TEMA 2

MÓDULO:
ARQUITECTURA E INFRAESTRUCTURA
BIG DATA

ARQUITECTURA BIG DATA

MARC PLANAGUMÀ I VALLS

Licenciado en Telecomunicaciones
por la UPC-ETSETB.
Data engineer.

Institut de Formació Contínua-IL3
UNIVERSITAT DE BARCELONA

ÍNDICE

OBJETIVOS

- Obtener una visión de conjunto del diseño de arquitecturas para generar plataformas de análisis *big data*.
- Entender los principios básicos de la arquitectura por capas big data.
- Comprender en detalle los objetivos y estrategias de cada una de las fases o capas dentro de las plataformas de datos.
- Conocer arquitecturas de referencia para el diseño de plataformas big data.

2. ARQUITECTURA BIG DATA

En el tema 1 hemos profundizado en detalle en el concepto de big data desde la óptica de un ingeniero de datos y hemos visto cómo sus características condicionan su trabajo. Las oportunidades que ofrece la extracción de valor de los datos es lo que alimenta el vertiginoso crecimiento del sector, mientras que las dificultades tecnológicas y organizativas que plantea el big data son los retos por los que existe tanta demanda de *data engineers*.

En este tema empezaremos a tratar la búsqueda de soluciones big data desde una perspectiva de arquitectura que englobe herramientas y paradigmas en una solución final.

2.1. PRINCIPIOS DE ARQUITECTURA BIG DATA

Para ser capaz de extraer información valiosa y significativa de los datos, necesitamos crear un entorno de trabajo **idóneo y eficiente**. El diseño de soluciones big data no suele enfocarse como la creación de una solución específica a un problema concreto, sino que su finalidad principal es la de **encontrar la verdad o el valor** que esconden los datos.

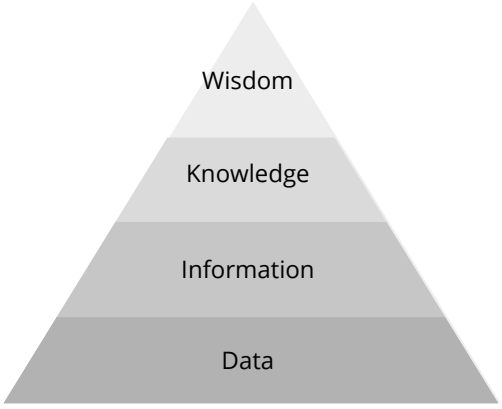
IMPORTANTE

Es muy común el uso del término en inglés ***insight*** para referirse a la conclusión de valor extraída de los datos.

Vamos a diferenciar el enfoque original de las soluciones analíticas (*problem driven*) del nuevo paradigma de analítica avanzada (*data driven*):

- El **enfoque dirigido por el problema (problem driven)**: se centra en encontrar solución a una pregunta muy precisa con unos objetivos muy concretos. En el sector de la industria esta descripción detallada se conoce como *caso de uso*. Una vez planteado el problema, se utiliza la tecnología y los expertos para proporcionar la solución que mejor responda a la pregunta.

- El **enfoque dirigido por los datos (data driven)**: parte de unas preguntas y objetivos mucho más generales (por ejemplo: ¿cómo puedo mejorar la producción? ¿Puedo encontrar un patrón de conducta? ¿Tengo errores o pérdidas no detectadas?). Las respuestas, que en muchos casos son a nuevas preguntas no imaginadas previamente, deben encontrarse en los datos. Este proceso sigue la denominada *jerarquía del conocimiento*, también conocida como ***pirámide DIKW***, siglas de los términos ingleses *data* (datos), *information* (información), *knowledge* (conocimiento) y *wisdom* (saber).



Jerarquía del conocimiento o pirámide DIKW.

Como define la pirámide DIKW, para llegar a la sabiduría debemos partir de una gran cantidad de datos, analizarlos para extraer información, relacionar esa información para generar conocimiento y acumularlo.

Este mismo proceso es el que queremos seguir cuando diseñamos una solución big data. Debemos buscar una solución que capacite el trabajo sobre datos digitales con distintas finalidades y desde distintas disciplinas. Por este motivo, las soluciones tecnológicas big data son llamadas, a menudo, *plataformas*, ya que persiguen la finalidad de **proveer un entorno de trabajo** más que una solución específica a un problema.

IMPORTANTE

La **arquitectura big data** se centra, pues, en la creación de plataformas de datos. Se trata de plataformas que proporcionan los servicios necesarios para el trabajo con los datos, como sería su captación, integración, persistencia, procesamiento, modelado, análisis y visualización, entre otros.

Niveles de servicio del análisis big data.
Fuente: Raj, Raman y Subramanian (2017).

La pirámide anterior representa los distintos niveles de servicio del mundo de la analítica big data: desde la plataforma de base hasta los *insights* extraídos de los datos, pasando por diferentes disciplinas de análisis de datos.

RECUERDA

El big data no aborda una sola tecnología o un solo ámbito de trabajo, sino un conjunto de herramientas, procesos y tecnologías sobre uno o varios sistemas de infraestructuras.

Como podemos observar, en la base se encuentra la **plataforma de datos** (*data platform*), que es el ámbito de trabajo principal de un *data engineer* y proporciona la base de todo el trabajo analítico que se llevará a cabo sobre los datos.

SABÍAS QUE...

Por lo general, la plataforma de datos es un producto interno, de vital importancia, en las compañías y organizaciones que realizan análisis big data.

Es en las siguientes capas donde encontramos servicios y disciplinas de otros sectores, como *data science* (DS) o *busines intelligence* (BI):

- Los DS son los responsables de crear productos analíticos como el análisis descriptivo, predictivo o prescriptivo.
- Los BI son los responsables de generar reportes, informes o visualizaciones, también llamados de forma genérica *dasboards*.

Tanto DS como BI son los principales clientes de las *data platforms* y, por tanto, también lo son de los data engineers (DE). Aunque exista una división por capas para diferenciar los niveles de servicio, en realidad, tal división del ámbito de trabajo entre perfiles DE, DS y BI no es tan clara, ya que se produce un solapamiento importante de conocimientos y responsabilidades.

Como hemos visto, todo el trabajo analítico reposa sobre una plataforma de datos. Para abordar todos esos desafíos de manera eficiente, existe una arquitectura de referencia en el sector. Se trata de la ***layer based architecture*** o **arquitectura basada en capas**, que aborda distintas necesidades con diferentes aproximaciones tecnológicas.

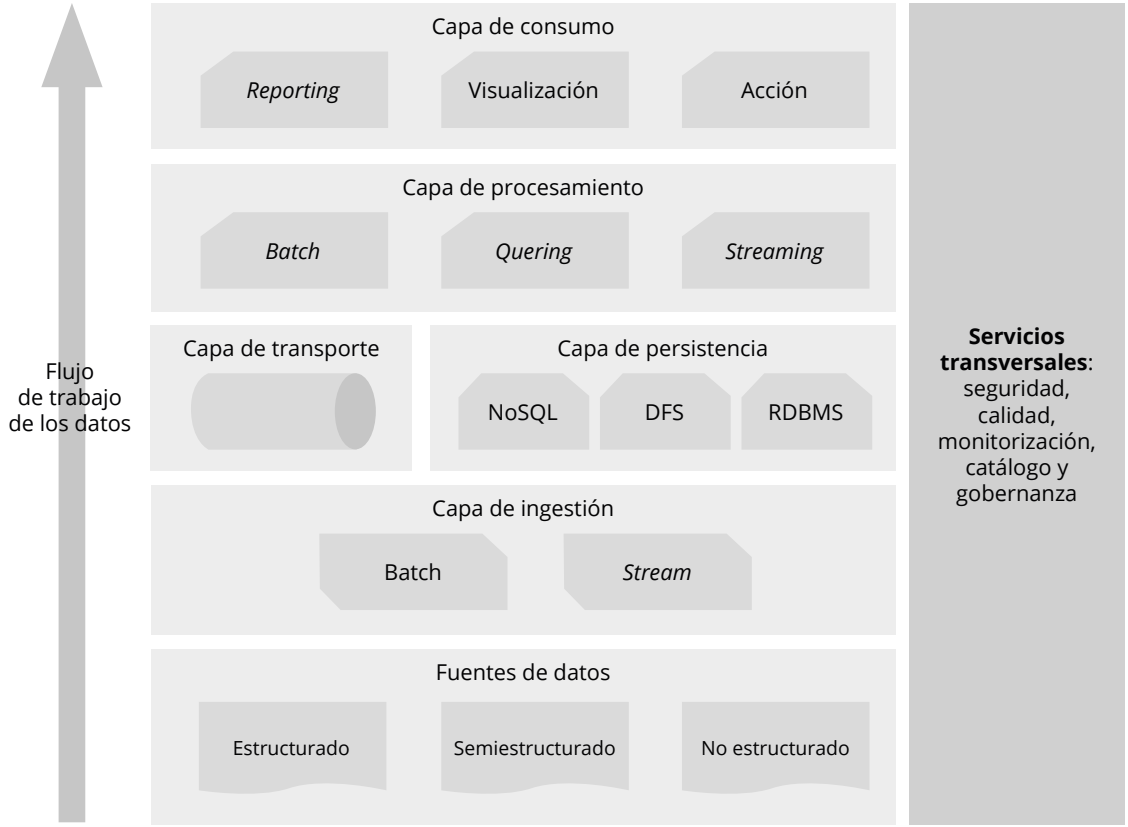
En este tema analizaremos distintas opciones de arquitectura para construir plataformas de datos, pero empezaremos por definir la arquitectura base de toda solución big data desde el punto de vista técnico.

2.2. ARQUITECTURA BASE O ARQUITECTURA POR CAPAS

Las **capas** permiten organizar diferentes componentes para abordar los problemas y representar **funciones únicas** como las siguientes:

- **Ingesta de fuentes de datos**: los datos provenientes de varios canales, con distintas tecnologías y comportamientos de llegada, deben ser ingeridos por el sistema sin perder ninguna de sus cualidades.
- **Mensajería y transporte de datos**: los datos ingeridos por el sistema deben poderse transportar, distribuir y/o entregar a otros componentes para que cada uno pueda cumplir su misión con los requerimientos deseados.
- **Almacenamiento de datos**: los datos ingeridos o generados dentro de la plataforma deben poderse persistir del modo más adecuado, eficiente y escalable obedeciendo a distintas demandas de capacidad de servicio y formato de los datos.
- **Análisis de datos**: los datos deben poderse procesar con acciones como acceso, exploración, indexado, transformación, agregado, cruce, cálculo y modelado, entre otras, para habilitar todas las disciplinas de análisis.
- **Consumo de datos**: los resultados también son datos que deben consumirse con el formato y las herramientas más adecuados para su finalidad de servicio. Visualizaciones, paneles, informes, indicadores e incluso acciones pueden ser opciones de consumo de los datos.

En el diagrama siguiente se representan las diferentes capas de una arquitectura por capas big data. Estas se pueden considerar un resumen de las fases por las que transitan las aplicaciones de análisis big data.



2.2.1. CAPA DE INGESTIÓN

IMPORTANTE

La capa de ingestión engloba las soluciones que permiten recoger, importar e integrar datos de fuentes externas dentro del sistema.

Es la responsable de **conectar distintas fuentes de datos** adaptándose a las distintas tecnologías de conexión y comunicación, así como de **resolver la transformación de formato** para hacer compatible el cumplimiento del contrato externo con la fuente y el contrato interno de consumo de los datos dentro de la plataforma.

También es la responsable de **adaptarse** a los distintos comportamientos de las fuentes dependiendo de cómo estas ponen a disposición los datos.

Las posibles **opciones** son:

- ***Push***: los datos son mandados directamente por la fuente al sistema de ingestión que es capaz de recogerlos.
- ***Pull***: los datos son recogidos por el sistema de ingestión periódicamente en una localización específica.
- **Batch**: los datos se encuentran agregados por lotes.
- **Stream**: los datos se reciben mensaje a mensaje de forma secuencial.

2.2.2. CAPA DE TRANSPORTE

IMPORTANTE

La capa de transporte engloba las soluciones de traslado de datos entre componentes. Los datos se transportan en mensajes entendiendo un **mensaje** como un objeto individual que define un **concepto atómico de dominio**.

Las modalidades de transporte pueden ser múltiples y tienen distintos comportamientos según las necesidades de distribución de los mensajes. Algunas de estas **opciones** son:

- **Colas**: contenedor de mensajes en el que el emisor puede dejar mensajes y el receptor consumirlos. Una vez consumido el mensaje por un receptor, este desaparece de la cola. Su comportamiento puede ser **FIFO** (primero entra, primero sale) o **LIFO** (último entra, primero sale).
- ***Topics***: contenedor de mensajes en los que el emisor puede dejar mensajes y múltiples receptores pueden consumirlos. El mensaje puede ser eliminado del *topic* una vez pasado un tiempo o puede persistirse de forma indefinida.

- **Enrutadores**: contenedor de mensajes en el que el emisor puede dejar mensajes y estos son dirigidos a un emisor determinado dependiendo de su naturaleza.
- **Pub/Sub**: contenedor de mensajes en el que el emisor puede dejar mensajes que reciben, de forma automática, todos los emisores que hayan hecho una suscripción.

2.2.3. CAPA DE PERSISTENCIA

IMPORTANTE

La capa de persistencia engloba las soluciones de **almacenamiento de datos** necesarias para la plataforma.

Estas soluciones pueden pertenecer a varias tipologías de almacenamiento, como las bases de datos NoSQL, los sistemas de ficheros distribuidos o las bases de datos relacionales.

Esta capa es la responsable de mantener la **durabilidad de los datos** y exponer su **acceso para su uso**.

Como ya hemos visto, las tecnologías de persistencia pueden variar mucho respecto al formato de los datos que guardan y a su forma de acceso. La decisión de qué tecnologías emplear dependerá de los requerimientos de los datos y de la solución de análisis, y afectará al diseño de la arquitectura dentro de la capa de persistencia.

2.2.4. CAPA DE PROCESAMIENTO

IMPORTANTE

La capa de procesamiento engloba las soluciones de **tratamiento de datos** destinadas a cubrir todas las necesidades de análisis de los equipos analíticos.

Estas capacidades se encuentran clasificadas dentro de **tres tipologías** de tratamiento de datos:

- **Quering**: acceso y transformación de datos a base de peticiones detalladas en leguajes de consulta.
- **Batch**: capacidad de acceso y transformación de lotes de datos a base de codificación de instrucciones en una aplicación denominada *job*.
- **Streaming**: capacidad de acceso y transformación de datos de forma continua y reactiva a su llegada a base de codificación de instrucciones en una aplicación denominada *topología*.

Esta es la capa responsable de **aportar las tecnologías** que permitan el desarrollo y despliegue de la lógica analítica que quiera aplicarse a los datos. Desde limpiar datos y crear modelos hasta extraer patrones o entrenar sistemas de aprendizaje automático (*machine learning*). Por ello, es una de las capas que requiere una mayor complejidad en el diseño de su arquitectura, como veremos más adelante.

2.2.5. CAPA DE CONSUMO

IMPORTANTE

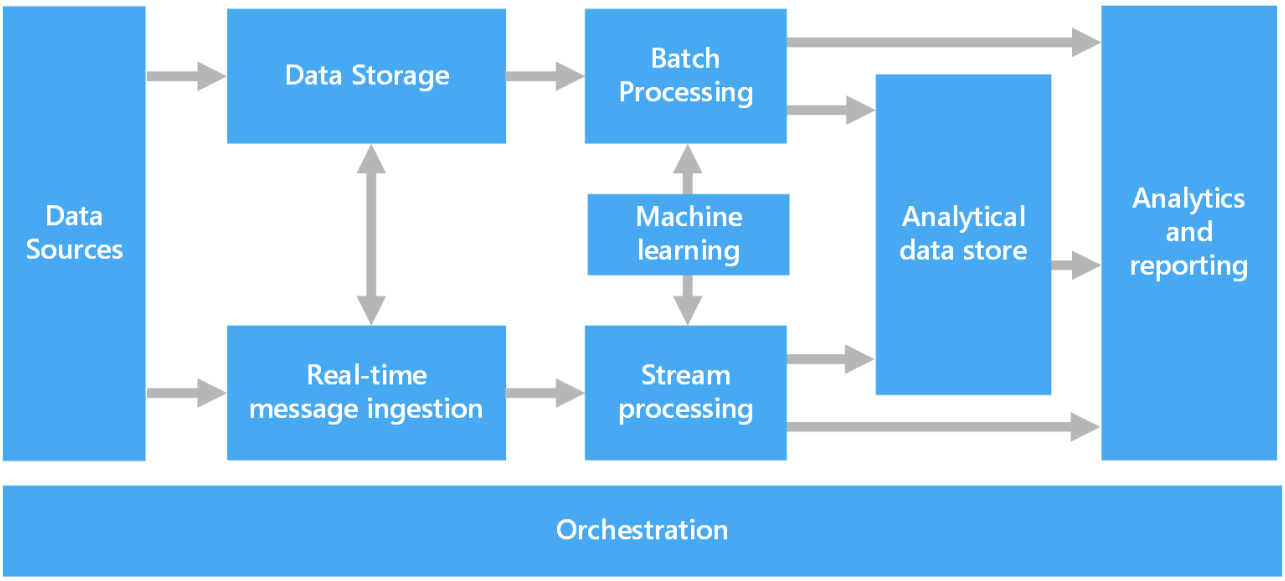
La capa de consumo engloba las soluciones de **presentación de los resultados** a los sistemas o actores que actúan como clientes.

Es la responsable tanto de dar forma al resultado final como de ser la **interfaz de operación** del servicio para el cliente final.

Los **resultados** se entregan principalmente de tres formas distintas:

- **Visualizaciones**: representación visual de los resultados de forma explicativa.
- **Datos**: representación cuantitativa y descriptiva de los resultados.
- **Acciones**: ejecución de una operación a partir de un resultado.

2.3. ARQUITECTURAS BIG DATA DE REFERENCIA



Componentes básicos de una arquitectura big data
Fuente: Big data architectures (Microsoft Azure).

A continuación veremos **tres arquitecturas de referencia** en el mercado que cumplen el patrón de la base *layer architecture*, aunque aportan distintos tipos de soluciones big data.

2.3.1. ARQUITECTURA DE DATA LAKE

El patrón de arquitectura de *data lake* fue creado por Jame Dixon (CTO de Pentaho) en 2010. Es uno de los patrones de referencia en las arquitecturas big data.

Su planteamiento principal es el de actuar como **repositorio único y central de datos** empezando por los datos en su formato original, también conocido como *formato raw* o *crudo*. Este requerimiento obliga a la arquitectura data lake a ser capaz de almacenar cualquier formato y estructura de datos.

Esta arquitectura también proporciona capacidad de **transformación** de los datos, así como la **persistencia** de todos los datos generados durante los procesos de transformación.

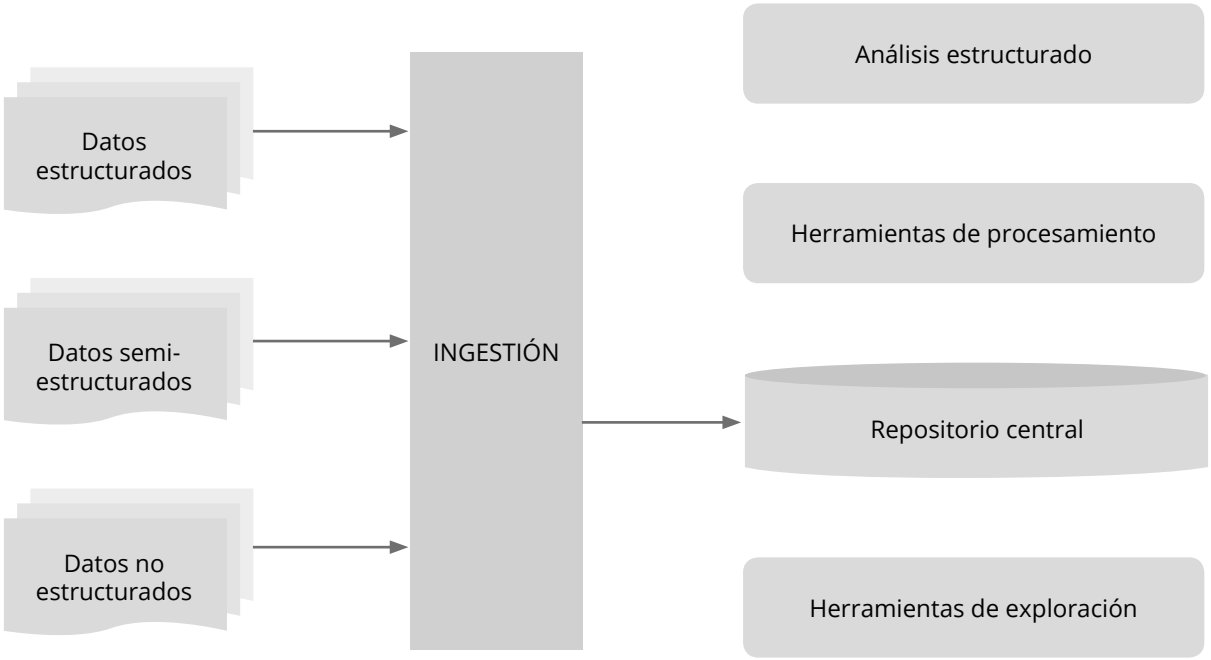
IMPORTANTE

La capacidad de unificar en un sistema los datos en distintos niveles de procesamiento permite lograr de forma eficiente la reutilización de la mayor parte de la infraestructura de datos para cualquier propósito analítico big data dentro de una compañía u organización y, al mismo tiempo, obtener los beneficios de los cambios de paradigma big data.

Los **data lakes** presentan las siguientes **características esenciales**:

- Administrar una gran cantidad de datos sin procesar.
- Retener los datos el mayor tiempo posible.
- Capacidad para gestionar la transformación de los datos.
- Soportar el cambio dinámico de esquema.

En el gráfico de la página siguiente se muestra una implementación del patrón data lake.



Esquema del patrón de arquitectura data lake.

La arquitectura obtiene datos sin procesar de diferentes orígenes en un sistema de almacenamiento central. Los datos recibidos deben conservarse el mayor tiempo posible en el almacenamiento de datos, conocido como **repositorio central**, generando lo que se denomina como *historificación de datos*.

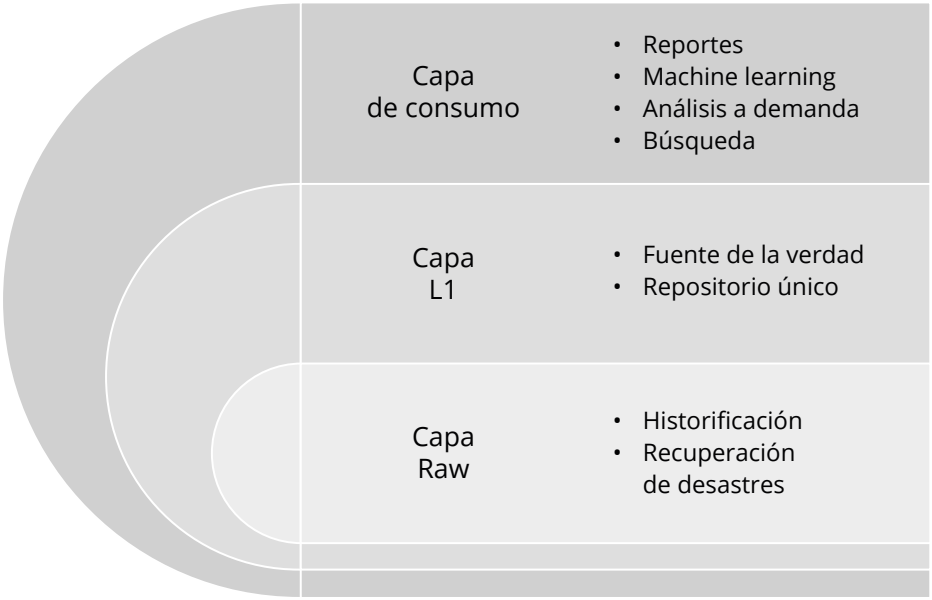
SABÍAS QUE...

En muchos casos, también los data lake disponen de estrategias de *disaster recovery*, que es como se conoce a las estrategias de recuperación de datos en caso de pérdida o accidente de datos.

El repositorio central se compone de una solución de sistema de ficheros distribuido, normalmente mediante la tecnología **HDFS de Hadoop**, pero también pueden ser otras, como S3 de AWS, ADL de Azure, Isilon de EMC, Manila de OpenStack, etc.

La arquitectura incorpora, en el servicio de persistencia, clústeres de procesamiento distribuido como Hadoop o Spark para añadir capacidades de **proceso y exploración** a los datos. La exploración nos permite buscar y acceder al histórico de datos almacenados.

El procesamiento consiente el empleo de técnicas de preparación de datos (como limpiado, filtrado, indexado, modelado, etc.) para transformarlos en formatos más estructurados.



Capas de datos dentro del repositorio central.

Dentro del repositorio de un data lake encontramos varias **capas de datos**:

- La primera es comúnmente llamada *capa raw* (dato crudo) o *landing* (dato de llegada). Aquí es donde se guarda todo el histórico de datos ingeridos en el formato original con el que llegó.

- La capacidad de procesamiento de la arquitectura data lake permite trabajar los datos almacenados en la capa raw de forma que se pueda homogenizar formato, socialización y estructura para crear una nueva capa llamada *L1* o *source of truth* (fuente de la verdad). En esta capa no se modifica, ni se agrega, ni se reduce el contenido de los datos originales que llegan a la plataforma. Su finalidad es poner todos los datos disponibles y operables y con unas altas capacidades de escalabilidad. Podemos concluir que la L1 es donde podemos encontrar en estado original todos los datos de una compañía.

- Por encima de L1, gracias otra vez a la fuerza de computación que aportan los clústeres de procesamiento distribuido, en los data lake, se pueden ir transformando los datos según las finalidades específicas hasta llegar a resultados finales de consumo. Llamamos entonces *capas de staging* a los resultados parciales dentro de un flujo de transformación y *capas de consumo* a los datos finales de dicho flujo.

IMPORTANTE

Los data lake proporcionan un mecanismo para capturar y explorar datos potencialmente útiles uniendo en una sola plataforma los servicios de repositorio central y el sistema de procesamiento de datos.

Su capacidad de almacenar y procesar cualquier formato de dato da a las plataformas data lake una ventaja competitiva enorme respecto a otras plataformas, y permite la creación real de un **único repositorio central** rompiendo los denominados *silos de datos* (es decir, la separación de datos en distintas plataformas que no son interoperables entre sí).

SABÍAS QUE...

Los data lake, en muchos casos, se crean para romper esos silos teniendo todos los datos históricos de la compañía u organización en una misma plataforma.

Los data lake se conectan como ingestores de otras plataformas de datos más específicas por tipos de análisis, estructuras y funcionalidades, como los *data warehouses*, las bases de datos analíticas, los cachés de datos u otras aplicaciones.

El bajo coste del almacenamiento de ficheros y la capacidad de operar directamente sobre ellos sin incurrir en grandes costes adicionales sitúan a los data lake como las plataformas de referencia como repositorio central, con mucha más competitividad que los sistemas transaccionales.

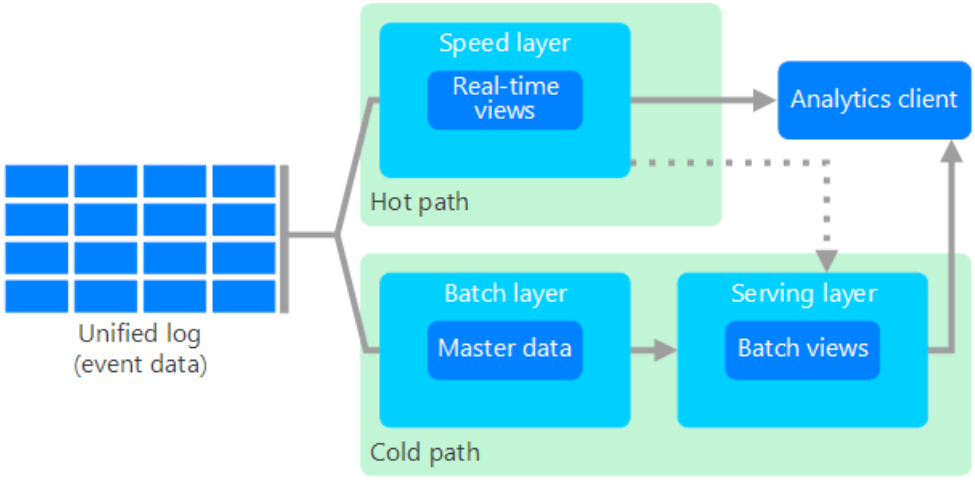
2.3.2. ARQUITECTURA LAMBDA

Para abordar los desafíos big data descritos anteriormente, se hace necesaria una arquitectura de procesamiento de datos que administre cantidades masivas de información para procesar rápidamente grandes lotes de datos (batch) y, al mismo tiempo, analizar transmisiones de datos en tiempo real (streaming).

En 2012, Nathan Marz propuso la arquitectura Lambda para dar solución a este reto en su publicación How to beat the CAP theorem.

Algunas **características fundamentales** de la arquitectura Lambda son:

- Los datos son tratados bajo los principios de atomicidad, inmutabilidad y agregación. Esto significa que no se cambian ni eliminan datos, sino que solo se añaden los nuevos resultados manteniendo los antiguos.
- Aporta su valor en el balanceo de la latencia, el rendimiento y la tolerancia a fallos entre los sistemas batch y streaming.
- Es capaz de interoperar, comparar o buscar la correlación del histórico de datos big data con la analítica en tiempo real.



Esquema del patrón de arquitectura Lambda.
Fuente: Big data architectures (Microsoft Azure).

El diagrama anterior representa la arquitectura Lambda con sus **tres capas primarias**, denominadas *capa de procesamiento por lotes* (batch), *capa de velocidad o de procesamiento en tiempo real* (*speed*) y *capa de servicio* (*serving*), que combina y agrega los resultados de las capas anteriores para responder a las consultas.

Vamos a detallar la **funcionalidad de cada capa**:

- **Capa de lote (batch)**: esta capa precalcula los resultados mediante un sistema de procesamiento distribuido sobre un repositorio de datos históricos de solo lectura. En cada ejecución se actualizan las vistas de resultados reemplazando las vistas precalculadas existentes. La periodicidad de cada ejecución se denomina *tamaño de ventana* y es la que marca cuál es el desfase de actualización de los datos. La capa batch proporciona un alto nivel de exactitud pero, sin embargo, tiene una alta latencia (precisión por encima de la latencia).

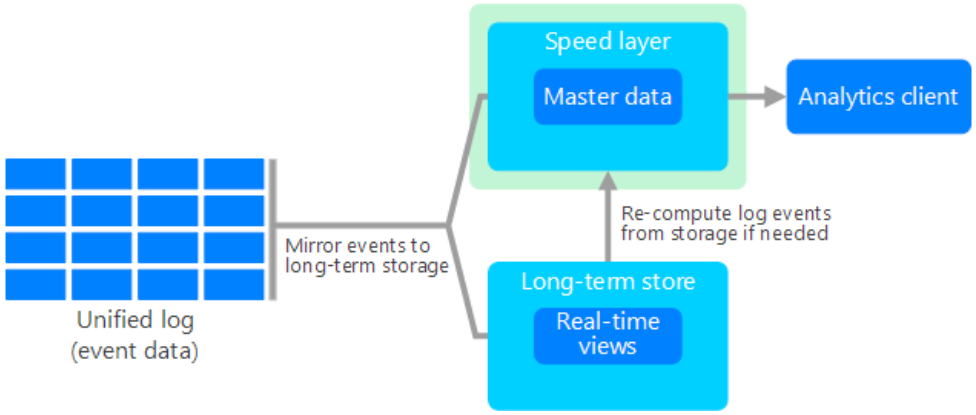
- **Capa de velocidad o en tiempo real (speed)**: esta capa procesa los flujos de datos en tiempo real y las vistas son casi instantáneas pero, al no usar tantos datos en el proceso, proporciona menos precisión (latencia por encima de la exactitud).

- **Capa de servicio (serving)**: esta capa almacena las vistas precalculadas de la capa batch y las vistas de la capa de velocidad para responder a las consultas al sistema. Los resultados provenientes de los datos históricos se capturan del preprocesamiento batch y los resultados provenientes de los últimos datos entrantes, que no fueron procesados por la capa batch, se capturan de la salida de la capa de velocidad.

2.3.3. ARQUITECTURA KAPPA (TIEMPO REAL)

La mayoría de las empresas modernas necesitan un procesamiento continuo y en tiempo real de los datos no estructurados para sus aplicaciones big data. En 2014, JayKreps propuso la arquitectura Kappa en su artículo Questioning the Lambda Architecture, donde señalaba los posibles puntos débiles de la arquitectura Lambda y cómo solucionarlos mediante una evolución. Su propuesta consiste en eliminar la capa batch dejando solamente la **capa streaming**.

La capa streaming se implementa a base de motores de procesamiento de eventos (procesadores de eventos) que tienen una capacidad de memoria considerable y que desencadenan una lógica de procesamiento concreta por cada evento específico que llega. La conexión de estos componentes de procesamiento de eventos, interconectando la salida de unos con la entrada de otros, genera lo que llamamos una ***topología***. El diseño de la topología es lo que define la solución analítica deseada.



Esquema del patrón de arquitectura Kappa.
Fuente: Big data architectures (Microsoft Azure).

Las implementaciones de streaming en tiempo real deben tener las siguientes **características**:

- Minimizar la latencia mediante el uso de grandes cantidades de memoria.
- Procesar los eventos de forma atómica e independiente entre sí, de modo que la solución sea fácilmente escalable.
- Proporcionar herramientas para una transformación de datos reactiva a la llegada del dato.
- No usar componentes centralizados que supongan un cuello de botella obligatorio para todos los flujos de trabajo.

El streaming, a diferencia del batch, no tiene un comienzo ni un fin desde un punto de vista temporal y está continuamente procesando nuevos datos a medida que van llegando. Pero un proceso batch se puede entender como un **stream acotado**; podríamos decir que el procesamiento batch es un subconjunto del procesamiento en streaming con inicio y fin. Así pues, la arquitectura Kappa, además de procesar los mensajes o eventos de entrada, también los almacena en un repositorio tal y como llegan (*longterm store*).

IMPORTANTE

La capacidad de reproducir la entrada de datos streaming original a partir de una selección de eventos entre dos fechas en el repositorio nos aporta la capacidad de recalcular la salida del análisis sobre todos los datos históricos deseados.

La arquitectura Kappa, pues, consiste en una simplificación de la arquitectura Lambda, en la que se elimina la capa batch y todo el procesamiento se realiza en una sola capa denominada *de tiempo real* o *speed layer*, dando soporte a procesamientos tanto batch como en tiempo real y tratándolo todo como un flujo de datos.

Podemos decir que los **cuatro pilares principales** de la arquitectura Kappa son los siguientes:

- **Todo es un stream**: las operaciones batch son un subconjunto de las operaciones de streaming, por lo que todo puede ser tratado como un stream.
- **Los datos de partida no se modifican**: los datos se almacenan sin ser transformados y las vistas se derivan de ellos. Un estado concreto se puede recalcular, puesto que la información de origen no se modifica.
- **Solo existe un flujo de procesamiento**: dado que mantenemos un solo flujo, el código, el mantenimiento y la actualización del sistema se ven reducidos considerablemente.
- **Posibilidad de volver a lanzar un procesamiento**: se puede modificar un procesamiento concreto y su configuración para variar los resultados obtenidos partiendo de los mismos datos de entrada.

Al prescindir de la capa batch, donde ya tenemos todos los datos y estos se pueden ordenar según sus atributos, perdemos *a priori* la capacidad de ordenar. Entonces, como requisito previo en la arquitectura Kappa, se tiene que garantizar que los eventos se lean y almacenen en el orden en el que se han generado. De esta forma, podremos variar un procesamiento concreto partiendo de una misma versión de los datos.

IDEAS CLAVE

- La arquitectura software es una de las responsabilidades clave de un data engineer que reúne distintas disciplinas y conocimientos como paradigmas de computación, programación, comunicación y almacenamiento para generar una solución completa llamada *plataforma*.

- La arquitectura por capas es la base de todas las arquitecturas de plataforma de análisis de datos big data porque cubre las diferentes necesidades con diversos enfoques y tecnologías, adaptándose a la escalabilidad de cada necesidad pero manteniendo la interoperabilidad entre capas. Esto permite obtener una plataforma enfocada al flujo natural del análisis de datos pero con las capacidades de escalabilidad big data.

- Hemos visto tres arquitecturas de referencia que cumplen la arquitectura por capas y están muy extendidas en el mercado. Aun así, no son las únicas; existen muchas más y otras que están por crear. Sin embrago, en todas las arquitecturas big data encontramos patrones de diseño repetidos y otros más novedosos. Uno de los objetivos de un data engineer es estar al día de nuevos patrones de diseño de arquitecturas.

BIBLIOGRAFÍA

AZARMI, B. (2016) *Scalable Big Data Architecture: A Practitioner's Guide to Choosing Relevant Big Data Architecture*. Nueva York: Apress. Disponible en: https://learning.oreilly.com/library/view/scalable-big-data/9781484213261/.

ISAACSON, C. (2014) *Understanding Big Data Scalability: Big Data Scalability Series, Part I*. Nueva Jersey: Prentice Hall. Disponible en: https://learning.oreilly.com/library/view/understanding-big-data/9780133599121/.

MARZ, N.; WARREN, J. (2015) *Big Data.Principles and best practices of scalable realtime data systems*. Nueva York: Manning. Disponible en: https://www.manning.com/books/big-data.

MISRA, P.; JOHN, T. (2017) *Data Lake for Enterprises*. Birmingham: Packt Publishing. Disponible en: https://learning.oreilly.com/library/view/data-lake-for/9781787281349/.

SHAH, H.; NITIN SAWANT, N. (2013) *Big Data Application Architecture Q&A: A Problem - Solution Approach*. Nueva York: Apress. Disponible en: https://learning.oreilly.com/library/view/big-data-application/9781430262923/.

RAJ, P.; RAMAN, A.; SUBRAMANIAN, H.(2017) *Architectural Patterns*. Birmingham: Packt Publishing. Disponible en: https://learning.oreilly.com/library/view/architectural-patterns/9781787287495/

RICHARDS, M. (2015) *Software Architecture Patterns*. Newton: O'Reilly Media, Inc. Disponible en: https://learning.oreilly.com/library/view/software-architecture-patterns/9781491971437/.