UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Investigación de sistemas de almacenamiento big data como apoyo a la generación de analíticas y modelos de aprendizaje automático en imagen médica.

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Serra Gil, Jaime

Tutor/a: Bosch Roig, Ignacio

Cotutor/a externo: GOMIS MAYA, ARMANDO

CURSO ACADÉMICO: 2022/2023

Resumen

Los sistemas de almacenamiento de imagen médica empleados en los hospitales (PACS) no han sido diseñados teniendo en cuenta la creación de herramientas analíticas o de aprendizaje automático, lo que hace que presenten ciertas limitaciones que dificultan el proceso de trabajo, dando lugar a que sea poco eficiente y automatizable.

El auge del *big data* ha supuesto el surgimiento de nuevos sistemas de almacenamiento y herramientas de procesado de datos que agilizan las tareas analíticas y de aprendizaje automático con cualquier tipo de datos. Tanto en datos no estructurados (imágenes, video, audio, …) como en datos tabulados. Es por ello, que el objetivo del presente proyecto consiste en realizar una investigación comparativa de tecnologías *big data* para la creación de un repositorio de imagen médica que sea capaz de agilizar y facilitar el proceso de trabajo para los nuevos perfiles profesionales basados en datos que están surgiendo.

Palabras clave: Big data, aprendizaje automático, imagen médica, analítica, sistemas de almacenamiento.

Resum

Els sistemes d'emmagatzematge d'imatge mèdica emprats als hospitals (PACS) no han estat dissenyats tenint en compte la creació d'eines analítiques o d'aprenentatge automàtic, cosa que fa que presentin certes limitacions que dificulten el procés de treball, cosa que fa que sigui poc eficient i automatitzable.

L'apogeu del *big data* ha suposat el sorgiment de nous sistemes d'emmagatzematge i eines de processament de dades que agiliten les tasques analítiques i d'aprenentatge automàtic amb qualsevol tipus de dades. Tant en dades no estructurades (imatges, vídeo, àudio, …) com en dades tabulades. És per això que l'objectiu del present projecte consisteix a fer una investigació comparativa de tecnologies *big data* per a la creació d'un repositori d'imatge mèdica que sigui capaç d'agilitzar i facilitar el procés de treball per als nous perfils professionals basats en dades que estan sorgint.

Paraules clau: Big data, aprenentatge automàtic, imatge mèdica, analítica, sistemes d'emmagatzematge.

Abstract

The medical image storage systems used in hospitals (PACS) have not been designed with the creation of analytical or machine learning tools in mind, which means that they have certain limitations that hinder the work process, making it inefficient and difficult to automate.

The rise of big data has led to the emergence of new storage systems and data processing tools that speed up analytical and machine learning tasks with any type of data. Both in unstructured data (images, video, audio, ...) and tabulated data. For this reason, the aim of this project is to carry out a comparative investigation of big data technologies for the creation of a medical image repository that is capable of speeding up and facilitating the work process for the new professional profiles based on data that are emerging.

Key words: Big data, machine learning, medical imaging, analytics, storage systems.

Índice

Índice de figuras

Índice de tablas

Glosario de términos

Término	Definición
Bases de datos relacionales	Sistemas de gestión de bases de datos que siguen el modelo relacional, estructurando la información en tablas y utilizando relaciones para vincular los datos.
Big Data	Se refiere al manejo y análisis de grandes volúmenes de datos que superan las capacidades de los métodos y herramientas tradicionales.
Consultas	Solicitudes o preguntas realizadas a una base de datos para obtener información específica.
DICOM	Siglas de Digital Imaging and Communications in Medicine, es el estándar utilizado para el intercambio, almacenamiento y transmisión de imágenes médicas.
Docker	Plataforma de código abierto que permite la creación, implementación y ejecución de aplicaciones en contenedores.
JSON	Acrónimo de JavaScript Object Notation, es un formato de datos ligero y de fácil lectura utilizado para el intercambio de información entre sistemas.
MongoDB	Base de datos NoSQL orientada a documentos, que permite el almacenamiento y consulta de datos de manera flexible y escalable.
PACS	Siglas de Picture Archiving and Communication System, se refiere a un sistema de gestión y almacenamiento de imágenes médicas, utilizado en entornos clínicos y de diagnóstico.
PostgreSQL	Sistema de gestión de bases de datos relacional de código abierto y robusto, conocido por su confiabilidad y capacidad para manejar grandes cantidades de datos.
Python	Lenguaje de programación de alto nivel, utilizado en el proyecto para el desarrollo de diversas funcionalidades y la manipulación de datos.
SQL	Lenguaje de consulta estructurado utilizado para comunicarse con bases de datos relacionales, permitiendo la manipulación y extracción de datos.
Tiempos de ejecución	Tiempo que toma completar la ejecución de una tarea o proceso.
Trino	Motor de consulta distribuido que permite el análisis y procesamiento de datos en tiempo real desde múltiples fuentes.

Capítulo 1. Introducción

1.1 Motivación

La tecnología de imágenes médicas ha experimentado un crecimiento exponencial en las últimas décadas, revolucionando el campo de la medicina y mejorando significativamente la calidad de vida de los pacientes. Actualmente, una amplia gama de técnicas de diagnóstico por imagen, incluidas tomografías computarizadas, radiografías, resonancias magnéticas y ultrasonidos, están disponibles para proporcionar información detallada sobre el estado del cuerpo humano.

Las técnicas de diagnóstico por imagen se han convertido en una herramienta indispensable para los profesionales de la salud, ya que les permiten visualizar y analizar estructuras anatómicas, detectar anomalías, identificar enfermedades y monitorear la efectividad de los tratamientos. Sin embargo, la creciente utilización de imágenes médicas ha generado una gran cantidad de datos que deben ser almacenados, administrados y analizados de manera efectiva.

Los sistemas de almacenamiento de imágenes médicas actualmente utilizados, conocidos como PACS (Sistemas de Comunicación y Archivo de Imágenes o en inglés *picture archiving and communication system*), fueron diseñados principalmente para la transmisión y almacenamiento de imágenes (ver **¡Error! No se encuentra el origen de la referencia.**). Estos sistemas se han concentrado en garantizar la integridad y la disponibilidad de las imágenes, pero no se han tenido en cuenta aspectos importantes para el análisis y la aplicación de herramientas de aprendizaje automático o análisis de *big data*.

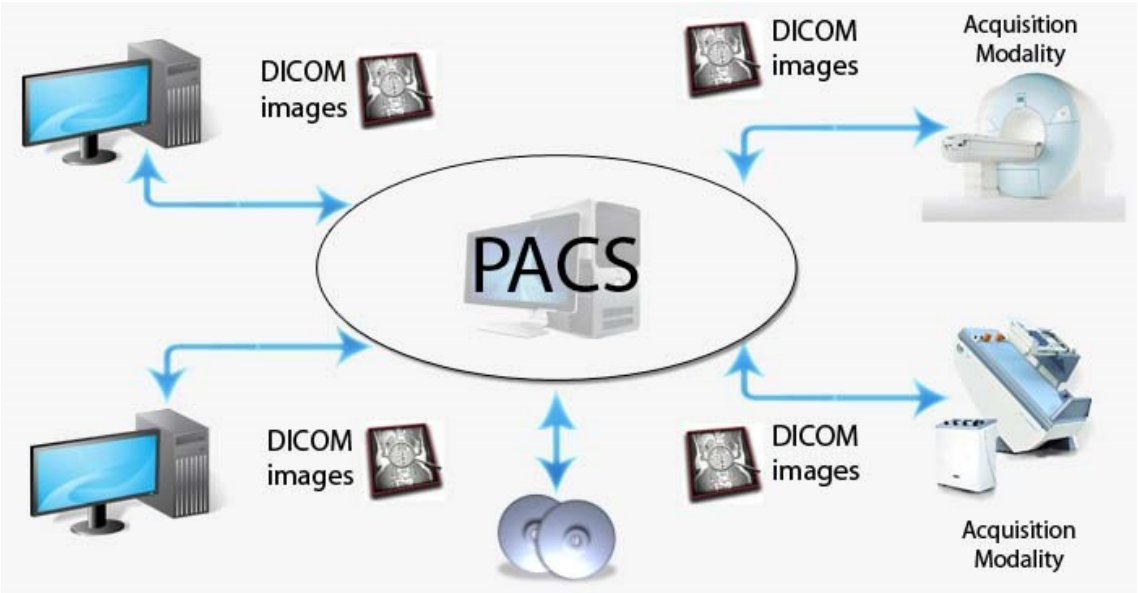


Figura 1: Esquema PACS (1).

La falta de integración entre las herramientas analíticas y los sistemas de almacenamiento de imágenes médicas ha generado problemas importantes. Los profesionales de la salud tienen dificultades para acceder y compartir datos de imágenes, así como para realizar análisis sofisticados que permiten descubrir patrones ocultos, realizar diagnósticos más precisos y tomar decisiones clínicas fundamentadas (2).

La importancia de aprovechar el potencial del *big data* en el campo de las imágenes médicas surge aquí. El auge del *big data* ha llevado a la creación de nuevos sistemas de almacenamiento y herramientas de procesamiento de datos que pueden almacenar, administrar y analizar cantidades significativas de datos, como imágenes médicas. Estos sistemas y herramientas permiten la

creación de bases de datos optimizadas para la búsqueda, filtración y aplicación de IA y aprendizaje automático.

El uso de técnicas de *big data* en el análisis de imágenes médicas ofrece una amplia gama de oportunidades. El análisis de grandes cantidades de datos está *"ayudando a tomar decisiones relativas a diagnóstico y tratamiento todo finalmente enfocado a una mejor atención al paciente"* (3).

1.2 Objetivos

Para el presente trabajo, se han definido una serie de objetivos en base a los cuales se busca obtener conclusiones significativas que contribuyan al conocimiento y la toma de decisiones en el ámbito de la gestión y análisis de imágenes médicas, así como en la selección de tecnologías adecuadas para futuros proyectos en este campo. Los objetivos son:

- Generación de tablas con información enriquecida que permita la rápida identificación y filtrado de las imágenes médicas deseadas.
- Comparativa de rendimiento en sistemas de bases de datos tabulares como PostgreSQL frente a una base de datos NoSQL orientada a documentos como MongoDB.
- Comparativa de tiempos de ejecución y requerimientos de almacenamiento para un caso de uso de analítica, teniendo en cuenta varias herramientas de procesado de datos como Spark o Trino en los repositorios de prueba anteriormente descritos.

1.3 Organización de la memoria

En este apartado de la memoria, se abordará la organización general del Trabajo de Fin de Grado, sintetizando los siguientes puntos:

Metodología de Trabajo: Se describe la metodología empleada en la investigación realizada, detallando la planificación y las tareas llevadas a cabo para abordar el desarrollo del trabajo.

Marco teórico: En este capítulo se aborda el marco teórico del proyecto, explorando los conceptos y tecnologías relevantes para su desarrollo. Este capítulo sienta las bases para una comprensión profunda de los temas abordados.

Herramientas y Programas: En este apartado se exploran en detalle las diversas herramientas y programas utilizados en el desarrollo del proyecto. Se destaca el papel fundamental que desempeñaron en la implementación de las soluciones propuestas y en el análisis de los datos obtenidos. Además, se describen los lenguajes de programación utilizados y el software empleado en cada etapa del proyecto.

Desarrollo y Resultados del Trabajo: Este capítulo describe la progresión del trabajo realizado y los resultados obtenidos. Se brinda una descripción completa de las etapas y métodos empleados en el proyecto, junto con los hallazgos y conclusiones derivados.

Conclusiones y Propuesta de Trabajo Futuro: Se resume las conclusiones, hallazgos y logros alcanzados en el trabajo realizado. Se discuten los resultados en relación con los objetivos planteados. Además, se presenta una propuesta para futuras investigaciones.

Capítulo 2. Metodología de Trabajo

2.1 Introducción

En este apartado se describirá la metodología que se utilizará en la investigación realizada para este Trabajo de Fin de Grado. Se explicará la organización de las tareas realizadas para abordar el desarrollo de este trabajo.

2.2 Organización de las tareas

Fases	Febrero				Marzo				Abril				Mayo				Junio				Julio			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Fase 1																								
Fase 2																								
Fase 3																								
Fase 4																								
Fase 5																								
Fase 6																								

Tabla 1: Diagrama de Gantt.

A continuación, se describen las fases que se han llevado a cabo en el proceso de desarrollo de este trabajo, tal como se muestra en la Tabla 1.

Fase 1. Estudio preeliminar: se ha llevado a cabo una investigación exhaustiva y adquisición de conocimientos sobre herramientas como Docker, el procesamiento de imágenes médicas en formato DICOM, y el uso de Python para manipular y analizar datos. Esto ha sentado las bases sólidas para el desarrollo del proyecto.

Fase 2. Limpieza de datos: Se ha realizado una revisión íntegra de los datos facilitados por el Grupo de Investigación Biomédica en Imagen del Hospital La Fe, con el objetivo de detectar y eliminar posibles documentos corruptos o inconsistentes. Esto asegura la calidad y fiabilidad de los datos utilizados en el trabajo.

Fase 3. Creación de las bases de datos: Se han creado dos modelos de base de datos, uno basado en SQL en PostgreSQL y otro en NoSQL en MongoDB, para almacenar y gestionar la información necesaria para el proyecto. Estos modelos de base de datos han sido diseñados de acuerdo con los requisitos específicos del trabajo y permiten un almacenamiento eficiente y estructurado de los datos.

Fase 4. Toma de medidas de tiempo de consulta: Se ha realizado una medición del tiempo de ejecución de diferentes consultas en las bases de datos utilizadas. Esto permite evaluar y comparar el rendimiento de cada base de datos en términos de velocidad y eficiencia.

Fase 5. Optimización de consultas para Big Data: Se ha utilizado la herramienta Trino para gestionar grandes cantidades de datos y optimizar las consultas realizadas sobre ellos. Esta herramienta permite realizar consultas distribuidas y paralelas, lo que mejora significativamente el rendimiento y la eficiencia en el procesamiento.

Fase 6. Redacción de la memoria: Se ha llevado a cabo la redacción del trabajo final de grado, que incluye la descripción detallada de todo el proceso de desarrollo, los resultados obtenidos, las conclusiones y las referencias utilizadas.

Capítulo 3. Marco teórico

3.1 Introducción

En este capítulo se aborda el marco teórico del presente trabajo, donde se exploran los conceptos y tecnologías relevantes para el desarrollo del proyecto. Este capítulo servirá como cimiento para el desarrollo y la comprensión profunda de los temas abordados en el proyecto.

3.2 Conceptos y tecnologías relevantes

3.2.1 Bases de datos

Una base de datos es una colección organizada de datos pertenecientes a un mismo dominio, con el propósito de permitir su acceso y manipulación de manera eficiente. Esto implica que los datos están estructurados y almacenados de forma adecuada para facilitar su posterior tratamiento y análisis.

El surgimiento de las bases de datos se originó por la necesidad de preservar la información a lo largo del tiempo, evitando su deterioro, y garantizando su accesibilidad para su posterior consulta. De esta manera, se buscaba encontrar una solución que permitiera el almacenamiento duradero de los datos y su disponibilidad para su recuperación en el futuro.

El diseño de las bases de datos se enfoca en facilitar la búsqueda de datos deseados por parte de los usuarios. Para lograr esto, la información se almacena siguiendo una estructura determinada basada en diversos parámetros. Este proceso se lleva a cabo mediante sistemas de gestión de bases de datos, los cuales automatizan el almacenamiento ordenado de los datos y su posterior recuperación de manera eficiente.

Las bases de datos ofrecen la capacidad de realizar acciones y consultas que permiten buscar, recuperar, actualizar y eliminar información de manera eficiente. Es sencillo obtener resultados pertinentes y correctos porque las consultas se utilizan para localizar y recuperar material específico que satisface los parámetros de búsqueda especificados.

Se pueden organizar las bases de datos según su naturaleza y características. Una forma común de clasificarlas es en base a su modelo de datos, distinguiendo entre bases de datos relacionales y no relacionales.

Durante un largo período, las bases de datos relacionales han sido ampliamente utilizadas, centrándose en el uso de relaciones entre los datos. Estas bases de datos permiten la recuperación y el almacenamiento de información a través de consultas, que suelen construirse utilizando el lenguaje SQL (Structured Query Language). Este lenguaje se ha convertido en un estándar implementado por los sistemas de gestión de bases de datos relacionales más populares.

En las bases de datos relacionales, el funcionamiento se basa en la inserción de datos en registros que se almacenan en tablas. Estas tablas permiten relacionar de manera sencilla los elementos entre sí y establecer conexiones entre ellas, lo que da lugar a relaciones entre los registros. Esta estructura de tablas y relaciones facilita la gestión y organización de los datos, así como la realización de consultas y operaciones complejas.

En una base de datos SQL, los datos se almacenan en tablas, que se componen de filas y columnas. Las columnas especifican los atributos de los datos, mientras que las filas contienen los valores correspondientes.

En el amplio panorama de los sistemas de gestión de bases de datos, destacan varios gestores populares, entre ellos MySQL, PostgreSQL, Oracle Database y Microsoft SQL Server. Para este proyecto, se ha seleccionado PostgreSQL como el gestor de base de datos (4).

Las bases de datos no relacionales o también llamadas NoSQL (Not Only SQL) son sistemas de administración de bases de datos que difieren de las bases de datos SQL típicas en su enfoque y

estructura. A diferencia de las bases de datos SQL, las bases de datos NoSQL no emplean un modelo relacional y no interactúan con los datos principalmente a través del lenguaje SQL.

Las bases de datos NoSQL contienen datos en formatos que no son tablas con filas y columnas, como documentos, gráficos, clave-valor o columnas anchas.

Estos gestores de bases de datos suelen ofrecer una mayor flexibilidad y apertura, lo que los hace altamente adaptables a las necesidades específicas de los proyectos. Esto permite una configuración más ágil y sencilla de las bases de datos, permitiendo ajustarlas de acuerdo a los requerimientos particulares de cada aplicación.

Este tipo de bases de datos resultan especialmente adecuadas para aplicaciones que manejan grandes volúmenes de datos, requieren tiempos de respuesta reducidos y necesitan modelos de datos flexibles. Son ampliamente utilizadas en entornos distribuidos donde la disponibilidad continua y el funcionamiento ininterrumpido son prioritarios. Estas bases de datos están diseñadas para optimizar el rendimiento y la eficiencia, permitiendo gestionar eficazmente la creciente cantidad de datos generados en el entorno actual. Su capacidad para manejar cargas de trabajo intensivas y su capacidad de escalamiento las convierten en una elección frecuente en aplicaciones de alto rendimiento.

En las bases de datos NoSQL, como contrapartida a las bases de datos relacionales, se utiliza un enfoque diferente para almacenar los datos. En lugar de organizar los datos en tablas y filas como en las bases de datos relacionales, en una base de datos NoSQL, se almacenan los registros como documentos JSON (JavaScript Object Notation). JSON es un formato de texto sencillo utilizado para el intercambio de datos. Cada documento en la base de datos NoSQL contiene los valores de los atributos en un solo objeto.

Esta estructura de documentos JSON permite una mayor flexibilidad en el almacenamiento y modelado de datos. Además, al utilizar documentos JSON, es posible representar estructuras de datos más complejas y anidadas, lo que resulta beneficioso en escenarios donde los datos no tienen una estructura rígida y se requiere una mayor flexibilidad en el modelado.

Una de las posibles desventajas de las bases de datos NoSQL es que pueden enfrentar problemas de compatibilidad. Debido a que cada base de datos NoSQL tiene su propio enfoque y estructura de almacenamiento de datos. Además, debido a su relativa novedad en comparación con las bases de datos relacionales, las bases de datos NoSQL pueden tener menos herramientas y recursos disponibles, así como una menor cantidad de documentación y soporte comunitario.

Dentro del ámbito de los sistemas de gestión de bases de datos NoSQL, existen diversas opciones destacadas, como MongoDB, Cassandra, Redis y Elasticsearch, entre otros. En el marco de este proyecto, se ha optado por utilizar MongoDB como el gestor de base de datos NoSQL (5).

3.2.2 Sistemas de consulta

Los sistemas de consultas desempeñan un papel crucial en las bases de datos al permitir a los usuarios buscar, recuperar y manipular datos de manera eficiente. Estos sistemas proporcionan un conjunto de herramientas, lenguajes y comandos que facilitan la formulación y ejecución de consultas, que son solicitudes específicas de información que deben cumplir ciertos criterios establecidos por el usuario.

En el ámbito de los sistemas de gestión de bases de datos relacionales, los sistemas de consultas SQL son fundamentales. SQL, o Structured Query Language, es un lenguaje estándar utilizado para interactuar con bases de datos relacionales. Ofrece una sintaxis específica y comandos como SELECT, INSERT, UPDATE y DELETE, que permiten a los usuarios formular consultas y realizar operaciones de manipulación de datos en la base de datos.

La versatilidad de los sistemas de consultas SQL radica en su capacidad para realizar una amplia gama de operaciones. Desde consultas simples que recuperan registros específicos de una tabla,

hasta consultas complejas que combinan datos de múltiples tablas y aplican criterios de filtrado sofisticados utilizando operadores lógicos y de comparación. Además, SQL proporciona funciones y operadores que permiten realizar cálculos, ordenar resultados y generar informes, ampliando aún más las capacidades de manipulación de datos.

En el contexto de las bases de datos NoSQL, los sistemas de consulta se utilizan para interactuar con datos almacenados en formatos no relacionales, como documentos, gráficos, clave-valor o columnas anchas. A diferencia de los sistemas de consultas SQL utilizados en bases de datos relacionales, los sistemas de consulta NoSQL suelen tener enfoques específicos para acceder y manipular los datos. Cada base de datos NoSQL puede ofrecer su propio lenguaje de consulta diseñado para aprovechar las características y estructuras de datos particulares de ese sistema.

Tomando como ejemplo MongoDB, una popular base de datos NoSQL, utiliza su propio lenguaje de consulta llamado MongoDB Query Language (MQL). MQL se basa en una sintaxis similar a JSON y permite realizar consultas complejas, filtrar datos, realizar agregaciones y operaciones de búsqueda en documentos almacenados. La flexibilidad de MQL y su capacidad para manejar estructuras de datos anidadas lo hacen especialmente adecuado para aplicaciones que requieren un modelado de datos flexible y no tienen una estructura rígida (6).

Es importante tener en cuenta que la eficiencia de las consultas depende en gran medida del diseño adecuado de la base de datos y de la optimización de las consultas realizadas. Un diseño eficiente implica organizar y estructurar los datos de manera que se ajusten a los patrones de acceso más comunes. Por otro lado, la optimización de consultas implica mejorar el rendimiento de las consultas mediante la selección adecuada de índices, la reescritura de consultas para aprovechar las optimizaciones internas del sistema y el uso de técnicas como la fragmentación de datos para distribuir la carga de consultas en sistemas distribuidos.

Además de los sistemas de consultas SQL y NoSQL, existen herramientas específicas que se enfocan en la optimización y ejecución eficiente de consultas en bases de datos. Un ejemplo destacado es Trino. Estas herramientas implementan técnicas avanzadas para mejorar el rendimiento de las consultas, como la proyección de columnas, el filtrado de datos y la ejecución de operaciones en memoria. Estas optimizaciones ayudan a reducir los tiempos de respuesta y aumentar la eficiencia de las consultas realizadas.

3.2.3 Big Data

El big data se refiere a conjuntos de datos grandes y complejos que contienen una mayor variedad de información, llegan en volúmenes crecientes y se generan a mayor velocidad. Estos conjuntos de datos son tan voluminosos que el software de procesamiento de datos tradicional no puede manejarlos. Sin embargo, el análisis de estos grandes volúmenes de datos puede ayudar a abordar problemas empresariales que antes eran difíciles de resolver (7).

El big data se caracteriza por las tres Vs: volumen, velocidad y variedad. El volumen se refiere a la cantidad de datos que se manejan, que puede llegar a ser de varios terabytes o petabytes. La velocidad se refiere a la rapidez con la que se reciben y procesan los datos, especialmente en tiempo real. La variedad se refiere a los diferentes tipos de datos disponibles, incluyendo datos no estructurados como texto, audio y video.

El valor del big data radica en descubrir su utilidad y la veracidad de los datos. Las empresas más exitosas, como las grandes compañías de tecnología, obtienen un gran valor de sus datos al analizarlos constantemente para mejorar la eficiencia y desarrollar nuevos productos.

El uso del big data abarca diversas áreas de negocio, como el desarrollo de productos, el mantenimiento predictivo, la mejora de la experiencia del cliente, la detección de fraudes y el cumplimiento normativo, el aprendizaje automático, la eficiencia operativa y la innovación.

Sin embargo, el big data también presenta desafíos. El volumen de datos está en constante crecimiento, lo que dificulta su almacenamiento y gestión efectiva. Además, se requiere un trabajo de curación y preparación de los datos antes de que puedan ser utilizados. También es necesario mantenerse al día con la evolución de las tecnologías de big data.

Para aprovechar al máximo el big data, se deben seguir prácticas recomendadas, como alinear el big data con los objetivos comerciales, abordar la escasez de habilidades mediante estándares y gobernanza, establecer un centro de excelencia para compartir conocimientos, integrar datos no estructurados con datos estructurados y planificar adecuadamente los entornos de descubrimiento y análisis. La adopción de un modelo de nube también es importante para brindar acceso a recursos y soportar los requisitos cambiantes del big data (8).

3.2.4 Archivos DICOM

El formato DICOM (Digital Imaging and Communications in Medicine) es el estándar internacional utilizado para imágenes médicas e información relacionada. Este formato desempeña un papel fundamental en la gestión y análisis de imágenes médicas, ya que permite representar y almacenar tanto imágenes 2D como 3D, junto con información clínica relevante del paciente y detalles del estudio.

Los archivos DICOM contienen una amplia gama de información que es crucial en el ámbito médico. Estos archivos están estructurados en base a *tags*, que representan campos específicos de datos. Estos campos incluyen datos del paciente, como nombre, edad y sexo, así como detalles sobre el tipo de examen realizado, la fecha del examen, los valores de píxeles de la imagen y la información del equipo utilizado para adquirir la imagen, entre otros datos relevantes. Los *tags* DICOM se organizan en grupos, cada uno de los cuales se enfoca en un aspecto particular de la imagen médica.

La estructura jerárquica de los *tags* DICOM es una de las características distintivas de este formato de archivo. Esta estructura permite una gran flexibilidad y extensibilidad en el almacenamiento de información médica. Los *tags* se pueden agregar, modificar o eliminar según las necesidades específicas de cada estudio o aplicación clínica. Además, el estándar DICOM garantiza la interoperabilidad y el intercambio seguro de imágenes médicas entre diferentes sistemas y dispositivos, asegurando que los datos se transmitan de manera precisa y sin pérdidas (9).

En nuestro trabajo, hemos trabajado con archivos DICOM para obtener imágenes médicas y analizar su contenido.

Figura 2: Ejemplo *tags* DICOM (10).

En el campo de la investigación médica y el diagnóstico por imágenes, los archivos DICOM desempeñan un papel fundamental. Permiten a los profesionales de la salud acceder y analizar imágenes médicas para el diagnóstico, seguimiento del tratamiento y planificación quirúrgica. Además, los archivos DICOM son compatibles con una variedad de herramientas de visualización y análisis médico, lo que facilita el trabajo con las imágenes y mejora la colaboración entre profesionales de la salud.

La adquisición y el procesamiento de imágenes médicas a partir de archivos DICOM requieren de herramientas y software especializados. Existen aplicaciones y bibliotecas que permiten la lectura, manipulación y análisis de archivos DICOM, lo que facilita la extracción de información relevante para su posterior estudio e interpretación.

Capítulo 4. Herramientas y Programas

4.1 Introducción

En este capítulo, se exploran en detalle las diferentes herramientas y programas que se utilizaron en el desarrollo de este proyecto. Estas herramientas desempeñaron un papel fundamental en la implementación de las soluciones propuestas y en el análisis de los datos obtenidos. A continuación, se describen los lenguajes de programación utilizados y el software empleado en cada etapa del proyecto.

4.2 Python

Python es un lenguaje de programación de alto nivel, interpretado y orientado a objetos. Python cuenta con una amplia gama de bibliotecas y paquetes especializados que ofrecen funcionalidades avanzadas y eficientes para diversas tareas (11).

Figura 3: Logo Python.

4.2.1 Bibliotecas de Python

Las bibliotecas de Python son módulos de software que contienen una serie de funciones y herramientas predefinidas, diseñadas para abordar necesidades específicas.

4.2.1.1 OS

La biblioteca OS (*Operating System*) es una biblioteca estándar de Python que proporciona una interfaz para conectarse con el sistema operativo subyacente. Esta biblioteca incluye una variedad de funciones que le permiten realizar actividades de administración de archivos y directorios, administración de procesos, operaciones de red, etc. (12).

La gestión de archivos ha sido una función crítica en el desarrollo de este trabajo, particularmente en términos de ubicación y manipulación de archivos DICOM. La biblioteca OS se utilizó para explorar el sistema de archivos, buscar y acceder a archivos DICOM almacenados en varias ubicaciones.

4.2.1.2 Numpy

Numpy es una biblioteca de Python diseñada específicamente para realizar operaciones numéricas en matrices y arreglos multidimensionales.

Ofrece una serie de funcionalidades que permiten el procesamiento eficiente y rápido de grandes cantidades de datos. Con Numpy, es posible realizar cálculos matemáticos complejos, manipular y transformar datos de manera eficiente, y realizar operaciones vectorizadas en matrices y

arreglos, lo que resulta en un mejor rendimiento y mayor velocidad en comparación con las operaciones tradicionales en Python (13).

4.2.1.3 Pandas

Pandas es un paquete de Python creado para el análisis de datos y la manipulación de conjuntos de datos. Esta biblioteca ofrece capacidades para leer y escribir datos de varias fuentes, limpiar y transformar datos, así como agregar y analizar datos.

Una de las características más importantes es su capacidad para manejar datos tabulares y de series de tiempo de manera eficiente y flexible. Los dos tipos de datos principales introducidos por esta biblioteca son series, una matriz unidimensional con etiquetas de índice, y DataFrames, una estructura de datos bidimensional con etiquetas de índice para filas y columnas (14).

Se ha utilizado la biblioteca de Pandas para el análisis, manejo y manipulación de datos tabulares, creando DataFrames con los documentos DICOM proporcionados. Además, se han utilizado procedimientos de filtrado, agrupación y agregación de datos, las transformaciones de datos, la combinación de conjuntos de datos y el manejo de datos faltantes gracias a esta librería.

4.2.1.4 Time

La biblioteca time de Python es una herramienta fundamental para trabajar con el tiempo en programas y scripts. Proporciona funciones y métodos que permiten medir el tiempo de ejecución (15).

Una de las funciones más utilizadas de la biblioteca time es time.time(), que devuelve el tiempo actual en segundos. Esta función es útil para medir el tiempo de ejecución de un fragmento de código y evaluar la eficiencia, se puede utilizar para calcular la duración de un proceso. Gracias a esta función se ha podido medir la ejecución de tiempos de consulta.

4.2.1.5 CSV

La biblioteca CSV de Python es una biblioteca incorporada que permite la lectura y escritura de archivos CSV, que son archivos de texto que almacenan datos en formato tabular, separando los valores por comas u otros delimitadores.

La biblioteca CSV proporciona funciones y métodos para leer datos de archivos CSV y convertirlos en estructuras de datos de Python, como listas o diccionarios. También ofrece funciones para escribir datos en un archivo CSV a partir de estructuras de datos de Python (16).

Se ha realizado pruebas con esta biblioteca para organizar los documentos DICOM en archivos CSV.

4.2.1.6 PyDicom

PyDicom es una biblioteca de Python creada con el propósito de simplificar la manipulación y procesamiento de imágenes médicas en formato DICOM.

PyDicom facilita la lectura, escritura y procesamiento de archivos DICOM. Esta biblioteca permite extraer información detallada de los metadatos de las imágenes, brindando acceso a los *tags* y sus correspondientes valores en cada documento DICOM (17).

Se ha utilizado PyDicom para acceder a los metadatos de los archivos DICOM y extraer la información relevante

4.2.1.7 Joblib

Joblib es una biblioteca diseñada para facilitar la ejecución paralela de tareas, especialmente en entornos de cómputo intensivo. Proporciona herramientas para la ejecución paralela eficiente de bucles y tareas repetitivas, lo que puede acelerar significativamente el procesamiento de datos y reducir el tiempo de ejecución en máquinas multi-core.

La clase Parallel de Joblib permite realizar cálculos en paralelo mediante la división de tareas en múltiples procesos. según el hardware y la configuración del sistema. Puede especificar el número de núcleos de CPU a utilizar y la estrategia de paralelización deseada. Parallel facilita la implementación de cálculos paralelos, ya que se encarga de la distribución y recopilación de los resultados automáticamente (18).

La clase Delayed se utiliza en combinación con Parallel para especificar las tareas que se deben ejecutar en paralelo.

Joblib se ha utilizado para facilitar los procesos de lectura de datos y la extracción de *tags* y valores realizados paralelamente.

4.2.1.8 Psycopg2

Psycopg2 es una biblioteca que permite la conexión y comunicación con bases de datos PostgreSQL. Esta biblioteca proporciona una interfaz sencilla y eficiente para interactuar con bases de datos PostgreSQL desde programas escritos en Python.

Psycopg2 permite establecer conexiones a bases de datos PostgreSQL, ejecutar consultas SQL, realizar transacciones, manejar errores y recuperar resultados de manera eficiente. Proporciona múltiples funciones y métodos que facilitan la manipulación de datos, la creación y modificación de tablas, la ejecución de consultas complejas y la gestión de transacciones (19).

Psycopg2 ha sido utilizada para crear la base de datos en PostgreSQL, la conexión con esta y la realización de consultas SQL para almacenar y recuperar información relevante.

4.2.1.9 SQLAlchemy

SQLAlchemy es una biblioteca de Python que facilita la interacción con bases de datos relacionales de manera eficiente y flexible. Proporciona una capa de abstracción sobre los motores de bases de datos, lo que permite escribir código independiente del sistema de gestión de bases de datos subyacente.

Con SQLAlchemy, se pueden realizar tareas comunes en bases de datos relacionales, como la creación de tablas, la ejecución de consultas SQL, la modificación de datos y la gestión de transacciones, de manera más fácil y legible. Además, ofrece una sintaxis orientada a objetos para trabajar con los datos de la base de datos, permitiendo el mapeo de objetos Python a tablas de base de datos y simplificando las operaciones CRUD (crear, leer, actualizar y eliminar) (20).

4.2.1.10 PyMongo

El módulo de Python llamado PyMongo ofrece una forma de comunicarse con MongoDB. PyMongo simplifica la conexión y el trabajo con datos en MongoDB usando Python, lo que permite acciones rápidas y eficientes de inserción, actualización, consulta y eliminación de documentos.

Algunas de las características clave de PyMongo son establecer conexiones con la base de datos, obtener acceso a colecciones y documentos dentro de MongoDB y ejecutar consultas y cambios

mediante el motor de consultas de MongoDB. Estas características son por lo que PyMongo se ha empleado (21).

4.2.1.11 Trino

La biblioteca Trino es una herramienta de procesamiento distribuido y de alto rendimiento diseñada para consultas SQL en grandes conjuntos de datos. Trino permite ejecutar consultas de manera rápida y eficiente.

Trino es altamente escalable y puede manejar grandes volúmenes de datos de forma eficiente mediante la utilización de clústeres distribuidos (22).

4.2.1.12 MinIO

La biblioteca MinIO para Python proporciona una interfaz sencilla para interactuar con MinIO desde una aplicación Python. Permite realizar operaciones como subir y descargar archivos, listar objetos, eliminar objetos y administrar la configuración y seguridad del almacenamiento de objetos.

En un entorno de Big Data, es fundamental contar con un sistema de archivos distribuido como MinIO. En lugar de utilizar el sistema de archivos del sistema operativo, se opta por MinIO como solución para el almacenamiento distribuido. MinIO proporciona una plataforma confiable y escalable que reemplaza al sistema de archivos convencional, permitiendo aprovechar al máximo el potencial de un sistema Big Data.

4.3 SQL

SQL (Structured Query Language) es un lenguaje de programación utilizado para manipular y gestionar bases de datos relacionales. Permite realizar consultas y operaciones en bases de datos, como la recuperación, inserción, actualización y eliminación de registros, la creación y eliminación de tablas y la definición de relaciones entre tablas.

SQL es un lenguaje declarativo, lo que significa que se describe lo que se quiere hacer con los datos en lugar de como hacerlo. Además, es compatible con diferentes sistemas de bases de datos relacionales, lo que facilita su uso en diferentes plataformas. SQL tiene la capacidad de realizar consultas complejas y manipular grandes cantidades de datos.

Figura 4: Logo SQL.

4.4 Entornos de desarrollo

Los entornos de desarrollo proporcionan un conjunto de herramientas y características que facilitan el proceso de desarrollo de software. En este proyecto, se han utilizado dos entornos principales dentro de Anaconda:

4.4.1 Anaconda

Anaconda es una plataforma de código abierto diseñada para la gestión de entornos de desarrollo en Python y R. Anaconda proporciona un conjunto de paquetes preinstalados.

4.4.1.1 Jupyter Notebook

Jupyter Notebook es una herramienta de programación interactiva que permite crear y compartir documentos que combinan código, texto explicativo y visualizaciones. Está diseñado para facilitar la exploración y análisis de datos, así como para la creación de informes técnicos y presentaciones.

El Notebook consiste en una interfaz basada en la web que se ejecuta en un navegador y que permite la edición y ejecución de celdas de código, lo que hace posible que los usuarios puedan iterar y experimentar con el código en tiempo real (23).

En la primera parte de tu proyecto, he utilizado Jupyter Notebook para aprender y comprender mejor los conceptos, experimentar con el código y visualizar los resultados.

4.4.1.2 Visual Studio Code

Visual Studio Code (VSC) es un editor de código fuente, es compatible con varios lenguajes de programación, tiene una interfaz de usuario intuitiva y fácil de usar. A diferencia de Jupyter Notebook, VSC no proporciona una funcionalidad interactiva similar a las celdas de código. En la segunda parte del proyecto, he utilizado Visual Studio Code como tu entorno de desarrollo principal, donde he escrito y ejecutado el código de manera más tradicional y trabajado en la implementación final del trabajo.

4.5 Docker

Docker es una plataforma de software que permite a los desarrolladores empacar, distribuir y ejecutar aplicaciones en contenedores. Un contenedor es una unidad de software que incluye todo lo necesario para ejecutar una aplicación, incluyendo código, bibliotecas, dependencias y configuraciones. Docker permite que los desarrolladores empaqueten una aplicación y todas sus dependencias en un contenedor, lo que garantiza que la aplicación se ejecute de manera consistente y sin problemas en cualquier entorno (24).

Además, Docker proporciona una manera fácil de administrar y escalar aplicaciones en diferentes entornos. Con Docker se puede asegurar que la aplicación se ejecuta de la misma manera en todas partes, lo que ayuda a reducir los problemas de configuración y compatibilidad entre diferentes sistemas y plataformas (25).

En este trabajo se ha utilizado imágenes de PostreSQL y Trino, donde he creado y configurado instancias de contenedores con versiones específicas de estas bases de datos y motores de consulta. Docker simplifica el proceso de configuración y gestión de estas herramientas, permitiendo una implementación más rápida y simplificada.

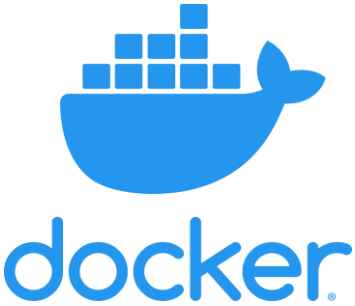


Figura 5: Logo Docker.

4.6 Bases de datos

Las bases de datos son sistemas de gestión de información que permiten almacenar y organizar datos de manera eficiente. En este proyecto, se han utilizado diferentes bases de datos:

4.6.1 PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto y altamente confiable. Basado en el modelo relacional permite almacenar y organizar datos en tablas con relaciones definidas entre ellas. Esto brinda una estructura coherente y eficiente para almacenar y recuperar información.

Tiene completo soporte de SQL, lo que permite realizar consultas de manera eficiente. Además, PostgreSQL tiene la capacidad de integrarse en distintos lenguajes de programación como Python (26).

Se ha utilizado PostgreSQL a través de una imagen en Docker donde se ha configurado adecuadamente.

Figura 6: Logo PostgreSQL

4.6.2 MongoDB

MongoDB es un sistema de gestión de bases de datos NoSQL orientado a documentos. A diferencia de las bases de datos relacionales, MongoDB no utiliza tablas para almacenar los datos, sino que utiliza un modelo flexible de documentos tipo JSON que se pueden organizar en colecciones. Esta estructura sin esquema fijo permite una gran flexibilidad y escalabilidad en el almacenamiento de datos.

En MongoDB, las consultas se realizan utilizando el lenguaje de consultas basado en documentos de MongoDB, esto permite realizar consultas flexibles y potentes para buscar y recuperar datos específicos de las colecciones (27).

Figura 10: Logo MongoDB

4.7 MinIO

MinIO es un sistema de almacenamiento de objetos de código abierto que se utiliza comúnmente como una alternativa de almacenamiento en la nube.

La arquitectura de MinIO se basa en la distribución horizontal y la escalabilidad, lo que permite manejar grandes volúmenes de datos de manera eficiente. Proporciona una interfaz sencilla y unificada para el almacenamiento y acceso de objetos, lo que facilita el desarrollo de aplicaciones que requieren almacenamiento de objetos a gran escala (28).

4.8 Interfaces de acceso

Las interfaces de acceso simplifican la interacción con las bases de datos y ofrecen funcionalidades adicionales para administrar y analizar los datos. En este proyecto, se utilizaron diversas interfaces:

4.8.1 DBeaver

DBeaver es una herramienta de administración de bases de datos que ofrece una interfaz gráfica de usuario para la gestión y el análisis de bases de datos.

Esta herramienta es compatible con una amplia variedad de bases de datos, incluyendo PostgreSQL. Proporciona una serie de funciones útiles para la administración de bases de datos, como la visualización y edición de tablas, la ejecución de consultas SQL, la importación y exportación de datos, y la conexión a múltiples bases de datos simultáneamente (29).

DBeaver se ha empleado para visualizar, comprobar y examinar la base de datos creada en PostgreSQL.

4.8.2 MongoDB Compass

MongoDB Compass es una interfaz gráfica de usuario diseñada para facilitar la interacción con bases de datos MongoDB. Proporciona una forma intuitiva y visual de explorar y administrar los datos almacenados en una base de datos MongoDB.

Permite realizar consultas de manera visual. Permite visualizar los resultados de las consultas en forma de tablas o gráficos, lo que facilita la comprensión y análisis de los datos.

También ofrece la posibilidad de crear y modificar índices, que ayudan a mejorar el rendimiento de las consultas. Además, permite realizar operaciones de inserción, actualización y eliminación de documentos de manera sencilla (30).

4.8.3 Trino

Trino es un motor de consultas distribuido de código abierto diseñado para el análisis interactivo de datos a gran escala.

Trino utiliza un optimizador de consultas inteligente que busca el mejor plan de ejecución para cada consulta, optimizando el rendimiento y reduciendo los tiempos de respuesta. Puede distribuir la carga de trabajo en diferentes nodos de un clúster, lo que permite procesar grandes volúmenes de datos de forma paralela. Además, es compatible con diferentes lenguajes por lo que puede interactuar con Python (31).

Se ha utilizado Trino para realizar análisis interactivo de datos a gran escala.

Figura 7: Logo Trino

Capítulo 5. Desarrollo y Resultados del Trabajo

5.1 Introducción

En este capítulo se describirá la progresión del trabajo realizado, así como los resultados obtenidos. Esta parte ofrecerá una descripción completa de las etapas y métodos del proyecto, así como sus resultados y conclusiones.

5.2 Desarrollo del trabajo

El proyecto se ha desarrollado en el Grupo de Investigación Biomédica en Imagen del Hospital La Fe, en colaboración con el servicio de Radiología. Para poner a prueba los métodos desarrollados se han empleado los estudios de imagen disponibles en el repositorio PACS del hospital, proporcionando aproximadamente 175,000 archivos DICOM para su utilización en el proyecto. Estos archivos constituyen una cantidad significativa de datos médicos que han sido analizados y procesados en las etapas posteriores del proyecto.

El objetivo final es la obtención de información comparativa para la implantación de un sistema de almacenamiento analítico de imagen médica con el fin de que quede integrado en la práctica diaria del grupo de imagen médica.

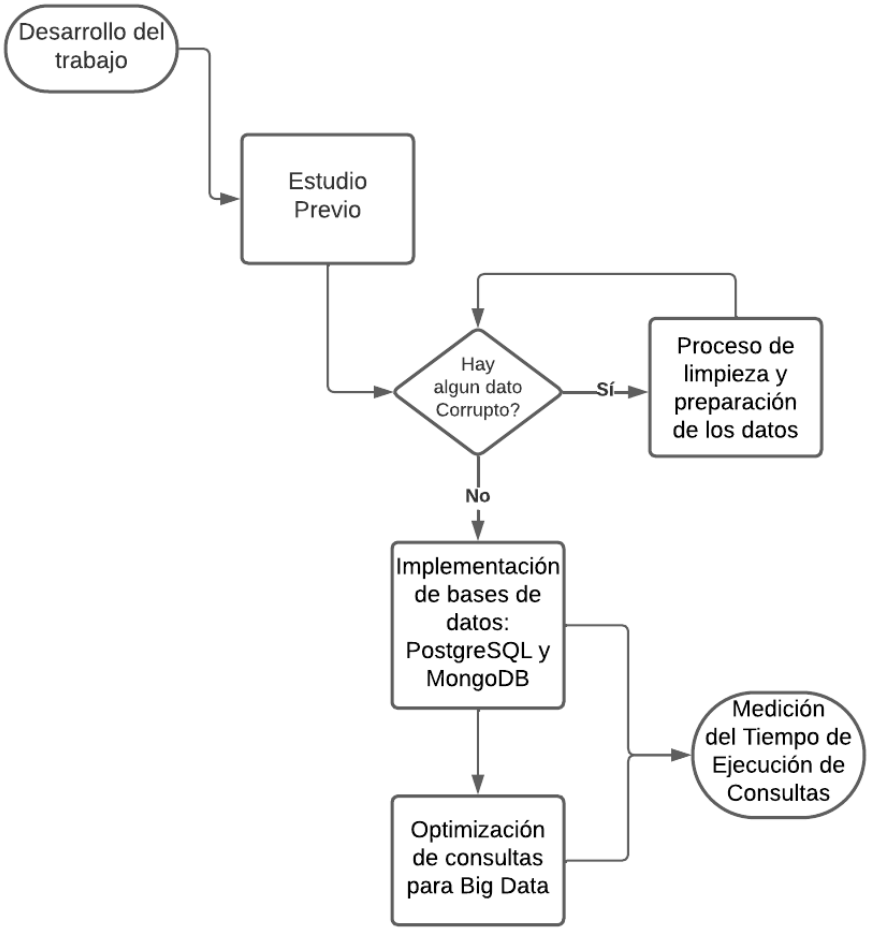


Figura 8: Diagrama de flujo del desarrollo del trabajo.

5.2.1 Estudio preliminar

Durante los primeros dos meses del proyecto, se ha realizado un estudio exhaustivo previo con el objetivo de obtener un entendimiento profundo de los requerimientos, objetivos y alcance del trabajo. Esta etapa ha sido fundamental para sentar las bases sólidas del proyecto y garantizar resultados.

En la fase de análisis preliminar, se han investigado diversas herramientas y tecnologías relevantes y fundamentales para el desarrollo del proyecto. Se ha invertido una gran cantidad de horas en la formación de cada una de ellas. En particular, se ha explorado el uso de Docker, que presenta una curva de aprendizaje elevada y es fundamental para el aprovechamiento de las demás herramientas. Como resultado, se ha adquirido un conocimiento sólido sobre su funcionamiento y utilidad. Se ha dedicado tiempo a comprender en profundidad cómo funciona Docker y a aprender a utilizarlo de manera efectiva.

En este proceso de formación, se ha adquirido experiencia práctica en la creación y configuración de imágenes personalizadas en Docker, aspecto de vital importancia. Se han explorado diversos comandos y conceptos, tales como la definición de Dockerfiles para especificar la configuración de la imagen, la instalación de dependencias, la configuración de variables de entorno y la exposición de puertos. Además, se han creado imágenes personalizadas de Docker para ejecutar aplicaciones específicas relacionadas con el procesamiento y análisis de imágenes médicas.

Adicionalmente, se ha profundizado en el uso de volúmenes en Docker, los cuales permiten almacenar y persistir datos fuera de los contenedores. Esta característica resulta especialmente valiosa al trabajar con aplicaciones que requieren almacenamiento de datos a largo plazo. Se ha comprendido cómo los volúmenes facilitan el intercambio de información entre diferentes componentes de la aplicación, permitiendo compartir y sincronizar datos entre contenedores.

Para abordar el manejo de los datos médicos en formato DICOM, se ha adquirido conocimiento sobre la estructura de estos archivos y se han investigado las mejores prácticas para su manipulación y procesamiento. Asimismo, se ha explorado el uso del lenguaje de programación Python y se ha comenzado a trabajar con archivos DICOM para obtener metadatos y explorar posibilidades de clasificación.

En la etapa inicial de trabajo, se ha utilizado Jupyter Notebook, con una cantidad reducida de datos, seleccionando menos del 10% de los archivos DICOM proporcionados para el proyecto. Mediante el uso de la biblioteca OS, se ha podido acceder a la ubicación de los archivos DICOM y recorrer los directorios para obtener una lista de los archivos disponibles. Esto ha permitido realizar un procesamiento automatizado de los archivos DICOM.

Una vez se han obtenido los nombres de los archivos DICOM, se ha podido utilizar la biblioteca PyDicom para leer y extraer los metadatos de cada archivo individualmente. Se ha comprendido la estructura de estos archivos, se han explorado las diferentes etiquetas y *tags* disponibles en los archivos DICOM y se ha utilizado PyDicom para acceder a los valores de estas.

Con los metadatos extraídos, se ha procedido a crear DataFrames en Python utilizando bibliotecas como pandas. Se ha utilizado cada etiqueta como columna en el DataFrame y se han asignado los valores correspondientes a cada una. Esto ha permitido organizar los datos de manera estructurada y ha facilitado su manipulación y análisis posterior (ver Figura 9).

Esta etapa de exploración inicial con Python y DataFrames ha proporcionado una visión general de los datos y ha permitido realizar operaciones simples, como la búsqueda y filtrado.

In [13]: df

Out[13]:

	Path	(0010, 2000)	(0020, 4000)	(0028, 0002)	(0040, 3001)	(0028, 0004)	(0008, 0005)	(0033, 1004)	(00e1, 1001)	(0008, 0008)	...	(01f7 0010)
0	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef1...			1		MONOCHROME2	ISO_IR 100	TAC RADIOTERAPIA EXTERNA	1	['ORIGINAL', 'PRIMARY', 'AXIAL', 'HELIX']	...	ELSCINT1
1	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef1...			1		MONOCHROME2	ISO_IR 100	TAC RADIOTERAPIA EXTERNA	1	['ORIGINAL', 'PRIMARY', 'AXIAL', 'HELIX']	...	ELSCINT1
2	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef1...			1		MONOCHROME2	ISO_IR 100	TAC RADIOTERAPIA EXTERNA	1	['ORIGINAL', 'PRIMARY', 'AXIAL', 'HELIX']	...	ELSCINT1
3	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef1...			1		MONOCHROME2	ISO_IR 100	TAC RADIOTERAPIA EXTERNA	1	['ORIGINAL', 'PRIMARY', 'AXIAL', 'HELIX']	...	ELSCINT1
4	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef1...			1		MONOCHROME2	ISO_IR 100	TAC RADIOTERAPIA EXTERNA	1	['ORIGINAL', 'PRIMARY', 'AXIAL', 'HELIX']	...	ELSCINT1
...	...	...	...	...	...	...	...	...	...	...	...	...
7495	E:\dipg\dipg\5ed05ff5b0141b280cb0f5b3\627a3240...	None	None	1	None	MONOCHROME2	ISO_IR 100	None	None	['ORIGINAL', 'PRIMARY', 'M', 'ND']	...	None
7496	E:\dipg\dipg\5ed05ff5b0141b280cb0f5b3\627a3240...	None	None	1	None	MONOCHROME2	ISO_IR 100	None	None	['ORIGINAL', 'PRIMARY', 'M', 'ND']	...	None
7497	E:\dipg\dipg\5ed05ff5b0141b280cb0f5b3\627a3240...	None	None	1	None	MONOCHROME2	ISO_IR 100	None	None	['ORIGINAL', 'PRIMARY', 'M', 'ND']	...	None
7498	E:\dipg\dipg\5ed05ff5b0141b280cb0f5b3\627a3240...	None	None	1	None	MONOCHROME2	ISO_IR 100	None	None	['ORIGINAL', 'PRIMARY', 'M', 'ND']	...	None
7499	E:\dipg\dipg\5ed05ff5b0141b280cb0f5b3\627a3240...	None	None	1	None	MONOCHROME2	ISO_IR 100	None	None	['ORIGINAL', 'PRIMARY', 'M', 'ND']	...	None

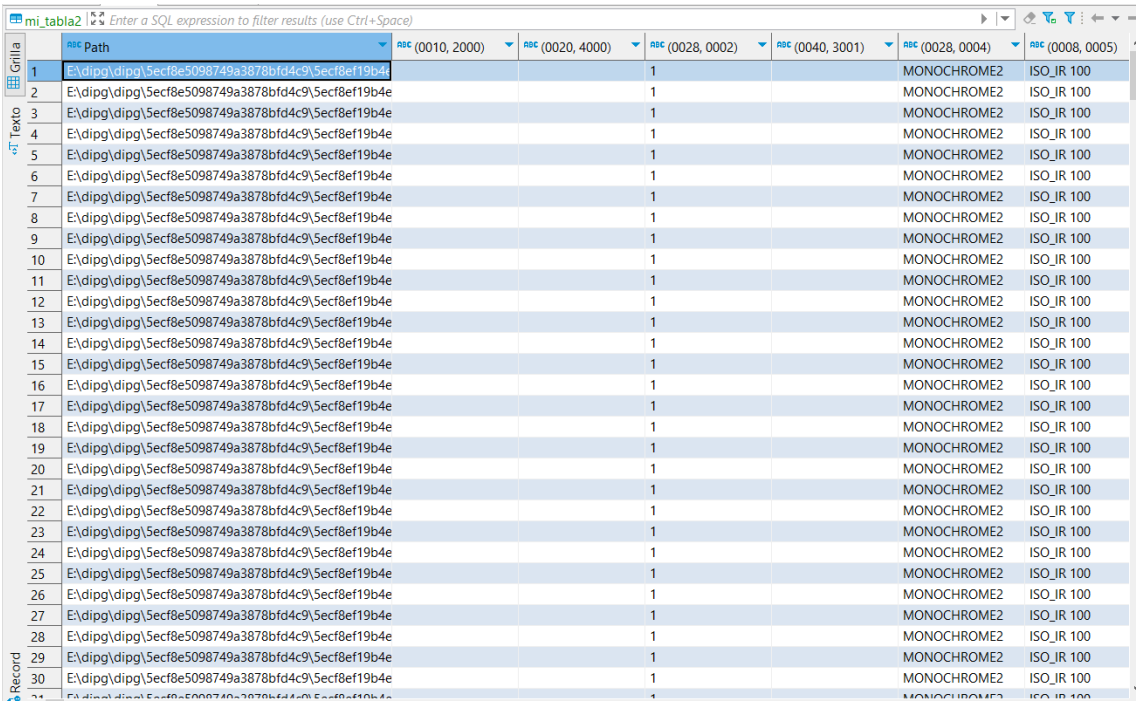
7500 rows × 587 columns

Figura 9: DataFrame 7500 documentos.

Después de haber creado el DataFrame con los metadatos de los archivos DICOM, el siguiente paso que se realizó fue convertir ese DataFrame en un archivo CSV. Esto es útil para almacenar los datos en un formato tabular, ver la estructura de los datos y medir unos primeros tiempos referencia a la hora de manejar unos pocos miles de archivos.

La siguiente acción fue crear una primera base de datos de prueba en PostgreSQL utilizando Docker para ver su funcionamiento. Sin embargo, surgieron problemas porque PostgreSQL rechazó algunos de los tipos de datos utilizados en valores de ciertas etiquetas a lo largo de este procedimiento. Este problema se resolvió al decidir convertir todos los datos en cadenas de texto antes de enviarlos a la base de datos.

Aunque esta solución implicó una transformación adicional de los datos y el uso de cadenas de texto en lugar de tipos de datos más específicos, permitió superar las limitaciones encontradas al trabajar con PostgreSQL y garantizar que los datos se pudieran almacenar correctamente en la base de datos (ver Figura 10).



mi_tabla2 | Enter a SQL expression to filter results (use Ctrl+Space)

	Path	(0010, 2000)	(0020, 4000)	(0028, 0002)	(0040, 3001)	(0028, 0004)	(0008, 0005)
1	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
2	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
3	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
4	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
5	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
6	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
7	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
8	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
9	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
10	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
11	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
12	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
13	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
14	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
15	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
16	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
17	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
18	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
19	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
20	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
21	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
22	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
23	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
24	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
25	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
26	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
27	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
28	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
29	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100
30	E:\dipg\dipg\5ecf8e5098749a3878bfd4c9\5ecf8ef19b4e			1		MONOCHROME2	ISO_IR 100

Figura 10: 7500 documentos en PostgreSQL

5.2.2 Proceso de limpieza y preparación de los datos

A continuación, se llevó a cabo la tarea de limpieza de datos, la cual se volvió necesaria al realizar pruebas con una mayor cantidad de documentos. Durante este proceso, se identificaron archivos corruptos que estaban generando errores en el sistema.

La limpieza de los datos, una etapa fundamental previa al procesamiento y análisis, se realizó con el objetivo de garantizar la integridad de los resultados y evitar posibles errores en las etapas posteriores del proyecto. Durante este proceso, se llevó a cabo una revisión exhaustiva de cada archivo DICOM para identificar y eliminar aquellos archivos corruptos que representaban un riesgo para la validez y confiabilidad del conjunto de datos.

De los 178,273 archivos totales, se determinó que 9,310 de ellos estaban dañados y no podían abrirse correctamente La limpieza de los datos fue realizada de manera automatizada mediante un script diseñado para recorrer todos los archivos DICOM del conjunto de datos. Durante este proceso, el script identificaba y detenía la ejecución en caso de encontrar un archivo dañado, permitiendo así su eliminación. Posteriormente, se realizaba una revisión manual de la carpeta en la que se encontraba dicho archivo para verificar la existencia de otros archivos corruptos. Una vez concluida la limpieza, se obtuvo un conjunto de datos depurado y apto para su posterior análisis.

5.2.3 Implementación de bases de datos: PostgreSQL y MongoDB

El siguiente objetivo fue crear dos bases de datos diferentes para comparar los resultados: una en PostgreSQL, una base de datos SQL, y otra en MongoDB, una base de datos NoSQL.

5.2.3.1 PostgreSQL

Para la base de datos PostgreSQL, se ha optado por utilizar Docker para facilitar su configuración y gestión. Se ha creado un archivo de composición de Docker, llamado *docker-compose.yml*, que

especifica la imagen de PostgreSQL a utilizar, así como el nombre de la base de datos, el usuario y la contraseña. Además, se han configurado los puertos a los que se debe conectar la base de datos. Para compartir los datos necesarios entre el contenedor de PostgreSQL y el entorno de desarrollo, se ha creado un volumen. Esto permite mantener la persistencia de los datos y facilita la transferencia de información entre diferentes componentes de la aplicación (ver Figura 11).

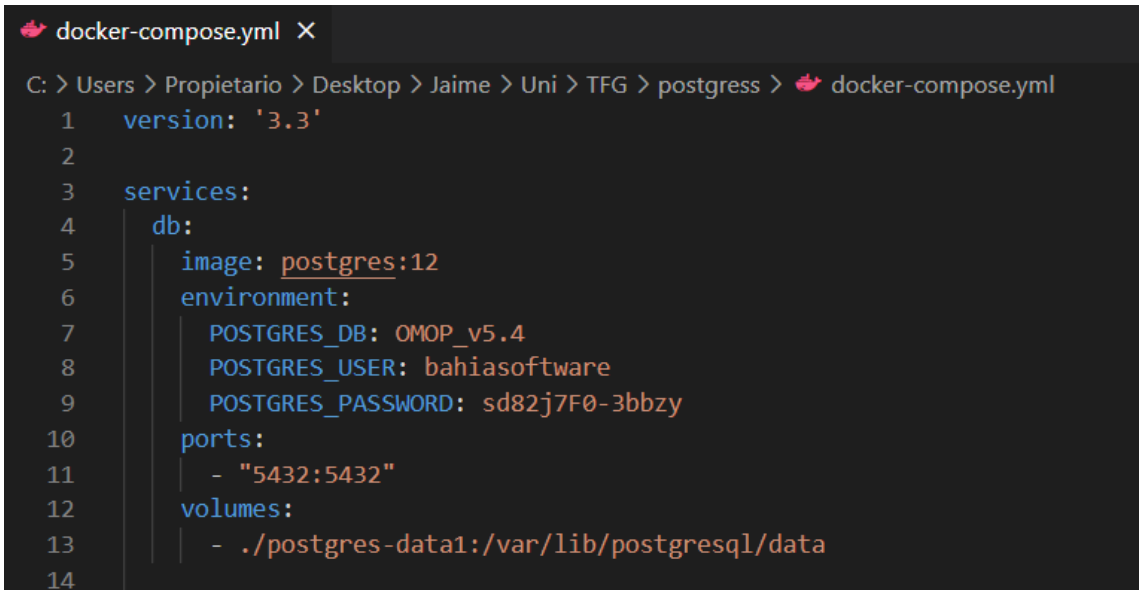


Figura 11: Configuración Docker-compose.yml.

Después de configurar y levantar (comando: docker-compose up) el contenedor de PostgreSQL utilizando Docker, el siguiente paso ha sido establecer una conexión entre Python y la base de datos. Para lograr esto, se ha utilizado la biblioteca psycopg2, que es una interfaz de Python para PostgreSQL.

Inicialmente, se intentó cargar todos los datos en la base de datos de una sola vez. Sin embargo, esto provocó un agotamiento de los recursos de memoria del ordenador debido al enfoque de almacenar todos los valores en un solo array. Para solucionar este problema, se han explorado otras opciones.

Uno de los enfoques ha sido realizar la carga de datos en grupos más pequeños, utilizando un enfoque de procesamiento por lotes (batch). Esto implicaba dividir los datos en grupos más manejables y cargarlos de forma incremental en la base de datos.

Sin embargo, se ha optado por otra solución que ha resultado más eficiente. En lugar de cargar los datos en lotes, se ha decidido añadir los archivos a la base de datos uno por uno. Esto se ha logrado utilizando ciclos y consultas individuales para cada archivo DICOM. Cada vez que se procesaba un archivo, se generaba una consulta SQL correspondiente para insertar los datos en la base de datos. Esta estrategia ha permitido manejar eficientemente grandes volúmenes de datos sin agotar los recursos del sistema.

Después de considerar cómo agregar los archivos DICOM a la base de datos, ha surgido otro problema relacionado con los diferentes conjuntos de etiquetas presentes en los archivos DICOM. Cada archivo puede tener un número variable de etiquetas, lo que dificulta la creación de una tabla con una estructura fija basada en el primer documento.

Para abordar este problema, se ha decidido realizar un recorrido inicial por todos los archivos DICOM y recopilar todas las etiquetas diferentes que se encuentren. Luego, se ha creado un documento especial que almacena los nombres de estas etiquetas. Por ejemplo, si el primer

archivo DICOM tenía alrededor de 300 etiquetas, se ha descubierto que en total hay más de 1700 etiquetas distintas en el conjunto de archivos.

Esta estrategia ha permitido obtener un conjunto completo de etiquetas que abarca todas las posibles variaciones encontradas en los archivos DICOM. Con esta información, se ha podido diseñar una tabla flexible que se ajuste a los diferentes conjuntos de etiquetas presentes en los archivos. La idea es que cada etiqueta se convierta en una columna de la tabla de la base de datos. Sin embargo, PostgreSQL tiene una limitación de 1600 columnas, por lo que se han seleccionado las primeras 1599 etiquetas más una columna adicional para almacenar la ruta del archivo (ver Figura 12 y Figura 13).

Column Name	#	Tipo de datos	Identidad	Collation	No Nulo	Por defecto	Comentario
(2005, 1359)	1.577	text		default	[ ]		
(2005, 130b)	1.578	text		default	[ ]		
(0019, 10ad)	1.579	text		default	[ ]		
(0021, 1058)	1.580	text		default	[ ]		
(0018, 0086)	1.581	text		default	[ ]		
(2005, 152c)	1.582	text		default	[ ]		
(0019, 1024)	1.583	text		default	[ ]		
(0029, 0013)	1.584	text		default	[ ]		
(2005, 132f)	1.585	text		default	[ ]		
(2005, 13c0)	1.586	text		default	[ ]		
(2001, 1201)	1.587	text		default	[ ]		
(2001, 100e)	1.588	text		default	[ ]		
(0029, 1018)	1.589	text		default	[ ]		
(2001, 1109)	1.590	text		default	[ ]		
(0040, 0243)	1.591	text		default	[ ]		
(2005, 1258)	1.592	text		default	[ ]		
(0021, 1178)	1.593	text		default	[ ]		
(01f7, 1074)	1.594	text		default	[ ]		
(0070, 0403)	1.595	text		default	[ ]		
(0040, 0254)	1.596	text		default	[ ]		
(0051, 0000)	1.597	text		default	[ ]		
(2001, 11cc)	1.598	text		default	[ ]		
(01f7, 1017)	1.599	text		default	[ ]		
(0018, 9307)	1.600	text		default	[ ]		

Figura 12: Algunos nombres y numero de columnas en PostgreSQL.

Al recopilar y almacenar los nombres de las etiquetas antes de la inserción de los archivos en la base de datos, se ha evitado la limitación de basar la estructura de la tabla en el primer documento. En cambio, se ha creado una tabla dinámica y adaptable que puede manejar eficientemente los diferentes conjuntos de etiquetas presentes en los archivos DICOM.

Este enfoque ha permitido un manejo más eficiente y preciso de los datos al tener en cuenta todas las etiquetas disponibles en los archivos DICOM, sin importar su variabilidad.

	path	(0008, 0060)
68	E:/dipg/dipg\5ecf94309b4e0113cca29ede\5ecfb1aadf94fd1d5c58ff73\1.2.840.113619.2.244.4120.8398801.11568.1319182608.815.dcm	MR
69	E:/dipg/dipg\5ecf94309b4e0113cca29ede\5ecfb1aadf94fd1d5c58ff73\1.2.840.113619.2.244.4120.8398801.11568.1319182608.816.dcm	MR
70	E:/dipg/dipg\5ecf94309b4e0113cca29ede\5ecfb1aadf94fd1d5c58ff73\1.2.840.113619.2.244.4120.8398801.11568.1319182608.817.dcm	MR
71	E:/dipg/dipg\5ecf94309b4e0113cca29ede\5ecfb1aadf94fd1d5c58ff73\1.2.840.113619.2.244.4120.8398801.11568.1319182608.818.dcm	MR
72	E:/dipg/dipg\5ecf94309b4e0113cca29ede\5ecfb1aadf94fd1d5c58ff73\1.2.840.113619.2.244.4120.8398801.11568.1319182608.819.dcm	MR
73	E:/dipg/dipg\5ecf94309b4e0113cca29ede\5ecfb1aadf94fd1d5c58ff73\1.2.840.113619.2.244.4120.8398801.11568.1319182608.820.dcm	MR
74	E:/dipg/dipg\5ecf94309b4e0113cca29ede\5ecfb1aadf94fd1d5c58ff73\1.2.840.113619.2.244.4120.8398801.11568.1319182608.821.dcm	MR
75	E:/dipg/dipg\5ecf94309b4e0113cca29ede\5ecfb1aadf94fd1d5c58ff73\1.2.840.113619.2.244.4120.8398801.11568.1319182608.822.dcm	MR
76	E:/dipg/dipg\5ecf94309b4e0113cca29ede\5ecfb1aadf94fd1d5c58ff73\1.2.840.113619.2.244.4120.8398801.11568.1319182608.823.dcm	MR
77	E:/dipg/dipg\5ecf94309b4e0113cca29ede\5ecfb1aadf94fd1d5c58ff73\1.2.840.113619.2.244.4120.8398801.11568.1319182608.824.dcm	MR
78	E:/dipg/dipg\5ecf94309b4e0113cca29ede\5ecfb1aadf94fd1d5c58ff73\1.2.840.113619.2.244.4120.8398801.11568.1319182608.825.dcm	MR
79	E:/dipg/dipg\5ecf94309b4e0113cca29ede\5ecfb1aadf94fd1d5c58ff73\1.2.840.113619.2.244.4120.8398801.11568.1319182608.826.dcm	MR
80	E:/dipg/dipg\5ecf94309b4e0113cca29ede\5ecfb1aadf94fd1d5c58ff73\1.2.840.113619.2.244.4120.8398801.11568.1319182608.827.dcm	MR

Figura 13: Columnas 'path' y '(0008, 0060)' en PostgreSQL.

En total, el proceso de carga de la base de datos utilizando esta estrategia tomó aproximadamente 7.2 horas (25944.939 segundos). Si bien este tiempo puede variar dependiendo del rendimiento del sistema y del volumen de datos a procesar, se logró insertar con éxito todos los archivos DICOM en la base de datos, preservando la integridad de los datos y la estructura flexible de la tabla.

5.2.3.2 MongoDB

En cuanto a MongoDB, se ha descargado la base de datos y se ha utilizado localmente. Este proceso ha sido más sencillo debido a la experiencia previa con PostgreSQL. En el script de Python, simplemente se ha establecido la conexión con MongoDB y, luego, se han ido agregando los archivos uno por uno a la base de datos. La estructura flexible de los datos en MongoDB ha permitido evitar cualquier problema con los *tags*, ya que cada documento puede tener sus propias etiquetas con sus respectivos valores (ver Figura 14).

Una de las ventajas de MongoDB es su flexibilidad en cuanto a la estructura de los datos. Cada documento puede tener su propia estructura y conjunto de etiquetas, lo que facilita el manejo de archivos DICOM con conjuntos de etiquetas y valores diferentes.

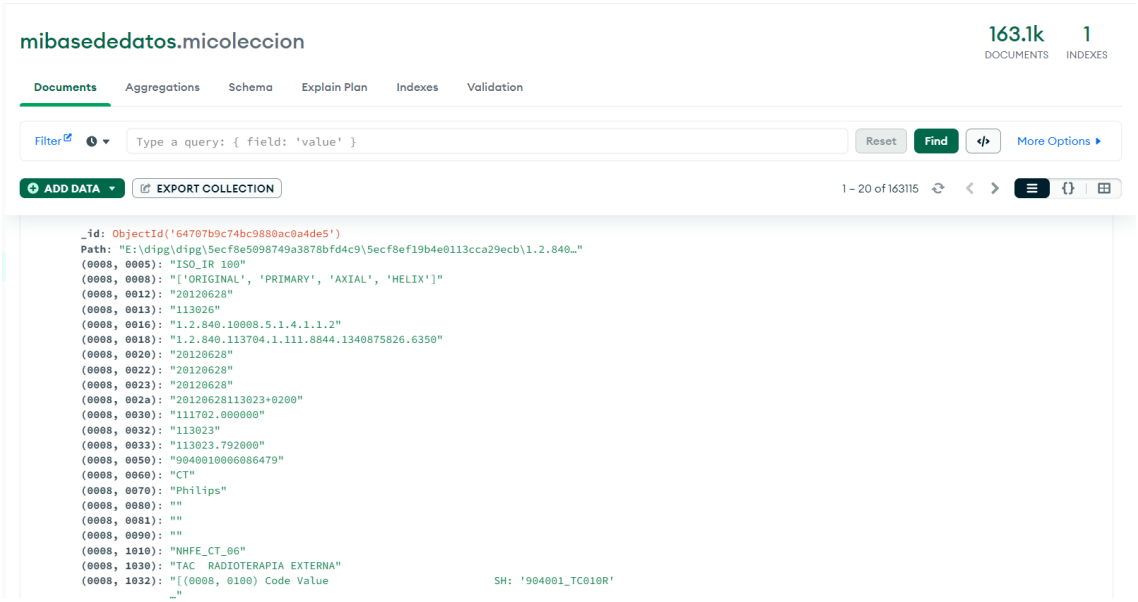


Figura 14: Ejemplo de documento en MongoDB.

Al final, el proceso de carga de los datos en la base de datos MongoDB llevó exactamente 8934.002 segundos, unas 2.48 horas.

5.2.4 Medición del Tiempo de Ejecución de Consultas en PostgreSQL y MongoDB

Una vez que se han construido ambas bases de datos, se ha procedido a realizar medidas sobre el tiempo de ejecución de consultas como parte del estudio. Se han analizado tanto PostgreSQL como MongoDB utilizando scripts de Python para llevar a cabo el análisis en cada una de ellas.

En el caso de PostgreSQL, se ha utilizado un script de Python para establecer la conexión con la base de datos y proponer una consulta SQL que recorra toda la tabla en busca de datos. La consulta utilizada es la siguiente (ver Figura 15):

```
query = "SELECT * FROM micoleccion WHERE \"(0008, 0060)\" = 'MR'"
```

Figura 15: Consulta PostgreSQL.

El objetivo de esta consulta era buscar todos los registros en la base de datos que tuvieran el valor "MR" (Resonancia Magnética) en la columna "(0008, 0060)".

Sin embargo, se ha encontrado que esta consulta genera errores de memoria, específicamente "Out of Memory", debido a la gran cantidad de datos a recuperar. Como resultado, se ha modificado la consulta para extraer solo la columna "path" de los registros que cumplen con la condición. La consulta final utilizada es la siguiente (ver Figura 16):

```
query = "SELECT \"path\" FROM micoleccion WHERE \"(0008, 0060)\" = 'MR'"
```

Figura 16: Consulta final PostgreSQL.

Para medir el tiempo de ejecución en PostgreSQL, se ha ejecutado la consulta múltiples veces, registrando el tiempo de inicio y finalización de cada ejecución. Se ha calculado el tiempo de ejecución para cada consulta y se ha obtenido la media y la desviación estándar de los tiempos registrados. Los resultados se mostrarán en la sección Resultados del trabajo.

En el caso de MongoDB, se ha utilizado un script de Python para establecer la conexión con la base de datos y proponer una consulta utilizando el filtro adecuado. La consulta en MongoDB se ha realizado utilizando el siguiente filtro (ver Figura 17):

```
filter = {"(0008, 0060)": 'MR'}
projection = {"path": 1}
```

Figura 17: Filtro para MongoDB.

Este filtro, al igual que la consulta en postgreSQL, tiene el objetivo de obtener el *path* de los documentos que tengan el valor “MR” en la etiqueta “(0008, 0060)”

Al igual que en PostgreSQL, se ejecutó la consulta en MongoDB múltiples veces, registrando el tiempo de inicio y finalización de cada ejecución. Se calculó el tiempo de ejecución para cada consulta y se obtuvo la media y la desviación típica de los tiempos registrados.

Estos cálculos permitieron comparar el rendimiento y la eficiencia de las consultas entre PostgreSQL y MongoDB.

5.2.5 Optimización de consultas para Big Data

En la etapa de Optimización de Consultas para Big Data, se ha implementado Trino, una herramienta de consulta distribuida diseñada para realizar consultas rápidas y eficientes en grandes conjuntos de datos. Para su implementación, se ha utilizado Docker para facilitar la configuración y gestión del entorno.

En el archivo de composición de Docker (docker-compose.yml), se ha añadido una imagen de Trino junto con la imagen de PostgreSQL. Los puertos correspondientes (8080:8080) se han configurado para establecer la comunicación con Trino. Además, se ha creado un volumen para compartir los catálogos necesarios entre el contenedor de Trino y el entorno de desarrollo. Esto ha permitido mantener la persistencia de los catálogos y ha facilitado la transferencia de información entre los componentes de la aplicación (ver Figura 18).

```
trino:
  image: trinodb/trino

  ports:
    - 8080:8080
  volumes:
    - C:/Users/Propietario/Desktop/Jaime/Uni/TFG/trino/properties:/etc/trino/catalog
```

Figura 18: Configuración de trino en el docker-compose.tml.

Los catálogos desempeñan un papel crucial en Trino, ya que definen la estructura y los metadatos de las fuentes de datos disponibles para las consultas. Se han creado archivos properties para cada catálogo, uno para MongoDB y otro para PostgreSQL. Estos archivos contienen la configuración necesaria para establecer la conexión con las respectivas bases de datos y definir las tablas, columnas y otros metadatos relevantes.

Aquí se muestra un ejemplo de los archivos *properties* utilizados para los catálogos de MongoDB (ver Figura 195) y PostgreSQL (ver Figura 206):

C: > Users > Propietario > Desktop > Jaime > Uni > TFG > trino > properties > mongodb.properties

```
1  connector.name=mongodb
2  mongodb.connection-url=mongodb://host.docker.internal:27017/mibasededatos
3  mongodb.schema-collection=collection_definition
```

Figura 19: Catalogo MongoDB para Trino.

C: > Users > Propietario > Desktop > Jaime > Uni > TFG > trino > properties > postgresql.properties

```
1  connector.name=postgresql
2  connection-url=jdbc:postgresql://db:5432/OMOP_v5.4
3  connection-user=bahiasoftware
4  connection-password=sd82j7F0-3bbzy
```

Figura 20: Catalogo PostreSQL para Trino.

En el caso de MongoDB, se ha procedido a definir el esquema de los datos utilizando el archivo que contenía los *tags* de los archivos DICOM. Se ha creado un archivo JSON denominado "collection_definition" que representa el esquema en MongoDB. Este archivo JSON define la estructura de los documentos en MongoDB, especificando los campos y los tipos de datos correspondientes. (ver Figura 217).

Figura 21: collection_definition (15 primeros campos).

Con la implementación de Trino, se han realizado nuevamente las consultas que previamente se llevaron a cabo en las bases de datos PostgreSQL y MongoDB. El objetivo ha sido evaluar el impacto de Trino en el rendimiento y el tiempo de ejecución de las consultas.

Utilizando Trino a través de scripts de Python, donde se ha conectado a ambos catálogos (ver Figura 22 y Figura 23) Se han ejecutado la misma consulta SQL (ver Figura 24) y se han tomado medidas del tiempo de ejecución en ambas bases de datos.

```
conn_mongodb = dbapi.connect(
    host='localhost',
    port=8080,
    catalog='mongodb',
    schema='mibasededatos',
    user='Jaime'
)
```

Figura 22: Conexión trino catalogo MongoDB

```
conn = dbapi.connect(
    host='localhost',
    port=8080,
    user='Jaime',
    catalog='postgresql',
    schema='public'
)
```

Figura 23: Conexión trino catalogo PostgreSQL.

```
query = "SELECT \"path\" FROM micoleccion WHERE \"(0008, 0060)\" = 'MR'"
```

Figura 24: Consulta SQL trino

5.3 Evaluación del rendimiento y eficiencia

Durante la fase de desarrollo del trabajo, se llevó a cabo una evaluación exhaustiva del rendimiento y la eficiencia de diferentes bases de datos, incluyendo PostgreSQL, MongoDB y varios motores de consulta, como Trino y los propios de cada base de datos. El objetivo principal de esta evaluación fue comprender cómo se comportaban estas bases de datos al ejecutar consultas y analizar el impacto de Trino como herramienta de motor de consultas en entornos de *big data*.

La misma consulta se ejecutó en cada una de las bases de datos un total de 5 veces, registrando el tiempo de ejecución en segundos. También se calculó la media y la desviación típica de los tiempos obtenidos para cada base de datos, utilizando el catálogo correspondiente.

En particular, se ha realizado una consulta que recorre toda la base de datos en busca de los documentos que cumplen ciertos criterios. Se eligió esta consulta específica debido a su naturaleza de exploración exhaustiva, que permite evaluar el rendimiento real de las bases de datos en la búsqueda y recuperación de datos. Se tomó en consideración que otro tipo de consultas podrían variar en función del orden en que los documentos estaban almacenados, lo que podría influir en el tiempo de ejecución.

Cabe mencionar que las mediciones se llevaron a cabo en un entorno de desarrollo específico, utilizando un ordenador con las siguientes especificaciones:

Procesador: Intel Core i7-7700HQ

Memoria RAM: 8 GB DDR4-2400

Tarjeta gráfica: NVIDIA GTX1050

Estas características del sistema pueden tener un impacto en el rendimiento y la velocidad de ejecución de las consultas en las bases de datos evaluadas. Es importante tener en cuenta que los resultados obtenidos pueden variar en función del hardware utilizado en diferentes entornos.

Antes de realizar la evaluación comparativa de las bases de datos, se llevó a cabo una búsqueda inicial utilizando un bucle en Python para recorrer los archivos deseados. Los tiempos de ejecución obtenidos en esta búsqueda fueron los siguientes:

Iteración	Tiempo de ejecución (s)
1	2872,190
2	2765,252
3	2928,541
4	2769,813
5	2914,239

Tabla 2: Tiempo de ejecución sin utilizar BBDD.

Calculando la media y la desviación típica de estos tiempos, se obtiene:

Métrica	Valor
Media (s)	2849.607
Desviación típica (s)	69.698

Tabla 3: Media y desviación típica sin utilizar BBDD.

Estos resultados proporcionan un punto de referencia inicial para comparar el rendimiento de las bases de datos evaluadas. A continuación, se analizarán los resultados obtenidos en PostgreSQL, MongoDB y Trino, y se compararán con el enfoque inicial para evaluar la eficiencia de cada base de datos en el procesamiento de consultas.

A continuación, se presentan los resultados obtenidos para la base de datos PostgreSQL:

Iteración	Tiempo de ejecución (s)
1	162.533
2	130.246
3	130.609
4	125.328
5	128.070

Tabla 4: Tiempo de ejecución utilizando PostgreSQL.

Métrica	Valor
Media (s)	135.337
Desviación típica (s)	11.712

Tabla 5: Media y desviación típica utilizando PostgreSQL.

Luego de realizar la comparativa entre PostgreSQL y la búsqueda sin utilizar una base de datos, se observa que utilizando PostgreSQL, el tiempo promedio para recuperar todos los documentos fue de aproximadamente 2 minutos. En contraste, cuando se realiza la búsqueda directamente a través del sistema de archivos del sistema operativo, el tiempo promedio fue de casi 50 minutos. Esto indica que hacer uso de una base de datos como PostgreSQL puede mejorar significativamente la eficiencia en la recuperación de datos.

Además, es importante mencionar que la creación de la base de datos en PostgreSQL tomó alrededor de 7.2 horas. En este caso, vale la pena considerar la utilización de una base de datos cuando se requiere realizar múltiples consultas posteriores. Aunque la creación inicial puede llevar más tiempo, una vez que la base de datos está configurada, las consultas posteriores pueden ejecutarse de manera más rápida y eficiente.

Si se requiere realizar un par de consultas aisladas, puede ser más conveniente realizar la búsqueda sin utilizar una base de datos. Sin embargo, cuando se prevé la necesidad de realizar múltiples consultas o consultas recurrentes, la creación y utilización de una base de datos, como PostgreSQL, puede proporcionar un mejor rendimiento y una mayor rapidez en la recuperación de datos. En este caso, la creación inicial de la base de datos puede llevar más tiempo, pero las consultas posteriores se ejecutarán de manera más eficiente y rápida.

A continuación, se presentan los resultados obtenidos para la base de datos MongoDB:

Iteración	Tiempo de ejecución (s)
1	0,001
2	0,02
3	0,006
4	0,002
5	0,005

Tabla 6: Tiempo de ejecución utilizando MongoDB.

Métrica	Valor
Media (s)	0,007
Desviación típica (s)	0,007

Tabla 7: Media y desviación típica utilizando MongoDB.

Al analizar los resultados de MongoDB, se observa que los tiempos de ejecución son considerablemente más bajos en comparación con PostgreSQL. El tiempo promedio para

recuperar todos los documentos utilizando MongoDB fue de tan solo 0,007 segundos, con una desviación típica de 0,007 segundos. Esto indica que MongoDB ofrece un rendimiento rápido y eficiente en la recuperación de datos.

En comparación con la búsqueda sin utilizar una base de datos, donde se tardaron casi 50 minutos, MongoDB demuestra ser extremadamente rápido en la ejecución de consultas. Incluso en comparación con PostgreSQL, MongoDB muestra un rendimiento notablemente superior.

Ante estos tiempos de ejecución tan rápidos obtenidos en MongoDB, se llevó a cabo una búsqueda para descartar cualquier posible error en el código o anomalía en los resultados. Se consultaron diversas fuentes y referencias especializadas, y se concluyó que los tiempos registrados en MongoDB son consistentes y se consideran normales para este tipo de base de datos (32) (33).

Además, es relevante mencionar que la creación de la base de datos en MongoDB tomó aproximadamente 2,48 horas, lo cual es considerablemente más rápido que la creación de la base de datos en PostgreSQL, que llevó alrededor de 7,2 horas. Esta diferencia en el tiempo de creación puede ser un factor a tener en cuenta al elegir una base de datos.

En base a los resultados obtenidos, se concluye que MongoDB es una opción muy favorable si se busca una recuperación ágil de datos y un rendimiento óptimo en consultas. Los tiempos de ejecución rápidos y consistentes, junto con el tiempo de creación de la base de datos más corto, respaldan la elección de MongoDB como una base de datos eficiente y rápida para el procesamiento de consultas.

A continuación, se presentan los resultados obtenidos utilizando el motor de consulta Trino:

Trino (catálogo PostgreSQL):

Iteración	Tiempo de ejecución (s)
1	91.017
2	90.634
3	89.074
4	92.416
5	87.110

Tabla 8: Tiempo de ejecución utilizando Trino con catalogo PostgreSQL.

Métrica	Valor
Media (s)	90.650
Desviación típica (s)	2.439

Tabla 9: Media y desviación típica utilizando Trino con catálogo PostgreSQL.

Trino (catálogo MongoDB):

Iteración	Tiempo de ejecución (s)
1	462.743
2	492.791
3	624.962
4	452.926
5	420.041

Tabla 10: Tiempo de ejecución utilizando Trino con catalogo MongoDB.

Métrica	Valor
Media (s)	490.493
Desviación típica (s)	74.899

Tabla 11: Media y desviación típica utilizando Trino con catálogo MongoDB.

Después de analizar los resultados obtenidos, se puede observar una notable reducción en el tiempo de ejecución en comparación con la búsqueda sin utilizar ninguna base de datos. Tanto Trino (catálogo PostgreSQL) como Trino (catálogo MongoDB) logran mejorar significativamente la eficiencia en la recuperación de datos.

En el caso de Trino utilizando el catálogo de PostgreSQL, se observa una disminución considerable en el tiempo de ejecución en comparación con la búsqueda sin base de datos. Esto indica que las consultas realizadas a través de Trino y utilizando el catálogo de PostgreSQL son más eficientes y rápidas.

Por otro lado, al utilizar Trino con el catálogo de MongoDB, se observa un aumento significativo en el tiempo de ejecución en comparación con la búsqueda sin Trino. El tiempo promedio para recuperar todos los documentos utilizando Trino y el catálogo de MongoDB fue de más de 8 minutos, en contraste con las milésimas de segundo obtenidas sin Trino. Se cree que este incremento se debe al hecho de que, al utilizar Trino, fue necesario esquematizar la base de datos de MongoDB para poder realizar consultas SQL. Esta esquematización puede haber afectado el rendimiento y el tiempo de ejecución de las consultas.

Es importante tener en cuenta que, en este caso, los resultados se obtuvieron utilizando solo un ordenador. Si se hubiese utilizado un clúster con múltiples máquinas, es probable que el tiempo de ejecución se hubiese reducido aún más. El uso de un clúster en Trino permite distribuir las tareas entre varias máquinas, lo que mejora la capacidad de procesamiento y reduce los tiempos de ejecución de las consultas.

Capítulo 6. Conclusiones y propuesta de trabajo futuro

6.1 Introducción

En este apartado se presentan las conclusiones derivadas del trabajo realizado, destacando los principales hallazgos y logros alcanzados. Se resumen los resultados obtenidos y se discuten en relación con los objetivos planteados. Asimismo, se plantea una propuesta de trabajo futuro, identificando posibles áreas de mejora o ampliación del proyecto.

6.2 Conclusiones

En el transcurso de este proyecto, se han logrado alcanzar los objetivos planteados, que consistían en la creación de bases de datos, la realización de una comparativa de rendimiento y la aplicación de herramientas de procesamiento de datos, particularmente enfocadas en el uso de Trino como motor de consulta. A través de estas etapas, se han obtenido resultados significativos que contribuyen a la comprensión y optimización de los sistemas de gestión de datos.

Se ha observado que MongoDB ha demostrado ser más eficiente tanto en la fase de creación como en el filtrado de datos, en comparación con PostgreSQL. La flexibilidad y escalabilidad de MongoDB en términos de almacenamiento y consulta de datos no estructurados ha permitido una manipulación ágil y eficaz de grandes volúmenes de información. Sin embargo, al aplicar Trino como motor de consulta, se ha obtenido un mejor rendimiento utilizando el catálogo de PostgreSQL. Esto se debe a que Trino requiere consultas SQL, las cuales no son compatibles de forma nativa con MongoDB. Por lo tanto, ha sido necesario realizar un proceso adicional de esquematización de los datos de MongoDB para que puedan ser consultados eficientemente por Trino.

6.3 Propuesta de trabajo futuro

Como propuesta de trabajo futuro, se sugiere explorar la implementación de un clúster de Trino con un mayor número de máquinas. Esto permitiría distribuir la carga de trabajo y mejorar significativamente los tiempos de ejecución, especialmente al procesar conjuntos de datos más grandes y complejos. Al aumentar la capacidad de procesamiento y distribución de consultas, se podrían obtener resultados aún más rápidos y precisos, lo que impulsaría la eficiencia general del sistema.

Además, se propone la incorporación de MinIO como herramienta para el manejo de datos en el entorno de trabajo. MinIO es una solución de almacenamiento en la nube de objetos que ofrece escalabilidad y acceso rápido a los datos. Al aprovechar las capacidades de MinIO, se podrían optimizar los procesos de almacenamiento y manipulación de datos, lo que contribuiría a mejorar la eficiencia y el rendimiento general del sistema.

Capítulo 7. Bibliografía

1. iambiomed. [En línea] [Citado el: 10 de 06 de 2023.] http://www.iambiomed.com/specialization/pacs.php.

2. Aiello, M., y otros. *The Challenges of Diagnostic Imaging in the Era of Big Data.* [En línea] 2019. [Citado el: 10 de 06 de 2023.] https://doi.org/10.3390/jcm8030316.

3. ERNESTINA MENASALVAS, CONSUELO GONZALO, ALEJANDRO RODRÍGUEZ-GONZÁLEZ. *BIG DATA EN SALUD: RETOS Y OPORTUNIDADES.* s.l. : Universidad Politécnica de Madrid.

4. Oracle. *What Is a Database?* [En línea] [Citado el: 10 de 06 de 2023.] https://www.oracle.com/database/what-is-database/.

5. Oracle. *What is NoSQL?* [En línea] [Citado el: 10 de 06 de 2023.] https://www.oracle.com/database/nosql/what-is-nosql/.

6. Devopedia. *MongoDB Query Language.* [En línea] [Citado el: 10 de 06 de 2023.] https://devopedia.org/mongodb-query-language.

7. slideshare. *IN Unidad 1: Introducción a la inteligencia de negocios.* [En línea] [Citado el: 10 de 06 de 2023.] https://www.slideshare.net/fparrale/in-unidad-1-introduccin-a-la-inteligencia-de-negocios?cv=1&session-id=0296ad6ba09c48648cec5064dd07739d.

8. Oracle. *What is Big Data?* [En línea] [Citado el: 10 de 06 de 2023.] https://www.oracle.com/big-data/what-is-big-data/.

9. DICOM. [En línea] [Citado el: 20 de 02 de 2023.] https://www.dicomstandard.org.

10. Innolitics. *DICOM Standad Browser.* [En línea] [Citado el: 15 de 06 de 2023.] https://dicom.innolitics.com/ciods/mr-image/general-series/00080060.

11. Python. *What is Python? Executive Summary.* [En línea] [Citado el: 21 de 02 de 2023.] https://www.python.org/doc/essays/blurb/.

12. Python. *OS.* [En línea] [Citado el: 21 de 02 de 2023.] https://docs.python.org/3/library/os.html.

13. Numpy. *What is Numpy.* [En línea] [Citado el: 21 de 02 de 2023.] https://numpy.org/devdocs/user/whatisnumpy.html.

14. Pydata. *Pandas Getting started.* [En línea] [Citado el: 21 de 02 de 2023.] https://pandas.pydata.org/docs/getting_started/index.html#getting-started.

15. Python. *time.* [En línea] [Citado el: 05 de 06 de 2023.] https://docs.python.org/3/library/time.html.

16. Python. *CSV.* [En línea] [Citado el: 22 de 02 de 2023.] https://docs.python.org/3/library/csv.html.

17. PyDicom. [En línea] [Citado el: 21 de 02 de 2023.] URL: https://pypi.org/project/pydicom/.

18. Joblib. *Joblib: running Python functions as pipeline jobs.* [En línea] [Citado el: 22 de 02 de 2023.] https://joblib.readthedocs.io/en/stable/#.

19. Pypi. *psycopg2.* [En línea] [Citado el: 13 de 03 de 2023.] https://pypi.org/project/psycopg2/.

20. SQLAlchemy. [En línea] [Citado el: 13 de 03 de 2023.] https://www.sqlalchemy.org.

21. MongoDB. *PyMongo.* [En línea] [Citado el: 17 de 04 de 2023.] https://api.mongodb.com/python/3.3.1/index.html.

22. Github. *Trinodb.* [En línea] [Citado el: 05 de 06 de 2023.] https://github.com/trinodb/trino-python-client.

23. Jupyter. *Project Jupyter Documentation.* [En línea] [Citado el: 20 de 02 de 2023.] https://docs.jupyter.org/en/latest/.

24. Oracle. *What is Docker?* [En línea] [Citado el: 6 de 02 de 2023.] https://www.oracle.com/cloud/cloud-native/container-registry/what-is-docker/.

25. Docker. *Docker Docs.* [En línea] [Citado el: 6 de 02 de 2023.] https://docs.docker.com/get-started/.

26. PostgreSQL. *About.* [En línea] [Citado el: 10 de 04 de 2023.] URL: https://www.postgresql.org/about/.

27. MongoDB. *What Is MongoDB?* [En línea] [Citado el: 17 de 04 de 2023.] https://www.mongodb.com/en/what-is-mongodb.

28. MinIO. *Docs.* [En línea] [Citado el: 25 de 06 de 2023.] https://min.io/docs/minio/kubernetes/upstream/.

29. Dbeaver. *About.* [En línea] [Citado el: 24 de 04 de 2023.] https://dbeaver.io/about/.

30. MongoDB. *What is MongoDB Compass?* [En línea] [Citado el: 17 de 04 de 2023.] https://www.mongodb.com/docs/compass/current/?_ga=2.249297346.475354277.1687283650-1097129599.1684925090.

31. Trino. *Docs.* [En línea] [Citado el: 05 de 06 de 2023.] https://trino.io/docs/current/index.html.

32. Pérez Román, Alberto. UNIVERSIDAD DE MÁLAGA. *Comparación de rendimiento entre bases de datos Relacionales, NoSQL y Blockchain.* [En línea] 17 de 03 de 2020. [Citado el: 25 de 06 de 2023.] https://riuma.uma.es/xmlui/bitstream/handle/10630/19413/Pérez%20Román%20Alberto%20Memoria.pdf?sequence=1&isAllowed=y.

33. Politowsk, Cristiano y Maran, Vinícius. Universidade Regional do Noroeste do Estado do Rio Grande do Sul. *Comparaçao de Performance entre PostgreSQL e MongoDB.* [En línea] [Citado el: 25 de 06 de 2023.] https://turing.pro.br/anais/ERBD-2014/artigos_aceitos/trilha_pesquisa/124500_1.pdf.