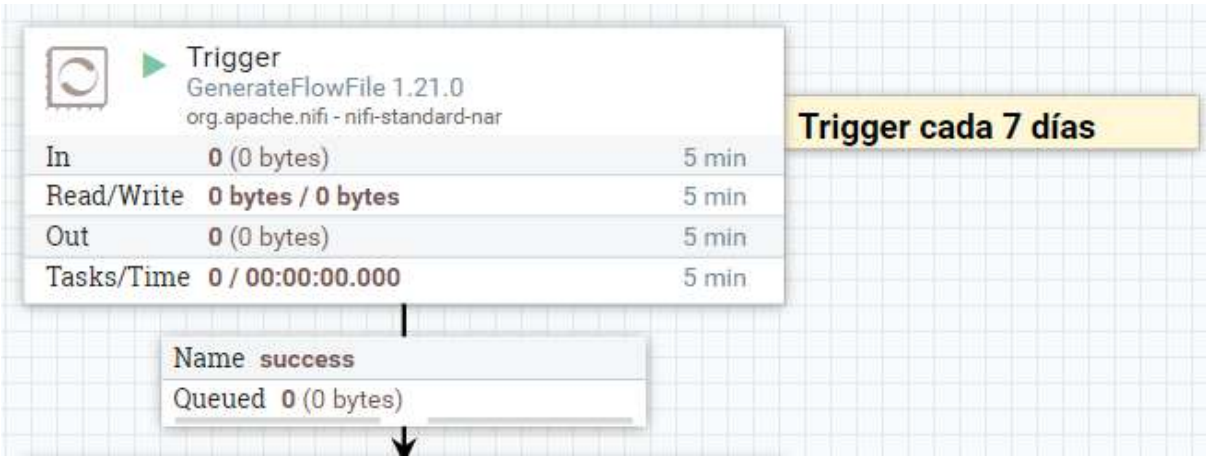

Figura 5: Se muestra el procesador disparador del proceso periódico de scraping

Como queremos que el proceso se ejecute automáticamente, configuramos un primer procesador que funcionará de disparador de todo el

proceso ETL. Genera un flowfile sin contenido con el fin de despertar el proceso cada semana, el cual llegará al siguiente procesador, ilustrado en la figura 6 y provocará su ejecución.

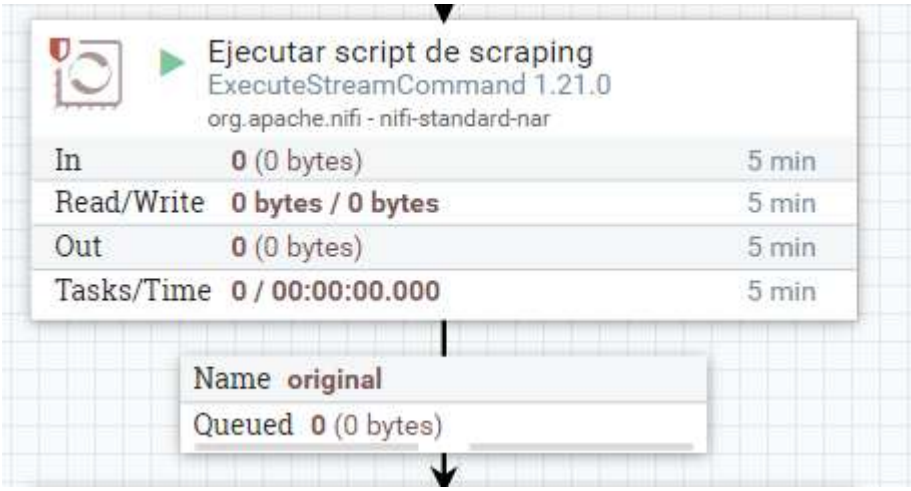

Figura 6: Se muestra el procesador encargado de ejecutar el script programado para realizar el scraping en Mercado Libre

El procesador ExecuteStreamCommand va a ejecutar el script en forma de ejecutable de Windows. Para esto, anteriormente con la librería pyinstaller transformamos el script python con la tarea de scraping en un archivo .exe

Una vez ejecutado el script, deja la información en una carpeta local de staging junto con los archivos generados por anteriores ejecuciones. El staging de datos se refiere al proceso de preparar y organizar los datos antes de ser procesados o almacenados de manera permanente en un sistema. Es una etapa intermedia en el flujo de trabajo de manejo de datos. Esto es, a forma de persistir los datos para luego poder darle otro uso si se quisiese, o simplemente como forma de resguardar la información en caso que se comprometa de alguna manera los almacenes de datos que emplearemos en el proceso de carga.

Continuando con la última parte del flujo de extracción referente a Nifi, podemos observar en la figura 7 la última parte del mismo.

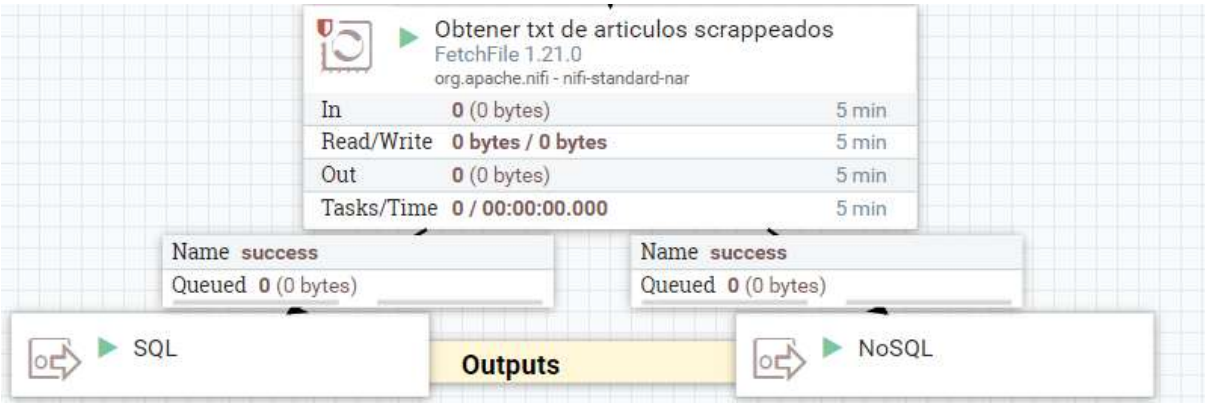

Figura 7 Se muestra el procesador encargado de tomar el archivo con el contenido de lo scrappeado de la carpeta de staging.

FetchFile se encarga de buscar en el directorio correspondiente y dirigir el archivo creado por el ejecutable que contiene el web scraping. Así, lo rutea a los dos procesadores de salida. A partir de este punto, el flujo comienza su etapa de transformación y carga. Siendo diferente el tratamiento de la data dependiendo su destino.

4.4.1.2 Implementación de scraping en Python

Al escribir la totalidad del pipeline utilizando únicamente Python, a diferencia del caso anterior, reducimos la complejidad a realizar una llamada al mismo como una función. En la figura 8 se muestra como se realizó la invocación al módulo de scraping.

```
[3]: import requests
     import json
     from bs4 import BeautifulSoup
     from datetime import datetime
     import sys

[4]: def scrapping_script():
         ....
```

Para extraer, simplemente llamamos a la función predefinida, que nos dejará la data en nuestra carpeta local de staging

```
[ ]: #extracción
     scrapping_script()
```

Figura 8. Se muestra la celda en donde se invoca en forma de función al script de scraping junto al script de scraping. Se redujo la función real con fin de mostrar en la imagen la invocación posterior.

Cómo se ve, el proceso de extracción se reduce a la invocación del módulo ‘scraping_script’. Además, esta solución permite modificar el método de scraping sin necesidad de posteriormente transformarlo a ejecutable. Podemos ver el código de forma clara en el pipeline, y no requerimos permisos

para modificar los contenidos de a un filesystem para accederlo, a diferencia de la solución en Nifi, donde el script se encuentra guardado en una carpeta.

4.5 Transformación

4.5.1 Herramientas con interfaz gráfica

La figura 9 detalla la etapa de transformación realizada en Apache Nifi.

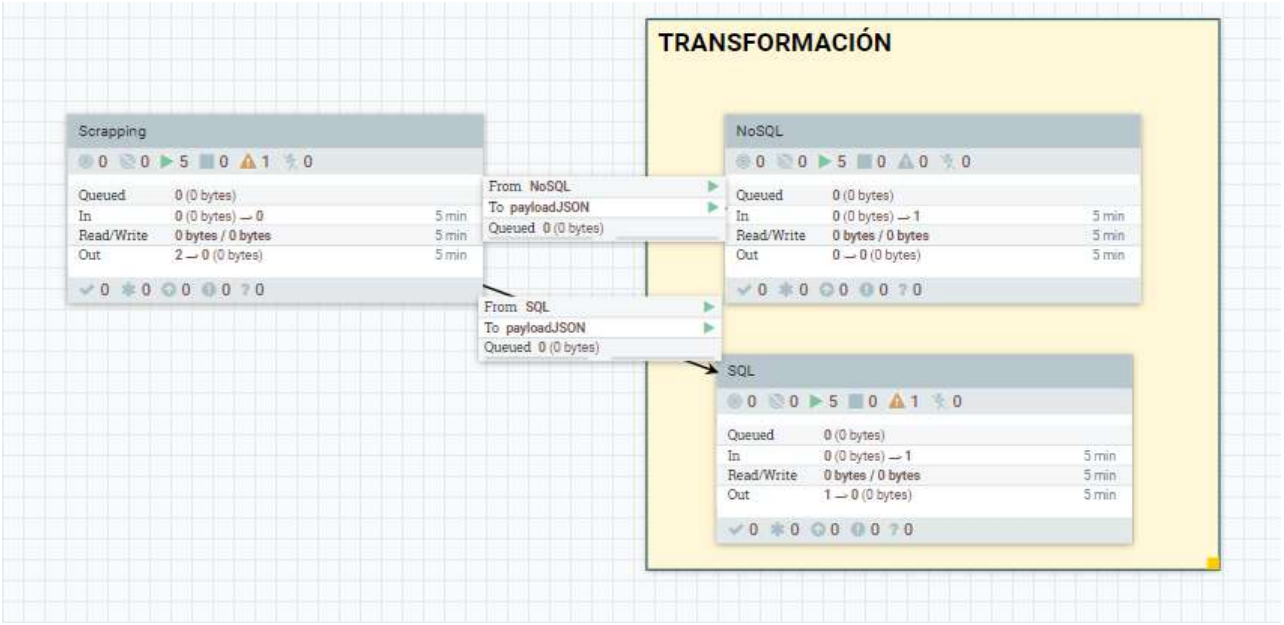

Figura 9 Se resalta la etapa de transformación en el proceso ETL Nifi

Como se ha mencionado, el flujo se divide en dos transformaciones que se ejecutan simultáneamente, una para cada motor de base de datos estudiado. Algo a resaltar, es que a diferencia de la transformación en código, no debemos manejar conceptos ni librerías especiales para ejecutar concurrentemente código (como por ejemplo, pthreads) o en el caso de Nifi, procesadores. Solo basta con diseñar una bifurcación en el flujo para que los flowfiles se dirijan en ambas direcciones y activen simultáneamente ambos subflujos.

Una rama procesa la data extraída para insertarla en PostgreSQL (proceso ‘SQL’ de la Figura 9), mientras que la otra procesa los datos para poder combinarlos en MongoDB (proceso ‘noSQL’ de la Figura 9).

Adentrándonos en el procesador ‘noSQL’, podemos ver su flujo de procesadores en la figura 10.

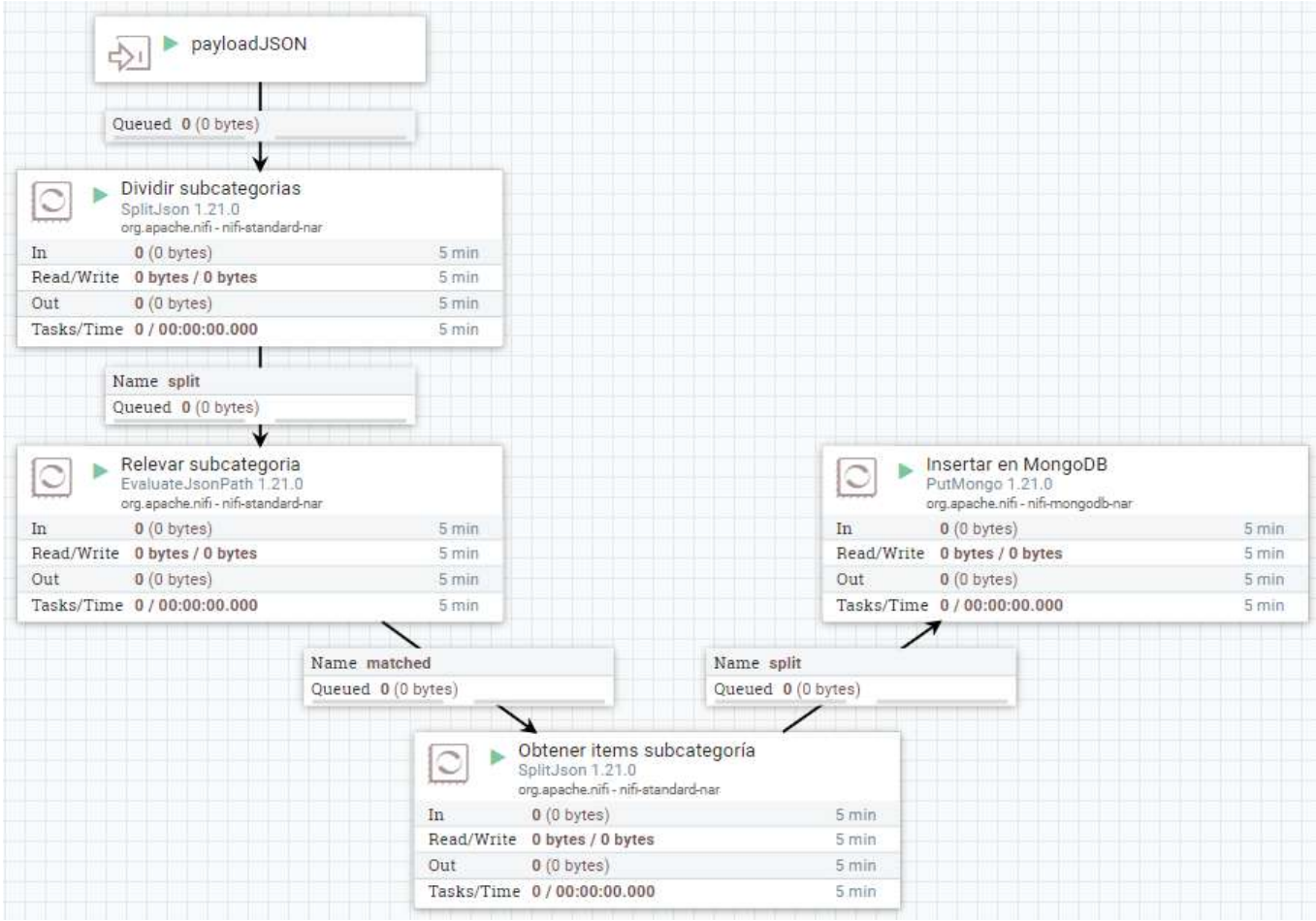

Figura 10. Se muestra en detalle el proceso ‘noSQL’ que inserta la data tomada de la carpeta de staging a mongoDB

Los pasos que realiza son los siguientes:

- Se recibe el payload como array JSON del scraping (entrada payloadJSON).
- Se divide el array JSON por categoría de producto (procesador SplitJson)
- Se divide los arrays pertenecientes a cada categoría de productos en JSON individuales de manera que cada uno represente un ítem (procesador SplitJson)
- Se inserta cada ítem en la colección MongoDB (procesador PutMongo)

Adentrándonos en el procesador ‘SQL’, podemos ver su flujo de procesadores en la figura 11.

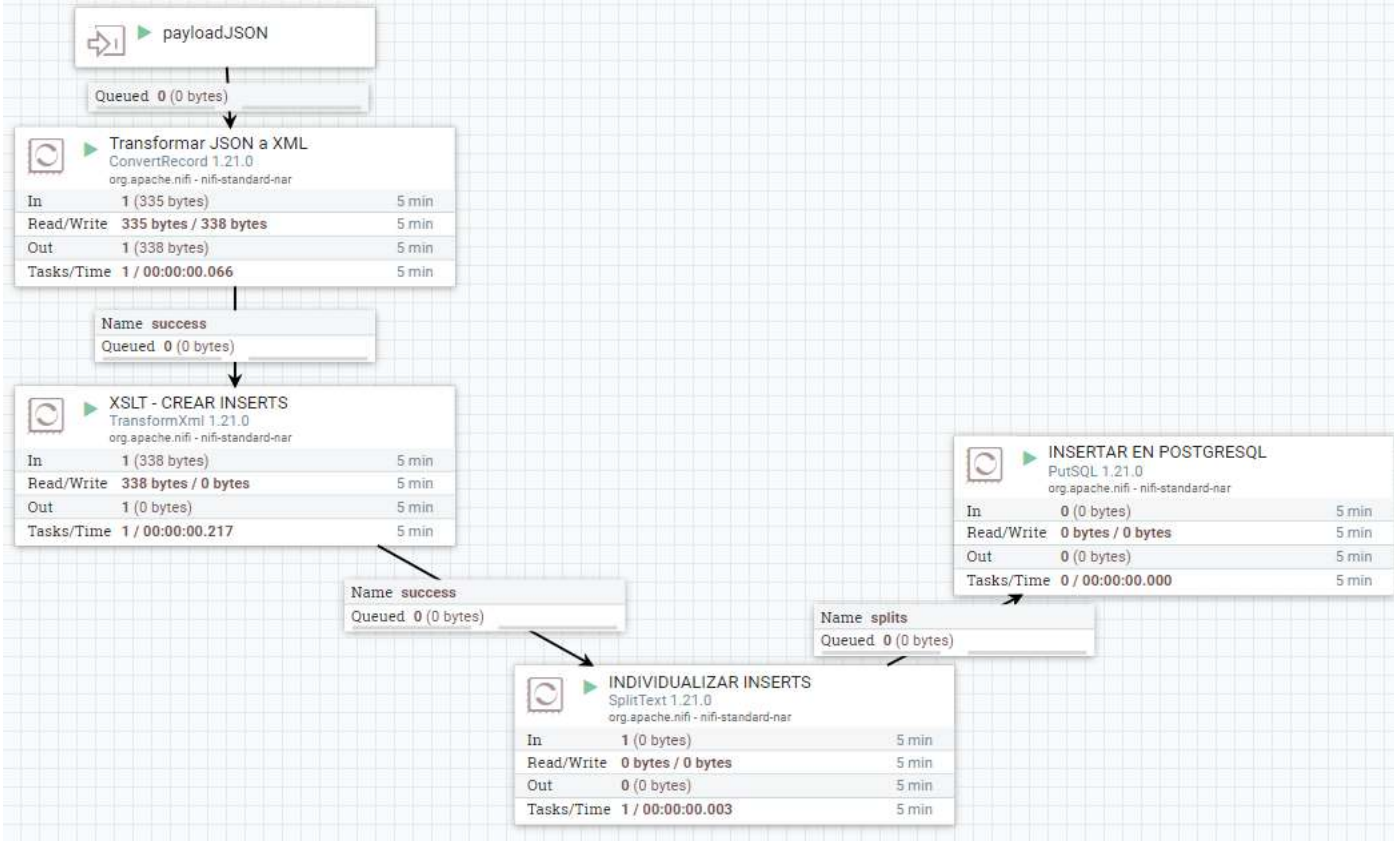

Figura 11 Se muestra en detalle el proceso ‘SQL’ que inserta la data tomada de la carpeta de staging a PostgreSQL

- Se recibe el payload como array JSON del scraping (entrada payloadJSON).
- Se convierte el array JSON a XML (ConvertRecord)
- Se realiza un parseo del XML mediante un archivo XSLT (archivo adjunto) que genera los INSERT STATEMENT a ejecutar en la base de datos (TransformXml - INSERT)
- Se ejecutan los INSERT STATEMENTS en PostgreSQL (PutSQL)

A su vez, durante cada paso del proceso de transformación en la herramienta se genera metadata sobre los flowfiles del flujo los cuales podemos obtener información importante referente al proceso. Por ejemplo, en el caso que tengamos múltiples scripts de scraping que obtengan datos de productos de múltiples fuentes, sería útil saber de donde provino un flowfile determinado.

Por eso, nuestro proceso Nifi, cuando ejecuta el script, asociará una variable ‘execution command’ referente al proceso de ejecución del script comentado en 3.1, que indicará de cual proviene, como se puede ver en la figura 12.

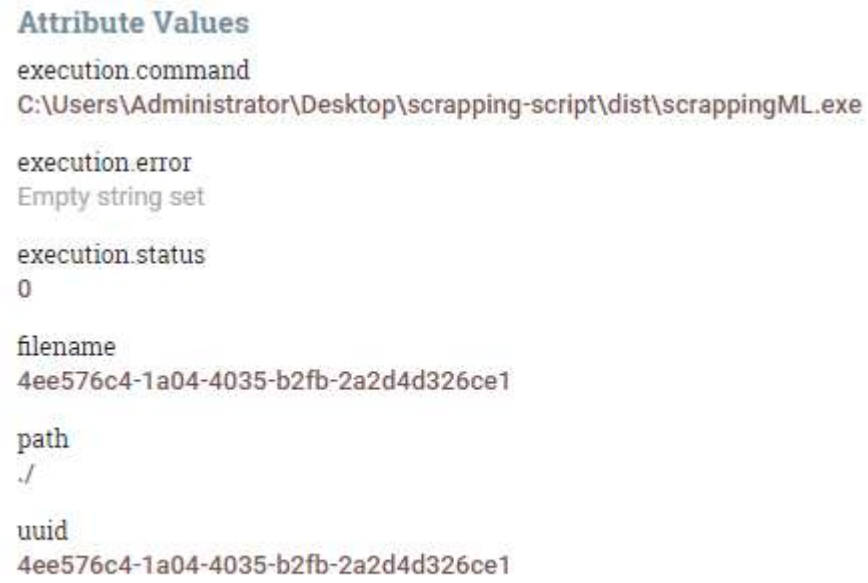

Figura 12. Se observa la metadata que Nifi recopiló de un flowfile durante su transcurso por todo el proceso.

En este caso, vemos que un flowfile determinado proviene del scraping de Mercado Libre (scrapingML.exe). Esta variable será arrastrada por todo el proceso, hasta su consumo en los procesadores finales. En caso de división del flowfile en varios ‘hijos’ del mismo (como por ejemplo, cuando separamos el array de JSON en registros individuales), ellos heredarán las variables de su padre. Estos tipos de datos en conjunto, no solo sirven para la detección de errores sino que se pueden utilizar para estudiar y mejorar el proceso.

Concluyendo el proceso de transformación para Nifi, este tipo de productos son activos indispensables en el ámbito del desarrollo de sistemas, ofreciendo un conjunto integral de características que simplifican los procesos de extracción, transformación y carga de datos. Por ejemplo, simplificamos la concurrencia y la recolección de la metadata. Su facilidad de uso convierte a este tipo de herramientas en un recurso crucial para gestionar los paisajes complejos de datos de las organizaciones actuales.

4.5.2 Transformación en notebooks de código.

La transformación de los datos utilizando código es mucho más directa.

Para el flujo correspondiente a la posterior carga en la base de datos noSQL simplemente se obtuvo el JSON generado y se procedió a configurar el conector

Haremos la transformación y carga en NoSQL. Para este caso en particular, las transformaciones son minimas ya que en nuestra carpeta local de staging los documentos se guardan en JSON, que es el formato admitido por MongoDB para sus colecciones de documentos. Utilizando la librería pymongo podemos insertar de forma simple y directa los datos adquiridos del portal de ML

```
import pymongo

with open(r'C:\Users\Administrator\Desktop\scrapping-script\scrappingfiles\\'+str(datetime.now()).
          replace(" ", "").replace(":","").replace('-','').replace(".","")[0:8]+'.json') as json_file:
    json_to_py_dict_ex = json.load(json_file)

client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["articulosML"]
collection = db["electronica"]

merge_subcategorias = [item['data'] for item in  json_to_py_dict_ex if 'data' in item]
merge_items_subcategorias = [diccionario for lista in merge_subcategorias for diccionario in lista]

collection.insert_many(merge_items_subcategorias)
```

Figura 13. Se observa la porción de código en notebook que realiza la transformación y carga en noSQL.

Por otro lado, la transformación para la posterior carga de estos datos a la base de datos PostgreSQL es más compleja, debiendo hacer uso de pySpark:

- Importamos las bibliotecas necesarias
  - Se importa la clase SparkSession de pyspark.sql para trabajar con Spark.
  - Se importan los tipos de datos y funciones necesarios de pyspark.sql.
  - Se importa la función `explode` de pyspark.sql.functions.

```
from pyspark.sql import SparkSession, types
from pyspark.sql.functions import explode
```

Figura 14. Se importan las librerías necesarias para la transformación utilizando spark.

- Configuramos la sesión spark
  - Se crea una instancia de SparkSession nombrada con la variable "spark" junto con configuraciones específicas:
  - Se establece el nombre de la aplicación como "TransformacionPostgres".
  - Se habilita el uso de Arrow para mejorar la velocidad de ejecución de las consultas SQL, utilizando un formato de datos binario y eficiente.

```
#transformacion sql
spark = SparkSession.builder \
    .appName("TransformacionPostgres") \
    .config("spark.sql.execution.arrow.enabled", "true") \
    .getOrCreate()
```

Figura 15. Se crea una sesión de Spark con configuraciones específicas.

- Leemos los datos del JSON generado por el proceso de extracción y lo guardamos como dataframe.

```
df_articulos = spark.read.json(r'C:\Users\Administrator\Desktop\scrapping-script\scrappingfiles\\'
                               +str(datetime.now()).replace(" ", "").replace(":","")
                               .replace('-','').replace(".","")[0:8]+'.json' ,multiLine=True)

df_articulos.printSchema()
df_articulos.show()
```

Figura 16. Se observa la porción de código en notebook que carga los datos extraídos en la anterior etapa y los transforma en dataframe.

Adicionalmente con printSchema() mostramos el esquema de datos generados, y con show() el contenido del mismo.

```
root
 |-- categoria: string (nullable = true)
 |-- data: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- document_date: string (nullable = true)
 |    |    |-- nombre_articulo: string (nullable = true)
 |    |    |-- precio: string (nullable = true)
 |    |    |-- subcategoria: string (nullable = true)
 |    |    |-- vendedor: string (nullable = true)
 |-- subcategoria: string (nullable = true)

+-----------+--------------------+------------+
|  categoria|                data|subcategoria|
+-----------+--------------------+------------+
|electronica|[{20231118, Celul...|   celulares|
|electronica|[{20231118, Helad...|   heladeras|
|electronica|[{20231118, Smart...| televisores|
+-----------+--------------------+------------+
```

Figura 17. Primero se muestra el esquema crudo de los datos, luego se los imprime como dataframe.

- Se aplican las transformaciones al dataframe.
    - Se seleccionan las columnas "categoria", "data" (que se al ser un array se explotará, es decir, expandirá en múltiples rows) y "subcategoria".
    - Se utiliza la función explode para expandir la columna "data" en

varias filas.

- Se seleccionan columnas específicas del dataframe resultante de la explosión.
- Se renombran algunas columnas usando la función `alias`.
- Se convierte la columna "precio" a decimal, con 10 dígitos en total y 2 de ellos decimales.
- Se convierte la columna "document_date" a timestamp.

```
df_exploded = df_articulos.select(
    col("categoria"),
    explode("data").alias("exploded_data"),
    col("subcategoria")
)

# Seleccionar campos específicos
transformed_df = df_exploded.select(
    col("subcategoria"),
    col("exploded_data.vendedor").alias("nombre_vendedor"),
    col("exploded_data.nombre_articulo").alias("nombre_articulo"),
    col("exploded_data.precio").cast(DecimalType(10, 2)).alias("precio"),
    col("exploded_data.document_date").cast(TimestampType()).alias("inserted_date")
)
```

Figura 18. Porción de código que transforma la data para dejarla lista para su inserción SQL.

Finalmente se muestran los resultados de la transformación con el fin de revisar que el resultado sea el correcto, ya que son los datos que posteriormente se insertan en la base de datos.

```
# Mostrar el DataFrame resultante
transformed_df.show(truncate=False)
transformed_df.printSchema()
```

Figura 19. Se observa la porción de código que en primer lugar imprime el dataframe generado, y en segundo lugar su esquema.

subcategoria	nombre_vendedor	nombre_articulo	precio	inserted_date
celulares	No especificado	Konka Indus Dual SIM 32 GB negro 3 GB RAM	61455.00	20231213
celulares	No especificado	Teléfono Celular Tcl 40 Nxtpaper midnight Blue Rv	199999.00	20231213
celulares	No especificado	Moto G32 128gb 6gb Ram Plateado	298699.00	20231213
celulares	No especificado	Kanji Fon Dual SIM 32 MB azul 32 MB RAM	33599.00	20231213
celulares	No especificado	Celular Quantum Q-rash 32gb 2gb Verde 1	100000.00	20231213
celulares	No especificado	Samsung Galaxy Z Flip5 8gb + 512gb Color Graphite	1099999.00	20231213
celulares	No especificado	Samsung Galaxy A14 (Mediatek) 5G 128 GB black 4 GB RAM	299999.00	20231213
celulares	No especificado	Celular Quantum Yolo 32 Gb 1 Gb Ram Android 10 8 Mp Rojo 5	55555.00	20231213
celulares	No especificado	Celular Tcl 405 64 Gb Dark Grey 2 Gb Ram	116999.00	20231213
celulares	No especificado	Celular Samsung Galaxy A04e 32 Gb Negro 3 Gb Ram	155999.00	20231213
celulares	No especificado	Xiaomi Redmi Note 11 - Star Blue 128GB 4GB de RAM	249999.00	20231213
celulares	No especificado	Moto G84 5g Color Viva magenta	599999.00	20231213
celulares	No especificado	Samsung Galaxy A04s 4 + 128gb Negro Color Verde	155799.00	20231213
celulares	No especificado	Samsung Galaxy A22 5G 5G 128 GB white 4 GB RAM	289999.00	20231213
celulares	No especificado	ZTE Blade V40 Design Dual SIM 128 GB starry black 6 GB RAM	256589.00	20231213
celulares	No especificado	Samsung A31 128gb Negro Bueno Liberado	103948.00	20231213
celulares	No especificado	Samsung Galaxy A23 5G 128 GB Awesome white 4 GB RAM	345899.00	20231213
celulares	No especificado	Celular Xiaomi Redmi 10c Mint Green Color Verde menta	188999.00	20231213
celulares	No especificado	Celular Motorola Moto E13 64gb 2gb Ram Azul Turquesa	168999.00	20231213
celulares	No especificado	Celular Quantum Q30 Negro 128gb 6gb Ram 4g Octacore 6.5	246999.00	20231213

Figura 20. Se imprimen las primeras 20 columnas del dataframe.

```
root
 |-- subcategoria: string (nullable = true)
 |-- nombre_vendedor: string (nullable = true)
 |-- nombre_articulo: string (nullable = true)
 |-- precio: decimal(10,2) (nullable = true)
 |-- inserted_date: timestamp (nullable = true)
```

Figura 21. Se imprime el esquema final antes de su inserción.

4.6 Carga

Como se presentó previamente, el proceso de carga es el último paso del proceso de ETL. Los datos que han sido previamente extraídos y transformados se cargarán en un almacén de datos o sistema de destino. Este proceso de carga es de vital importancia para asegurar que los datos estén listos para su análisis y consulta, brindando una base sólida para la toma de decisiones.

Se introducirá a dos dimensiones críticas en relación con la carga de datos: la carga en SQL relacionado a Data Warehousing y la carga en noSQL teniendo en cuenta ambos procesos utilizados anteriormente (transformación con herramientas drag n drop, y su contraparte, con código). Estos enfoques presentan diferencias notables en términos de estructura y tecnologías utilizadas. Observaremos cómo cada uno de ellos se adapta a diversos contextos y necesidades de las organizaciones, ofreciendo un panorama de las opciones disponibles en la gestión de datos durante esta etapa.

4.6.1 Carga en SQL y Data Warehousing

Para poder cargar los datos transformados primero hay que definir el modelo de base de datos relacional para su organización. El modelo debe reflejar de manera lógica las relaciones entre los diferentes elementos manipulados durante el proceso. Esto facilita la posterior generación de diagramas precisos y comprensibles. Sumado a esto, el modelo debe estar diseñado para permitir consultas eficientes que extraigan los datos necesarios de manera rápida y precisa.

Para nuestro modelo, teniendo en cuenta el problema que incluye productos, vendedores, categorías y subcategorías, podríamos en un principio pensar el siguiente esquema:

- Categoría(ID_categoría, nombre_categoría)
- Subcategoría(ID_subcategoría, ID_categoría, nombre_subcategoría)
- Producto(ID_producto, ID_subcategoría, ID_vendedor, nombre_producto, precio, fecha)
- Vendedor(ID_vendedor, nombre_vendedor)

Este esquema está en la cuarta forma normal, por lo que puede servir para nuestro posterior uso de análisis de datos. De hecho, podríamos pensarlo como una versión extremadamente simplificada y plana del modelo de datos perteneciente a la fuente de los mismos, Mercado Libre.

En dicha plataforma constantemente se están cargando nuevos productos, registrando nuevos vendedores, eliminando productos, eliminando vendedores, editando los valores de los productos (el precio, principalmente), adicionalmente registrando compras, etcétera. Por lo que este modelo está relacionado para el manejo de transacciones en tiempo real.

Nuestro objetivo es el análisis de datos, por lo que lo más correcto sería usar un modelo que, a diferencia de OLTP, no de gran peso a la concurrencia ya que los usuarios serán únicamente 2: las herramientas de transformación que se encargan de depositarlos, y por otro lado, las herramientas de reporting que posteriormente utilizaremos para reportarlos y analizarlos. Una vez nuestras herramientas depositen los datos, estos no serán modificados ni eliminados.

Nuestro interés es que se realicen múltiples selecciones de datos, posiblemente complejas, relacionando gran cantidad de entidades entre sí. Además se debe tener en cuenta que por lo general no se sabe qué tipos de selecciones se harán, ni qué relaciones se harán entre las entidades al momento en que los analistas realicen los reportes. Si se desea relacionar dos entidades muy lejanas entre sí (es decir, con muchas entidades de por medio para llegar a una conexión), probablemente la consulta se torne compleja y costosa de ejecutar. Las bases de datos OLAP cumplen estos requerimientos.

Un ejemplo de modelo para adaptar el modelo anterior OLTP a DW (data warehouse) es el siguiente:

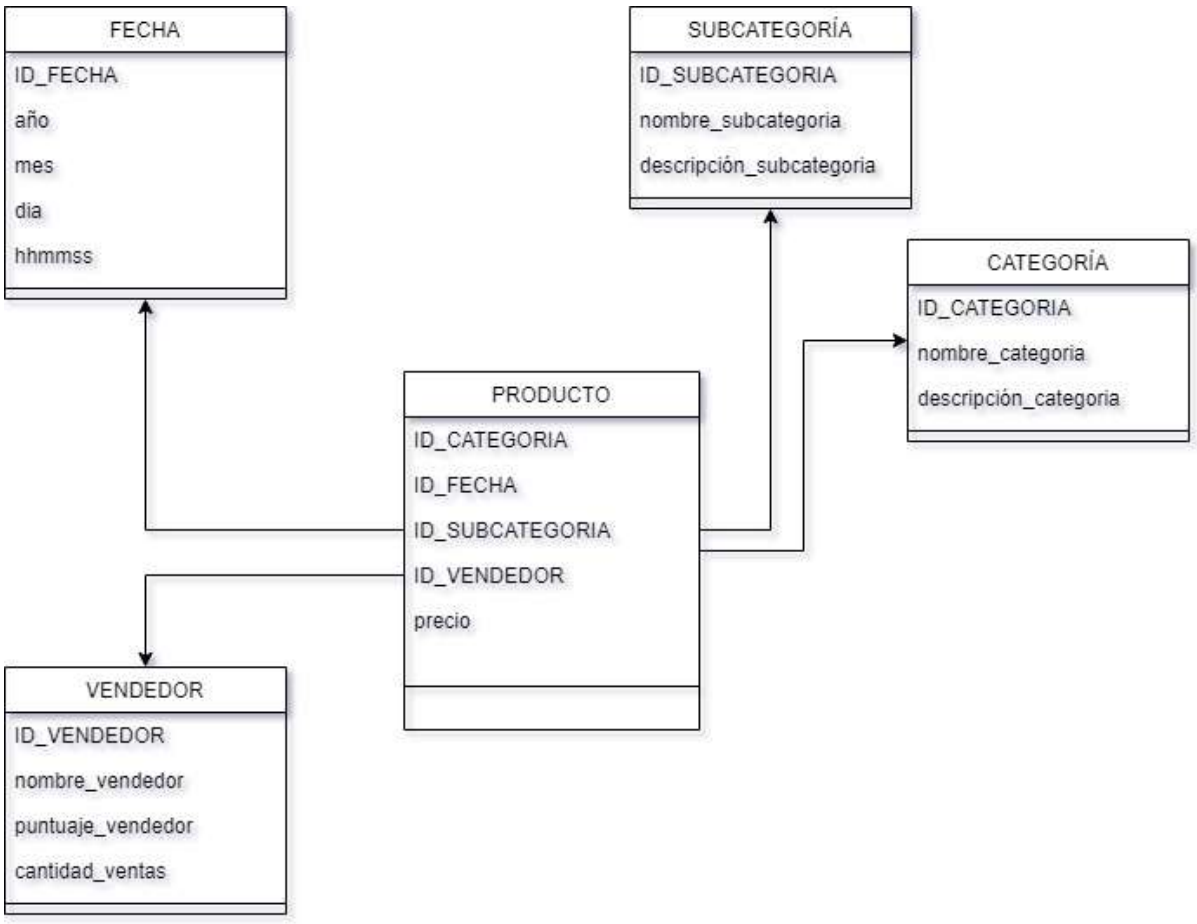

Figura 22. Se muestra un esquema de data warehouse con forma de estrella.

La tabla de facto es PRODUCTO, que posee el dato cuantitativo de precio, el cual posteriormente se filtrará, se agrupará y se explorará mediante los datos cualitativos de las demás tablas de dimensión que lo rodean.

La clave de la tabla PRODUCTO es el conjunto de todas las claves de las tablas de dimensiones que la rodean.

Cabe aclarar que este modelo de DW tiene forma de estrella, pero hay otros tipos de formas de organizar los datos, como por ejemplo copo de nieve, en donde hay dimensiones que a su vez dependen de otra dimensión, a diferencia del esquema estrella. Podríamos haberlo implementado si hacíamos que SUBCATEGORÍA esté relacionado directamente con CATEGORÍA en lugar de con PRODUCTO:

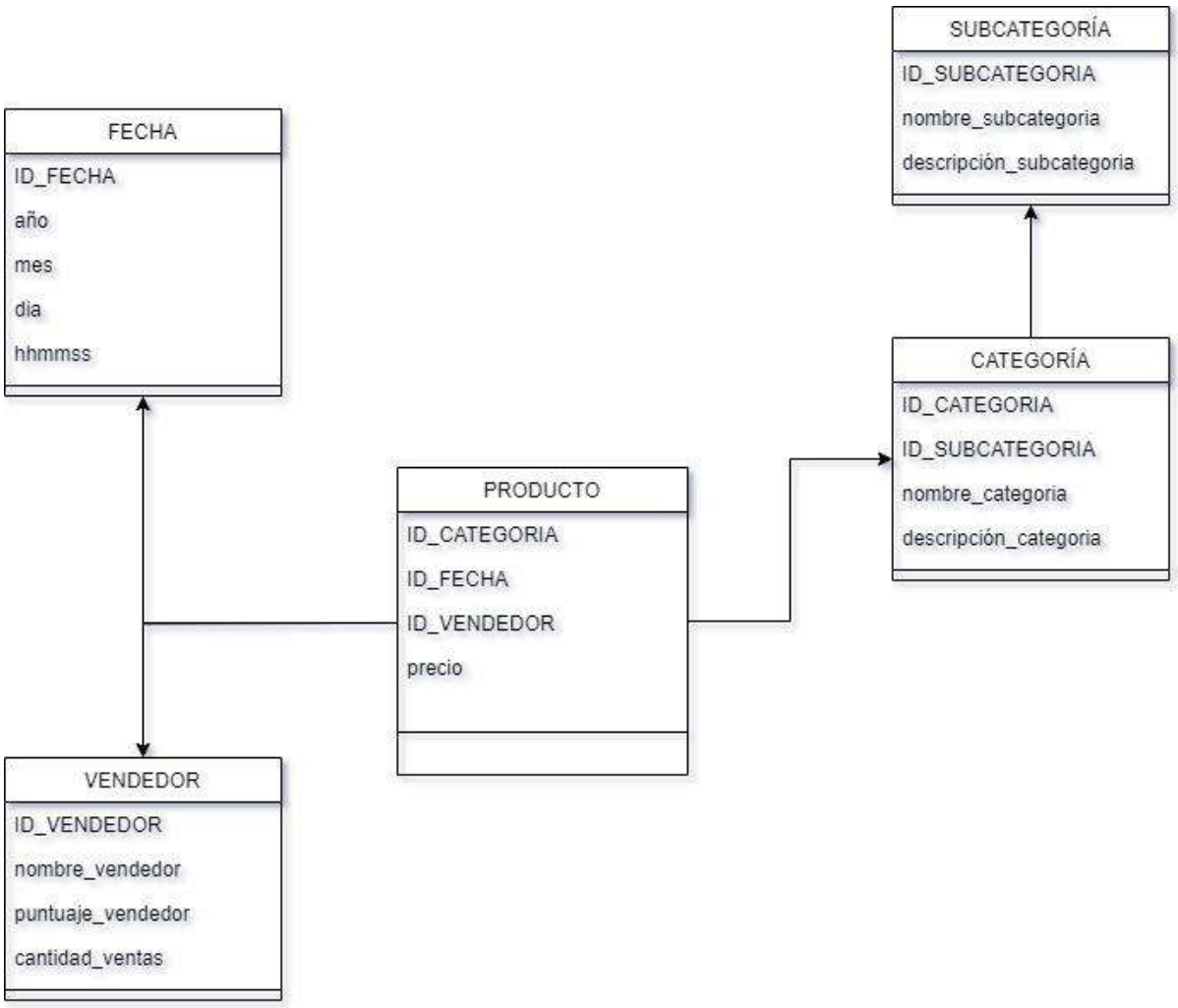

Figura 23. Se muestra un esquema de data warehouse con forma de copo de nieve.

A pesar de esto, como se vió en la introducción, únicamente se analizaran celulares, televisores y heladeras. Por esta razón, en la práctica implementaremos únicamente la siguiente tabla, que representa la categoría ELECTRONICOS:

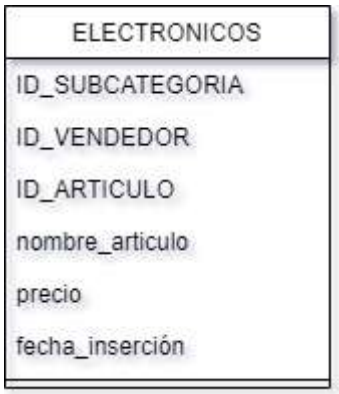

Figura 24. Se muestra la versión reducida del datawarehouse empleada en el desarrollo

Donde la subcategoría será: o “celular”, o “televisor”, o “heladera”. El ID del vendedor será su nombre de usuario en MercadoLibre, el ID de artículo será autogenerado (id incremental), mientras que fecha_inserción representa el día, mes y año de su integración a la base de datos.

Como el análisis será acotado y está enfocado en el uso de las herramientas y técnicas en sí, la complejidad de los reportes a generar irán acorde a este objetivo. Por lo tanto, no se relaciona la información de los vendedores más allá de su nombre, por eso se puede prescindir de su tabla propia.

4.6.1.1 Carga SQL en Nifi

Para la carga SQL se utilizó el procesador PutSQL. Este se utiliza para ejecutar instrucciones SQL en una base de datos relacional, como inserciones (este caso), actualizaciones o eliminaciones de datos en una base de datos a partir de flujos de datos gestionados por Nifi.

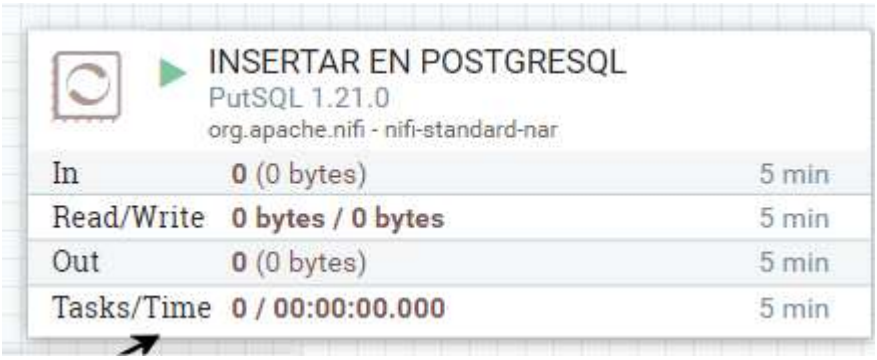

Figura 25. Se muestra la vista del procesador PutSQL en el proceso Nifi.

Se configuraron propiedades clave, como la URL de conexión a la base de datos, el nombre de usuario y la contraseña. Esto se configura creando un conector en la propiedad JDBC Connection Pool del procesador. Se llamó al conector ‘local_PG’:

Required field

Property	Value
JDBC Connection Pool	**local_PG**
SQL Statement	No value set
Support Fragmented Transactions	false
Database Session AutoCommit	false
Transaction Timeout	No value set
Batch Size	**1000**
Obtain Generated Keys	false
Rollback On Failure	**true**

Figura 26. Se muestran las propiedades básicas del procesador. Vemos que puede procesar hasta 1000 instrucciones por lote, y que mantendrá el mensaje en la cola de entrada en caso de fallar (propiedad ‘Rollback On Failure’)

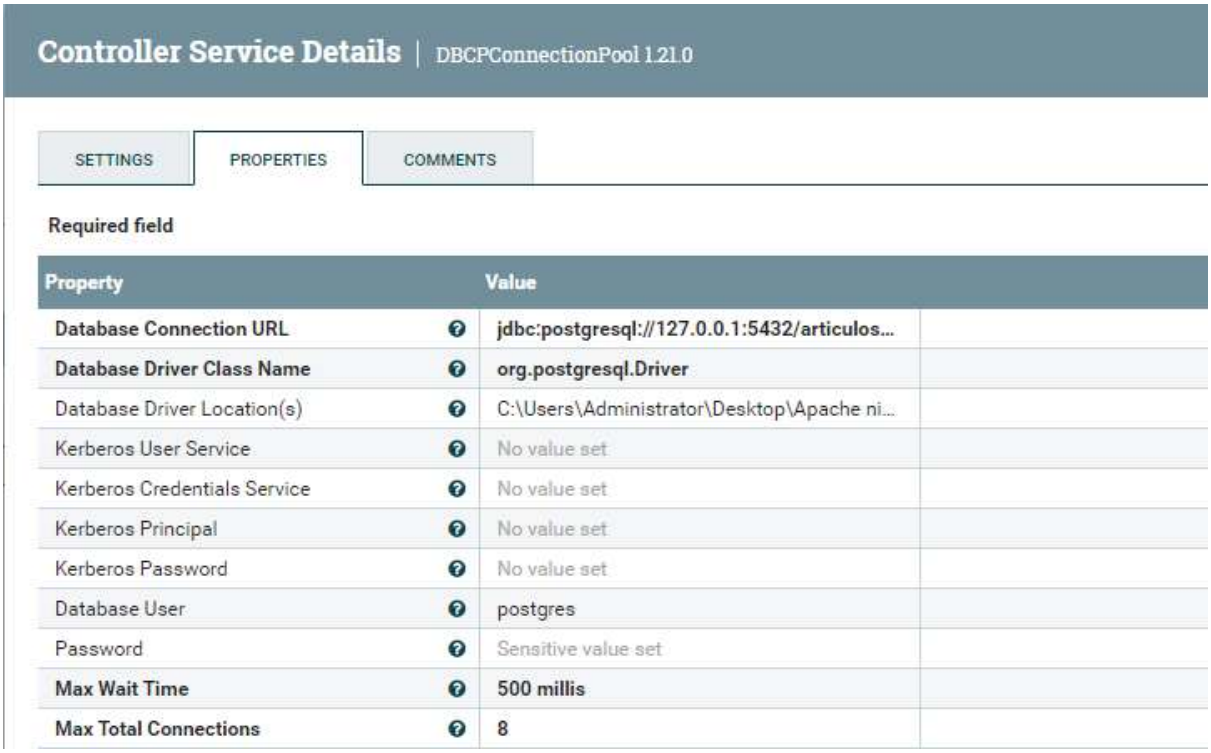

Figura 27. Se muestran las propiedades básicas del conector 'local_PG', especificado en las propiedades básicas del procesador PutSQL. Vemos que en primer lugar tiene la URL de la BD, en segundo lugar se especificó el driver a utilizar, y en segundo lugar la localización del mismo. En Database User y Password tenemos las credenciales.

4.6.1.2 Carga SQL en notebooks

Para la inserción de datos SQL en notebooks, se puede utilizar también pySpark. Con el dataframe generado a partir de la transformación, que es unión compatible con la tabla en nuestra base de datos, se procede a especificar la URL de la base de datos, junto con las propiedades de conexión, que incluyen usuario, contraseña y el mismo driver utilizado para Nifi.

Mediante el método write se puede escribir en la base de datos. El formato jdbc indica que se debe utilizar el driver, mientras que el modo 'append' indica la técnica de inserción incremental de la tabla, agregando registros sin eliminar los anteriores.

Dentro de las opciones (métodos option) especificamos el usuario, la contraseña y el driver a utilizar.

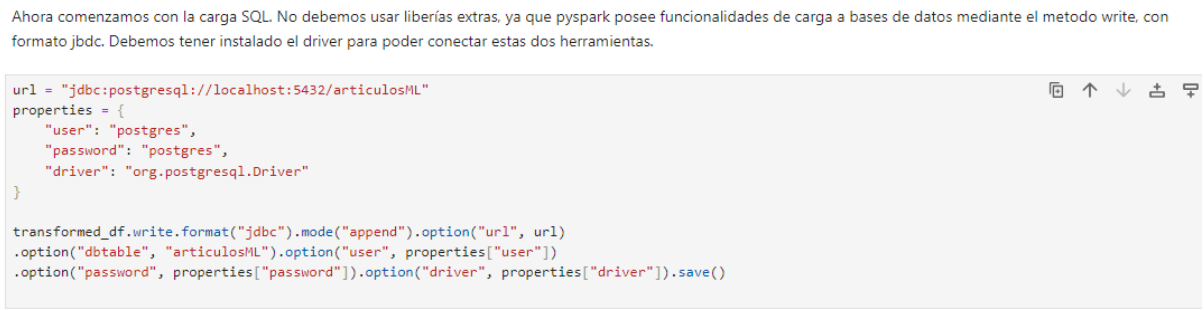

Figura 28. Se muestra el código encargado de la inserción SQL utilizando Spark en notebooks.

4.6.2 Carga en NoSQL

Para el proceso ETL utilizando tecnologías noSQL se trabajó con MongoDB. Con su naturaleza basada en documentos, presentó una opción sumamente ventajosa para la etapa de transformación. El hecho de que MongoDB almacene datos en documentos JSON facilitó de manera significativa la transformación de datos. En este contexto, la transformación se limitó al simple traslado del JSON extraído mediante técnicas de scraping a la colección correspondiente en MongoDB, preservando intacta la estructura original de los datos, independientemente de si se usó la herramienta con GUI o código.

Este enfoque, que aprovecha la flexibilidad y simplicidad intrínseca de MongoDB, demuestra ser una estrategia efectiva para la implementación de ETL en el entorno NoSQL específicamente para nuestro problema. Además, resulta esencial mencionar que este enfoque simplificado en la fase de transformación puede contribuir de manera significativa a la reducción de los costos operativos, al tiempo que garantiza una mayor eficiencia en la gestión y explotación de datos, lo cual es particularmente relevante en un mundo caracterizado por la creciente cantidad de información generada y utilizada en diversas aplicaciones y sistemas.

Este ahorro no solo viene de la simplicidad de la transformación, sino que además prescindimos de diseñar un data warehouse y disponer de la estructura para mantenerlo. En un contexto empresarial podría significar un gran ahorro en cuanto a presupuesto para un determinado proyecto, aunque veremos posteriormente si esto lo vale durante el proceso de análisis de los datos, teniendo en cuenta la complejidad de este problema.

En consecuencia, esta elección de utilizar MongoDB como parte integral del proceso ETL no solo se alinea con las tendencias actuales en el ámbito de la tecnología de la información, sino que también representa un paso crucial en la evolución de las prácticas de gestión de datos. Este enfoque presenta un claro ejemplo de cómo las soluciones NoSQL, como MongoDB, pueden contribuir de manera significativa a la eficacia de los procesos ETL y, en última instancia, a la mejora de la toma de decisiones basada en datos, un aspecto fundamental en la investigación y el desarrollo en el campo de la informática y los sistemas de información.

4.6.2.1 Carga noSQL en Nifi

Para la carga noSQL en Nifi, se utilizó el procesador PutMongo. Este toma documentos JSON desde los flujos de datos de entrada de NiFi y realiza la operación especificada en la colección configurada dentro de sus propiedades.

Figura 29. Se muestra la vista del procesador PutMongo en el proceso Nifi.

A su vez, se necesita un conector MongoDB para que se puedan realizar las operaciones. De la misma forma que configuramos el conector para la inserción Nifi en SQL, creamos el conector 'MongoDBControllerService' dentro de las propiedades del procesador PutMongo.

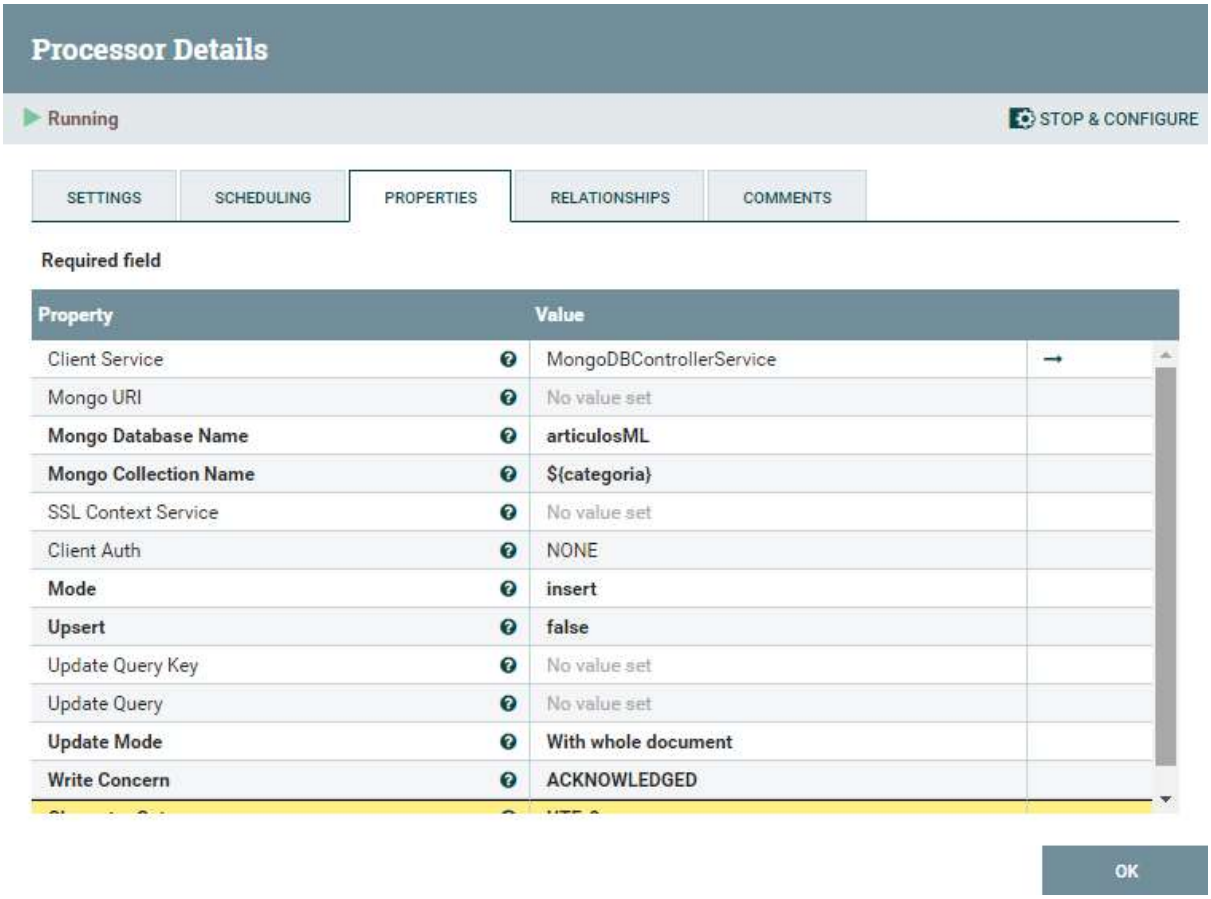

Figura 30. Se muestra la configuración del procesador PutSQL en el proceso Nifi. Se especificó el nombre de la BD, el nombre de la colección, en este caso parametrizada y el tipo de operación.

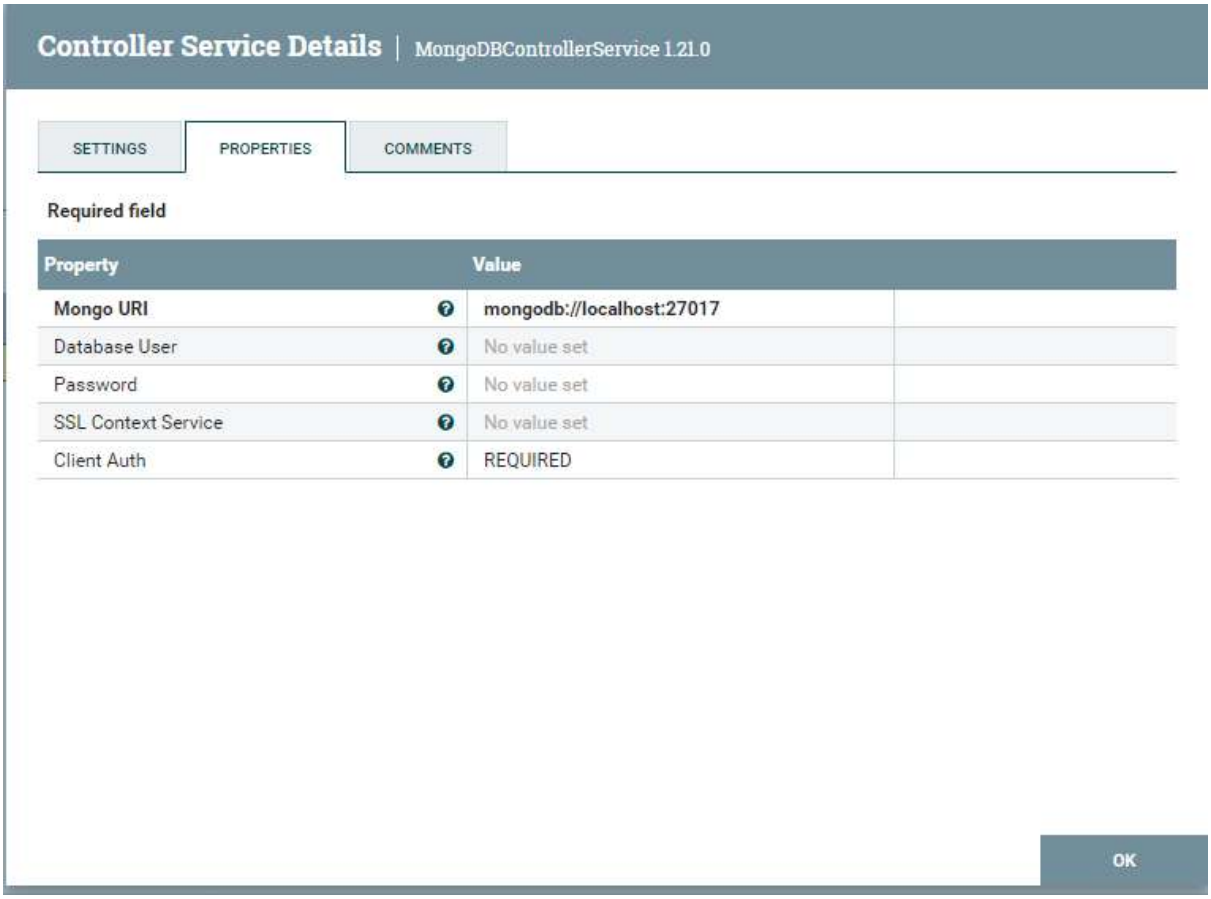

Figura 31. Se muestra la configuración del conector MongoDBControllerService definido en las configuraciones del procesador PutSQL en el proceso Nifi. Se especificó la URL de la BD

4.6.2.2 Carga noSQL en notebooks

La carga noSQL en notebooks ya fue descrita durante la fase de transformación, en la Figura 13. Mediante la librería pymongo se instanció la conexión a la base de datos utilizando el método MongoClient, que recibe de parámetro la URL. Esta se guardó en una variable, por la cual podemos acceder a la colección en forma de diccionario.

```
import pymongo

with open(r'C:\Users\Administrator\Desktop\scrapping-script\scrappingfiles\\'+str(datetime.now()).
          replace(" ", "").replace(":","").replace('-','').replace(".","")[0:8]+'.json') as json_file:
    json_to_py_dict_ex = json.load(json_file)

client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["articulosML"]
collection = db["electronica"]

merge_subcategorias = [item['data'] for item in  json_to_py_dict_ex if 'data' in item]
merge_items_subcategorias = [diccionario for lista in merge_subcategorias for diccionario in lista]

collection.insert_many(merge_items_subcategorias)
```

Figura 13.

Luego se realiza un merge de las listas de artículos para cada subcategoría (celulares, heladeras y televisores). Posteriormente con insert_many() se insertan en la base de datos MongoDB todos los ítems contenidos en esa estructura de datos.

4.7 Reporting

Recapitulando, reporting se refiere a la presentación de datos de una manera estructurada y narrativa. Es el proceso de recopilar, organizar y comunicar información sobre un conjunto de datos específico.

El objetivo principal del reporting es proporcionar información clara y comprensible al usuario. Los informes suelen ser estáticos y se generan periódicamente, como informes semanales, mensuales o trimestrales. Un informe típico incluye texto descriptivo, tablas, gráficos y resúmenes que explican los hallazgos clave y las tendencias en los datos.

Los avances tecnológicos, demandas cambiantes y una comprensión cada vez más profunda de la importancia de la efectividad en, y durante, la presentación de datos, han dado lugar al nacimiento del proceso de ‘data visualization’ o ‘data viz’, que a diferencia del reporting tradicional, introduce la interactividad con la data. Este concepto se convirtió en un pilar fundamental de la visualización de datos. Los usuarios pueden explorar datos, aplicar filtros, hacer clic en elementos de la visualización para obtener detalles y cambiar las perspectivas en tiempo real.

4.7.1 Reporting utilizando herramientas empresariales con GUI.

Se realizó un tablero en PowerBI, conectándose a la base de datos PostgreSQL, y con la API REST del BCRA para obtener el índice de inflación mensual desde 1990 hasta 2023. La herramienta se encarga de convertir el JSON a una estructura de tabla relacional. En la figura 32 se puede ver el esquema de datos que nos ofrece la herramienta.

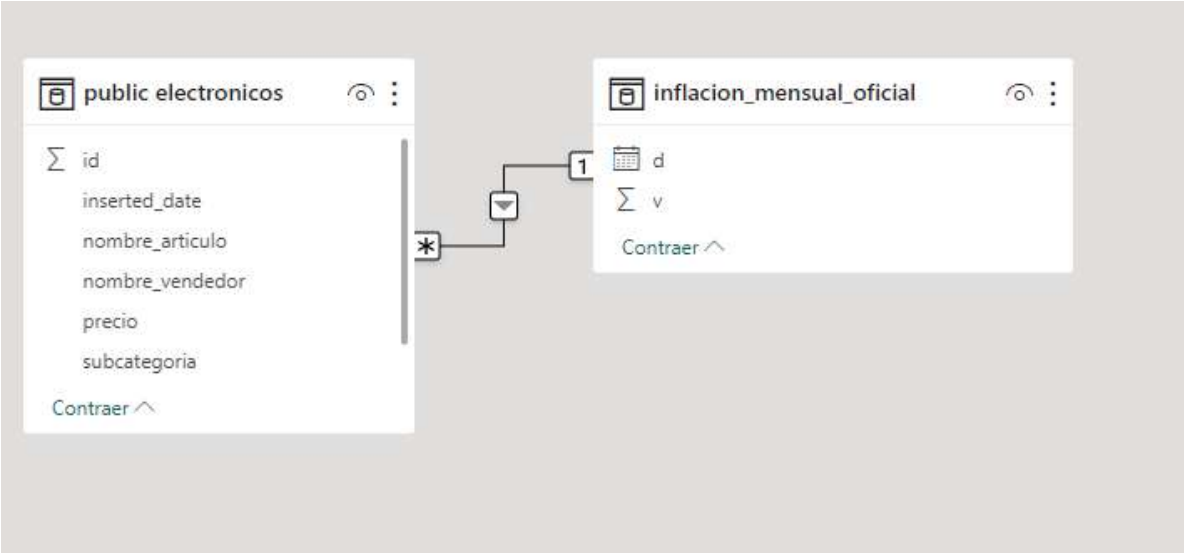

Figura 32. Se muestra el esquema automáticamente generado por powerBI relacionando las dos fuentes de datos.

Vemos además como pudimos definir una relación entre ambos conjuntos de datos, mediante la fecha de publicación de los artículos, junto con la fecha de emisión del comunicado de inflación mensual oficial (campo d). Esto simplemente arrastrando un campo hacia el otro, sin necesidad siquiera de definir una cardinalidad, ya que esta es detectada automáticamente En el caso de haber utilizado el modelo de data warehouse de copo de nieve, el proceso de relacionar los datos hubiera sido igual de simple, y no hubiésemos tenido que preocuparnos por la unión de los datos.

Cabe aclarar que para traer los datos almacenados en el proceso de carga hasta el entorno de reportes se eligió elegir como fuente de los mismos a la base PostgreSQL ante la otra base de datos en MongoDB debido a la disponibilidad de conectores por defecto para este DBMS que nos ofrece la herramienta de Microsoft.

En cambio para poder conectar MongoDB debemos primero instalar el driver ODBC correspondiente*[1]. En el mismo se configura su nombre y a que base de datos se conectará. Luego se puede testear la conexión para comprobar que el driver está correctamente conectado con la base. Posteriormente se puede seleccionar el conector de tipo ODBC en la interfaz de PowerBI, indicándole el nombre configurado anteriormente. Esto traerá la data de la misma forma que hizo con PostgreSQL.

Procedemos luego a realizar un tablero con los datos obtenidos durante el proceso. La Figura 33 muestra la comparación entre los índices de inflación tras los precios de los celulares más económicos de tres marcas.

[1] *https://www.devart.com/odbc/mongodb/

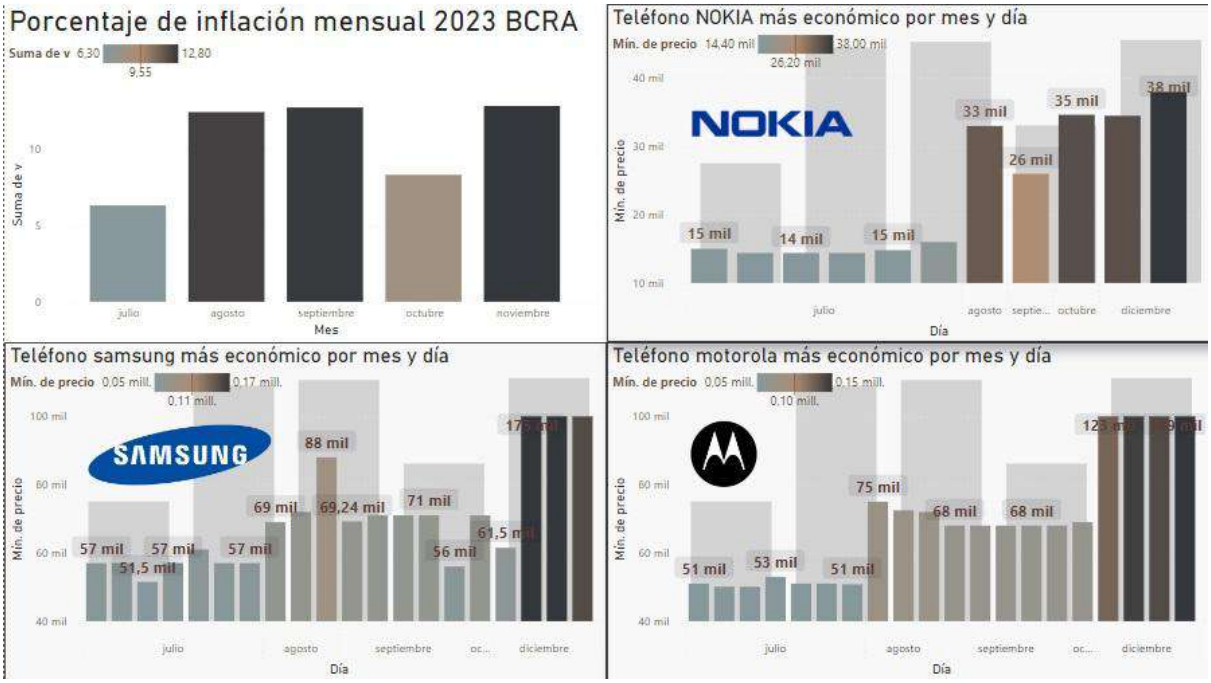

Figura 33. Se muestra en el primer cuadrante el porcentaje de inflación mensual desde julio hasta noviembre 2023. En el consiguiente, la comparación de este primer cuadrante frente al precio del dispositivo móvil más barato publicado en la plataforma perteneciente a Nokia, Samsung, y Motorola en ese orden.

Para realizar este tablero no se tuvo que escribir ninguna línea de código, simplemente se utilizó las opciones que ofrece la interfaz para la creación y customización de visualizaciones. La gama para elegir es amplia, además de que se puede potenciar utilizando lenguajes como DAX (Data Analysis Expression) que permiten personalizar y potenciar las capacidades de la herramienta. Por otra parte, también se puede combinar con Python y R.

Aún así, exceptuando estas últimas funcionalidades, un usuario sin grandes conocimientos en cuanto a bases de datos o programación se puede desenvolver fácilmente por la plataforma, incluso integrar sus trabajos con otras herramientas como Excel, para la actualización automática de los mismos.

Sin embargo, es esencial analizar esta situación con lupa. La fórmula no siempre demuestra calzar perfectamente en todas las organizaciones o proyectos, ya que en ciertas circunstancias no cumple con los requisitos de agilidad, flexibilidad y profundidad que se requieren. Power BI se presenta como una herramienta de fácil manejo con diseños sólidos y costos medianamente accesibles. A pesar de esto, para algunas organizaciones, estas ventajas pueden no satisfacer completamente todas las necesidades particulares, ya que por ejemplo se tiene una gran deficiencia en rendimiento, lo que es un golpe bajo si estamos trabajando en un entorno de Big Data, donde se necesita que millones de registros sean analizados de forma ágil y quizá con frecuentes actualizaciones. La herramienta guarda todos los datos exportados para utilizar en tableros en memoria. Si tenemos un dataset de 5 millones de registros, es

probable que, dependiendo de las especificaciones en donde se ejecute el programa, sea inusable. Por lo tanto, sería necesario antes un proceso embudo que sintetice los datos para luego reportarlos.

4.7.2 Reporting mediante notebooks utilizando librerías de código

Con librerías de código, se logra obtener la mayor capacidad de la personalización y precisión deseada de los informes y visualizaciones de datos. Esto permite crear informes altamente adaptados a las necesidades de la organización respecto a los datasets.

De todos modos utilizarlas para reporting puede requerir un nivel significativo de habilidades técnicas en programación y visualización de datos. La curva de aprendizaje es empinada para quienes no están familiarizados con el desarrollo de software en particular.

Junto con esto, existe una relación de personalización/tiempo. En primer lugar, para aprender los módulos que se están usando con el fin de sacarles el mayor provecho. En segundo lugar, dependiendo de las librerías y lenguajes que se utilicen, puede haber limitaciones en cuanto a las capacidades de visualización o generación de informes, y es posible que se necesiten combinar múltiples herramientas o módulos. En el caso del lenguaje que vamos a utilizar, Python, las librerías más utilizadas son las siguientes:

- Matplotlib: proporciona una amplia gama de herramientas para crear gráficos estáticos de alta calidad.
- Seaborn: basada en Matplotlib pero simplifica la creación de gráficos y rediseña su estética.
- Plotly: librería de visualización interactiva. Permite crear visualizaciones interactivas en línea y fuera de línea.
- Bokeh: se utiliza para crear visualizaciones interactivas y aplicaciones web.
- Altair: se basa en la gramática de Vega-Lite, que es una gramática desarrollada por la Universidad de Washington que permite crear gráficos de manera sencilla y eficaz

Se realizó el siguiente reporte utilizando librerías de código Python en notebooks Jupyter, utilizando pandas para la manipulación y agregación de los datos, junto con matplotlib para realizar los gráficos. Se obtuvieron los datos desde MongoDB mediante la librería pymongo, y mediante la función json_normalize() de pandas, la convertimos en dataframe. También podríamos haber realizado algo similar con PostgreSQL utilizando la librería psycopg2,

trayendo la tabla mediante un SELECT, que a su vez mediante la función pandas read_sql_query() devuelve los datos en un dataframe.

En la figura 34 se puede observar el gráfico realizado con un concepto similar al que se hizo en PowerBI, solo que para las tres categorías, se selecciona el artículo más barato publicado de cada mes, y se compara con la inflación obtenida por la API BCRA

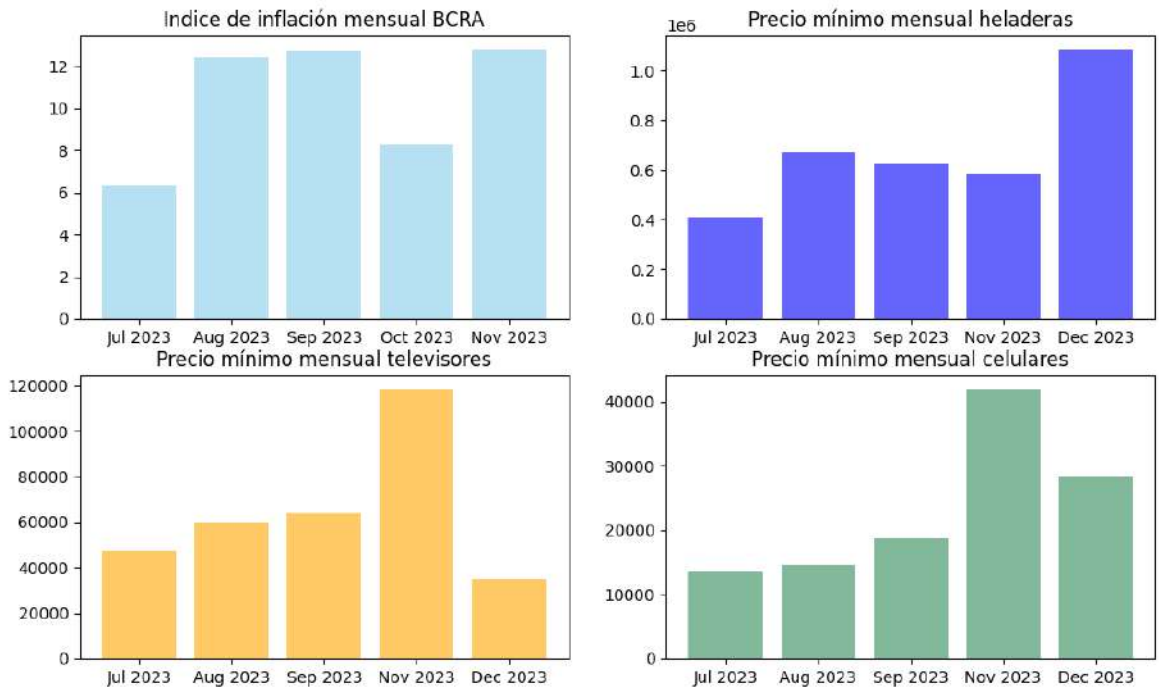

Figura 34. Se muestra el precio mínimo de heladeras, televisores y celulares desde Julio hasta Diciembre 2023, junto con la inflación del BCRA, la cual hasta ahora solo tiene datos hasta Noviembre 2023

Como se puede apreciar, este gráfico es más simple que el realizado en PowerBI, conceptual y visualmente. Esto se debe a que, a pesar de que hacer el reporte utilizando código nos da un mayor grado de libertad, requiere mucho más tiempo y esfuerzo. Manejar los conjuntos de datos, agregarlos, y diseñar las gráficas sin poder verlos directamente como se hizo en la herramienta drag and drop, dificulta la confección del mismo, incluso para una persona con cierto grado de conocimiento en programación.

5. Mediciones

La eficiencia es un factor determinante que moldea el rendimiento y la

viabilidad de los sistemas ETL y un modelo de costos es necesario para realizar recomendaciones sobre las herramientas [18], ya que inciden en la infraestructura y contexto de las organización. Se evaluará para este tipo de procesos desde múltiples dimensiones: computacional (uso de CPU, uso de memoria y tiempo de ejecución), económica y en términos de recursos humanos. Al abordar la complejidad inherente a la integración de datos, nos enfocaremos en evaluar cómo las diferentes herramientas y técnicas anteriormente presentadas, influyen en el rendimiento general de los procesos ETL.

Estas mediciones no son únicamente técnicas, sino que al interrelacionarse aportan una visión holística que abarca desde el consumo de recursos que provienen de la ejecución de las tareas hasta el impacto económico y la carga de trabajo sobre los recursos humanos.

Estas métricas interrelacionadas pueden ser fundamentales para tomar decisiones informadas en la selección y optimización de sistemas ETL.

5.1 Consumos de recursos de cómputo

Para determinar el consumo de recursos de cada herramienta utilizada, haremos uso de Windows Performance Analyzer (WPA) y de Windows Performance Recorder (WPR)[2] para que determine en porcentaje la repartición de capacidades de cómputo del CPU y memoria para Apache Nifi, PowerBI y Jupyter Notebooks.

Se realizó 4 veces una grabación del uso de cada recurso activando al proceso ETL en todas sus versiones (Es decir, con Nifi y con notebooks python, más PowerBI) utilizando WPR, que guardó 4 archivos .etl los cuales se analizaron con WPA, que muestra indicadores de uso de los recursos en forma de indicadores clave de performance (KPIs, (Key Performance Indicators)).

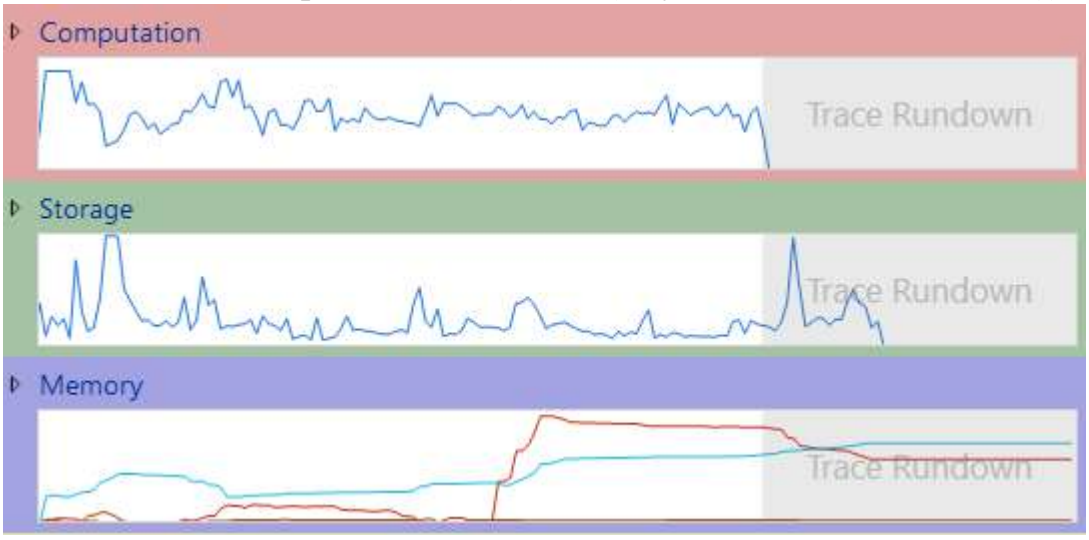

Figura 39. Se muestran gráficos del uso de los recursos de la computadora mediante la ejecución del

[2] learn.microsoft.com/en-us/windows-hardware/test/wpt/windows-performance-analyzer

proceso ETL.

Los gráficos de las figuras 40 y 41 muestran en su eje vertical, el porcentaje de uso de la CPU, mientras que el horizontal representa el tiempo transcurrido. En todas las grabaciones se obtuvo un resultado similar, por lo que procedemos a mostrar las gráficas provenientes de la primera ejecución.

5.1.1 Consumo de recursos de cómputo para ETL con Apache Nifi y Python

Una vez iniciada la grabación con WPR, se ejecutó el pipeline basado en código, y luego se disparó manualmente el programado con Apache Nifi. El gráfico de uso de CPU que obtenemos está en la Figura 40, y representando la línea violeta a los procesos asociados a la notebook, mientras que las naranjas a los asociados con Apache Nifi:

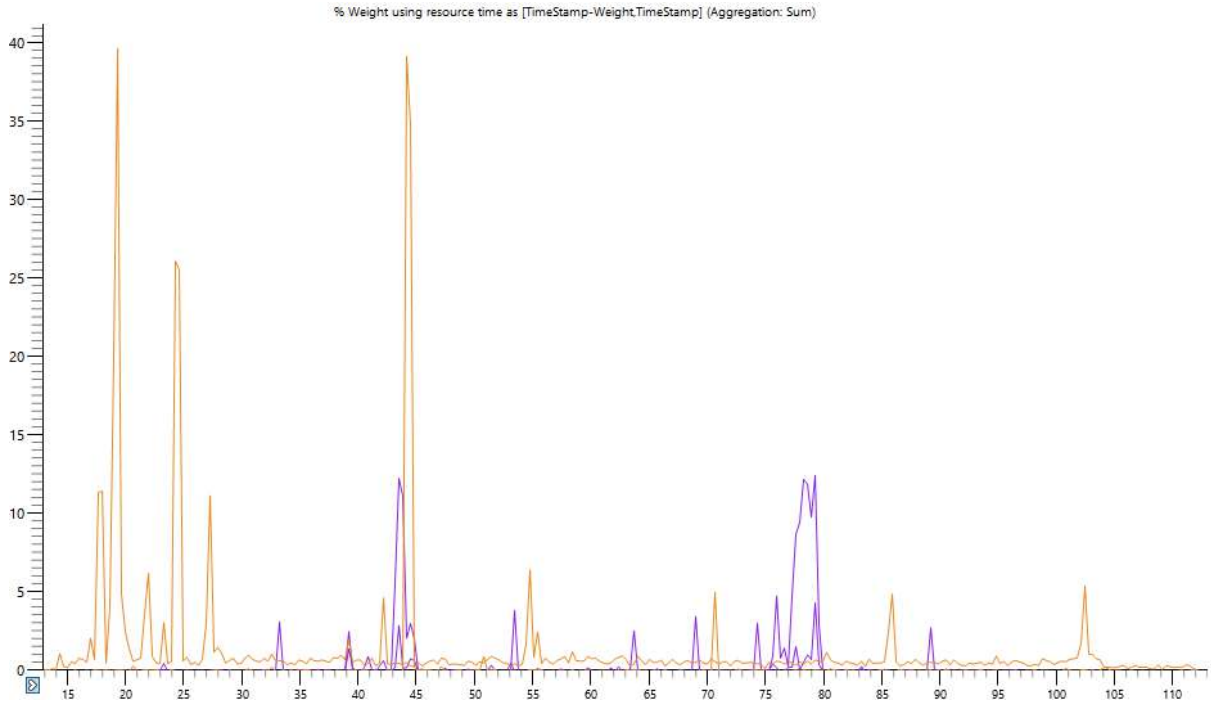

Figura 40. Se muestra el gráfico de porcentaje de uso de la CPU para Python y para Nifi, al ejecutar el flujo ETL.

Se puede apreciar que el servidor Nifi estuvo en ejecución toda la corrida, al igual que el servidor Jupyter Notebook, que luego de terminar la ejecución de la notebook, su línea de consumo no se llega a apreciar en el gráfico debido a que tiene muy poco uso de CPU, a diferencia del de Apache, que a razón de tener más funcionalidades, y por su naturaleza de aplicación drag n drop, es más pesado.

Vemos una notable diferencia en cuanto a sus picos máximos.

Vemos que el pico máximo para el proceso correspondiente a la notebook fue de un 13% de uso de la CPU. En cambio, para los procesos referentes a la

ejecución de Apache Nifi, fueron de casi un %40, con otros pico menor del %25 de uso.

Esta marcada disparidad de consumos también se presentan para el uso de memoria, donde también se ve una clara diferencia entre las dos herramientas. En el siguiente gráfico correspondiente a la Figura 41 se ve en amarillo el uso de memoria producido por Apache Nifi, mientras que en violeta el uso de memoria producido por Python y Jupyter Notebooks:

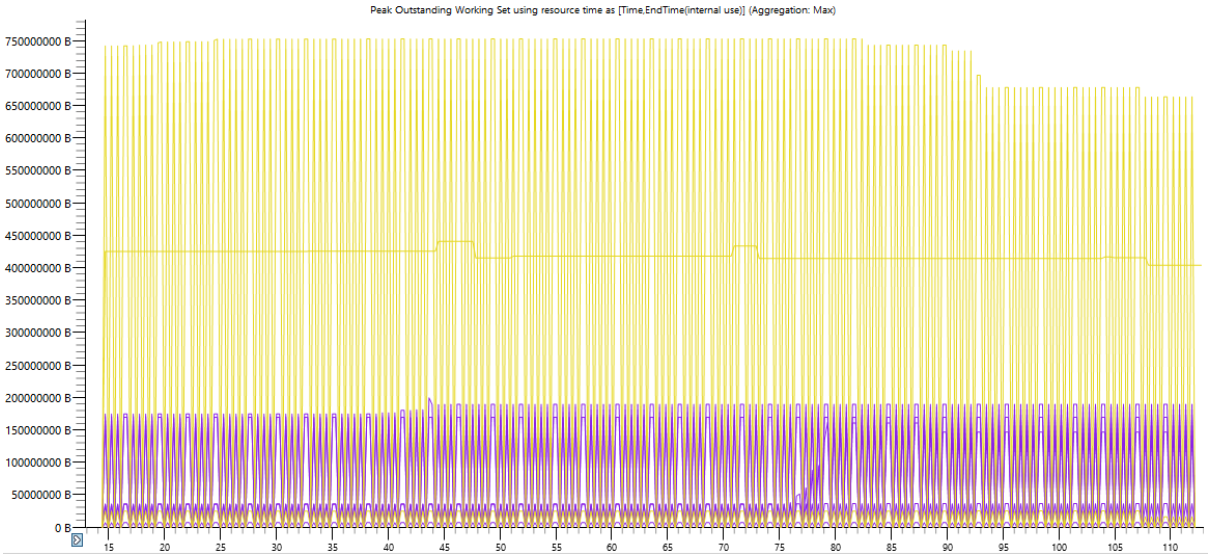

Figura 41. Se muestra el gráfico de porcentaje de uso de memoria para Python y para Nifi, al ejecutar el flujo ETL.

Vemos en cuanto al uso de memoria, que la herramienta drag n drop es casi 5 veces más exigente en cuanto al recurso que su contraparte en código. Tenemos tres procesos en amarillo correspondientes a Apache Nifi, que sumados representan aproximadamente un uso constante de 1GB de memoria. Por otro lado, los procesos violetas sumados, usan aproximadamente 160 MB.

En cuanto a tiempo, el procesamiento en Nifi de un flowfile de principio a fin, es de aproximadamente 8.7 segundos. Obtenemos esta información accediendo a la metadata de uno de ellos:

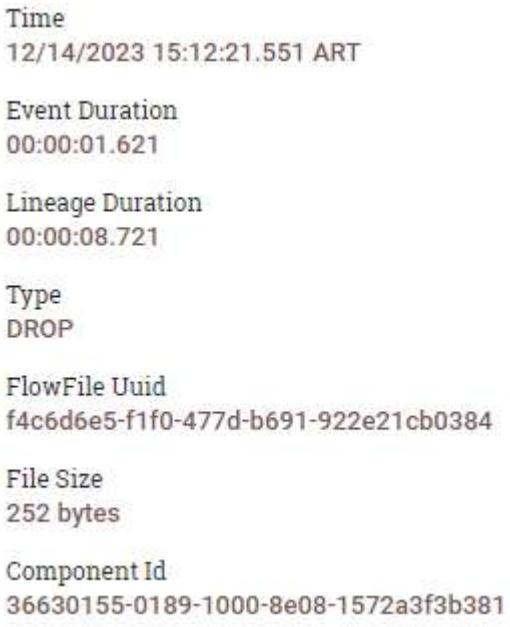

Figura 42. Se ven algunos de los atributos de la metadata de un flowfile en particular.

El atributo 'Lineage Duration' nos indica la duración de un mensaje desde su nacimiento hasta su fin.

Podemos simular lo mismo en nuestra notebook de Jupyter. Para esto se guardó el tiempo de inicio de la ejecución de la notebook en una variable llamada 'inicio', mediante la librería datetime y su método now() que nos devuelve el timestamp actual para el momento dado. Al final de la corrida, se imprime la resta del timestamp actual, frente a la variable 'inicio' definida al comienzo. Nos da un resultado similar, de aproximadamente 5.6 segundos:

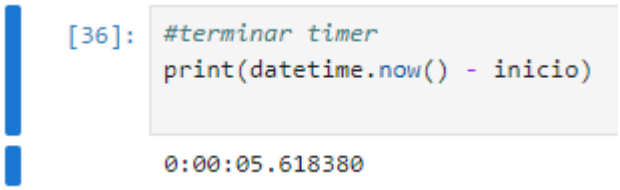

Figura 43. Se muestra el tiempo que tarda en correr la notebook referente al proceso de ETL.

5.1.2 Consumo de recursos de cómputo para herramientas de reporting PowerBI y Python

Luego de ejecutar el proceso de ETL, decidimos iniciar las herramientas de reporting, primero ejecutando PowerBI con el reporte confeccionado, y luego ejecutando la notebook python que realiza el reporting. Se obtuvo el siguiente gráfico de consumo expresado en la Figura 44, donde la línea verde representa a los procesos asociado a PowerBI, mientras que la línea violeta representa los procesos asociados a Python:

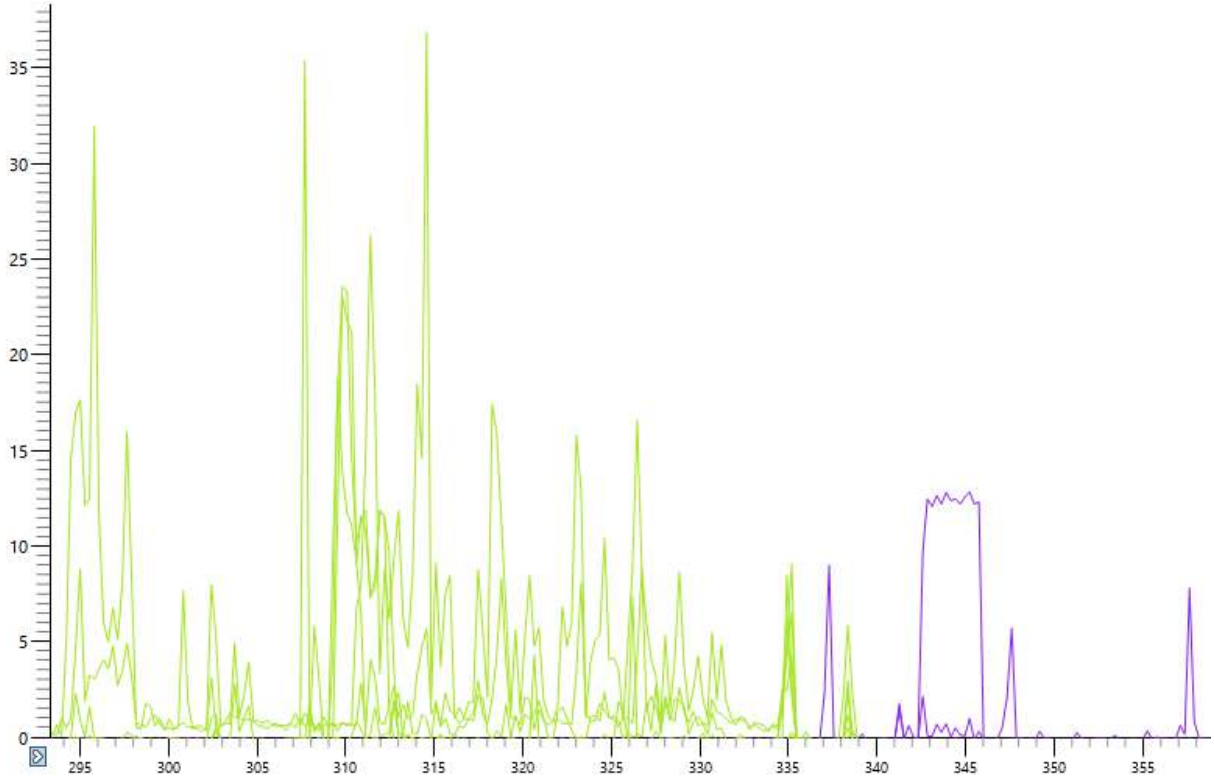

Figura 44. Se muestra el gráfico de porcentaje de uso de la CPU para PowerBI y para Python, al ejecutar ambas herramientas de reporting.

Se puede observar una clara diferencia de uso de CPU. Microsoft PowerBI consume en general siempre más capacidad de cómputo que su contraparte. A su vez, se muestran picos al interactuar con el reporte. Por un lado los picos de PowerBI representan la interacción del usuario con el mismo reporte, que clickea los gráficos para acceder a más detalle. En cambio, los picos en Python representan la ejecución de las celdas en la notebook.

El reporte realizado con código Python en notebooks parece ser un proceso mucho más liviano, requiriendo capacidad de cómputo únicamente al ejecutarse el script, debido a que no hay interacción posterior.

El pico máximo de powerBI llegó a consumir casi un 35% de la capacidad de cómputo otorgada para ese instante. Mientras que el pico máximo para el reporting en Python requirió 13% de la capacidad total entregada para el instante dado.

El uso de memoria puede verse en la figura 45, donde el verde nuevamente representa a los procesos asociados a PowerBI, mientras que el violeta representa los procesos asociados a Python y Jupyter Notebooks:

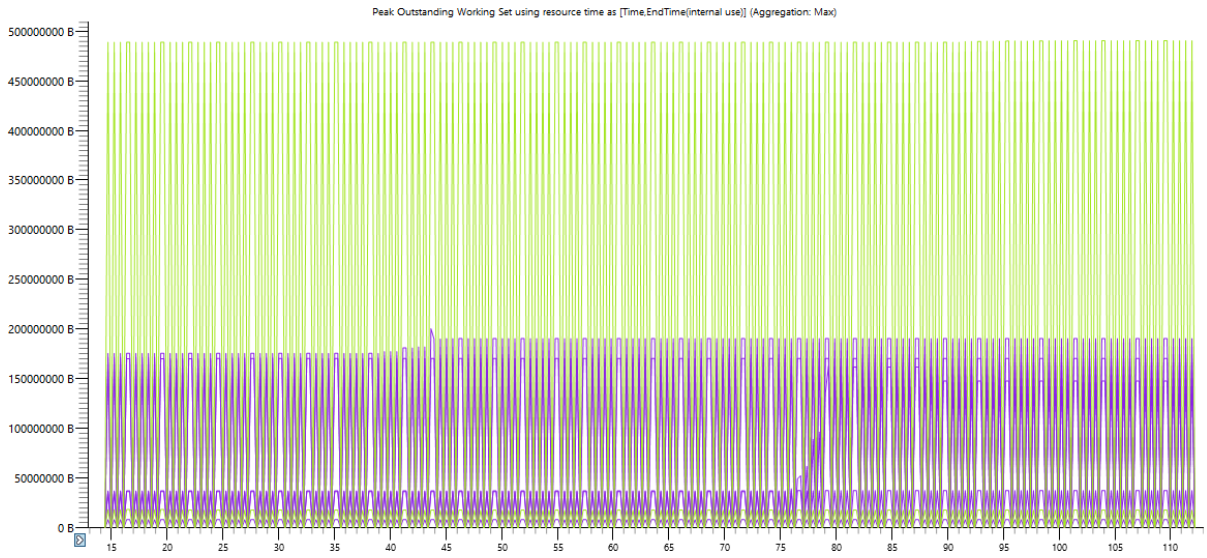

Figura 45. Se muestra el gráfico de porcentaje de uso de memoria para PowerBI y para Python, al ejecutar ambas herramientas de reporting. PowerBI tiene procesos con picos de alrededor de 500 megabytes, mientras que los procesos con pico máximo referentes a Python son de 200 megabytes.

Se puede notar que PowerBI Desktop entre todos sus procesos consume casi 3 veces más memoria que los procesos Python. Sumando el promedio de uso de memoria de cada proceso, tenemos que la herramienta drag n drop consume alrededor de 1.2GB de memoria. Por otro lado, Python consume en promedio con todos sus procesos, 484 MB. Nuevamente vemos que la herramienta de Microsoft es mucho más exigente, tanto en memoria como en CPU. Esto es algo a tener en cuenta a la hora de elegir la herramienta, ya que la relación entre facilidad de uso y consumo parecen ser aspectos correlativos, al tener interfaces más pesadas y gráficas más complejas.

Cabe aclarar que la medición se realizó mediante snapshots virtuales de memoria. Por lo que, para un instante dado, se mide el uso de la memoria de un programa. Es por eso que tenemos la gráfica de Python con picos constantes, en lugar de ser una línea recta.

En cuanto al tiempo se realizaron mediciones diferentes para ambas técnicas. Para el reporting en notebooks de código, se tomó el tiempo y se determinó que todo el proceso de reporte tarda 2 segundos.

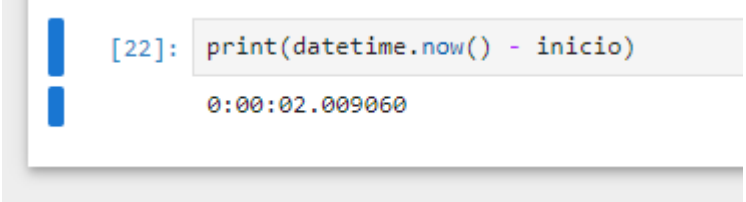

Figura 46. Se muestra el tiempo total de ejecución del reporte en código.

En cuanto al reporte en PowerBI, se midió el tiempo con cronómetro manualmente en inicializar el reporte ya guardado y dió como resultado 1:14

minutos. Luego, se actualizaron los datos de los productos provenientes de la base de datos PostgreSQL. Este procedimiento tardó 14 segundos.

Por lo tanto, se ve una clara ventaja en cuanto a tiempo para los reportes de código. De todos modos este factor para esta etapa del proceso no es tan crucial, ya que los reportes son la etapa final del proceso y más que eficiencia en los mismos se busca que sean claros y cumplan el propósito de informar.

5.2 Costos en cloud:

Antes de la era del Big Data, las herramientas tradicionales de ETL estaban diseñadas para procesar fuentes provenientes de sistemas de información internos o proveedores externos en volúmenes de datos razonables.

Ahora, hay varios procesos en los cuales los volúmenes de datos son enormes, y este caudal no es adecuado para el enfoque clásico de ETL, requiriendo implementar tecnologías cloud [19].

Para calcular los costos que podría llegar a consumir el proceso ETL en cloud, utilizaremos Azure Pricing Calculator[3]. Esta herramienta permite calcular el costo por período de tiempo y uso de recursos computacionales de las diferentes herramientas necesarias para poder llevar nuestro sistema a cloud.

Si vamos a trabajar con procesos de ETL basados en pipelines gráficos, junto con powerBI y cuentas de almacenamiento para crear nuestras carpetas de staging o landing, Microsoft ofrece la posibilidad de optar por el paquete de productos 'Enterprise data warehouse', el cual se ilustra en la Figura 47.

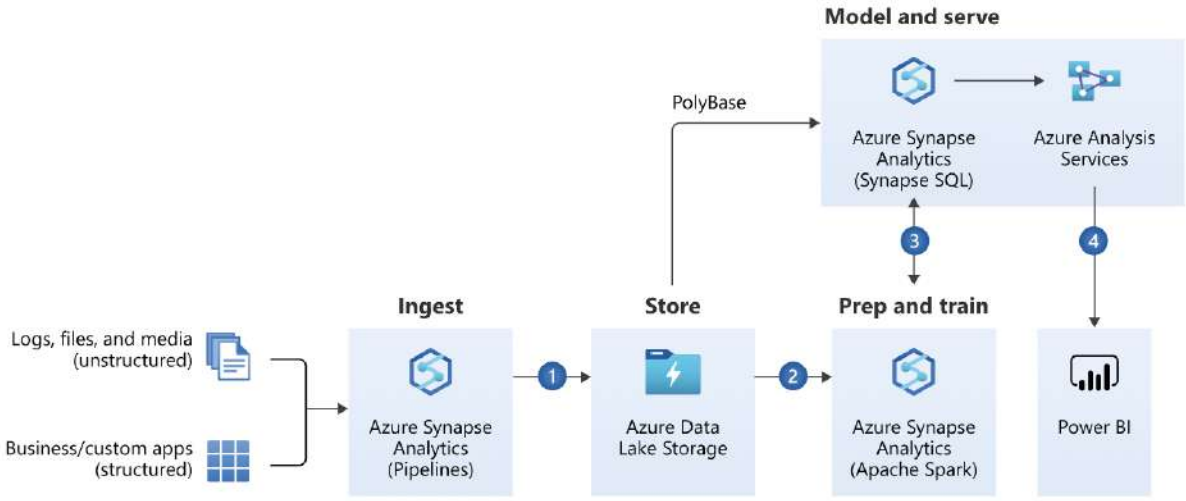

Figura 47. Se muestra el esquema del paquete de productos, relacionando los mismos.

[3] azure.microsoft.com/en-us/pricing/calculator

Este producto ejemplifica un modelo de sistema que utiliza en un principio Azure Synapse Analytics, en donde se moldean los pipelines utilizando Data Factory, que es una herramienta similar a Apache Nifi. Luego se almacenan los datos generados en Azure Data Lake Storage, y de ahí se derivan a Synapse Analytics para realizar reportes en código, o a PowerBI para crear los tableros. La calculadora provee el costo total del paquete, junto con el costo individual de cada producto mensualmente:

Microsoft Azure Estimate

Enterprise Data Warehouse

Service category	Service type	Region	Description	Estimated monthly cost	Estimated upfront cost
Databases	Azure Synapse Analytics	East US 2	Tier: Compute Optimized Gen2, Dedicated SQL Pools: DWU 100 x 1 Month, 1 TB of storage with Geo-redundant disaster recovery; East US 2 Region, 100 GB of data collected per day, 7 days of Hot Cache, 30 days of total retention, 7 times estimated data compression, 730 Hours of 2 x Extra Small (2 vCores) Engine Instances, 730 Hours of 2 x 1 vCore Data Management Instances	$1.755,90	$0,00
Analytics	Azure Analysis Services	East US 2	Developer (Hours), 5 Instance(s), 720 Hours	$475,20	$0,00
Analytics	Power BI Embedded	East US 2	1 node(s) x 1 Month, Node type: A1, 1 Virtual Core(s), 3GB RAM, 1-300 Peak renders/hour	$735,91	$0,00
Storage	Storage Accounts	East US	Data Lake Storage Gen2, Standard, LRS Redundancy, Hot Access Tier, Flat Namespace File Structure, 1,000 GB Capacity - Pay as you go, Write operations: 4 MB x 100,000 operations, Read operations: 4 MB x 100,000 operations, 100,000 Iterative read operations, 100,000 Archive High Priority Read, 100,000 Iterative write operations, 100,000 Other operations. 1,000 GB Data Retrieval, 1,000 GB Archive High Priority Retrieval, 1,000 GB Data Write	$15.820,80	$0,00
Support		Support		$0,00	$0,00
		Licensing Program	Microsoft Customer Agreement (MCA)		
		Billing Account			
		Billing Profile			
		Total		$18.787,81	$0,00

Disclaimer

All prices shown are in United States – Dollar ($) USD. This is a summary estimate, not a quote. For up to date pricing information please visit https://azure.microsoft.com/pricing/calculator/

This estimate was created at 11/23/2023 7:10:12 PM UTC.

Figura 48. Se muestra una tabla de costos por el paquete con la descripción detallada de cada producto.

Por contraparte, si se necesita un sistema más simple y por lo tanto económico, se puede optar por combinar Azure Functions junto con Storage Accounts.

Azure Functions es un servicio de cómputo sin servidor (serverless). Permite ejecutar funciones individuales o piezas de código en la nube sin preocuparse por la infraestructura que le subyace. Azure se encarga de la administración y escalabilidad automática de los recursos de cómputo (capacidad de cómputo elástica). Esto quiere decir que tiene ventajas en cuanto a costos, ya que escala automáticamente el número de instancias en ejecución según la carga de trabajo. Las funciones se ejecutan en respuesta a eventos como cambios en datos almacenados en Storage Accounts, mensajes en colas de Azure Queue para procesos real time, entre otros; y solo consumen recursos

cuando están en ejecución, lo que puede ser más rentable en comparación con los modelos tradicionales de cómputo. A su vez estas pueden escribirse en C#, F#, Node.js, Python, y Java.

En general, es posible construir arquitecturas basadas en eventos y responder dinámicamente a cambios en la aplicación tal como fue resuelto con la opción de código.

Es importante recalcar, que además el primer millón de ejecuciones y 400 Terabytes procesados utilizando esta herramienta son gratuitos. Por lo tanto, teniendo en cuenta el proceso de ETL y reporting en código que se desarrolló, el cual se puede agrupar en 18 funciones (si tomamos una celda como una función), con un promedio de medio segundo cada una, las cuales se ejecutan semanalmente, este producto para este escenario particular nos resultaría gratuito.

De todos modos, y arbitrariamente, calculamos los costos de Azure Functions para un exagerado escenario de 10 millones de ejecuciones mensuales, cada una de 1000 milisegundos:

Microsoft Azure Estimate

Your Estimate

Service category	Service type	Region	Description	Estimated monthly cost	Estimated upfront cost
Compute	Azure Functions	West US	Consumption tier, Pay as you go, 128 MB memory, 100 milliseconds execution time, 0 executions/mo	$0,00	$0,00
Storage	Storage Accounts	East US	Block Blob Storage, General Purpose V2, Hierarchical Namespace, LRS Redundancy, Hot Access Tier, 1,000 GB Capacity - Pay as you go, 10 x 10,000 Write operations, 10 x 10,000 Read operations, 10 x 10,000 Iterative Read operations, 10 x 100 Iterative Write operations, 1,000 GB Data Retrieval, 1,000 GB Data Write, 1,000 GB Index, 1 x 10,000 Other operations	$52,41	$0,00
Compute	Azure Functions	West US	Consumption tier, Pay as you go, 128 MB memory, 1,000 milliseconds execution time, 10,000,000 executions/mo	$15,40	$0,00
Support		Support		$0,00	$0,00
		Licensing Program	Microsoft Customer Agreement (MCA)		
		Billing Account			
		Billing Profile			
		Total		$67,81	$0,00

Disclaimer

All prices shown are in United States – Dollar ($) USD. This is a summary estimate, not a quote. For up to date pricing information please visit https://azure.microsoft.com/pricing/calculator/
This estimate was created at 11/24/2023 2:28:26 PM UTC.

Figura 49. Se muestran los costos de realizar 10 millones de ejecuciones mensuales en Azure Functions.

Lo que nos lleva a un total de 67 USD, siendo 15.40 USD pertenecientes a Azure Functions, y el resto perteneciente al producto Azure Storage Accounts, que es un servicio de almacenamiento de datos en la nube. Es necesario el espacio de almacenamiento para las tablas o carpetas de staging, así como para los reportes en código.

Posee componentes tales como blobs, que son objetos binarios grandes (archivos, imágenes, etc), tablas para guardar datos estructurados y administradores de discos para máquinas virtuales.

Así, es posible ver que utilizar herramientas con GUI en un entorno de

Big Data y en cloud, en general es mucho más costoso que realizar el sistema únicamente con código, teniendo en cuenta que la primera solución estaba por encima de los 18.000 USD mensuales, mientras que la segunda apenas 67 USD mensuales, que de todos modos pueden escalar dependiendo de la intensidad del uso de las funciones o las necesidades de agrandar los espacios de almacenamiento.

Por último, hay que aportar también la última visión, y es la de los recursos organizacionales, y cómo estos pueden influir en la balanza de costos y en las decisiones a la hora de crear el sistema.

5.2 Recursos organizacionales:

Uno de los desafíos más grandes de la implementación de un sistema en una organización está representado por la complejidad de la administración de recursos humanos [17]. Programar de manera tradicional tiene una pronunciada curva de aprendizaje que requiere no solamente aprender un lenguaje de programación, sino que además se necesitan otras habilidades sustanciales, como la comprensión de algoritmos y estructuras de datos o debuggeo [13].

El no coding o programación drag n drop es una forma de programar sin utilizar códigos o scripts o conjuntos de comandos. Las personas aprenden fácilmente a través de indicadores visuales, y deben ocuparse en un principio únicamente por comprender cómo funciona la plataforma de desarrollo, las funcionalidades que ofrece y las limitaciones que puede presentar.

A pesar de tener una curva de aprendizaje más aplanada, esta forma de construir sistemas posee marcadas limitaciones. Una de las más grandes, es que dependiendo de la herramienta a utilizar, se puede no ser dueño del código fuente, por lo que se puede generar una dependencia absoluta hacia el proveedor.

Por ejemplo, al plantear una solución ETL en Azure, con el objetivo de escalarla en un sentido de mejorar su eficiencia para entornos Big Data, se genera una fuerte dependencia a un equipo de desarrolladores y administradores ajeno a nuestra organización. A su vez, las funcionalidades se encuentran limitadas a lo que la herramienta permite, y lo que no se permite no tiene forma de realizarse con la misma.

Por otro lado, las plataformas sin código contribuyen a aumentar la cantidad de desarrolladores dado un momento de creciente demanda de mano de obra dedicada al software. Empresarios, pequeñas y medianas empresas encuentran beneficios al disminuir los tiempos de gestión, automatizar procesos, y explorar nuevas líneas de negocio sin sobrecargar a los desarrolladores. Este

enfoque también resulta valioso para product owners, ya que permite centrarse en la creación y diseño de un producto, en lugar de sus aspectos técnicos.

Los costos y la conveniencia de utilizar herramientas drag n drop o de código varían significativamente según el tipo de organización y el sistema que se esté construyendo. Una gran empresa a menudo puede invertir en herramientas más costosas y personalizadas debido a sus mayores presupuestos. Así es que pueden priorizar soluciones más robustas y escalables, incluso si implican una curva de aprendizaje más pronunciada, como conlleva utilizar únicamente código.

Las PYMEs generalmente buscan soluciones más asequibles y pueden favorecer herramientas que ofrezcan un buen equilibrio entre costo y funcionalidad. Las herramientas fáciles de implementar y utilizar son preferidas, ya que se pueden tener equipos más pequeños y a su vez asignar menos recursos a la capacitación del personal disponible.

Por otro lado, en cuanto a herramientas ‘no code’, la facilidad de implementación propicia a la rapidez de la adición de nuevas funcionalidades a los sistemas de ETL, pero a su vez pueden haber limitaciones en cuanto a escalabilidad debido a la simplicidad intrínseca de las herramientas y dependencia del proveedor. La programación basada en código permite mayor control y optimización para escalabilidad. Por ejemplo, mediante código Python podemos realizar balanceos de carga para nuestros sistemas. Sin embargo, esto es mucho más complicado de replicar en Nifi, quitándole la ventaja de la simplicidad por la cual se suele elegir la herramienta. Otras consideraciones incluyen la complejidad del proyecto, velocidad de desarrollo, y evolución a largo plazo. La elección depende de requisitos, habilidades del equipo, presupuesto, y visión para la aplicación. En algunos casos, combinar ambos enfoques puede ser beneficioso, como es el caso de Azure Data Factory, en el cual sus pipelines de datos combinan tareas predefinidas, junto con la posibilidad de integrar notebooks de código en el flujo, a diferencia de Nifi.

En el contexto de Big Data, estas consideraciones adquieren una dimensión aún más relevante. La gestión y procesamiento de grandes volúmenes de datos requiere soluciones escalables y eficientes, donde le damos la derecha a la programación tradicional. Por otro lado, las herramientas 'no code', al simplificar la implementación, pueden ser atractivas para proyectos más pequeños, pero pueden enfrentar desafíos significativos al escalar a entornos Big Data.

6. Conclusiones finales

A lo largo de la investigación, se examinaron las implicaciones de la incorporación de tecnologías ETL en procesos de ejecución periódica por lotes, destacando tanto los desafíos como los beneficios experimentados para este tipo

de proceso, para el cual además, se llevaron a cabo implementaciones que simularon un entorno organizacional específico, permitiendo explorar teórica y prácticamente cómo los diferentes tipos de herramientas podrían contribuir a la optimización de los procesos de datos en contextos simulados.

Estos escenarios ficticios proporcionaron una evaluación precisa de las posibles ventajas y desafíos asociados con la adopción de tecnologías ETL avanzadas. El hincapié en utilizar herramientas visuales frente a construir un sistema únicamente con código generó un cuestionamiento para el cual se enunciarán ventajas y desventajas de cada enfoque.

Para entornos drag n drop se asoció una mayor velocidad de desarrollo, facilitando la participación de usuarios no técnicos. Sin embargo, esto puede venir a expensas de la personalización y optimización precisas que ofrecen los enfoques basados en código, donde los desarrolladores tienen un control más granular sobre cada aspecto del proceso ETL. Se concluyó que la elección entre herramientas visuales y desarrollo basado en código debe basarse en las necesidades y habilidades específicas de la organización. Las herramientas visuales son ideales para implementaciones rápidas y situaciones donde la participación de usuarios no técnicos es crucial. Por otro lado, el desarrollo basado en código puede ser preferible cuando se busca un control total sobre la personalización, la optimización del rendimiento y la adaptabilidad a requisitos complejos y cambiantes.

En el contexto de Big Data, el impacto de la elección entre SQL y NoSQL para un determinado proceso se amplifica, ya que las demandas de velocidad, escalabilidad y flexibilidad son aún más pronunciadas. Así como en entornos de desarrollo tradicionales, la comparación entre estas tecnologías se asemeja a la dicotomía entre herramientas visuales y desarrollo basado en código, con matices particulares que influyen en la gestión de grandes volúmenes de datos.

Las bases de datos NoSQL, con su capacidad para manejar datos no estructurados y su escalabilidad horizontal, encuentran su nicho en entornos de Big Data. Al igual que las interfaces de arrastrar y soltar, proporcionan una respuesta ágil en cuanto a implementación. Sin embargo, difiere en cuanto a la facilidad de manejar el volumen de datos característico de este contexto. La flexibilidad en el modelo de datos de NoSQL permite adaptarse fácilmente a cambios en la estructura y tipos de datos, facilitando la ingesta rápida de grandes volúmenes de información en escenarios donde la velocidad de procesamiento es esencial. No obstante, la simplicidad inherente a las bases de datos NoSQL puede presentar desafíos en términos de consistencia y control (excluyendo el caso de MongoDB que sí puede cumplir con las propiedades ACID), especialmente cuando se trata de gestionar grandes conjuntos de datos. En el contexto de Big Data, donde la integridad de los datos y la capacidad de realizar análisis complejos son cruciales, las bases de datos SQL ofrecen una estructura más rígida y relaciones definidas, brindando mayor garantía de

coherencia y precisión en los resultados.

Este balance también se manifiesta de manera notable en el contexto del reporting, especialmente al comparar Power BI con notebooks de código como Jupyter Notebooks. Al igual que en el desarrollo y en la gestión de bases de datos, la elección entre estas herramientas en el ámbito del reporting refleja nuevamente la tensión entre las variables de facilidad de uso y flexibilidad.

PowerBI, al igual que las herramientas de arrastrar y soltar, proporciona una experiencia visual e intuitiva para la creación de informes. Su interfaz gráfica permite a los usuarios construir dashboards y visualizaciones de datos de manera rápida y sin necesidad de programación extensiva. Es una opción atractiva para aquellos que buscan presentar datos de manera efectiva sin sumergirse en detalles de codificación.

En contraste, los notebooks de código ofrecen un nivel de control y personalización más profundo. Al utilizar lenguajes de programación como Python, se puede escribir código para generar visualizaciones altamente personalizadas y realizar análisis más avanzados. Esto proporciona una flexibilidad considerable a expensas de una empinada curva de aprendizaje y la carencia de simplicidad que ofrecen las herramientas de drag n drop. Aún cuando un recurso sepa programar, de todos modos, le llevaría tiempo adaptarse a una librería de visualización de datos como las usadas en este trabajo de investigación, para poder llegar al expertise de replicar un reporte realizado en una herramienta visual, teniendo lo mejor de los dos mundos: eficiencia de código y un reporte vistoso.

La elección entre Power BI o entornos del estilo, y notebooks de código para reporting dependerá de las necesidades y habilidades específicas del recurso. Power BI es una opción para aquellos que buscan una solución rápida y visualmente atractiva sin la necesidad de conocimientos profundos en programación. Por otro lado, los notebooks de código son preferibles cuando se requiere un control total sobre la visualización y se valora la capacidad de realizar análisis complejos y personalizados, o en otro extremo, cuando el gráfico o reporte a realizar es demasiado simple como para involucrar una herramienta semejante a su contraparte. La decisión se centra nuevamente en encontrar el equilibrio adecuado entre la accesibilidad, flexibilidad y simplicidad según los requisitos específicos de reporting.

A continuación se presentan cuadros comparativos a modo de síntesis, comparando herramientas de ETL drag n drop versus código, SQL versus noSQL, y por último, reporting drag n drop versus reporting en código.

ETL	No code o ‘drag n drop’	Basadas en código
Implementación	Permite a personas no técnicas construir aplicaciones y flujos de trabajo rápidamente sin necesidad de conocimientos profundos de programación. La simplicidad puede llevar a limitaciones en la personalización y complejidad de las soluciones.	Se necesita más conocimiento para construir un flujo de datos, tiene una curva de aprendizaje más empinada. Sin embargo la personalización es ilimitada, lo que lleva a ser útil cuando las soluciones requeridas son complejas.
Escalabilidad	Puede enfrentar desafíos significativos al escalar a sistemas complejos o grandes volúmenes de datos.	Optimización personalizada para escalabilidad y rendimiento eficiente, especialmente en entornos Big Data.
Costos	Los costos iniciales en recursos humanos pueden ser menores. Por otra parte los SaaS suelen ser más costosos	Puede requerir mayores inversiones iniciales en términos de tiempo y recursos humanos.
Mantenimiento	En sistemas grandes el mantenimiento se torna complejo.	Permite un ajuste fino y una adaptación continua.
Integración	Hay que evaluar la dificultad de integración del producto con los demás sistemas existentes	Facilidad de integración con los demás sistemas de la organización

Figura 50. Cuadro comparativo entre tecnologías code y no code para procesos de ETL

Bases de datos	noSQL	SQL
Distribución	Facilita distribuir grandes cantidades de información.	Facilita distribuir las bases de datos relacionadas.
Escalabilidad	Fácil escalabilidad horizontal	Escalabilidad horizontal limitada
Consultas	La falta de un lenguaje de consulta estándar puede dificultar la interoperabilidad	Lenguaje estandarizado para cualquier base SQL.
Estructura y consistencia	Admite datos no estructurados o con esquemas flexibles pero puede haber menos garantía de integridad y consistencia.	Garantiza integridad y consistencia de datos con esquemas predefinidos pero puede ser menos flexible para datos no estructurados o con cambios frecuentes.

Figura 51. Cuadro comparativo entre tecnologías SQL y noSQL

Herramientas de reporting	Visuales	Código
Desarrollo	Permite a usuarios no técnicos crear informes visuales de manera rápida y sencilla, pero puede tener limitaciones en la personalización avanzada dependiendo de la herramienta.	Permite un control detallado sobre la visualización y personalización avanzada pero requiere habilidades de programación y también de diseño.
Licenciamiento y costos asociados	Pueden existir costos asociados con licencias y capacidades premium.	Las librerías y módulos son de uso gratuito.

Integración	Hay que evaluar las posibilidades de integración del reporte con los procesos y sistemas pre existentes en la organización.	Mayor flexibilidad para integrar informes con otros procesos y sistemas.

Figura 52. Cuadro comparativo entre tecnologías code y no code para procesos de reporting.

En conclusión, la elección entre herramientas de código y herramientas drag and drop no debe ser excluyente. Un escenario híbrido, donde se integran elementos de ambas metodologías, puede ofrecer lo mejor de ambos mundos.

En las etapas iniciales del desarrollo, especialmente durante la prototipación y conceptualización, la simplicidad y rapidez de las herramientas gráficas pueden permitir una iteración más ágil. Sin embargo, a medida que el proyecto evoluciona y se profundiza, la eficiencia y el control que proporciona el código pueden convertirse en una gran ventaja. Existen herramientas que tienen esta posibilidad y permiten diagramar flujos con procesadores predeterminados, que luego se pueden reemplazar por módulos de código, representados en notebooks, como es el caso de Azure Data Factory, visto en la sección de costos.

El proceso de extracción desarrollado en este trabajo, cuenta con una actividad muy personalizada como lo es el script de web scraping. En este caso se observa una mayor ventaja en la implementación directa del código, ya que facilita su mantenimiento e implementación. Sin embargo, durante la etapa de reporting, pudimos objetivamente realizar un mejor reporte con la herramienta perteneciente al grupo de drag n drop debido a su facilidad de uso. Para las etapas de transformación y carga encontramos ventajas y desventajas para ambas opciones, teniendo que lidiar con drivers y elementos externos durante el desarrollo en notebooks de código, mientras que esto se resolvió en Nifi con simples configuraciones sin necesidad de revisar compatibilidades entre las herramientas. Por otro lado, la herramienta drag n drop nos coartó de poder utilizar una herramienta valiosa para los entornos de Big Data como lo es Apache Spark.

En resumen, la dualidad entre herramientas de código y drag and drop no es una dicotomía rígida, sino más bien una oportunidad para aprovechar lo mejor de ambos enfoques dependiendo de las actividades que deba realizar nuestro proceso.

7. Referencias bibliográficas

[1] Extraction Transformation and Loading (ETL) of Data Using ETL Tools Manish Manoj Singh 2013

[2] Big Data Processing: Batch-based processing and stream-based processing Benjelloun, Sarah El Aissi, Mohamed Yassine, Loukili Lakhrissi, Younes Ali, Safae Chougrad, Hiba Boushaki, Abdessamad. 2020.

[3] A Study of Apache Kafka in Big Data Stream Processing - Bhole Rahul Hiraman, ,Chapte Viresh M, C Karve Abhijeet 2018

[4] A Systematic Literature Review on Big Data Extraction, Transformation and Loading (ETL) - Joshua Chibuike Nwokeji y Richard Matovu

[5] Big data: Promises and problems.N. Gudivada, R. A. Baeza-Yates, and V. V. Raghavan 2015.

[6] An Overview and Classification of Mediated Query Systems - Ruxandra Domenig, Klaus R. Dittrich 1999

[7] Modern Federated Database Systems: An Overview - Azevedo Leonardo, F. de S. Soares, Elton & Souza, Renan & Ferreira Moreno, Marcio 2020.

[8] Big Data ETL Implementation Approaches: A Systematic Literature Review Joshua C. Nwokeji, Faisal Aqlan, Anugu Apoorva, Ayodele Olagunju

[9] THE DATA WAREHOUSE TOOLKIT: THE DEFINITIVE GUIDE TO DIMENSIONAL MODELING RALPH KIMBALL 2013

[10] Business Intelligence System - Hans Peter Luhn 1958

[11] API BCRA - estadisticasbcra.com/api/documentacion

[12] The Data Warehouse ETL Toolkit - Ralph Kimball, Joe Caserta 2004.

[13] Nonprogrammers are building more of the world's software – a computer scientist explains 'no-code' - Tam Nguyen 2022

[14] ¿What is Apache Spark? IBM - ibm.com/es-es/topics/apache-spark

[15] Value Proposition and ETL Process in Big Data Environment - Prateek Kumar, Veena Gaded 2019

[16] On-Demand Big Data Integration: A Hybrid ETL Approach for Reproducible Scientific Research - Pradeeban Kathiravelu, Ashish Sharma, Helena Galhardas, Peter Van Roy, Luıs Veiga 2018

[17] Big Data ETL Process and Its Impact on Text Mining Analysis for Employees' Reviews - Laura Gabriela Tanasescu, Andreea Vines, Ana Ramona Bologa, Claudia AntalVaida 2022

[18] Next-generation ETL Framework to address the challenges posed by Big Data - Syed Muhammad Fawad Ali 2018

[19] Analysis of Cloud based ETL in the Era of IoT and Big Data - Geno Stefanov 2019

[20] Generalized Big Data Test Framework for ETL migration - Kunal Sharma, Vahida Attar

Apéndice

Despliegue del proceso

Para desplegar el proceso y poder simular lo mismo que se ejecutará durante la realización de este trabajo debemos instalar y/o configurar las siguientes herramientas:

- ➔ Jupyter Notebooks
- ➔ Apache Nifi
- ➔ PostgreSQL
- ➔ PgAdmin
- ➔ MongoDB
- ➔ CompassDB
- ➔ PowerBI Desktop

Juppyter Notebooks

Para correr el desarrollo basado en código se deben importar las notebooks de código adjuntas, con extensión .ipynb en un entorno de desarrollo como google collab o jupyter notebooks También se puede realizar en VSCODE descargando la extensión correspondiente. En caso de utilizar jupyter notebooks el puerto por defecto para ejecutarlas será el 8888

Apache Nifi

Se puede descargar e instalar esta herramienta desde su página oficial nifi.apache.org . Para poder ejecutarlo se debe correr el proceso nifi.bat, dentro de la carpeta bin. Se levantará un host local en el puerto 8080. Se debe importar como template el xml adjunto.

PostgreSQL y PgAdmin

Se pueden descargar estas herramientas desde su página oficial postgresql.org. Para crear la db modelada en este trabajo junto con los datos recopilados hasta la fecha se debe ejecutar el script backup articulosML.sql

MongoDB y CompassDB

Se pueden descargar e instalar estas herramientas desde su página oficial mongodb.com. Para obtener una réplica de la db utilizada en este trabajo junto con los datos recopilados hasta la fecha se debe importar dentro de una colección llamada electrónica en una base de datos llamada articulosML el json adjunto.

PowerBI Desktop

Se puede descargar e instalar esta herramienta desde su página oficial powerbi.microsoft.com. Para importar el tablero interactivo creado en este trabajo, se debe importar el archivo tableroARTICULOSML.pbix