Universidad del País Vasco Euskal Herriko Unibertsitatea

BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA EN TECNOLOGÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO DE UNA PLATAFORMA BIG DATA PARA PREDICCIÓN DE PATOLOGÍAS A PARTIR DE RESULTADOS MÉDICOS

Alumno: Iñigo Gutierrez Narbaiza

Director: Koldo Espinosa Acereda

Curso: 2019-2020

Fecha: Bilbao, Febrero de 2020

Resumen/Abstract/Laburpena

Resumen

El fin de este proyecto es implementar una plataforma Big Data orientada al análisis médico. El principal objetivo es realizar un análisis de ciertos factores físicos de una enfermedad con el fin de facilitar su estudio, diagnosis y prevención. Para ello, se realiza un estudio de las plataformas Big Data, a continuación, se presentan varias herramientas para cada etapa de una plataforma Big Data. Haciendo uso del estudio realizado, se realiza la elección de las herramientas más eficientes para cada fase. Finalmente, se implementa un caso de uso comprobando así el correcto funcionamiento de nuestra plataforma.

Abstract

The purpose of this project is to implement a Big Data platform directed to medical analysis. The main objective is to perform an analysis of certain physical factors of a disease in order to facilitate its study, diagnosis and prevention. For this, a study of Big Data platforms is carried out, below several tools are presented for each stage of a Big Data platform. Using the study carried out, the most efficient tools are chosen for each phase. Finally, a use case is implemented, thus checking the correct functioning of our platform.

Laburpena

Proiektu honen helburua azterketa medikoetara bideratutako Big Data plataforma ezartzea da. Helburu nagusia gaixotasun baten zenbait faktore fisikoetako azterketa bat egitea da, azterketa, diagnostikoa eta prebentzioa errazteko. Horretarako, Big Data plataformetako azterketa egiten da, ondoren Big Data plataforma baten fase bakoitzeko hainbat tresna aurkeztuko dira. Egindako azterketa erabiliz, tresna eraginkorrenak fase bakoitzerako aukeratzen dira. Azkenik, erabilera kasu bat inplementatzen da, horrela gure plataformaren funtzionamendu egokia egiaztatzen da.

Universidad del País Vasco | Euskal Herriko Unibertsitatea

BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

Contenido

Lista de ilustraciones

1. Introducción

1.1 ¿Qué es el Big Data?

El ser humano se trata de una especie social, racional y dotada de lenguaje, lo que conllevó a que comenzara a recopilar información de manera sistemática y de forma consciente. En un principio se recopilaba de forma analógica, pero una vez aparecieron los primeros ordenadores, se comenzó a trasladar esta información analógica a bases digitales, lo cual presentaba ciertas ventajas como la reducción del espacio físico ocupado por los datos, mayor seguridad y mayor facilidad a la hora de clasificarlos.

La aparición de nuevas tecnologías como los smartphones o las redes sociales, han supuesto un aumento masivo de los datos generados. De tal forma que las herramientas tradicionales no son capaces de gestionarlos, quedándose estas herramientas obsoletas. Por lo tanto, se comienzan a investigar otras metodologías y herramientas para poder procesar tales cantidades de datos, creando de este modo el Big Data.

1.2 Las 5V's

Debido a la gran cantidad de datos que se analizan, la mayoría de los analistas y profesionales consideran macrodatos a conjuntos de datos que van desde 30-50 Terabytes a varios Petabytes. Al tratarse de una cantidad tan voluminosa de datos se debe tener en cuenta la calidad de los mismos, es por ello que el Big Data cuenta con cinco características esenciales, las 5 Vs(1):

Volumen: la cantidad de datos generados y guardados.
Variedad: el tipo y naturaleza de los datos para ayudar a las personas a analizar los datos y usar los resultados de forma eficaz. Los macrodatos usan textos, imágenes, audio y vídeo.
Velocidad: la velocidad a la cual se generan y procesan los datos para cumplir las exigencias y desafíos de su análisis.
Veracidad: la calidad de los datos capturados puede variar mucho y así afectar a los resultados del análisis.
Valor: los datos generados deben ser útiles, accionables y tener valor.

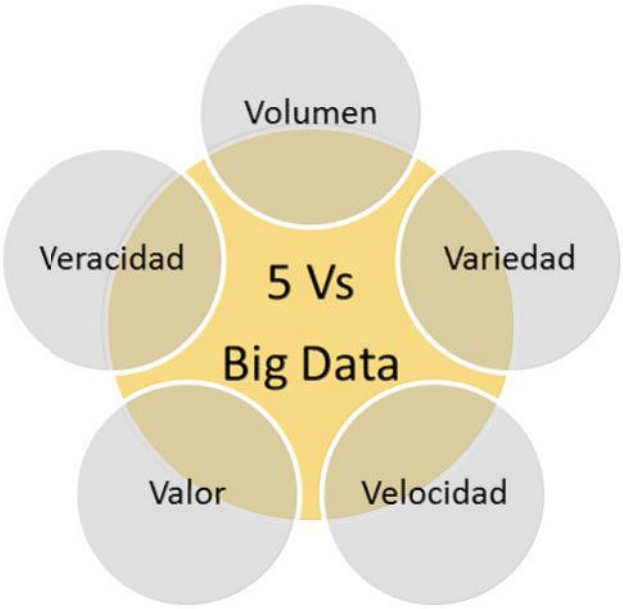


Ilustración 1- Las 5 Vs del Big Data

1.3 Estructura del documento

El documento comienza con una contextualización del trabajo en el mundo actual, seguido de los objetivos en los que se enfoca y los posibles beneficios que aporta, técnicos, económicos y sociales.

A continuación, se muestra el estado del arte, es decir, se analizan los distintos usos que se aplican a las actuales herramientas y tecnologías.

Tras ello, se procede a clasificar y analizar las distintas alternativas existentes, alternativas estudiadas anteriormente en el estado del arte, se concluye este apartado con la elección final de ciertas herramientas y metodologías específicas.

Seguidamente, se expone tanto el diseño como la descripción de la solución propuesta.

En siguiente lugar, se realiza una descripción de las tareas realizadas y se confecciona un diagrama de Gantt, el cual muestra las tareas de manera secuencial y temporal.

Finalmente, se muestran unas propuestas de mejora como una serie de conclusiones acerca del trabajo.

2. Contexto

2.1. ¿Por qué es bueno utilizar Big Data?

Como se ha comentado anteriormente, debido a la aparición de nuevas tecnologías, la cantidad de datos generados ha aumentado masivamente en los últimos años.

A día de hoy, las grandes empresas de almacenamiento de datos están utilizando capacidades para este almacenamiento cercanas a los 10 Zettabytes. El 80% de dichos datos se han generado en los últimos años. Por lo que el uso de una tecnología que proporciona ventajas tanto en volumen, variedad, velocidad, veracidad y valor, supondrá una mejor gestión de dichos datos.

El Big Data nos ofrece reducción de coste, las grandes tecnologías de datos, aportan importantes ventajas en términos de costes cuando se trata de almacenar grandes cantidades de datos, además de identificar maneras más eficientes de hacer negocios.

Con la velocidad y la capacidad de análisis de ciertas herramientas, combinada con la capacidad de analizar nuevas fuentes de datos, las empresas pueden analizar la información inmediatamente y tomar decisiones basadas en lo que han aprendido.

2.2. Necesidad del Big Data

La digitalización de las compañías y, la evolución de la tecnología y los mercados, han convertido a la banda ancha y a Internet en dos elementos fundamentales del desarrollo de las sociedades. Casi el 50% de las grandes empresas considera fundamental la inversión en soluciones digitales dentro de la estrategia de innovación de su empresa, y la otra mitad que "es bastante importante". Mientras que, según datos del Informe de la Sociedad de la Información 2016 (SIE) de Fundación Telefónica, tan solo el 6% piensa que este tipo de inversiones tiene una importancia relativa.(2)

En los últimos años el uso de la red y la digitalización ha sido un factor clave en el avance de la sociedad, de modo que el número de usuarios de la red aumenta de forma exponencial. Este hecho ha posibilitado a las empresas a recopilar distinta información de sus usuarios, de forma que pudiesen ofertar productos más acordes a sus necesidades o los conocidos sistemas de recomendación.

A parte de los usos comerciales, se pueden encontrar una infinidad de prácticas respecto a esta tecnología. A continuación, se expondrán ciertos ejemplos:

Predicción de catástrofes: Las grandes cantidades de datos disponibles se utilizan en la detección de eventos como incendios o terremotos, de tal manera que se pueda predecir su impacto y generar una reacción temprana.

Categorización y reconocimiento: De lugares, caras o personas, mediante el análisis del gran volumen de datos de este tipo disponible online.

Medicina: La medicina genómica personalizada (aún en el campo de la investigación) analiza e integra datos genómicos y clínicos para el diagnóstico precoz y una mejor aplicación de las terapias.

Comportamiento inteligente de servicios públicos: Utilizando la información proveniente de datos recopilados por sensores inteligentes puede mejorarse la distribución y consumo de recursos fundamentales como el agua o la energía eléctrica.

Investigación y desarrollo: Algunas empresas con fuerte componente investigadora, como las farmacéuticas, realizan análisis de grandes volúmenes de documentación (por ejemplo, artículos científicos) y otro tipo de datos históricos para mejorar el desarrollo de sus productos.

2.3. Situación del trabajo en el mundo actual

Los datos recogidos provienen de distintas fuentes, tradicionalmente en la medicina se ha basado el análisis de datos en métodos cuantitativos, debido a su sencillez respecto al análisis cualitativo.

La dificultad del análisis cualitativo proviene de la obtención de los datos, el hecho de no tener una estructura definida, el número de datos disponibles o la incertidumbre a la hora de almacenar dichos datos.

El uso del Big Data, haciendo uso de las distintas herramientas y tecnologías que nos ofrece, nos da la opción de poder gestionar dichos datos desde otra perspectiva.

2.4. Posibles usos del proyecto

Los datos en el sector salud sobre los que poder aplicar técnicas analíticas de Big Data son muy variados. La información extraída de todos estos datos tiene el potencial de lograr una medicina más eficaz.

2.4.1. Medicina poblacional

Anteriormente, la investigación en salud se ha centrado en analizar a ciertos grupos de personasrepresentantes de la población general y extrapolar los resultados obtenidos

al resto de la población. Este enfoque empezó a cuestionarse con el desarrollo de las tecnologías de análisis del genoma humano. Sin embargo, ambas aproximaciones están resultando ser complementarias.

La medicina poblacional en este sentido ahorra tiempo y costes gracias a las tecnologías Big Data, porque analizar los datos puede ayudar a detectar aquellas situaciones en las que la mejor opción sería aplicar un tratamiento común para toda una población o grupo de alto riesgo, como en el caso de las enfermedades de elevada frecuencia.

2.4.2. Medicina preventiva

Conseguir identificar y anticiparse a las necesidades de los pacientes, los centros sanitarios o los laboratorios clínicos. Analizar el impacto de lo social y el estudio de los históricos de datos permite definir políticas preventivas. Ejemplos de estas políticas pueden ser:

Campañas de salud ambiental y laboral.
Estudios de la probabilidad de la aparición o progresión de una enfermedad.
Vigilancia de brotes epidémicos y situaciones de emergencia sanitaria.

La medicina preventiva, entendida así como promoción de la salud, parte del análisis de datos para aumentar el bienestar social en todos los sentidos, ya que nos hace conscientes de nuestro estado de salud física y mental.

2.4.3. Medicina participativa

El anteriormente mencionado auge del internet posibilita la creación y participación de comunidades online de salud. La influencia online y la capacidad para desarrollar propuestas de forma conjunta basadas en técnicas Big Data de análisis de texto libre, análisis del sentimiento y normalización y codificación automática de datos médicos.

Con la información que genera el análisis de los datos, el individuo sano decide sobre su salud y el paciente decide sobre su enfermedad. Se consigue con ello una mayor implicación de todos los afectados, lo que facilita entre otros un mejor diagnóstico y una mayor adherencia a los tratamientos.

3. Objetivos y alcance

3.1. Objetivo principal

El objetivo principal del trabajo es elaborar el diseño de una plataforma que une las diferentes etapas que se pueden encontrar en un diseño de Big Data, con el fin de almacenar y analizar datos médicos. Para ello se realizará tanto un estudio teórico del Big Data como el diseño de una solución al problema.

3.1.1. Estudio teórico

Mediante el estudio teórico se busca crear una base sólida acerca del Big Data, necesaria para poder elegir correctamente las diferentes herramientas utilizadas en cada fase de la solución.

3.1.2. Diseño de una solución

El objetivo es diseñar una solución y comprobar su correcto funcionamiento mediante casos de uso, para ello se debe diseñar una arquitectura que conecte las diferentes herramientas.

3.2. Objetivos secundarios

3.2.1. Implementar un caso de uso que soporte la solución diseñada

A la hora de comprobar el funcionamiento del diseño planteado, se realiza un caso de uso, el cual aparte de demostrar el correcto funcionamiento, nos aporte un valor para mejorar los análisis médicos.

3.2.2. Comprobar el uso correcto en el análisis cuantitativo y cualitativo

Uno de los objetivos secundarios del presente proyecto es poder analizar todo tipo de datos, tanto de manera cualitativa como cuantitativa, para poder realizar un análisis más eficaz de todo tipo.

4. Beneficios que aporta el trabajo

A la hora de comentar los posibles beneficios del trabajo, se deben tener en cuenta tres aspectos: el beneficio técnico, el beneficio económico y el beneficio social.

4.1. Beneficios técnicos

Los beneficios técnicos del diseño de una plataforma Big Data enfocada a la resolución de problemas del ámbito médico son, obtener una herramienta capaz de realizar un análisis técnico de diferentes fuentes de datos, mostrar el funcionamiento de las diferentes herramientas que engloban la plataforma.

Incluso se entenderá mejor el funcionamiento de las herramientas gracias a la realización de un caso de uso.

4.2. Beneficios económicos

El desarrollo de una plataforma con estos objetivos proporciona unos costes menores a la hora de realizar estudios o investigaciones, dado que proporciona una mejor toma de decisiones y una optimización de los procesos.

Por otro lado, existe un menor riesgo de fallar en el resultado, por lo que existirán menos denuncias o acusaciones de fallos debidos al factor humano.

4.3. Beneficios sociales

Dentro de los beneficios sociales contamos con un mejor tratamiento de los datos de los pacientes, proporcionando de esta forma una mayor precisión y aceleración en el proceso de detección de enfermedades o patologías.

5. Estudio del estado del arte

5.1. Arquitectura Big Data

Una de las claves del auge del Big Data consiste en la posibilidad de gestionar no solo datos estructurados, como venían haciendo las bases de datos tradicionales, sino también ser capaces de gestionar datos no estructurados.

¿Cuáles son las principales ventajas del Big Data frente a las bases de datos tradicionales? Además de ser capaces de gestionar una cantidad mayor de datos en un tiempo menor, también tienen la capacidad de procesar información de distintos formatos, tipos o frecuencia en la que se generan.

5.1.1. Capas de la arquitectura

La capacidad de gestionar tantos datos sin la necesidad de que sean estructurados, es posible gracias a la arquitectura de 5 capas que forman cualquier proyecto de Big Data.

Las 5 capas son las siguientes: fuentes de datos, recolección de datos, almacenamiento, análisis y procesamiento.

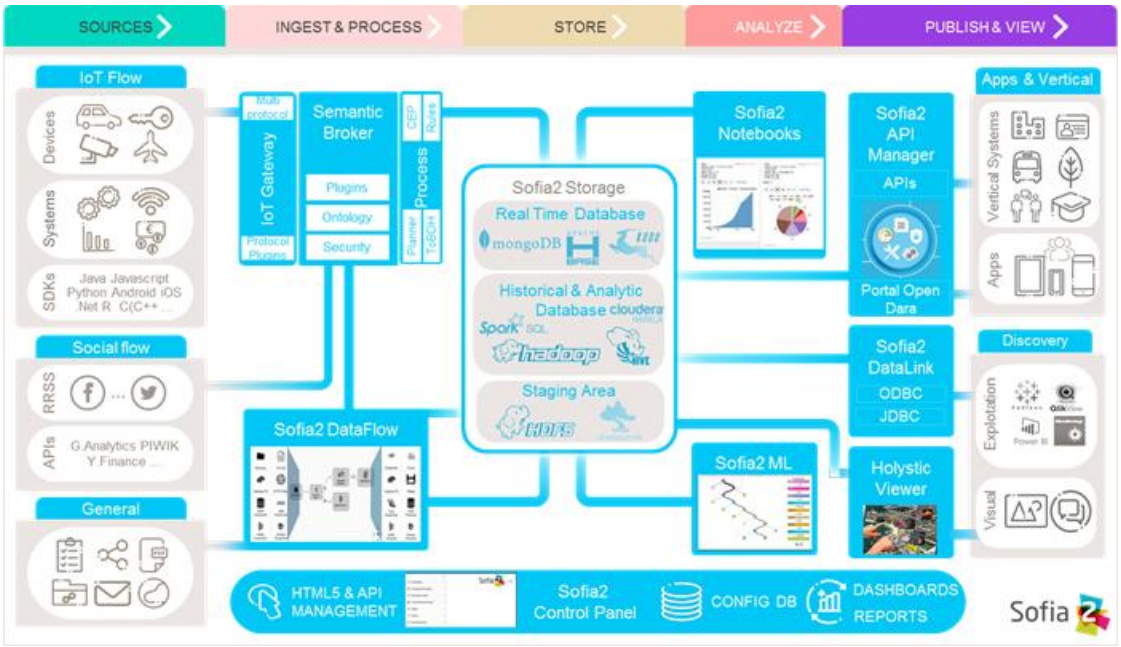


Ilustración 2-Capas arquitectura Big Data-
https://about.sofia2.com/2016/06/30/protocolos-de-automatizacion/

5.1.2. Fuentes de datos

Dada la inmensa cantidad de datos que se procesan, se procede a comentar la clasificación de los mismos. En primer lugar, se distinguen los datos según origen y categoría.

Por lo que concierne al origen, se clasificarán en 5 grupos:

Web y redes sociales
Comunicación entre máquinas
Transacciones
Biométricos
Generados por personas.

Por otro lado, existen tres categorías principales según la forma y disposición de los datos, los datos estructurados, los no estructurados y los semiestructurados.(3)

Cuando hablamos de datos estructurados nos referimos a la información que se suele encontrar en la mayoría de bases de datos. Son archivos de tipo texto que se suelen mostrar en filas y columnas con títulos.

Estos datos pueden ser ordenados y procesados fácilmente. Se podría comparar con un archivador perfectamente organizado donde todo está identificado, etiquetado y tiene fácil acceso. Se trata de información que tiene perfectamente definido la longitud, el formato y el tamaño de sus datos, normalmente se almacenan en formato de tabla, hojas de cálculo o en bases de datos relacionales.

	nombre	color	edad	altura	peso	puntuacion
1:	Paco	Rojo	24	182	74.8	83
2:	Juan	Green	30	170	70.1	500
3:	Andres	Amarillo	41	169	60.0	20
4:	Natalia	Green	22	183	75.0	865
5:	Vanesa	Verde	31	178	83.9	221
6:	Miriam	Rojo	35	172	76.2	413
7:	Juan	Amarillo	22	164	68.0	902

Ilustración 3-Ejemplo datos estructurados

Respecto a los datos no estructurados, constituyen la gran mayoría de los datos almacenados por una empresa. El 80 % de la información relevante para un negocio se

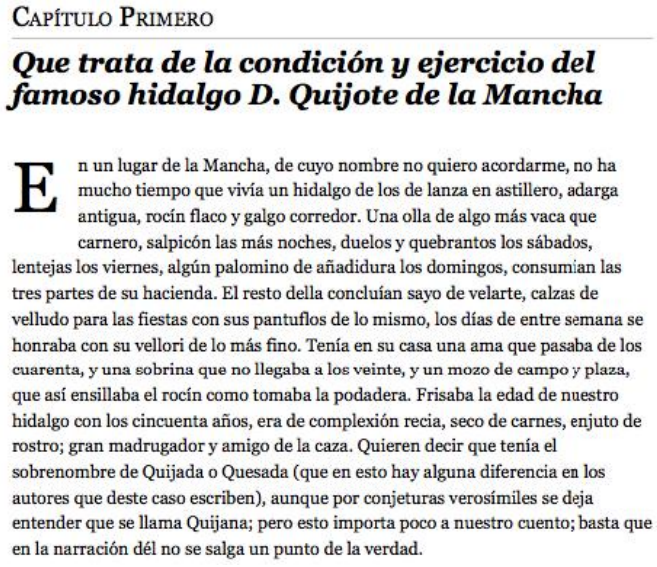
origina en forma no estructurada, la cual puede encontrarse principalmente en formato de texto.

Los datos no estructurados, generalmente son datos binarios que no tienen estructura interna identificable. Es un conglomerado masivo y desorganizado de varios objetos que no tienen valor hasta que se identifican y almacenan de manera organizada. Una vez que se organizan, los elementos que conforman su contenido pueden ser buscados y categorizados (al menos hasta cierto punto) para obtener información.

Idealmente, toda esta información podría ser convertida en datos estructurados, sin embargo, sería algo costoso y requeriría mucho tiempo.

Existen ciertos tipos de datos no estructurados como pueden ser: correos electrónicos, archivos de procesador de texto, archivos PDF, hojas de cálculo, imágenes digitales, video, audio o publicaciones en redes sociales.

Actualmente los datos no estructurados a pesar de tratarse de datos que suponen un problema en el momento de ser utilizados, debido a su complejidad en cuanto a su estructuración, se están desarrollando tecnologías y servicios para ayudar a solventarlo.(4)



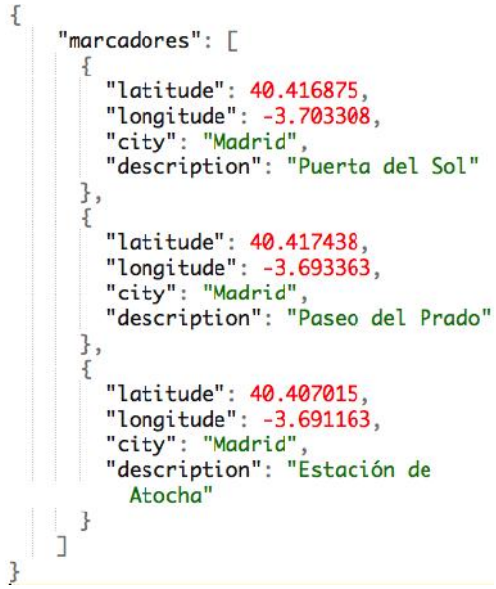
CAPÍTULO PRIMERO

Que trata de la condición y ejercicio del famoso hidalgo D. Quijote de la Mancha

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados, lentejas los viernes, algún palomino de añadidura los domingos, consumían las tres partes de su hacienda. El resto della concluían sayo de velarte, calzas de velludo para las fiestas con sus pantuflos de lo mismo, los días de entre semana se honraba con su vellori de lo más fino. Tenía en su casa una ama que pasaba de los cuarenta, y una sobrina que no llegaba a los veinte, y un mozo de campo y plaza, que así ensillaba el rocín como tomaba la podadera. Frisaba la edad de nuestro hidalgo con los cincuenta años, era de complexión recia, seco de carnes, enjuto de rostro; gran madrugador y amigo de la caza. Quieren decir que tenía el sobrenombre de Quijada o Quesada (que en esto hay alguna diferencia en los autores que deste caso escriben), aunque por conjeturas verosímiles se deja entender que se llama Quijana; pero esto importa poco a nuestro cuento; basta que en la narración dél no se salga un punto de la verdad.

Ilustración 4-Ejemplo datos no estructurados

Por último, los datos semiestructurados son una mezcla de los dos anteriores. No presentan una estructura perfectamente definida como los datos estructurados, pero si presentan una organización definida en sus metadatos donde describen los objetos y sus relaciones, y que en algunos casos están aceptados por convención, como por ejemplo los formatos HTML, XML o JSON. (5)



```
{
  "marcadores": [
    {
      "latitude": 40.416875,
      "longitude": -3.703308,
      "city": "Madrid",
      "description": "Puerta del Sol"
    },
    {
      "latitude": 40.417438,
      "longitude": -3.693363,
      "city": "Madrid",
      "description": "Paseo del Prado"
    },
    {
      "latitude": 40.407015,
      "longitude": -3.691163,
      "city": "Madrid",
      "description": "Estación de
        Atocha"
    }
  ]
}
```


Ilustración 5- Ejemplo datos semiestructurados

Una vez analizados la inmensa variedad de datos que se pueden encontrar hoy en día, se procede a estudiar diferentes formas de obtener los mismos. El provenir de los datos es la etapa en proyectos Big Data en la que se estudian los datos de la fuente de origen, se establecen procesos para su tratamiento, y estos son volcados de manera coherente en sistemas de almacenamiento. *"Es una etapa crucial, necesaria y a menudo infravalorada"*, José Felipe Ortega, científico de datos de la URJC.(6)

5.1.2.1. Minería de datos

A pesar de no tratarse de una técnica enfocada al Big Data, ya que data de los años 70, la minería de datos o el *Data Mining*analiza los grandes volúmenes de datos, para ello, sintetiza, identifica y agrupa patrones de comportamiento entre los datos. Generalmente los datos que analiza pertenecen a clientes y consumidores.

Como ejemplo para el uso de *Data Mining*, podría aplicarse al caso de necesitar patrones de conducta de clientes, periodos de contratación de un servicio determinado

o periodos de compra, fuga a otras compañías, o incluso riesgos de estafas a partir de patrones sospechosos o inusuales.

Data Mining requiere de Big Data para agilizar su procesamiento y gestión de los datos y, a la vez, Big Data requiere de *Data Mining* para el análisis predictivo de datos y poder detectar tendencias. Podríamos decir que hay una integración mutua entre técnica y herramienta.

La tecnología Big Data es capaz de capturar, almacenar, gestionar y procesar de forma rápida y veraz grandes cantidades de datos sacándole partido de ellos. Fundamentalmente, se enfoca al análisis predictivo y a detectar tendencias, sirviéndose de distintas técnicas, entre ellas las de minería de datos. A través de la definición de modelos y el uso de las diferentes tecnologías se busca convertir los datos en un activo de gran valor.(7)

5.1.2.2. Web Scraping

Esta técnica consiste en la extracción de información en línea de diferentes sitios web, hoy en día se trata de una de las prácticas más comunes. No se trata de una técnica novedosa, hace años se utilizaba de forma manual, a pesar de que hoy en día se haya conseguido automatizar, lo que permite ahorrar mucha energía y tiempo, además de mejorar la precisión.(8)

Existen diferentes programas de web scraping, los cuales han sido desarrollados para extraer datos de manera rápida, adecuada y organizada, todo ello emulando el comportamiento humano para no ser ni detectado ni bloqueado.(9)

5.1.2.3. API

La interfaz de programación de aplicaciones, también conocida como API, en inglés, *application programming interface*, es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. Hoy en día podemos encontrar tres tipos de API:

> *Privado*: Las API solo se pueden usar internamente, así que las empresas tienen un mayor control sobre ellas. Esto le da a las empresas un mayor control sobre sus API.
> *De partners*: Las API se comparten con partners empresariales específicos, lo cual puede ofrecer flujos de ingresos adicionales, sin comprometer la calidad.
> *Público*: Todos tienen acceso a las API, esto permite que terceros desarrollen aplicaciones que interactúan con su API, y puede ser un recurso para innovar.

Se ha conseguido un gran avance en el Big Data gracias a la economía API. El uso de API públicas ha facilitado a los desarrolladores acceder al código fuente de las compañías y de programar aplicaciones que ayudan a atraer más usuarios, siendo el resultado un gran avance para la compañía sin apenas coste adicional.(10)(11)

Como ejemplo de API se ha elegido la de Twitter, ya que se trata de una de las redes sociales más utilizada hoy en día. Como su propia página web nos indica,las API son la forma en que los programas informáticos "hablan" entre sí para solicitarse y enviarse información. Para esto, se le permite a la aplicación del software que llame a lo que se denomina punto de conexión: una dirección que corresponde a un tipo específico de información que proporcionamos (generalmente, los puntos de conexión son únicos, como los números telefónicos). Twitter permite acceder a partes de nuestro servicio mediante las API para permitirles a las personas crear software que se integre con Twitter, como una solución que ayuda a una empresa a responder a la opinión del cliente en Twitter.(12)

La API de Twitter incluye una gran variedad de puntos de conexión, que se dividen en cinco grupos principales:

Cuentas y usuarios
Tweets y respuestas
Mensajes directos
Anuncios
Herramientas y SDK del editor

5.1.3. Recolección de datos

La recolección de datos se refiere a las maneras en las que se pueden obtener e importar datos, ya sea para uso inmediato o para ser almacenados. Importarlos también incluye el proceso de prepararlos para un análisis. En un sentido más amplio, la ingesta de datos puede ser entendida como un flujo dirigido entre dos o más sistemas que resulta en una operación fluida e independiente.

En la época del Big Data, la ingesta manual ya es una rareza. Las compañías tienen numerosas fuentes de datos que funcionan las 24 horas del día. Los ingresos vienen en una variedad de formatos, por lo que una conversión a similares es necesaria. Así, cada vez más organizaciones están implementando la automatización para hacer más eficiente la ingesta de datos.(13)

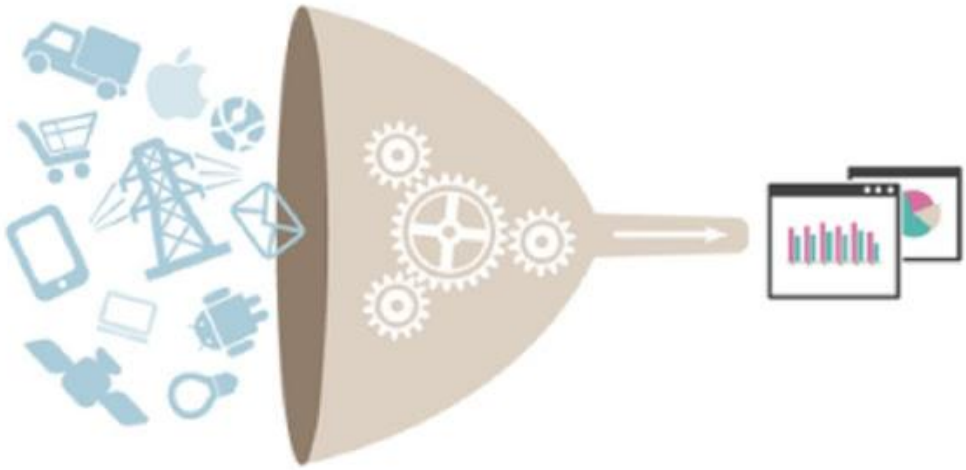


Ilustración 6-Recolección de datos-https://www.todaysoftmag.com/article/2196/real-time-stream-processing-in-the-big-data-realm

Existen dos herramientas principales en esta etapa, se tratan de Kafka y Flume.

5.1.3.1. Kafka

Apache Kafka es un sistema de intermediación de mensajes basado en el modelo publicador/suscriptor. Se considera un sistema persistente, escalable, replicado y tolerante a fallos. A estas características se añade la velocidad de lecturas y escrituras que lo convierten en una herramienta excelente para comunicaciones en streaming

Kafka proporciona multitud de conectores que le hacen conectarse a casi cualquier fuente de datos, como conectores para ActiveMQ, IBM MQ, JDBC, JMS, Replicator, etc. y también proporciona multitud de conectores que le hacen almacenar los datos en cualquier sitio, como conectores para HDFS, Amazon S3, Elasticsearch, JDBC, etc.

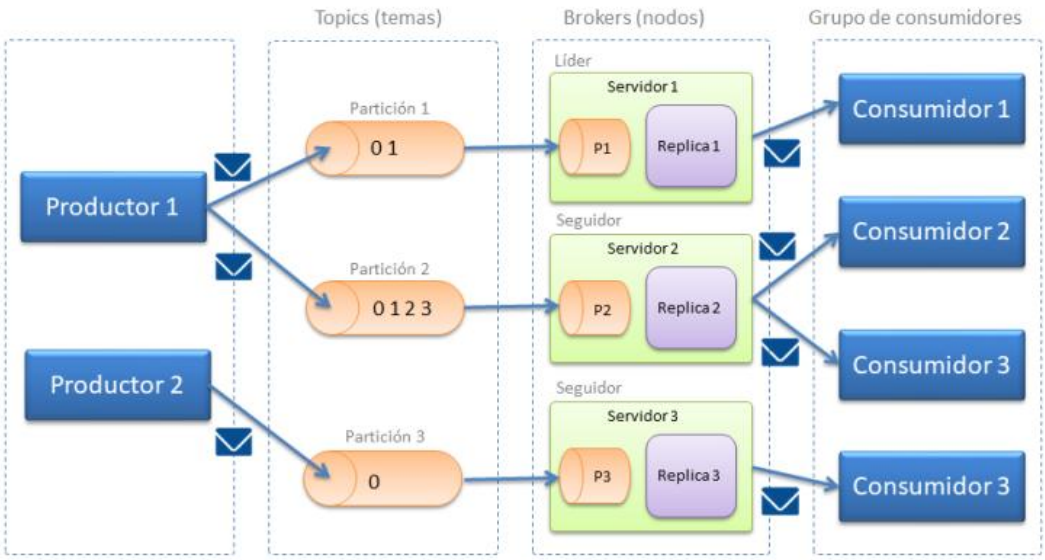


Ilustración 7-Arquitectura Kafka-http://www.diegocalvo.es/en/apache-kafka/kafka-architecture/

En la imagen anterior podemos observar la arquitectura por la que se rige esta herramienta, la cual consta de:

Topic (tema): Categorías en las que clasificar los mensajes enviados a Kafka.
Producer (productor): Clientes conectados responsables de publicar los mensajes. Estos mensajes son publicados sobre uno o varios *topics*.
Consumer (consumidor): Clientes conectados suscritos a uno o varios *topics* encargados de consumir los mensajes.
Broker (nodos): Nodos que forman el cluster.
Offset: es el indicador que indica a cada consumidor el último elemento que ha leído. Esto hace que si se cae el sistema no se pierdan los datos.

¿Cómo funciona? Apache Kafka divide cada topic (mensaje) en particiones. Cada partición es una secuencia ordenada de mensajes y cada partición es consumida por un único consumidor. Aunque se consuma la partición no se substrae por el consumidor.

A cada topic se le puede definir un número de particiones, en función del número de servidores y de conexiones que vayamos a tener. Esto aumenta considerablemente la disponibilidad. Cada topic tiene un offset para que cada consumidor indique qué

mensaje quiere que se le devuelva. A mayor número de particiones más tardará el productor (escribe mensajes) en guardar el mensaje, pero tardará menos el consumidor (lee mensajes) en recuperarlo. La idea está pensada para procesamiento en paralelo. Cada mensaje publicado en un topic se entrega a una instancia de consumidor dentro de cada grupo de consumidores suscriptores.

Apache Kafka se distribuye junto con Zookeeper para su instalación. Cuando solicitas un mensaje no te conectas a Kafka sino a Zookeeper. Zookeeper es una forma muy cómoda de escalar horizontalmente. Zookeeper está compuesto por brokers que actúan como líder de una o más particiones, el líder es activo y el seguidor pasivo.(14)

5.1.3.2. Flume

Apache Flume es un servicio distribuido que mueve de forma fiable y eficiente grandes cantidades de datos, especialmente logs. Es ideal para aplicaciones de analíticas en línea en entornos Hadoop.

Flume tiene una arquitectura sencilla y flexible basada en flujos de datos en streaming, que permite construir múltiples flujos por donde viajan los eventos a través de diferentes agentes hasta que alcanzan el destino final.

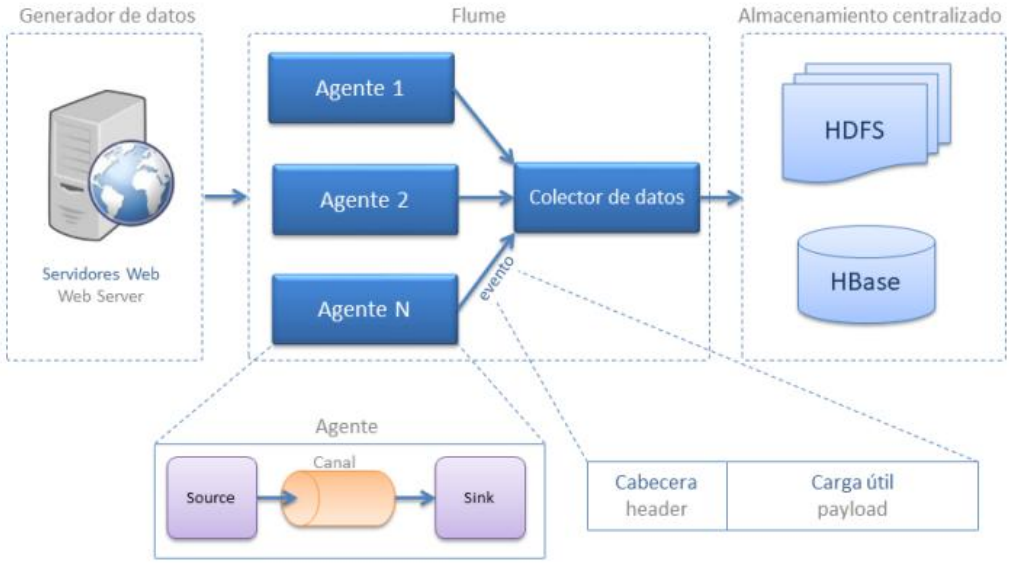


Ilustración 8-Arquitectura Flume

En la imagen anterior podemos observar la arquitectura por la que se rige esta herramienta, la cual consta de:

Evento: representa la unidad de datos en Flume, cuenta de una carga de bytes y una cabecera opcional.

Flujo de datos: describe el movimiento de eventos del origen al destino, cabe destacar que un evento puede pasar por agentes de manera encadenada antes de llegar a su destino.

Cliente: implementación de interfaz que opera en el punto de origen de los eventos y los entrega a los agentes. Los clientes suelen operar en el espacio de proceso de la aplicación de la que están consumiendo datos. Flume Log4j Appender es un ejemplo de cliente.

Agente: es un proceso independiente que se encarga de recibir, guardar y enviar eventos. Está compuesto por:

- **Source**: implementación de una interfaz que puede consumir eventos que le son enviados con un mecanismo específico y ponerlos en el canal. Avro es un ejemplo de interfaz.
- **Canal**: almacén transitorio de eventos. Los eventos se entregan al canal a través de las fuentes que operan en el agente. Un evento permanece en el canal hasta que un receptor lo quite para su posterior transporte. JDBC es un ejemplo de canal que utiliza una base de datos incrustada respaldada por el sistema de archivos para conservar los eventos hasta que sean eliminados por un receptor.

Sink: implementación de una interfaz que permite consumir eventos de un canal y transmitirlos al siguiente agente o a al destino final. Flume HDFS es un ejemplo de Sink.

(15)

5.1.4. Almacenamiento de datos

El desarrollo de las tecnologías y del internet ha incrementado de forma vertiginosa el volumen de datos que manejan las grandes empresas. En consecuencia, esto ha evolucionado el almacenamiento de datos, es decir, decidir cómo, cuándo y qué se almacena.(16)

5.1.4.1. SQL

Una base de datos es relacional cuando esta cumple con el modelo relacional, que se refiere a la relación que existe entre las distintas entidades o tablas de la base. Los datos se almacenan en diferentes tablas y las relaciones se establecen usando claves primarias u otras llaves conocidas como claves externas o foráneas.

Existen un sin número de sistemas de gestión de bases de datos relacionales y cada una de ellas posee una forma diferente de manejar su información. Algunos ejemplos de RDBMS son: Oracle, MySQL, SQL Server, entre otras. Sin embargo, con el paso de los años, estos se fueron unificando y universalizando para dar paso a mejores técnicas y mejores formas de manejo. Así nació SQL.(17)

SQL (Structured Query Language) es un lenguaje estándar e interactivo de acceso a bases de datos relacionales, que permite especificar diversos tipos de operaciones en ellas, gracias a la utilización del álgebra y de cálculos relacionales.

A la hora de hablar de las bases de datos relacionales, tienen como condición una estructura mínima de almacenamiento que consta de la tabla, el registro y el campo. Las tablas de una BD guardan información de individuos o unidades de una misma naturaleza con una serie de atributos en común.

Una BD contendrá tablas que a su vez contendrán registros y en estos se encontrarán los datos distribuidos en una serie de campos. Cada registro de la tabla guarda la información particular de una unidad o miembro de un mismo grupo. El SGBD cumple la función de interface entre el usuario y la BD, permitiéndonos interactuar con ella mediante SQL.(18)

5.1.4.2. NoSQL

Las bases de datos NoSQL están diseñadas específicamente para modelos de datos específicos y tienen esquemas flexibles para crear aplicaciones modernas. Las bases de datos NoSQL son ampliamente reconocidas porque son fáciles de desarrollar, su

funcionalidad y el rendimiento a escala. Usan una variedad de modelos de datos, que incluyen documentos, gráficos, clave-valor, en memoria y búsqueda. (19)

Estos tipos de bases de datos están optimizados específicamente para aplicaciones que requieren grandes volúmenes de datos, baja latencia y modelos de datos flexibles, lo que se logra mediante la flexibilización de algunas de las restricciones de coherencia de datos en otras bases de datos. Como ya se ha comentado anteriormente, existen distintos tipos de bases de datos:

- Las de datos clave-valor, son el modelo de base de datos NoSQL más popular, además de ser la más sencilla en cuanto a funcionalidad. En este tipo de sistema, cada elemento está identificado por una llave única, lo que permite la recuperación de la información de forma muy rápida, información que habitualmente está almacenada como un objeto binario. Se caracterizan por ser muy eficientes tanto para las lecturas como para las escrituras. Algunos ejemplos de este tipo son Cassandra, BigTable o HBase.
- Las de datos documentales, este tipo almacena la información como un documento, generalmente utilizando para ello una estructura simple como JSON o XML y donde se utiliza una clave única para cada registro. Este tipo de implementación permite, además de realizar búsquedas por clave–valor, realizar consultas más avanzadas sobre el contenido del documento. Son las bases de datos NoSQL más versátiles. Se pueden utilizar en gran cantidad de proyectos, incluyendo muchos que tradicionalmente funcionarían sobre bases de datos relacionales. Algunos ejemplos de este tipo son MongoDB o CouchDB.
- Las de datos en grafo, en este tipo de bases de datos, la información se representa como nodos de un grafo y sus relaciones con las aristas del mismo, de manera que se puede hacer uso de la teoría de grafos para recorrerla. Para sacar el máximo rendimiento a este tipo de bases de datos, su estructura debe estar totalmente normalizada, de forma que cada tabla tenga una sola columna y cada relación dos. Este tipo de bases de datos ofrece una navegación más eficiente entre relaciones que en un modelo relacional. Algunos ejemplos de este tipo son Neo4j, InfoGrid o Virtuoso.
- Las orientadas a objetos, en este tipo, la información se representa mediante objetos, de la misma forma que son representados en los lenguajes de programación orientada a objetos (POO) como ocurre en JAVA, C# o Visual Basic y .NET. Algunos ejemplos de este tipo de bases de datos son Zope, Gemstone o Db4o.(20)

5.1.4.3. MongoDB

MongoDB es una base de datos distribuida, basada en documentos y de uso general que ha sido diseñada para desarrolladores de aplicaciones modernas y para la era de la nube.

Se trata de una base de datos NoSQL optimizada para trabajar con grupos de datos que varían con frecuencia, o que son semiestructurados o no estructurados. Se emplea para almacenar datos de aplicaciones móviles y de sistemas de gestión de contenidos, entre otros.

MongoDB guarda la estructura de los datos en documentos BSON con un esquema dinámico, lo que implica que no existe un esquema predefinido. Los elementos de los datos se denominan documentos y se guardan en colecciones. Una colección puede tener un número indeterminado de documentos. Comparando con una base de datos relacional, se puede decir que las colecciones son como tablas y los documentos son registros en la tabla. La diferencia es que en una base de datos relacional cada registro en una tabla tiene la misma cantidad de campos, mientras que en MongoDB cada documento en una colección puede tener diferentes campos. En un documento, se pueden agregar, eliminar, modificar o renombrar nuevos campos en cualquier momento, ya que no hay un esquema predefinido. La estructura de un documento es simple y compuesta por pares llave/valor, parecido a las matrices asociativas en un lenguaje de programación, esto es debido a que MongoDB sigue el formato de JSON. En MongoDB la clave es el nombre del campo y el valor es su contenido, los cuales se separan mediante el uso de ":". Como valor se pueden usar números, cadenas o datos binarios como imágenes o cualquier otro.

Cabe destacar que MongoDB soporta una amplia variedad de lenguajes de programación, como vienen siendo: C, C++, C#/.NET, Java, JavaScript, Node.js, Perl, PHP, Python, Ruby y Scala.

Al tratarse de una de las bases de datos más conocidas del panorama actual, es empleada por grandes compañías como Bosch, Facebook, eBay, Adobe, Google, Sega, Sap, Nokia, Cisco, Paypal y Telefónica.(21)

5.1.4.4. Apache Cassandra

Apache Cassandra fue lanzada en el año 2008 y fue creada inicialmente por Facebook, más tarde fue traspasada a la Fundación Apache, convirtiéndose en una herramienta Open Source, que a día de hoy la sigue manteniendo.

Se trata de una base de datos que permite grandes volúmenes de información en forma distribuida, la cual escala linealmente y que no sigue un patrón maestro-esclavo. Está basada en un modelo de almacenamiento de «clave-valor», de código abierto que está escrita en Java.

Su objetivo principal es la escalabilidad lineal y la disponibilidad. La arquitectura distribuida de Cassandra está basada en una serie de nodos iguales que se comunican con un protocolo P2P con lo que la redundancia es máxima.(22)

El modelo de datos de Cassandra consiste en particionar las filas, que son reorganizadas en tablas. Las claves primarias de cada tabla tienen un primer componente que es la clave de partición. Dentro de una partición, las filas son agrupadas por las columnas restantes de la clave. Las demás columnas pueden ser indexadas por separado de la clave primaria. Las tablas se pueden crear, eliminar y alterar en tiempo de ejecución sin bloquear actualizaciones y consultas.

En las versiones iniciales utilizaba un API propia para poder acceder a la base de datos. En los últimos tiempos están apostando por un lenguaje denominado CQL (Cassandra Query Language, no confundir con Contextual Query Language) que posee una sintaxis similar a SQL aunque con muchas menos funcionalidades. Esto hace que iniciarse en el uso de la misma sea más sencillo.

Apache Cassandra se trata de una herramienta muy popular, la cual es utilizada por grandes empresas como Walmart, Apple, Cisco, Facebook, IBM, Netflix, Reddit, SoundCloud y Twitter.(23)

5.1.5. Procesamiento y análisis de datos

El análisis de grandes datos es el proceso de examinar grandes cantidades de datos de una variedad de tipos para descubrir patrones ocultos, correlaciones desconocidas y otra información útil. Tal información puede proporcionar ventajas competitivas a través de organizaciones rivales y resultar en beneficios para el negocio, tales como el marketing más efectivo y mayores ingresos.

El objetivo principal de este análisis es ayudar a las empresas a tomar mejores decisiones de negocios, al permitir a los científicos y otros usuarios de datos analizar grandes volúmenes de datos transaccionales, así como otras fuentes de datos que puedan haber quedado sin explotar por la inteligencia de negocio convencional.

El análisis de Big Data puede hacerse con herramientas de software de uso común en el marco de disciplinas analíticas avanzadas, como el análisis predictivo y la minería de

datos. Sin embargo, las fuentes de datos no estructurados utilizados para el análisis de grandes datos no pueden ser gestionados con las bases de datos tradicionales, como se ha comentado anteriormente. Como resultado, una nueva clase de tecnología de datos grandes ha surgido, y está siendo utilizada en muchos análisis de datos grandes. Las tecnologías relacionadas con el análisis de datos incluyen bases de datos grandes NoSQL, Hadoop y MapReduce. Estas tecnologías forman el núcleo de un marco de software de código abierto, que soporta el procesamiento de grandes volúmenes de datos a través de sistemas en clúster.(24)(25)

Existen dos grandes bloques en los que se puede enfocar un proyecto gestionado mediante Big Data. Por un lado, estarían las necesidades más de estructura: capacidades de almacenamiento, transmisión y transformación de datos, y por otro lado, las necesidades de explotación o analítica del dato para extraer información relevante y conclusiones de negocio.(26)

5.1.5.1. Hadoop

Hadoop se trata del sistema más utilizado en Big Data para ofrecer capacidades analíticas avanzadas, facilita el almacenamiento de información y permite hacer consultas complejas sobre las bases de datos existentes, resolviéndolas con rapidez.

Es un tipo de solución Big Data, es una plataforma que utiliza un sistema de código abierto para almacenar, procesar y analizar grandes volúmenes de datos; cientos de terabytes, petabytes o incluso más. Permite también ejecutar aplicaciones en clusters de hardware básicos.

Hadoop surgió como una iniciativa open source para resolver los problemas asociados al Big Data y a la aparición del Data Science, y es conocida como la plataforma de código abierto que lidera el ranking de plataformas Big Data. Incluso en parte de la industria del almacenamiento y aplicaciones analíticas se ha convertido en sinónimo de Big Data en su argot.

Entre sus puntos clave se encuentran su capacidad de almacenamiento y procesamiento local, gracias a ellos consigue escalar desde unos pocos servidores hasta miles de máquinas, teniendo todas las máquinas una misma calidad de servicio.

Su sistema de distribución por nodos hace que Hadoop nos proporcione un almacenamiento masivo para cualquier tipo de datos con una gran capacidad de procesamiento y de gestionar tareas o trabajos de tamaño prácticamente ilimitados.

Esta gran capacidad de procesamiento es gracias, también, a que los grandes conjuntos de datos que almacena se encuentran ubicados en clusters de computadoras que utilizan modelos sencillos de programación.

Es utilizado por empresas como Facebook, Yahoo!, eBay, IBM, LinkedIn, Twitter o The New York Times entre otras.

La biblioteca Hadoop utiliza modelos de programación simples para el almacenamiento y procesamiento distribuido de grandes conjuntos de datos en clusters, dando redundancia para no perder nada y, al mismo tiempo, aprovechando muchos procesos a la vez.(27)(28)

5.1.5.2. Elasticsearch

Elasticsearch es un servidor de búsqueda basado en Lucene. Provee un motor de búsqueda de texto completo y distribuido. Está desarrollado en Java y publicado como código abierto bajo las condiciones de la licencia Apache.

Shay Banon creó Compass en 2004, llegó a la conclusión de que habría que "crear una solución de búsqueda escalable". Entonces creó "una solución construida para ser distribuida desde el comienzo" con la interfaz JSON sobre HTTP, muy común y adecuada para lenguajes de programación que no sean Java.

Elasticsearch utiliza Query DSL (Lenguaje de dominio específico) para realizar las consultas a los documentos indexados. Es un lenguaje sumamente flexible y de gran alcance, además de simple, que permite conocer y explorar los datos de la mejor manera. Al ser utilizado a través de una interfaz de tipo JSON, las consultas son muy sencillas de leer y, lo más importante, de depurar.

Este servidor de búsqueda permite el procesamiento de grandes cantidades de datos y ver la evolución de estos en tiempo real. Además, proporciona gráficos que ayudan a comprender con más facilidad la información obtenida. Nos permite indexar y analizar en tiempo real un gran volumen de datos y hacer consultas sobre ellos. Un ejemplo de uso son las consultas de texto completo; al estar los datos indexados, los resultados se obtienen de forma muy rápida. Es distribuido, haciendo que los índices se puedan dividir en fragmentos y cada uno teniendo cero o más réplicas. Cada nodo alberga uno o más fragmentos, actuando como un coordinador para delegar operaciones a los fragmentos correctos.

Mozilla, SoundCloud, GitHub, Foursquare, eBay, Walmart, Optum y Etsy son algunas de las empresas que han empleado esta herramienta de Big Data.(29)

5.1.5.3. Apache Spark

Apache Spark es un framework de computación en clúster open-source. El código base del proyecto Spark fue donado más tarde a la Apache Software Foundation que se encarga de su mantenimiento desde entonces. Spark proporciona un interfaz para la programación de clusters completos con paralelismo de datos implícito y tolerancia a fallos.

Se puede considerar un sistema de computación en clúster de propósito general y orientado a la velocidad. Proporciona APIs en Java, Scala, Python y R. También proporciona un motor optimizado que soporta la ejecución de grafos en general. También soporta un conjunto extenso y rico de herramientas de alto nivel entre las que se incluyen Spark SQL, MLlib para implementar machine learning, GraphX para el procesamiento de grafos y Spark Streaming.(30)

La principal ventaja de este sistema de computación es que permite dividir o paralelizar el trabajo, ya que normalmente se instala en un clúster de máquina. La idea es que tengamos n máquinas, por ejemplo diez máquinas, y cada una de esas instancias va a tener instalada una versión de Apache Spark.

De esta manera, cuando tengamos que procesar una gran cantidad de datos, por ejemplo un fichero muy grande, podemos dividir el mismo en diez partes, y cada máquina se encargará de una décima parte del fichero, y al final lo uniremos. Con esto estamos ganando velocidad, y la velocidad es clave en el mundo del Big Data. Por lo que la característica más destacable de esta herramienta de Big Data es su velocidad, siendo 100 veces más rápida que Hadoop.(31)

En cuanto a la segunda parte, la de analítica o Data Science. Aquí las dos principales alternativas en el ámbito del software libre y con grandes comunidades de usuarios son R y Python.(32)

5.1.5.4. Lenguaje R

R es un lenguaje y entorno de programación para análisis estadístico y gráfico. Se podría decir que es un dialecto libre del lenguaje S, un lenguaje de programación estadístico, desarrollado por Robert Gentleman y Ross Ihaka del Departamento de Estadística de la Universidad de Auckland en 1993.

Su potencia, robustez y versatilidad lo convierten en el software elegido por los Data Scientist de la comunidad científica internacional de las universidades, empresas, industrias e instituciones más prestigiosas del mundo.

Este lenguaje consta de claras ventajas como vienen siendo el tratarse de un software libre, su estabilidad y la gran cantidad de herramientas que se pueden encontrar en él. Al tratarse de un lenguaje muy común, cuenta con una extensa comunidad de usuarios, por lo que hay disponible una gran cantidad de librerías. R es muy usado por estadistas y data miners.(33)

5.1.5.5. Python

Se trata de un lenguaje de programación multiparadigma, esto significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos: orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.

Una característica importante de Python es la resolución dinámica de nombres; es decir, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado enlace dinámico de métodos).

Otro objetivo del diseño del lenguaje es la facilidad de extensión. Se pueden escribir nuevos módulos fácilmente en C o C++. Python puede incluirse en aplicaciones que necesitan una interfaz programable.

Python cuenta con la ventaja de que sólo hay que tener unos conocimientos mínimos de informática para poder usarla, lo que hace que tenga una gran comunidad de usuarios con la opción de crear sus propias librerías. El inconveniente de este lenguaje de Big Data es la velocidad, ya que es notablemente más lento que sus competidores.(34)

Ambos tienen interfaces y conexiones con otras plataformas de desarrollo Big Data como son Hadoop y todo su ecosistema, o Spark, así como se integran con los proveedores de computación en la nube.

Aunque cada uno tiene sus defensores y detractores, la realidad es que pueden convivir incluso en un mismo proyecto. Existen casos de uso reales en los que las tareas de conexión a fuentes de datos, su descarga y transformación están implementadas en Python, y las fases de modelado, predicción y visualización de resultados están programadas con R. Esto no quiere decir que siempre sea ideal hacerlo así, ya que en determinado tipo de algoritmos que aplicamos, la mejor solución pasa por implementarlos en Python, debe estudiarse cada necesidad particular.(35)

5.1.6. Streaming

Los datos de streaming se generan constantemente a partir de miles de fuentes de datos, que normalmente envían los registros de datos simultáneamente en conjuntos de tamaño pequeño. Los datos de streaming incluyen diversos tipos de datos, como archivos de registros generados por los clientes que utilizan sus aplicaciones móviles o web, compras electrónicas, actividades de los jugadores en un juego, información de redes sociales, operaciones bursátiles o servicios geoespaciales, así como telemetría de dispositivos conectados o instrumentación en centros de datos.

Estos datos deben procesarse de forma secuencial y gradual registro por registro o en ventanas de tiempo graduales, y se utilizan para una amplia variedad de tipos de análisis, como correlaciones, agregaciones, filtrado y muestreo. La información derivada del análisis aporta a las empresas visibilidad de numerosos aspectos del negocio y de las actividades de los clientes, como el uso del servicio (para la medición/facturación), la actividad del servidor, los clics en un sitio web y la ubicación geográfica de dispositivos, personas y mercancías, y les permite responder con rapidez ante cualquier situación que surja. Por ejemplo, las empresas pueden supervisar los cambios en la opinión pública de sus marcas y productos al analizar constantemente las transmisiones de los medios sociales y responder rápidamente cuando sea necesario.(36)

Existen dos formas de procesar los datos de streaming, el procesamiento de transmisiones y el procesamiento por lotes. El procesamiento por lotes se puede utilizar para procesar consultas arbitrarias a diferentes conjuntos de datos. Normalmente, ofrece resultados derivados de todos los datos que reúne y facilita un análisis profundo de los conjuntos de Big Data. A diferencia de este, el procesamiento de transmisiones requiere la introducción de una secuencia de datos y la actualización gradual de las métricas, los informes y las estadísticas de resumen como respuesta a cada registro de datos de llegada, es más adecuado para las funciones de respuesta y supervisión en tiempo real.(37)

5.1.6.1 Storm

Apache Storm es un sistema que sirve para recuperar streams de datos en tiempo real desde múltiples fuentes de manera distribuida, tolerante a fallos y en alta disponibilidad. Storm está principalmente pensado para trabajar con datos que deben ser analizados en tiempo real, por ejemplo datos de sensores que se emiten con una

alta frecuencia o datos que provengan de las redes sociales donde a veces es importante saber qué se está compartiendo en este momento.

Se compone de dos partes principalmente. La primera es la que se denomina Spout y es la encargada de recoger el flujo de datos de entrada. La segunda se denomina Bolt y es la encargada del procesado o transformación de los datos.

En la documentación oficial representan los Spouts con grifos simulando la entrada de un stream de datos al sistema y a los Bolts con un rayo que es donde se realizan las acciones pertinentes con los datos de entrada.

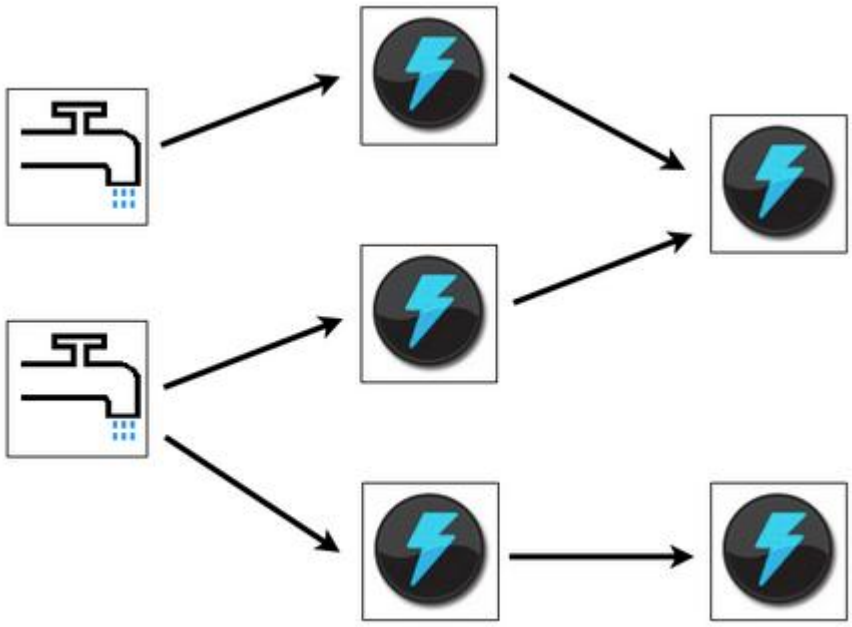


Ilustración 9-Apache Storm

Uno de los puntos fuertes de Storm es que podemos crear una topología donde añadimos instancias de Bolts y Spouts para que escale el sistema, desplegándola en el cluster de Storm que es quien se encargará de particionar los datos de entrada y redistribuirlos por los diferentes componentes.

La arquitectura de Storm es bastante sencilla, se divide en los siguientes componentes:

El **master node** ejecuta el demonio llamado Nimbus responsable de distribuir el código a través del cluster (similar al JobTracker de Hadoop). Realiza también la asignación y monitorización de las tareas en las distintas máquinas del cluster.

Los **worker nodes** ejecutan el demonio Supervisor encargado de recoger y procesar los trabajos asignados en la máquina donde corre. Estos nodos ejecutan una porción de la topología para que así se puedan distribuir los trabajos a lo largo del cluster. Si fallara un worker node, el demonio Nimbus se daría cuenta y redirigiría el trabajo a otro worker node.

Zookeeper: aunque no es un componente como tal de Storm, sí que es necesario montar un Apache Zookeeper, ya que será el encargado de la coordinación entre el Nimbus y los Supervisors. También es el encargado de mantener el estado ya que el Nimbus y los Supervisors son stateless.

El componente Spout de Storm es el encargado de la ingesta de los datos en el sistema, por ejemplo, si tenemos que leer un fichero de texto y contar las palabras, el componente que recibiría los streams del fichero sería el Spout. Otro típico ejemplo sería Twitter. Si queremos recoger determinados tweets para posteriormente procesarlos o realizar algún tipo de analítica sobre ellos, el encargado de conectar con el API de Twitter y recoger los datos sería el Spout.

El Bolt es encargado de consumir las tuplas que emite el Spout, las procesa en función de lo que dicte el algoritmo que programamos sobre los streams de entrada y puede emitirlos a otro Bolt. Es recomendable que cada Bolt realice una única tarea. Si necesitamos realizar varios cálculos o transformaciones sobre los datos que le llegan al Bolt, lo mejor es que se dividan en distintos Bolt para mejorar la eficiencia y la escalabilidad. Tanto el Spout como el Bolt emiten tuplas que serán enviadas a los Spouts que estén suscritos a ese determinado stream configurado en la topología. Por ejemplo, si queremos contar las veces que aparece cada palabra en un texto, podemos hacer un Bolt que se encargue de contar las que empiezan por vocal y otro para las que empiezan por consonante. El Spout sería el encargado de redirigir a uno u otro Bolt.(38)

5.1.6.2. Apache Spark Streaming

Apache Spark Streaming es una extensión de la API core de Spark, que da respuesta al procesamiento de datos en tiempo real de forma escalable, con alto rendimiento y tolerancia a fallos. Fue desarrollado por la Universidad de California en Berkeley, actualmente Databricks se encarga de dar soporte y realizar mejoras.

Como idea general se podría decir que Spark Streaming toma un flujo de datos continuo, lo convierte en un flujo discreto denominado DStream, para que el core de Spark lo pueda procesar.

Ilustración 10-Spark Streaming-http://www.diegocalvo.es/spark-streaming/spark-streaming-2/

Esta herramienta puede ingerir datos de multitud de fuentes, tales como Kafka, Flume, RabbitMQ, Kinesis, ZeroMQ o sockets TCP; y es capaz de procesar datos utilizando una multitud de algoritmos de machine learning o de grafos y funciones tales como map, reduce, join y window. Finalizado el procesamiento, los datos son almacenados en sistema de ficheros, base de datos para poder presentarse en dashboards, por ejemplo.

Dstream o stream discreto: no es más que una abstracción proporcionada por Spark Streaming que representa a una secuencia de RDDs ordenados en el tiempo que cada uno de ellos guarda datos de un intervalo concreto. Con esta abstracción se consigue que el core lo analice sin enterarse de que está procesando un flujo de datos, ya que el trabajo de crear y coordinar los RDDs es realizado por Spark Streaming.(39)

6. Análisis de alternativas

Como se ha comentado con anterioridad en el apartado 5.1.1 Capas de la arquitectura, en un proyecto de Big Data podemos encontrar 5 secciones: fuentes de datos, recolección, almacenamiento, procesamiento y visualización.

En el presente apartado se realizará una valoración de las características de las herramientas vistas en el estudio del estado del arte, apartado 5. Para ello, se tendrán en cuenta tanto puntos positivos como negativos a la hora de enfocarlas hacia el presente proyecto, no se valorará en ningún momento qué herramienta es mejor o peor.

6.1. Recolección de datos

6.1.1. Kafka

En primer lugar, nos encontramos con una herramienta de mensajería capaz de manejar un alto volumen de datos a alta velocidad. Habiendo estudiado previamente el funcionamiento de Kafka, podemos comprobar su facilidad a la hora de escalar horizontalmente mediante la adición de nodos adicionales.

Por otro lado, una de sus mejores ventajas es la tolerancia a fallas, ya que los mensajes no se envían al usuario hasta este se lo pide, por lo que si en un momento dado el usuario no está preparado no tendrá problemas por perdida de información.

En cuanto a la compatibilidad, nos encontramos con una herramienta que funciona prácticamente con cualquier plataforma.

Por último, Kafka retiene temporalmente los datos en disco, ya que los mensajes no se eliminan tras ser consumidos, los mensajes son eliminados según la política definida por el usuario, ya sea una vez alcanzado un límite de tiempo o tras alcanzar una determinada capacidad.

6.1.2. Flume

La característica principal de Flume podría considerarse que se trata de una herramienta completamente integrada en el entorno de Hadoop. Una desventaja es su dificultad para escalar, ya que añadir más consumidores significa cambiar la topología del pipeline, y añadir un nuevo destino.

Flume es robusta y tolerante a fallos, con mecanismos de fiabilidad configurables y de conmutación por error y recuperación.

Por último, Flume soporta canales efímeros basados en memoria y canales duraderos basados en ficheros. Incluso utilizando canales duraderos, los mensajes que se queden en el canal no estarán disponibles en el destino hasta que se recupere el agente.

6.2. Almacenamiento de datos

En primer lugar, se tratarán los dos tipos de bases de datos existentes, relacionales y no relacionales, y a continuación se analizarán ciertas herramientas y tecnologías.

6.2.1. SQL

Las bases relacionales se rigen por el lenguaje de programación SQL, mediante el cual se crean tablas, consultas y se inserta, elimina y actualiza información. Constan de unos estándares bien definidos.

Las bases de datos relacionales tienen ciertas desventajas. Por ejemplo, cuando tienden a crecer demasiado en el almacenamiento, y el mantenimiento es sumamente difícil y costoso, suelen presentar fallas en tiempo de respuesta. Un cambio en la estructura puede convertirse en un proceso costoso, dado que, si el diagrama de Entidad Relación no lo soporta, entonces esto implica tener que realizar una modificación en la estructura de la base de datos y, posiblemente, detener el sistema por un tiempo moderado hasta terminar el proceso.

6.2.2. NoSQL

Una vez vistas las características de las bases de datos relacionales, se van a tratar las bases de datos no relacionales, o también conocidas como NoSQL. Estas bases de datos buscan mejorar el rendimiento sobre las bases relacionales, pero también dentro de las ventajas conllevan algunas desventajas.

La principal ventaja que nos ofrece una base de datos NoSQL es su versatilidad, la sencillez a la hora de agregar un nuevo campo a la colección. Dado que se basa en documentos con notación JSON, como podemos ver en la ilustración 11, los nuevos campos simplemente se agregan sobre el documento y el sistema sigue operando sin necesidad de configuraciones extras.

```
[
      {
        "description": "quarter",
        "mode": "REQUIRED",
        "name": "qtr",
        "type": "STRING"
      },
      {
        "description": "sales representative",
        "mode": "NULLABLE",
        "name": "rep",
        "type": "STRING"
      },
      {
        "description": "total sales",
        "mode": "NULLABLE",
        "name": "sales",
        "type": "INTEGER"
      }
    ]
```

Ilustración 11-Notación JSON-https://cloud.google.com/bigquery/docs/loading-data-cloud-storage-json?hl=es-419

Al soportar una estructura distribuida, si durante la operación el desempeño de los servidores baja, se instalan nuevos nodos operativos que balancean la carga de trabajo, es decir, crece horizontalmente.

Hablando de rentabilidad, las bases de datos no relacionales de código abierto no requieren tarifas de licencia y pueden ejecutarse en hardware de precio bajo.

A pesar de las ventajas que nos ofrecen estas bases de datos, existen ciertas desventajas como se ha comentado anteriormente, por ejemplo, la necesidad de implementar un código propio a la hora de realizar una BD fiable y coherente.

Por otro lado, existen problemas de compatibilidad, las bases de datos NoSQL tienen pocos estándares en común, cada una de ellas tiene su propia API, las interfaces de consulta son únicas, etc. Esta falta de normas significa que es imposible cambiar simplemente de un proveedor a otro, por si no quedara satisfecho con el servicio.

6.2.3. MongoDB

Como se ha podido comprobar, MongoDB prioriza la individualidad de los documentos, por lo que su almacenamiento se basa en formato de documentos.

Por otro lado, MongoDB se rige por su propio lenguaje de consulta, lo que supone aprenderlo a la hora de implementar esta herramienta en un proyecto.

La principal desventaja de MongoDB es que carece de *Joins*. Es decir, que si se necesita consultar datos de dos o más colecciones se debe realizar más de una consulta.

6.2.4. Apache Cassandra

Apache Cassandra es una herramienta de gestión de bases de datos open source diseñada por Facebook y adoptada posteriormente por Apache. Se caracteriza por tener una arquitectura escalable, al tratarse de un diseño masterless, todos los nodos son iguales, en todos estos nodos se puede escribir y leer.

A diferencia de MongoDB, está orientado a columnas y directamente almacena sus datos y los recupera por columnas. Esta herramienta resulta ideal a la hora de trabajar con datos semiestructurados y cuando se espera tener un crecimiento casi exponencial de la base de datos.

A nivel de sintaxis, Cassandra consta de un lenguaje muy similar a SQL, el CQL (Lenguaje de Consulta Cassandra), por lo que la transición desde una base de datos relacional es muy sencilla.

Por último, Cassandra comprime hasta un 80% de los datos sin que ello suponga un gasto de recursos.

6.3. Procesamiento de datos en batch

En este apartado se analizarán únicamente las herramientas orientadas a la estructura de los datos, previamente estudiadas en el apartado de estudio del estado del arte.

6.3.1. Hadoop

La arquitectura de Hadoop permite llevar a cabo un análisis eficaz de grandes datos no estructurados, añadiéndoles un valor que puede ayudar a tomar decisiones estratégicas, mejorar los procesos de producción, ahorrar costes, hacer un seguimiento de lo que opina la clientela o extraer conclusiones científicas, pongamos por caso.

Hadoop nos permite un crecimiento fácil, sin estar atados a las características iniciales del diseño, haciendo uso de decenas de servidores de bajo costo que, a diferencia de la base de datos relacional, no puede escalar. Gracias al procesamiento distribuido de MapReduce, los archivos se dividen en bloques de forma sencilla.

Al incrementar el número de nodos del sistema también ganamos en capacidad de almacenamiento y procesamiento. A su vez, es posible agregar o acceder a nuevas y diferentes fuentes de datos (estructurados, semiestructurados y no estructurados).

Por último, este software permite ejecutar procesamientos y realizar análisis muy rápidos, y si un equipo se cae, siempre hay otra copia disponible, con lo que es posible la recuperación de datos en caso de producirse fallos.

6.3.2. Elasticsearch

Se trata de un sistema distribuido que permite trabajar con un gran volumen de datos de cualquier tipo y un sistema de analítica que permite realizar consultas en tiempo real. Está publicado como open source (código abierto) bajo licencia Apache y desarrollado en Java.

Elasticsearch tiene una poderosa DSL (Línea de Abonado Digital) basada en JSON, que permite a los equipos de desarrollo construir consultas complejas y afinarlas para recibir los resultados más precisos de una búsqueda. También proporciona una forma de clasificar y agrupar los resultados.

Este software está orientado a documentos, no utiliza esquemas, sino documentos JSON e intenta detectar la estructura de datos, indexar los datos y hacer que se pueda buscar.

Por último, Elasticsearch permite escalar horizontalmente, permite extender los recursos y equilibrar la carga entre los nodos de un cluster.

6.3.3. Apache Spark

Esta herramienta tiene un predecesor conocido en el Big Data, MapReduce, el cual revolucionó la manera de trabajar con grandes conjuntos de datos ofreciendo un modelo relativamente simple para escribir programas que se podían ejecutar paralelamente en cientos y miles de máquinas al mismo tiempo.

Spark mantiene la escalabilidad lineal y la tolerancia a fallos de MapReduce, pero amplía sus bondades gracias a varias funcionalidades: DAG y RDD.

DAG (Grafo Acíclico Dirigido) es un grafo dirigido que no tiene ciclos, como se puede ver en la ilustración 12, es decir, para cada nodo del grafo no hay un camino directo que comience y finalice en dicho nodo. Un vértice se conecta a otro, pero nunca a sí mismo. Cada tarea de Spark crea un DAG de etapas de trabajo para que se ejecuten en un determinado cluster. En comparación con MapReduce, el cual crea un DAG con dos estados predefinidos (Map y Reduce), los grafos DAG creados por Spark pueden tener cualquier número de etapas.

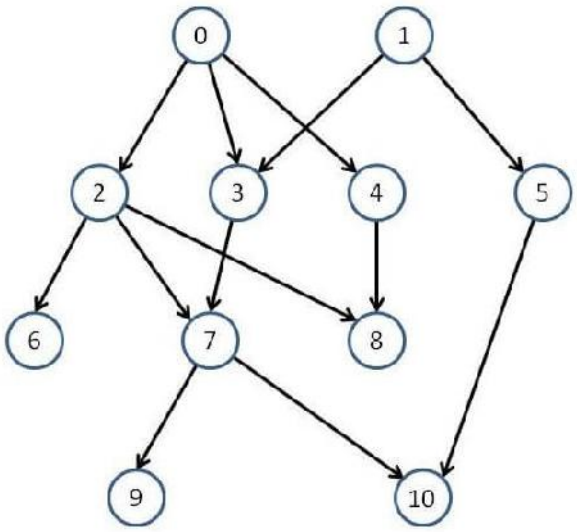


Ilustración 12- DAG- https://geekytheory.com/apache-spark-que-es-y-como-funciona

Por otro lado, nos encontramos con el RDD (Resilient Distributed Dast), el cual permite a los programadores realizar operaciones sobre grandes cantidades de datos en clusters de una manera rápida y tolerante a fallos.

Existen dos tipos de transformaciones, Narrow y Wide, se utiliza Narrow cuando los datos que se necesitan tratar están en la misma partición del RDD y no es necesario realizar una mezcla de dichos datos para obtenerlos todos. Por otro lado, se utiliza Wide cuando la lógica de la aplicación necesita datos que se encuentran en diferentes particiones de un RDD y es necesario mezclar dichas particiones para agrupar los datos necesarios en un RDD determinado. Se puede ver un ejemplo gráfico de cómo funciona cada una de ellas en la ilustración 13.

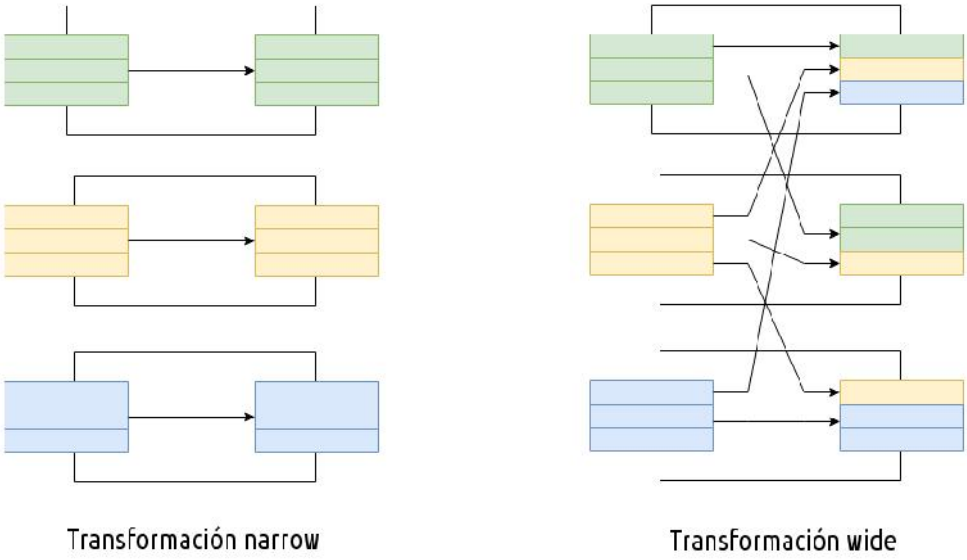


Ilustración 13- Transformación Narrow y Wide- https://geekytheory.com/apache-spark-que-es-y-como-funciona

6.4. Procesamiento de datos en streaming

6.4.1. Apache Storm

Storm es un sistema extremadamente rápido y es capaz de procesar hasta más de un millón de registros por segundo por nodo en un cluster de modesto tamaño.

Storm está dotado de una API simple y fácil de usar, y también es programable en diferentes lenguajes como Ruby, Python, JavaScript o Perl.

Por último, destacar que es un sistema escalable y tolerante a fallos.

6.4.2. Spark Streaming

Spark Streaming es una herramienta de Apache Spark utilizada para el procesamiento de datos en tiempo real.

Para empezar, Spark es un framework de análisis distribuido en memoria y nos permite ir más allá de las operaciones en batch de Hadoop MapReduce.

Por otro lado, Spark puede coexistir con distintas arquitecturas de Big Data y entiende el lenguaje SQL.

6.5. Elección de herramientas

En esta sección se determinará cada herramienta de cada capa de la arquitectura.

Empezaremos eligiendo la herramienta del procesamiento y análisis de los datos, sin tener mucho en cuenta el entorno streaming, puesto que los datos en tiempo real utilizados en este trabajo son casi insignificantes. En segundo lugar, se determinará la herramienta de recolección de datos y, por último, la herramienta de almacenamiento.

6.5.1. Hadoop vs Spark

Velocidad

Como se ha comentado anteriormente, Apache Spark es hasta 100 veces más rápido que Map Reduce, que es la tecnología que utiliza Hadoop. Esta ventaja se debe a que a diferencia de Hadoop, Spark trabaja en memoria RAM, ahorrándose almacenar los resultados intermedios en disco, de forma que acelera enormemente los tiempos de procesamiento. En cuanto a Hadoop, necesita menos memoria para el almacenamiento, y al ocuparse de eliminar los datos cuando no son ya necesarios, no produce pérdidas de rendimiento significativas para aplicaciones pesadas.

Por otro lado, Apache Spark consta de DAG, un planificador que establece las tareas a realizar y optimiza los cálculos.

Complejidad

Por un lado, nos encontramos con Hadoop el cual se programa principalmente con Java, a pesar de contar con la compatibilidad con otros lenguajes. Por otro lado, nos encontramos con Apache Spark, el cual es muy conocido por su facilidad de uso, ya que viene con API fáciles de usar para Scala, Java, Python y Spark SQL.

No podemos obviar la existencia del modo interactivo de Spark, el cual permite que tanto los desarrolladores como los usuarios puedan tener comentarios inmediatos sobre consultas y otras acciones.

Conclusión

A pesar de haber comentado el hecho de que los datos en streaming no son fundamentales en este proceso, cabe destacar que Hadoop necesita tecnología adicional para procesar estos datos. En cambio, Apache Spark cuenta con su propia tecnología, Apache Spark Streaming, por lo que la herramienta elegida es *Apache Spark*.

6.5.2. Kafka vs Flume

Una vez elegida nuestra herramienta de procesamiento de datos, tomaremos ciertas decisiones en base a esto.

Compatibilidad

Como se ha comentado, la herramienta escogida como *core* se trata de Apache Spark. Flume está muy bien integrado con el ecosistema de Hadoop, a pesar de ser compatible con otros sistemas. En cambio, Kafka se puede utilizar en prácticamente cualquier plataforma. Una ventaja significativa de Flume es su eficiencia al trabajar con HDFS.

Funcionamiento

El uso de logs de Kafka frente al uso de eventos de Flume lo hace más completo y polivalente, dado que, no está sujeto siempre al mismo tipo de comunicación.

Escalabilidad

Dada la necesidad de Flume de cambiar de topología a la hora de añadir nuevos clientes, la escalabilidad de Kafka es significativamente superior.

Envío de datos

En la herramienta Kafka, el consumidor es el encargado de pedir la información que se quiere enviar, al contrario de Flume, la cual es de tipo push, es decir, la propia herramienta envía la información, de modo que existen situaciones en las que el consumidor recibe una información que no es capaz de procesar.

Conclusión

A pesar de las ventajas que nos ofrece Flume a la hora de trabajar con HDFS, Kafka nos ofrece mayores ventajas en nuestro proyecto, como pueden ser la escalabilidad o el envío de datos. A pesar de estas características, el hecho de que la herramienta elegida para procesar los datos sea Apache Spark, la compatibilidad que nos ofrece Kafka con el mismo es una ventaja bastante notable, por lo que la herramienta escogida es *Kafka*.

6.5.3. SQL vs NoSQL

Escalabilidad

Las bases de datos SQL escalan normalmente de forma vertical, en cambio, las bases de datos NoSQL escalan de forma horizontal, es decir, gestiona una mayor cantidad de datos al añadir o fragmentar servidores. Esto hace que NoSQL sea la opción elegida normalmente en proyectos donde se manejan bases de datos grandes o con cambios frecuentes.

Soporte

Las bases de datos SQL reciben un gran apoyo por parte de sus vendedores y existe una gran cantidad de recursos disponibles gracias a la cantidad de años que llevan activas. En cambio, las bases de datos NoSQL, al tratarse de unas herramientas relativamente nuevas, no están tan asentadas en el entorno, por lo que hay que recurrir a comunidades o foros específicos a la hora de solucionar un problema.

Estructura

La estructura de las bases de datos NoSQL es mucho más flexible y dinámica, se debe a que las bases de datos SQL están basadas en tablas, mientras que las bases de datos NoSQL están basadas en clave–valor, mapeo de columnas o grafos.

Conclusión

A pesar de que el soporte recibido por las bases de datos SQL es mucho mayor, el hecho de que los datos manejados en este proyecto no estén estructurados y aumenten considerablemente cada poco tiempo nos hacedecantarnos hacia un lado, por lo tanto, la tecnología escogida es **NoSQL**.

6.5.4. MongoDB vs Cassandra

Una vez optado por un tipo de base de datos, se estudiarán las distintas características entre ciertas herramientas usuarias de esa tecnología, con el fin de decantarnos por una de ellas.

Lenguaje

Por defecto, las bases de datos NoSQL suelen tener lenguajes propios, pero este no es el caso de Cassandra, la cual utiliza el lenguaje CQL, derivado de SQL, en cambio, MongoDB utiliza un lenguaje parecido a un flujo de instrucciones mediante JSON.

Escalabilidad

Cassandra ofrece escalabilidad y alta disponibilidad con un mínimo de administración, mientras que MongoDB es una gran alternativa si se gestionan datos no estructurados o estructurados sin una clara definición.

Almacenamiento

MongoDB almacena todo en documentos, en cambio Cassandra maneja columnas con ancho flexible y la información se almacena en objetos similares a las tablas de una base de datos relacional.

Conclusión

MongoDB se trata claramente de una herramienta más rica que Apache Cassandra, en cambio, el hecho de tener que aprender JSON y a usar la propia herramienta MongoDB es una clara desventaja frente a Cassandra, ya que su lenguaje es muy parecido a SQL y consta de una mayor escalabilidad. Por lo tanto, la herramienta escogida es Apache Cassandra.

7. Descripción de la solución propuesta

7.1. Diseño de alto nivel

En esta sección, se describen los componentes principales de nuestra plataforma y las conexiones que se establecen entre sí.

La herramienta Kafka es la encargada de la recolección de datos, los cuales pueden provenir de múltiples fuentes y pueden tratarse de datos tanto estructurados como no estructurados.

Se cuenta con Kafka Connector para poder establecer la conexión con Spark, la cual nos ayuda a procesar la información recibida.

Por último, los datos se almacenan en la base de datos Cassandra, la cual ha sido escogida debido a su flexibilidad y su escalabilidad frente a otras bases de datos, la cual permitirá aumentar la plataforma cuando se desee.

7.2. Caso de uso

Para demostrar la consistencia de la arquitectura planteada, se realizará un caso de uso, el cual mostrará la información recopilada de los posibles enfermos, comprobando que los datos han sido procesados correctamente.

Se ha decidido recoger ciertos datos obtenidos de análisis hematológicos y estudiar su relación con la diabetes de tipo 2. Estos datos serán enviados y procesados, realizando un análisis de los mismos determinando los pacientes que se encuentran en riesgo y deban realizarse un estudio más específico. De esta manera, se generará una lista de pacientes la cual podrá ser almacenada en Cassandra.

8. Implementación de caso de uso

En este apartado se desarrollará un caso de uso, mostrando el funcionamiento de la arquitectura mostrada en el apartado anterior.

En este caso hemos accedido a una serie de datos hematológicos obtenidos mediante análisis de sangre. Por otro lado, se ha realizado un estudio de la relación de dichos factores con la diabetes de tipo 2.(40)

Las relaciones obtenidas son las siguientes:

- Basofilos (x10E3 cells/uL) [0-0.015]
- Eosinofilos (x10E3 cells/uL)[0-0.005]
- Hematocritos (L/L)[0.42-0.53]
- Hemoglobina (g/L)[130-180]
- Linfocitos (x10E3 cells/uL)[0.205-0.455]
- Monocitos (x10E3 cells/uL)[0-0.15]
- Neutrofilos (x10E3 cells/uL)[0.2-0.65]
- Plaquetas (x10E3 cells/uL)[0.42-0.53]

El caso de uso ha sido desarrollado en el lenguaje de programación Python, el cual, es un lenguaje de programación orientado a objetos, que se puede utilizar con Spark, Kafka y Cassandra.

El análisis consiste en determinar qué pacientes participantes en un análisis de sangre tienen mayor probabilidad de ser enfermos de diabetes de Tipo 2.

8.1. Kafka Producer

Antes de comenzar con el caso de uso, es necesario instalar la última versión de Kafka e iniciar un servidor. Una vez instalado abrimos dos terminales, iniciando la instancia Zookeeper en uno y un servidor Kafka en el otro.

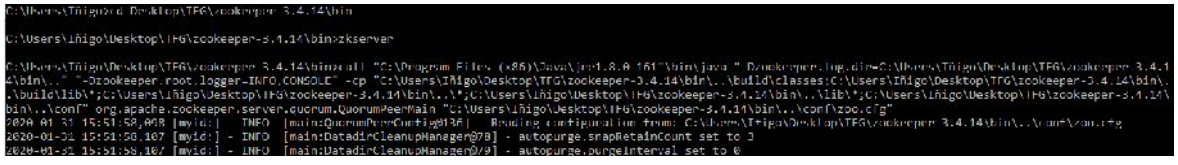


Ilustración 14-Instancia Zookeeper

Ilustración 15-Servidor Kafka

A la hora de realizar una conexión entre un productor y un consumidor de Kafka es necesario crear un topic, en nuestro caso será tfg.

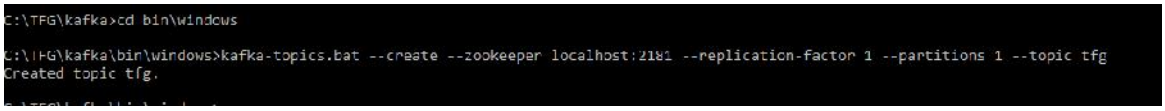


Ilustración 16-Topic Kafka

Una vez determinado el topic se prueba que existe una conexión creando una instancia productor y otra consumidor, cada una de ellas en un terminal distinto.

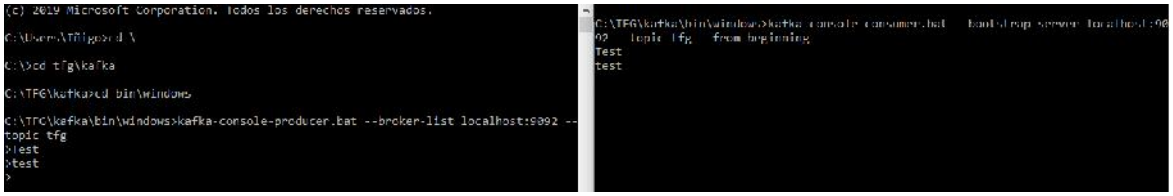


Ilustración 17-Productor y consumidor Kafka

Como se ha comentado anteriormente, en este análisis se estudian 8 factores, los cuales son enviados por el productor, esto se hará ejecutando el script EnvioDatosPacientes.py.

Para ello, se recogen los datos del dataset, el cual está dividido en columnas. Acto seguido, se convierten los datos de cada extracción en formato JSON, y seenvían mediante la conexión Kafka creada anteriormente.

```python
def enviarDatos(extraccion,basophils,hematocrit,hemoglobin,lymphocites,monocytes,eosinophils,neutrophils,platetes):

    extraccion=extraccion[-1]
    extraccion.pop(-1)
    basophils=basophils[-1]
    basophils.pop(-1)
    hematocrit=hematocrit[-1]
    hematocrit.pop(-1)
    hemoglobin=hemoglobin[-1]
    hemoglobin.pop(-1)
    lymphocites=lymphocites[-1]
    lymphocites.pop(-1)
    monocytes=monocytes[-1]
    monocytes.pop(-1)
    eosinophils=eosinophils[-1]
    eosinophils.pop(-1)
    neutrophils=neutrophils[-1]
    neutrophils.pop(-1)
    platetes=platetes[-1]
    platetes.pop(-1)

    data={
        'extraccion':extraccion,
        'basophils':basophils,
        'hematocrit':hematocrit,
        'hemoglobin':hemoglobin,
        'lymphocites':lymphocites,
        'monocytes':monocytes,
        'eosinophils':eosinophils,
        'neutrophils':neutrophils,
        'platetes':platetes
    }

    data=json.dumps(data)
    try:
        producer.send(topic=topic,value=data)
        logger.debug('Enviado datos a Kafka de la extracion:',extraccion[1])
    except exception as e:
        logger.warn('Fallo de envio')
```

Ilustración 18-Algoritmo envío de datos

Podemos ver el resultado obtenido en la Ilustración 19, el cual ha sido satisfactorio. Se observan una gran cantidad de mensajes, confirmando el envío de cada extracción.

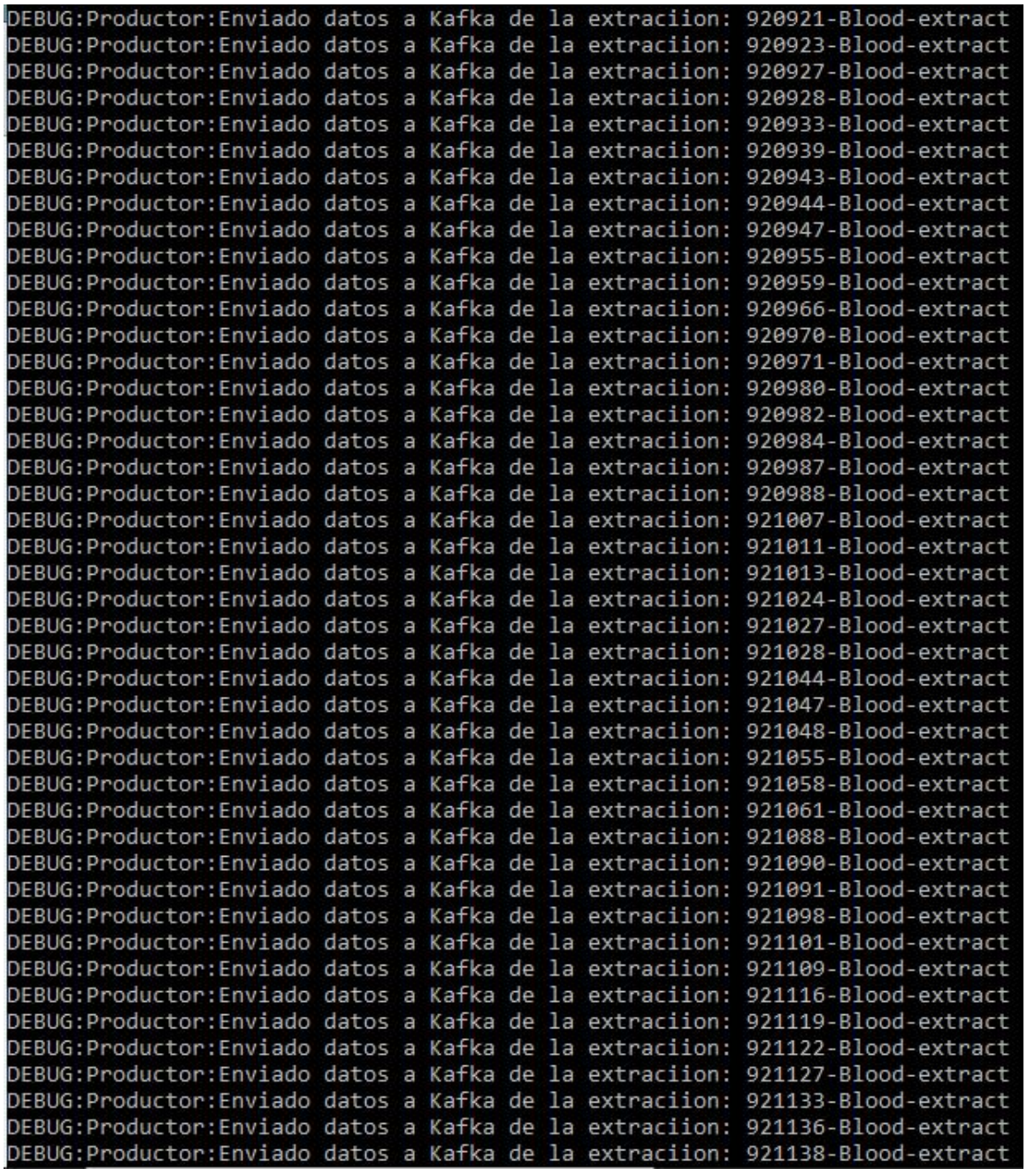


Ilustración 19-Envio de datos de los pacientes

8.2. Kafka Consumer

Mediante la librería pyspark, seremos capaces de procesar la información y de obtener unos resultados. Esto se puede ver en streamPacientes.py.

Para conectar con Kafka se realiza una conexión del Zookeeper pidiendo el topic mencionado anteriormente, en nuestro caso "tfg". Una vez configurada la conexión se realiza un Stream con Kafka, pidiendo que se envíen los mensajes con el topic seleccionado.

Se archiva cada uno de los mensajes recibidos, diferenciando cada uno de los factores.

```
def procesarStream(consumer):

    for message in consumer:

        extraccion.append((record.get('extraccion'))
        basophils.append((record.get('basophils'))
        eosinophils.append((record.get('eosinophils'))
        hematocrit.append((record.get('hematocrit'))
        hemoglobin.append((record.get('hemoglobin'))
        lymphocites.append((record.get('lymphocites'))
        monocytes.append((record.get('monocytes'))
        neutrophils.append((record.get('neutrophils'))
        platetes.append((record.get('platetes'))

return extraccion,basophils,hematocrit,hemoglobin,lymphocites,monocytes,eosinophils,neutrophils,platetes
```

Ilustración 20-Recepción de mensajes

Finalmente, se procesa toda la información recibida mostrando qué pacientes tienen riesgo de haber contraído o contraer en un futuro diabetes de Tipo 2. Esto se realiza mediante un algoritmo el cual comprueba cada uno de los factores claves de cada extracción teniendo en cuenta los parámetros normales y mostrando en cada una de las extracciones que factores se no llegan o se exceden de estos valores.

```
def mostrarResultados(extraccion,basophils,hematocrit,hemoglobin,lymphocites,monocytes,eosinophils,neutrophils,platetes):
    for i in range(len(extraccion)-1):
    i=i+1

    razones=[]

    if(basophils[i]>0.015):
        razones.append("basofilos")

    if(eosinophils[i]>0.005):
        razones.append("eosinofilos")

    if(hematocrit[i]<0.42 or hematocrit[i]>0.53):
        razones.append("hematocritos")

    if(hemoglobin[i]<130 or hemoglobin[i]>180):
        razones.append("hemoglobina")

    if(lymphocites[i]<0.205 or lymphocites[i]>0.455):
        razones.append("linfocitos")

    if(monocytes[i]>0.15):
        razones.append("monocitos")

    if(neutrophils[i]<0.2 or neutrophils[i]>0.65):
        razones.append("neutrofilos")

    if(platetes[i]<0.42 or platetes[i]>0.53):
        razones.append("plaquetas")

    print('El paciente de la extraccion', extraccion[i] ,'debe realizarse un estudio debido a:')

    print(", ".join(map(str,razones)))
    print("")
```

Ilustración 21-Algoritmo filtrado de mensajes

8.3. Resultados

En este apartado se comprueba si la conexión y el algoritmo funcionan correctamente.

```
El paciente de la extraccion 921055-Blood-extract debe realizarse un estudio debido a:
eosinofilos, linfocitos, plaquetas

El paciente de la extraccion 921058-Blood-extract debe realizarse un estudio debido a:
eosinofilos, linfocitos, monocitos, neutrofilos, plaquetas

El paciente de la extraccion 921061-Blood-extract debe realizarse un estudio debido a:
eosinofilos, linfocitos, monocitos, plaquetas

El paciente de la extraccion 921088-Blood-extract debe realizarse un estudio debido a:
eosinofilos, linfocitos, neutrofilos, plaquetas

El paciente de la extraccion 921090-Blood-extract debe realizarse un estudio debido a:
eosinofilos, linfocitos, monocitos, neutrofilos, plaquetas

El paciente de la extraccion 921091-Blood-extract debe realizarse un estudio debido a:
eosinofilos, linfocitos, neutrofilos, plaquetas

El paciente de la extraccion 921098-Blood-extract debe realizarse un estudio debido a:
eosinofilos, linfocitos, neutrofilos, plaquetas

El paciente de la extraccion 921101-Blood-extract debe realizarse un estudio debido a:
eosinofilos, hematocritos, hemoglobina, linfocitos, plaquetas

El paciente de la extraccion 921109-Blood-extract debe realizarse un estudio debido a:
eosinofilos, hematocritos, hemoglobina, linfocitos, monocitos, plaquetas

El paciente de la extraccion 921116-Blood-extract debe realizarse un estudio debido a:
eosinofilos, linfocitos, monocitos, neutrofilos, plaquetas

El paciente de la extraccion 921119-Blood-extract debe realizarse un estudio debido a:
eosinofilos, hematocritos, hemoglobina, linfocitos, neutrofilos, plaquetas

El paciente de la extraccion 921122-Blood-extract debe realizarse un estudio debido a:
eosinofilos, linfocitos, monocitos, plaquetas

El paciente de la extraccion 921127-Blood-extract debe realizarse un estudio debido a:
eosinofilos, hematocritos, hemoglobina, linfocitos, monocitos, plaquetas

El paciente de la extraccion 921133-Blood-extract debe realizarse un estudio debido a:
eosinofilos, hematocritos, hemoglobina, linfocitos, monocitos, plaquetas

El paciente de la extraccion 921136-Blood-extract debe realizarse un estudio debido a:
eosinofilos, hematocritos, hemoglobina, linfocitos, monocitos, plaquetas
```

Ilustración 22-Resultados stream

Se puede ver claramente un número considerable de pacientes que deben realizarse el estudio.

9. Tareas y diagrama de Gantt

En este apartado se expone y se explica cada una de las tareas realizadas para el desarrollo de este proyecto.

9.1 Tareas

T.1. Documentación

La documentación ha sido desarrollada a lo largo de todo el proyecto.

T.2. Familiarización con la temática

Se ha realizado un estudio de la temática tratada en este proyecto, el Big Data, que se ha visto plasmada en el presente documento en la sección del estudio del estado del arte.

T.3. Diseño previo de la arquitectura

Se plantea un diseño genérico de una arquitectura Big Data, explicando brevemente las características de cada fase de la misma.

T.4. Análisis de herramientas

Se realiza un estudio de las diferentes herramientas que pueden emplearse en cada una de las fases del tratamiento de los datos en la plataforma.

T.4.1. Fuentes y recolección de datos

Se estudia el funcionamiento de las herramientas que pueden ser utilizadas a la hora de obtener los datos, como pueden ser las APIs, la minería de datos, Kafka y Flume.

T.4.2. Procesamiento de datos

Se analizan las distintas herramientas de procesamiento de datos, que son Hadoop, ElasticSearch, Apache Spark, el Lenguaje R y Python.

T.4.3. Almacenamiento de datos

Se estudian los diferentes modelos de bases de datos y las herramientas utilizadas. Se detallan las características de las bases de datos relacionales y no relacionales, y por otro lado las características de MongoDB y Apache Cassandra.

T.5. Diseño de una solución

T.5.1. Comparación de las herramientas

Una vez realizado el estudio de las diferentes herramientas, se estudia la compatibilidad de las mismas y se eligela mejor opción para cada fase.

T.5.2. Diseño de la plataforma

Realizar un diseño respecto a las herramientas escogidas y comprender la conexión entre las distintas fases y herramientas.

T.6. Demostración de la solución

T.6.1. Diseño de caso de uso

Implementar un caso de uso que permita comprobar el correcto funcionamiento de la plataforma.

T.6.2. Implementación de caso de uso

Implementar el caso de uso diseñado anteriormente, mostrando y estudiando su funcionamiento.

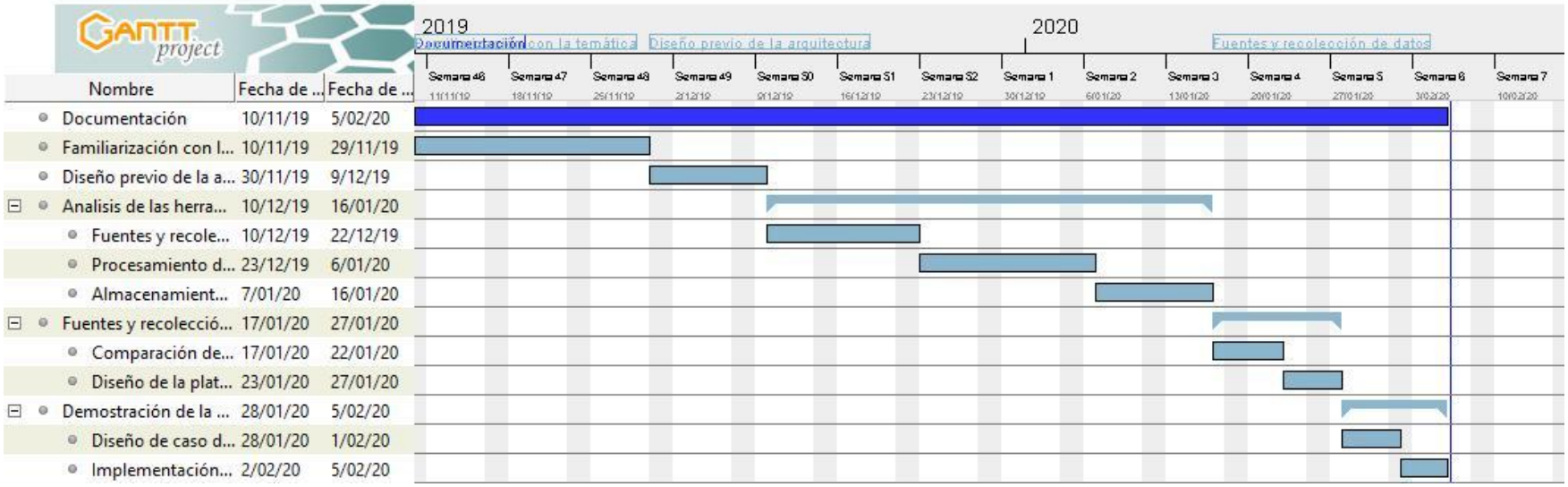


Ilustración 23-Diagrama de Gantt

10. Conclusiones

El Big Data se trata de una tecnología con un gran potencial a la hora de analizar datos, esto ofrece una gran ventaja al realizar ciertos estudios, ofreciendo un mejor diagnóstico y prevención mayoritariamente. De esta forma se puede comprobar cómo ciertas tecnologías o métodos son desplazadas o sustituidas por herramientas cada vez más rápidas y eficientes. Esta tecnología se encuentra en pleno auge, siendo cada vez mayor la necesidad de implementarla en grandes proyectos, debido a la gran cantidad de datos que se procesan hoy en día.

Los objetivos propuestos se han cumplido, consiguiendo una plataforma funcional, como se ha podido comprobar en el caso de uso. La implementación realizada en este proyecto, se trata de una implementación sencilla, la cual puede ser ampliada añadiendo más clusters. Gracias a este proyecto, se ha podido mostrar el uso de ciertas herramientas de la plataforma, aportando de esta manera una visión tanto técnica como práctica.

Tras haber recopilado información sobre la relación entre la diabetes de tipo 2 y ciertos factores hematológicos, se ha implementado una herramienta capaz de enviar esta información mediante Kafka, siendo el Consumer el que implementa un algoritmo capaz de generar una lista que indica qué pacientes deberían realizarse un análisis médico más riguroso, por ser individuos con riesgo de padecer esta enfermedad.

El caso realizado se trata de un ejemplo básico de lo que se podría llegar a implementar en la plataforma diseñada, el ejemplo se ha realizado buscando la comprobación del correcto funcionamiento de la plataforma, lo cual se ha conseguido.

Este caso de uso tiene diferentes aspectos a mejorar para hacerlo más atractivo a la hora de procesar la información. Además del aumento de nodos es posible mejorar:

- Realización de un algoritmo de precisión más preciso para el análisis de datos, teniendo en cuenta una mayor cantidad de factores.
- Obtener una fuente de datos con más información, ya que, al tratarse de datos médicos, la información disponible a nivel de usuario es escasa.

Referencias

1. **www.leonup.com. [En línea] http://www.leonup.com/es/2018/05/04/big-data-las-5-vs-del-big-data/.**

2. tendencias21.net. [En línea] https://www.tendencias21.net/telefonica/La-necesidad-del-uso-del-Big-Data-en-las-empresas_a2202.html.

3. www.bit.es. [En línea] https://www.bit.es/knowledge-center/tipos-de-datos-en-big-data/.

4. smarterworkspaces.kyocera.es. [En línea] https://smarterworkspaces.kyocera.es/blog/diferencia-datos-estructurados-no-estructurados/.

5. www.diegocalvo.es. [En línea] http://www.diegocalvo.es/tipos-de-datos-estructurados-semiestructurados-y-no-estructurados/.

6. Pulid, Francisco Javier. https://es.slideshare.net/FranciscoJavierPulid. [En línea] https://es.slideshare.net/FranciscoJavierPulid/obtencin-de-datos-en-bigdata.

7. www.deustoformacion.com. [En línea] https://www.deustoformacion.com/blog/gestion-empresas/que-es-mineria-datos-big-data.

8. towardsdatascience.com. [En línea] https://towardsdatascience.com/big-data-what-is-web-scraping-and-how-to-use-it-74e7e8b58fd6.

9. www.7puentes.com. [En línea] http://www.7puentes.com/blog/2018/04/09/web-scraping-el-gran-aliado-de-big-data/.

10. www.redhat.com. [En línea] https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces#targetText=Una%20API%20es%20un%20conjunto,de%20saber%20c%C3%B3mo%20est%C3%A1n%20implementados..

11. www.allerin.com. [En línea] https://www.allerin.com/blog/open-apis-in-big-data-analytics.

12. help.twitter.com. [En línea] https://help.twitter.com/es/rules-and-policies/twitter-api.

13. http://www.icorp.com.mx. [En línea] http://www.icorp.com.mx/blog/automatizar-ingesta-de-datos/.

14. http://www.diegocalvo.es. [En línea] http://www.diegocalvo.es/kafka/.

15. http://www.diegocalvo.es. [En línea] http://www.diegocalvo.es/flume/.

16. revistabyte.es. [En línea] https://revistabyte.es/actualidad-byte/big-data/almacenamiento-en-tiempos-big-data/.

17. bigdata.black. [En línea] http://bigdata.black/infrastructure/storage/sql-nosql-differences/.

18. deletesql.com. [En línea] http://deletesql.com/viewtopic.php?f=5&t=4.

19. aws.amazon.com. [En línea] https://aws.amazon.com/es/nosql/.

20. www.acens.com. [En línea] https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf.

21. www.iebschool.com. [En línea] https://www.iebschool.com/blog/mejores-herramientas-big-data/.

22. openwebinars.net. [En línea] https://openwebinars.net/blog/que-es-apache-cassandra/.

23. es.wikipedia.org. [En línea] https://es.wikipedia.org/wiki/Apache_Cassandra#Alternativas_populares.

24. searchdatacenter.techtarget.com. [En línea] https://searchdatacenter.techtarget.com/es/definicion/Analisis-de-big-data#targetText=El%20an%C3%A1lisis%20de%20'grandes%20datos,desconocidas%20y%20otra%20informaci%C3%B3n%20%C3%BAtil.&targetText=Las%20tecnolog%C3%ADas%20relacionadas%20con%20el,grandes%20.

25. www.teldat.com. [En línea] https://www.teldat.com/blog/es/procesado-de-big-data-base-de-datos-de-big-data-clusters-nosql-mapreduce/.

26. www.iic.uam.es. [En línea] http://www.iic.uam.es/innovacion/herramientas-big-data-para-empresa/.

27. www.brainsins.com. [En línea] https://www.brainsins.com/es/blog/principales-tecnologias-big-data-hadoop/107625.

28. www.deustoformacion.com. [En línea] https://www.deustoformacion.com/blog/marketing-digital/que-es-hadoop-que-vinculacion-tiene-con-big-data.

29. es.wikipedia.org. [En línea] https://es.wikipedia.org/wiki/Elasticsearch.

30. openwebinars.net. [En línea] https://openwebinars.net/blog/que-es-apache-spark/.

31. es.wikipedia.org. [En línea] https://es.wikipedia.org/wiki/Apache_Spark.

32. piperlab.es. [En línea] https://piperlab.es/2018/12/11/los-lenguajes-de-programacion-mas-utilizados-en-big-data/.

33. www.maximaformacion.es. [En línea] https://www.maximaformacion.es/blog-dat/guia-rapida-introduccion-al-lenguaje-r/.

34. www.muylinux.com. [En línea] https://www.muylinux.com/2019/06/28/principales-tecnologias-big-data-2019/.

35. es.wikipedia.org. [En línea] https://es.wikipedia.org/wiki/Macrodatos#Tecnolog%C3%ADa.

36. aws.amazon.com. [En línea] https://aws.amazon.com/es/streaming-data/.

37. Ortega, José Felipe. https://fdocuments.ec. [En línea] https://fdocuments.ec/document/obtencion-de-datos-en-bigdata.html.

38. https://www.adictosaltrabajo.com. [En línea] https://www.adictosaltrabajo.com/2014/09/22/introduccion-storm/.

39. http://www.diegocalvo.es/. [En línea] http://www.diegocalvo.es/spark-streaming/.

40. figshare.com. [En línea] https://figshare.com/articles/Hematology_Raw_Data/1605230/1.

41. https://archive.ics.uci.edu. *https://archive.ics.uci.edu*. [En línea] https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Coimbra.

42. www.powerdata.es. [En línea] https://www.powerdata.es/big-data.

43. es.wikipedia.org. [En línea] https://es.wikipedia.org/wiki/Macrodatos.

44. ideasparatuempresa.vodafone.es. [En línea] https://ideasparatuempresa.vodafone.es/big-data-desde-los-inicios-hoy/.

45. es.wikipedia.org. [En línea] https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones.

46. https://bites.futurespace.es. [En línea] https://bites.futurespace.es/2019/01/15/ingesta-es-mas-que-una-mudanza-de-datos/.