ADEN

ARQUITECTURAS BIG DATA, APACHE SPARK Y KAFKA

AUTOR: GASTÓN ADDATI

CONTENIDO

INTRODUCCIÓN

Trabajando con datos podemos comprender que su manejo no es sencillo de llevar, nos podemos encontrar con grandes cantidades de estos. Optar por una manera práctica y óptima de procesarlos es lo mejor para nuestro proyecto. Los modelos de arquitectura de big data llevan un tiempo ya en libre circulación, veremos dos, los modelos Lambda y Kappa.

El modelo de arquitectura Lambda nos permite diseñar a nuestra conveniencia la manera de analizar los conjuntos gigantes de datos, de forma tal que se puedan obtener los datos que estamos buscando de forma clara y certera. De esta forma podremos analizar grandes cantidades de datos de manera ordenada y bajo los parámetros que necesitemos establecer.

El modelo de arquitectura Kappa es una respuesta a la complejidad establecida por el anterior y el gran consumo de recursos que necesita. El modelo Kappa nos permite fraccionar la información por capas, analizar desde allí simplificando un poco el proceso. Como la finalidad y búsqueda de resultados es similar, ambos modelos conviven y aplicar uno u otro dependerá del proyecto a llevar a cabo.

01

LA IMPORTANCIA DE LAS ARQUITECTURAS DE BIG DATA

Por lo visto hasta el momento, podemos deducir que, en líneas generales, los proyectos de Big Data consisten en un proceso que puede resultar tan simple como complejo a la vez. Ese proceso consiste en capturar los datos (Ingesta), almacenarlos (en proyectos de big data en un datalake), procesarlos y, por último, mostrar o presentar los resultados obtenidos.

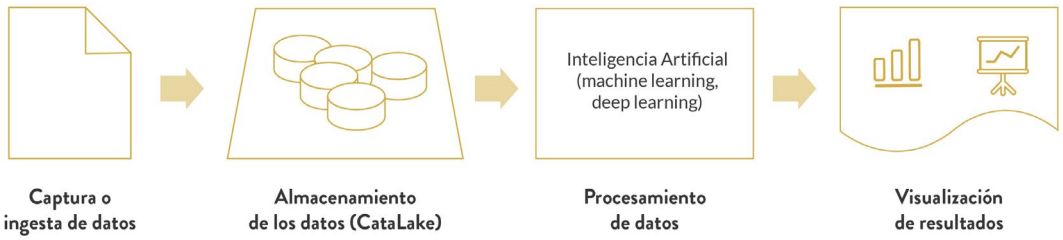


Figura 1: Proceso de manejo de datos. (Elaboración propia).

Un ejemplo práctico y sencillo para comprender este proceso podría ser el siguiente:

Imagínese que trabaja en un banco y que desea implementar un proyecto de big data para disminuir el fraude en tiempo real, que se puede generar o producir por el robo de datos de tarjetas de crédito.

- **El proceso de captura de datos:** podrían ser todas las transacciones que están ocurriendo en este instante, con las tarjetas de crédito emitidas.
- **El proceso de almacenamiento:** podría ser almacenar toda esa información en el DataLake (aunque hay veces que el proceso de almacenamiento se realiza en una etapa posterior por cuestiones de performance).

- **El procesamiento de los datos:** puede ser que, a partir de un modelo de inteligencia artificial, nuestro algoritmo pueda clasificar a una operación con cierto porcentaje de probabilidad de ser una transacción normal o bien de clasificar con cierto porcentaje de probabilidad que una operación es fraudulenta.
- **El proceso de visualizar los datos:** podría darse también en tiempo real para monitorear actividad por actividad, aquellas transacciones que son anuladas en tiempo real, para evitar caer en un fraude.

Por supuesto, toda esta información se podrá ir almacenando, y con el correr del tiempo podríamos ir obteniendo estadísticos descriptivos respecto de la cantidad de operaciones fraudulentas, cantidad de operaciones por día, mes, que son realizadas por nuestros clientes, importe de las transacciones, etc.

Si bien este proceso no pareciera ser muy complejo, conlleva que especialistas en big data puedan comprender la importancia de elegir una correcta arquitectura que sea capaz no sólo de dar respuesta al problema que se quiere resolver, y a los objetivos que se quieren cumplir, sino que además, permita tener todas aquellas características técnicas (eficiencia, solvencia, redundancia) y todas aquellas características que el negocio necesitará (como por ejemplo: Escalabilidad y Flexibilidad a los cambios).

Si bien existen algunas arquitecturas de referencia que ayudan a los ingenieros o especialistas en big data a diseñar y a modelar una correcta arquitectura, es importante que el lector sepa que hay ciertos componentes de las soluciones de big data que deben ser considerados con especial atención, para no tener problemas de performance durante el desarrollo del proyecto.

En el ejemplo que vimos anteriormente, debemos de considerar por ejemplo que se trata de un proyecto de Big data en tiempo real (*streaming*). Esa característica, como veremos luego, hará que nuestra arquitectura deba tener ciertos aspectos contemplados, para que el resultado sea el esperado.

Veamos a continuación un ejemplo de un modelo de arquitectura de referencia que resulta muy práctica para comprender de manera integral cualquier proyecto de Big Data:

Figura 2: Arquitectura de referencia para Big Data. (Jesús Montoya Sanchez de Pablo[1]).

Este modelo de arquitectura se suele emplear cuando se trabaja en el entendimiento del negocio, del problema que se pretende resolver, del contexto que rodea a nuestra organización. Podemos observar cómo a partir de este modelo se pueden tener en consideración cuáles serán las fuentes de datos desde donde obtendremos los datos, cuál será el modelo de la plataforma de datos, es decir, como lo almacenaremos, donde lo almacenaremos, de que forma procesaremos, etc. Por último, nos permite establecer como explotaremos los datos, es decir, que herramientas y/o aplicaciones utilizaremos para generar el valor que buscamos.

Este modelo de referencia, es de gran ayuda sin dudas, pero cuando se trata de cuestiones sumamente técnicas que implican el diseño técnico a bajo nivel de la tecnología, los expertos en big data basan su diseño en dos grandes arquitecturas que mundialmente son *"aceptadas" y "utilizadas"*. Nos referimos a las arquitecturas **Lambda y Kappa.**

[1] https://empresas.blogthinkbig.com/como-transformar-una-companiaiii-profundizando-en-la-arquitectura-de-referencia

02

ARQUITECTURA LAMBDA

Hemos visto que la arquitectura de Big Data, es un proceso importante que puede condicionar la efectividad de los resultados que queremos lograr al analizar grandes conjuntos de datos. Esta arquitectura también consiste en diseñar de forma personalizada los métodos de análisis no convencionales para los conjuntos gigantes de datos, de forma tal que se puedan obtener los datos que estamos buscando de forma clara y certera. Si recordamos el ejemplo simple que mencionábamos con anterioridad (el proyecto de Big Data del Banco que busca evitar el fraude en tiempo real) y también recordamos algunas características del Big Data en cuanto a sus V's (volumen, variedad, velocidad, veracidad y valor); notaremos en de alguna forma, para el proyecto del Banco, estas V's pueden resultar ser condicionantes de éxito. Es por este motivo que el diseño de la arquitectura de Big Data para el tratamiento técnico de los datos, se basan principalmente estas V 's que son intrínsecas a cualquier tipo de proyecto.

La arquitectura Lambda o "*'Lambda Architecture*" surge con el objetivo de dar respuesta global a diferentes necesidades de análisis Big Data.

Esta arquitectura fue planteada y explicada por su autor Nathan Marz en el libro llamado: Big Data: Principles and best practices of scalable realtime data systems, en el año 2015.

El eje central de la arquitectura Lambda radica en implementar sistemas de big data que pueden combinar los dos tipos de procesamiento, ya sea que se trate de procesamientos tipo batch o bien en tiempo real (stream). Para lograr esto, la arquitectura se especifica en CAPAS de abstracción.

Recordemos que el procesamiento de datos en modo batch, es aquel que nos permite procesar volúmenes de datos en tiempos espaciados, por ejemplo, cada 10 minutos, 1 hora o diario. Para ello el sistema dispone de lotes o batch en el que almacena toda la información que va obteniendo hasta completar un periodo.

Por ejemplo, este tipo de procesamiento pueden ser las transacciones de venta de determinados productos a lo largo de un periodo. Si el volumen de datos es elevado, este procesamiento podría demorarse varios minutos o incluso horas (eso dependerá de varios factores técnicos), y en este caso en particular es probable que quienes tomen decisiones, o bien el negocio mismo, estén dispuestos a esperar ese tiempo para poder hacerlo.

En cambio, el procesamiento de datos en modo stream o tiempo semi-real, es aquel que necesita procesar volúmenes de datos en tiempos lo más parecido a tiempo real que se pueda. Recordemos el caso que mencionamos antes, del banco que quiere evitar el fraude de las transacciones en tiempo real.

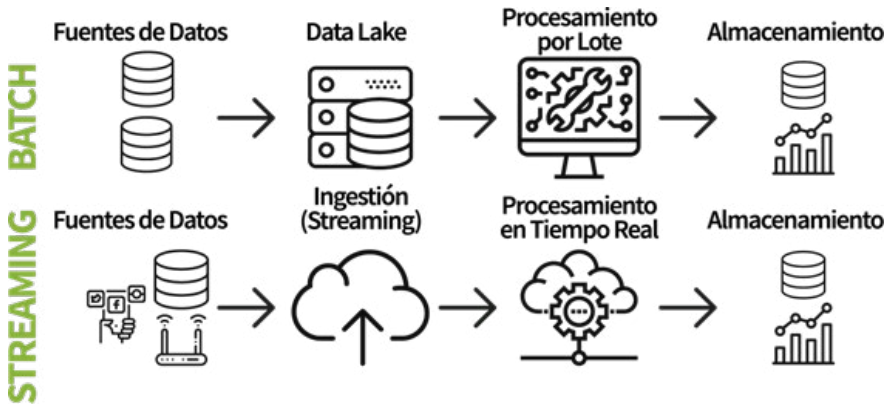


Figura 3: Combinación de procesamientos. (Pandaid [1]).

La arquitectura Lambda combina el procesamiento de datos: batch y stream, buscando las ventajas que nos ofrece cada uno de ellos.

Concretamente la arquitectura Lambda, consiste en 3(tres) grandes CAPAS

1. **Batch Layer:** se gestiona la información en crudo. Los datos nuevos se añaden a los ya existentes. Se hace un tratamiento batch cuyo resultado serán los denominados "Batch Views", que se utilizarán en la capa que "sirve" los datos para ofrecer la información ya transformada al exterior.
2. **La capa Serving Layer:** indexa los Batch Views generadas en el paso anterior de forma que puedan ser consultadas con baja latencia.
3. **La capa de streaming o Speed Layer**, compensa la alta latencia de las escrituras que ocurre en la serving layer y solo tiene en cuenta los datos nuevos.

La idea principal de la arquitectura Lambda es que toda la información que entra al sistema sea replicada en la capa de velocidad (*speed layer*) y en la capa de segmentos (*batch layer*) para que la información esté disponible para generar vistas en tiempo real en la capa de velocidad (*speed layer*) y vistas batch en la capa de servicios (*serving layer*). De esta manera, cualquier consulta realizada al sistema puede ser resuelta combinando resultados de las vistas batch (*batch views*) y de las vistas en tiempo real (*real-time views*).

[1] https://www.pandaid.com/big-data-procesos-en-tiempo-real/

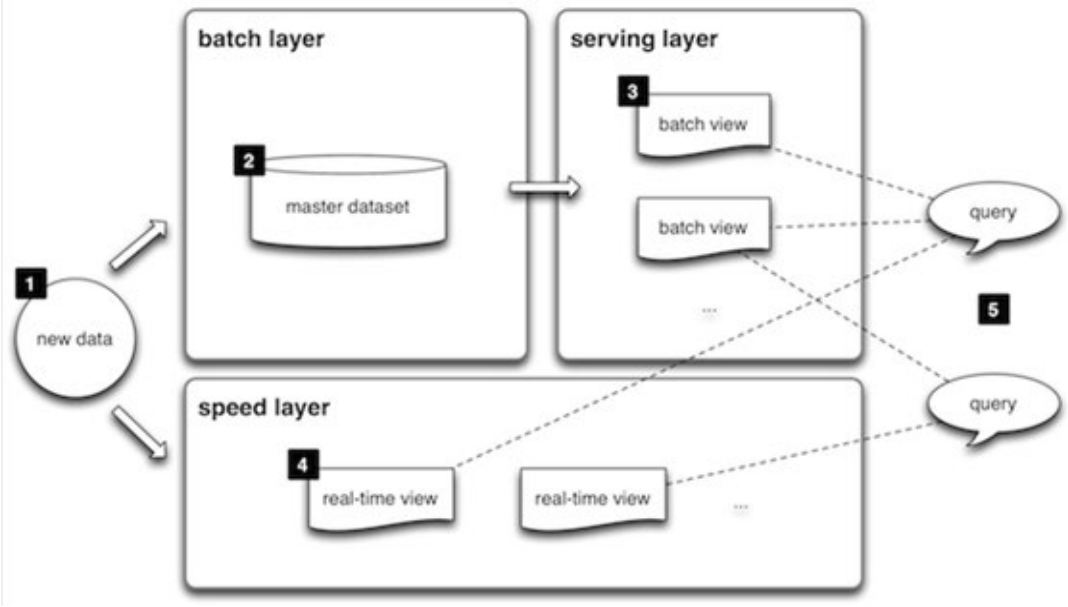


Figura 4: Lambda Kappa. (Analisisdedatos.net[1'']).

La arquitectura Lambda está orientada a aplicaciones que han sido construidas alrededor de transformaciones asíncronas complejas que necesitan ejecutarse con latencia baja (segundos a minutos). Un buen ejemplo es un sistema de recomendación de noticias, el cual necesita extraer las noticias de diferentes fuentes, procesarlas y normalizarlas, indexarlas, ponderarlas y almacenarlas para que estén disponibles.

Una de las preguntas que seguramente se estará realizando el lector, es ¿Qué ocurre con todas las tecnologías que hemos visto hasta el momento? ¿Dónde están? ¿Dónde se ubican en esta arquitectura?

Recordemos que la arquitectura representa capas de abstracción, pero en cada una de las capas de esta arquitectura se utilizan tecnologías especializadas para cada propósito. Aunque siempre será posible utilizar distintas opciones, una opción popular es utilizar **Apache Sqoop + Hadoop HDFS + Hive** para capturar, almacenar y procesar datos **en forma batch**, y como veremos más adelante, **Apache Kafka + HBase + Spark** para capturar, almacenar y procesar datos **en forma stream** (por citar algunos ejemplos populares que permitan relacionar todos los conceptos que estamos desarrollando).

La siguiente imagen, describe algunas tecnologías que podrían aplicarse en diversas capas de arquitectura de un proyecto de big data:

[1''] https://analisisdedatos.net/bigData/eco/lambdaKappa.php

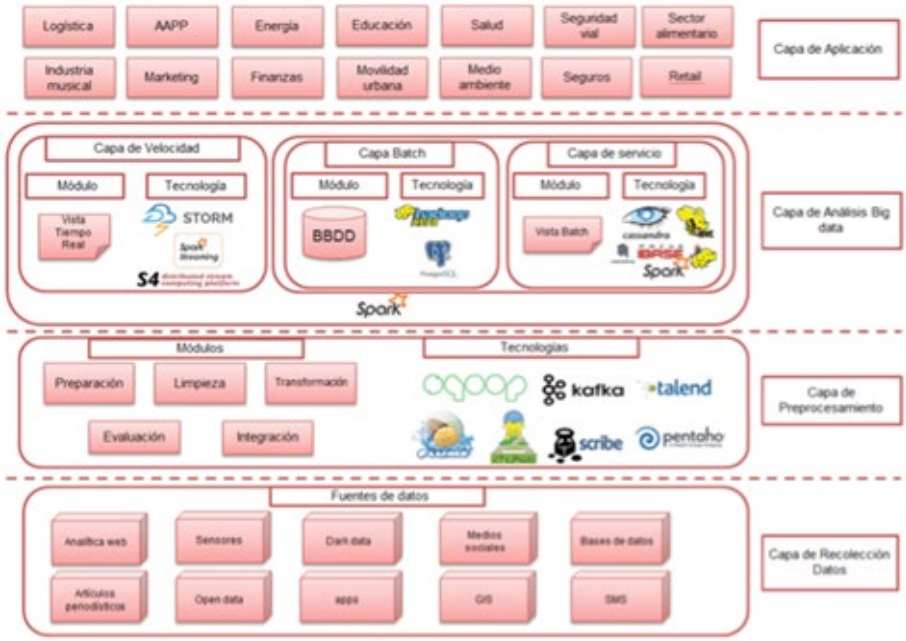
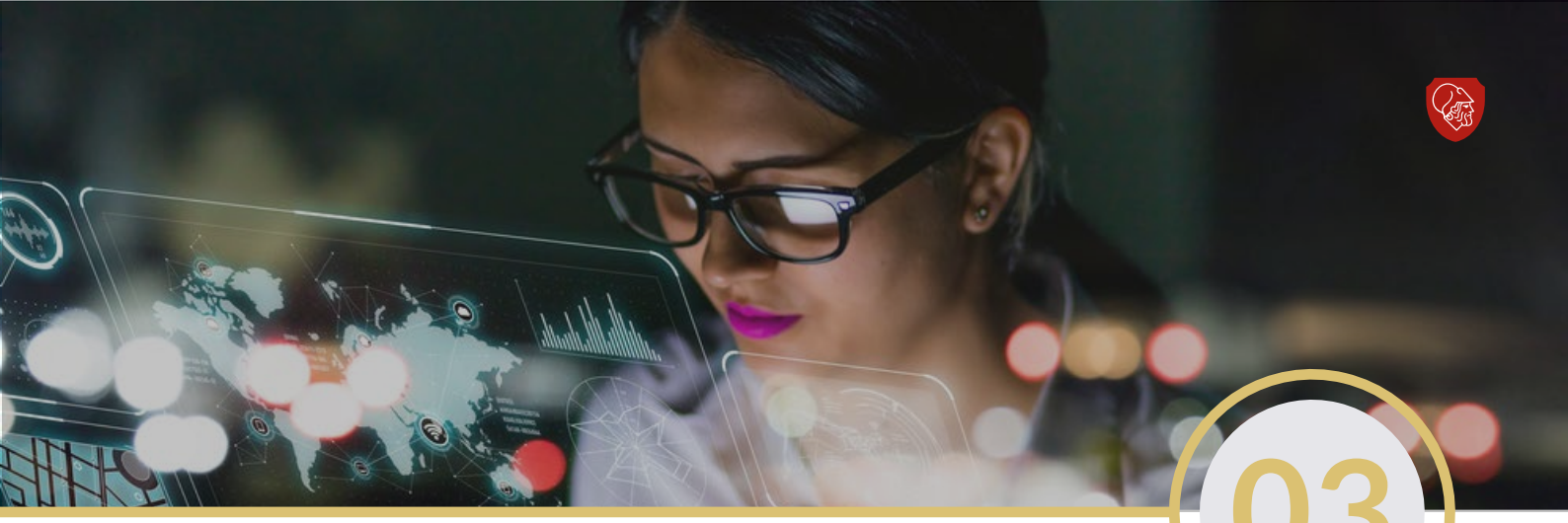


Figura 5: Soluciones. (Grupo Fractalia[2]).

[2] https://fractaliasystems.com/arquitectura-lambda-y-tecnologias-para-el-diseno-de-soluciones-big-data-1



03

ARQUITECTURA KAPPA

Una desventaja de la arquitectura lambda es su complejidad. La lógica de procesamiento aparece en dos lugares diferentes (las rutas de acceso inactiva y activa) y esto conduce a la duplicación de la lógica de cálculo y a la complejidad de administrar la arquitectura de ambas rutas de acceso.

La **arquitectura kappa** fue propuesta por Jay Kreps[2'] en el año 2014 como alternativa a la arquitectura lambda[2''] .

El autor criticaba el consumo innecesario de recursos que supone mantener y tratar los mismos datos, con el objetivo de obtener resultados similares, en dos sistemas distintos (la capa *batch* y la capa *streaming*).

Kreps opina que el procesamiento por lotes también se puede llevar a cabo en la **capa streaming.** Y, como consecuencia, en su idea de Kappa aboga por suprimir la capa de segmentos, quedándose solo con la de streaming y la de consulta, y pasando a considerar todo como un flujo de datos ininterrumpido, sin final definido, en el que aplicar las operaciones. Es por eso por lo que la arquitectura Kappa tiene los mismos objetivos básicos que la arquitectura lambda, pero con una diferencia importante: todos los flujos de datos atraviesan una única ruta de acceso, para lo que usan un sistema de procesamiento de flujos.

[2'] https://www.oreilly.com/people/jay-kreps/
[2''] https://www.oreilly.com/radar/questioning-the-lambda-architecture/

EXTRA

NOTA: para la arquitectura Kappa 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.

En la arquitectura Kappa, tal como puede observarse en el diagrama de su arquitectura, solo se tienen 2 (dos) capas: La capa de tiempo real y la capa de servicios.

La capa de tiempo real lanza tareas de procesamiento de los datos. Normalmente, una única tarea es lanzada para permitir el procesamiento de los datos en tiempo real.

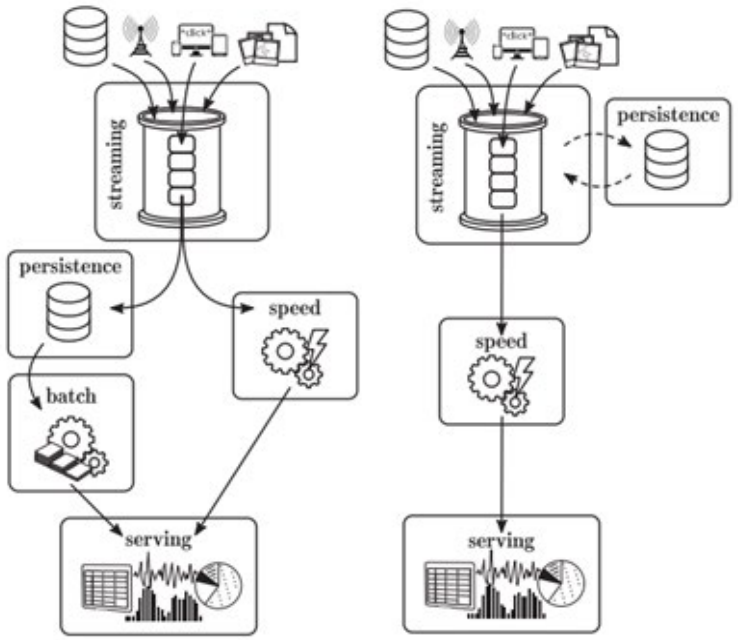
El reprocesado de datos únicamente se realiza cuando alguna parte del código de la tarea necesita ser modificada. Esto se consigue lanzando otra tarea modificando y replicando en ella todos los datos previos.

Los resultados de esta nueva tarea se enviarán a una nueva tabla de salida y, cuando esta tarea haya finalizado el procesamiento de los datos, se deberán cambiar las aplicaciones para que consuman de la nueva tabla de salida. Finalmente, similarmente a la arquitectura Lambda, la capa de servicios es utilizada para consultar los resultados.

En resumidas cuentas, la arquitectura Kappa Es 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 Real-time Layer, dando soporte a procesamientos tanto batch como en tiempo real.

- **Los datos son almacenados** sin ser transformados y las vistas se derivan de ellos. Un estado concreto puede ser recalculado puesto que la información de origen no se modifica.
- **Solo existe un flujo de procesamiento:** puesto que mantenemos un solo flujo, el código, el mantenimiento y la actualización del sistema se ven reducidos considerablemente.

La arquitectura Kappa consiste en eliminar la capa batch dejando solamente la capa de streaming.



(a) The Lambda Architecture. (b) The Kappa Architecture.

Figura 6: Procesamiento en tiempo real. (Wingerath, W., Gessert, F., Friedrich, S. & Ritter, N., 2016).

04

ARQUITECTURA KAPPA O ARQUITECTURA LAMBDA ¿CUÁL ELEGIR?

La pregunta crucial que toda persona quiere conocer es cuando conviene utilizar una arquitectura en lugar de otra. Es decir, en qué proyectos conviene optar por la arquitectura Lambda y cuales conviene la arquitectura Kappa.

Esta respuesta no existe hasta el momento, bajo un criterio que pueda ser consensuado o establecido. No existe de momento una regla a seguir o una parametrización que le permita al lector tomar la decisión basada en ciertos parámetros, sobre cual elegir. Lo cierto es que estas arquitecturas, fueron pensadas y creadas para ser una guía de referencia en cómo se pueden tratar los datos.

Algún ingeniero especialista en el tratamiento de datos, y en casos sumamente específicos podría concluir que la arquitectura Lambda es una arquitectura un poco obsoleta, y que en cambio podría concluir que la arquitectura Kappa es la más adecuada. Esos criterios, vale decir, pueden ser considerados sobre los supuestos de un proyecto, en el entorno de ese proyecto, y con las condiciones de ese proyecto. Y lo que es más confuso aún, lo más probable es que otro ingeniero especialista en datos, tal vez no concuerde con el mismo criterio anterior.

La única respuesta es que depende de lo que se quiera realizar un modelo se impone sobre otro, y lo que, si hoy es claro y contundente, es que aquellos proyectos de big data que requieren trabajar bajo modelo Streaming son los que más interesados estarán en optimizar las arquitecturas, porque en ese tipo de proyectos, hablar de 100 milisegundos de retraso, puede generar problemas. En cambio, en los proyectos que son del tipo Batch, lo cierto es que parecería ser indistinto el modelo a elegir.

05

APACHE SPARK

Spark es un framework ultrarrápido para el almacenamiento, procesamiento y análisis de grandes volúmenes de datos. Es de código abierto y se encuentra gestionado, al igual que vimos anteriormente, por la *Apache Software Foundation*.

Apache Spark está especialmente diseñado para su implementación en big data y machine learning. Posee una potencia de procesamiento que agiliza la detección de patrones en los datos, la clasificación organizada de la información, la ejecución de cómputo intensivo sobre los datos y el procesamiento paralelo en clústers.

Apache Spark nació en 2009 en la Universidad de Berkeley, a partir de un paper que publicó Google, y fue a partir de ese entonces, que su evolución ha sido permanente. Su desarrollo está basado en un lenguaje de programación muy potente y muy utilizado en proyectos de Big Data que se llama SCALA. Tanto Scala como JAVA son lenguajes muy populares y muy empleados por los programadores e ingenieros que implementan soluciones de big data.

En la actualidad es una de las aplicaciones más utilizadas dentro del ecosistema de hadoop. Iremos viendo detenidamente el por qué esto es así y sobre todo qué ventajas tiene Spark respecto de lo que vimos anteriormente con MapReduce.

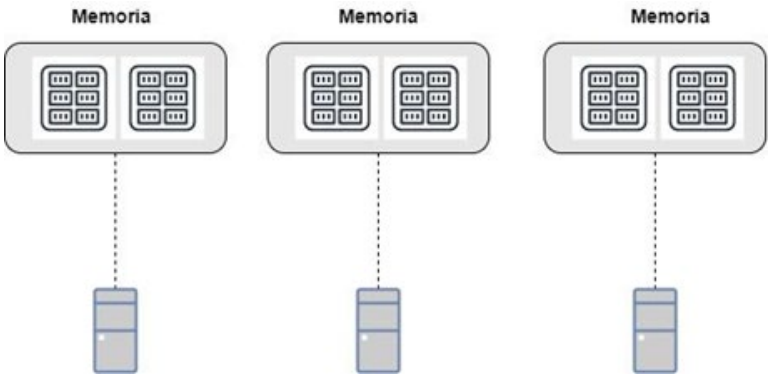
De manera sintetizada podemos decir que Apache Spark es un sistema que, por supuesto se basa en Hadoop MapReduce y, además, **permite dividir o paralelizar el trabajo.** La idea es que tengamos una cierta cantidad de máquinas, por ejemplo 20 (veinte) máquinas, y cada una de esas instancias va a tener instalada una versión de Apache Spark. De esta manera, cuando tengamos que procesar una gran cantidad de datos, vamos a poder dividir el mismo en 20 partes, y cada máquina se encargará de una parte. Con estas cuestiones, es notorio que estaremos optimizando la velocidad de procesamiento algo que sabemos es fundamental en el mundo del big data.

Esta síntesis que de manera intuitiva permite comprender el funcionamiento general de Spark, es en verdad, un derivado de unos conceptos fundamentales en la computación que se denominan Computación Distribuida y Computación en Paralelo.

- **Computación Distribuida:** es un modelo para resolver problemas de computación masiva utilizando un gran número de ordenadores organizados en clústeres incrustados en una infraestructura de telecomunicaciones distribuida[3].

Este modelo informático permite hacer grandes cálculos utilizando miles de ordenadores de voluntarios. El gran aporte y beneficio de esto, es la velocidad, la performance y sobre todo considerando que muchas computadoras pueden trabajar de manera colaborativa para hacer más rápido un trabajo determinado.

Por lo general en la computación distribuida, podemos encontrarnos con un esquema, similar al siguiente:



- **Computación en Paralelo:** tradicionalmente, los programas informáticos se han escrito para el cómputo en serie. Para resolver un problema, se construye un algoritmo y se implementa como un flujo en serie de instrucciones. Estas instrucciones se ejecutan en una unidad central de procesamiento en un computadora o máquina.

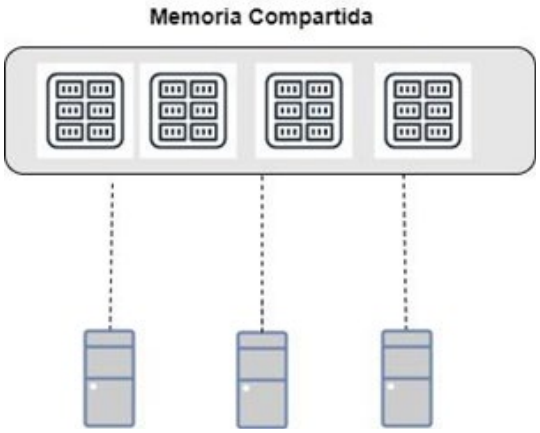
En este tipo de programas, sólo puede ejecutarse una instrucción a la vez y después de que la instrucción ha terminado, se ejecuta la siguiente.

La computación en paralelo, por el contrario, utiliza simultáneamente múltiples elementos de procesamiento para resolver un problema. Esto se logra mediante la división del problema en partes independientes de modo que cada elemento de procesamiento pueda ejecutar su parte del algoritmo de manera simultánea con los otros

Por supuesto, esto también genera grandes beneficios, porque también se optimizan los tiempos de procesamiento y de resolución de problemas.

[3] https://es.wikipedia.org/wiki/Computaci%C3%B3n_distribuida

Generalmente en la computación en paralelo podemos encontrarnos con un esquema como el siguiente:



Apache Spark tiene la característica de que provee procesamiento paralelo y distribuido de datos, provee escalabilidad, redundancia a fallos, y, sobre todo, utiliza hardware tradicional, es decir, no requiere de un hardware específico para poder funcionar.

Vemos a continuación cómo es el funcionamiento de Apache Spark un poco más en detalle:

Para poder comprender el funcionamiento de Spark necesitamos recordar algunos conceptos que previamente estudiamos en relación a MapReduce.

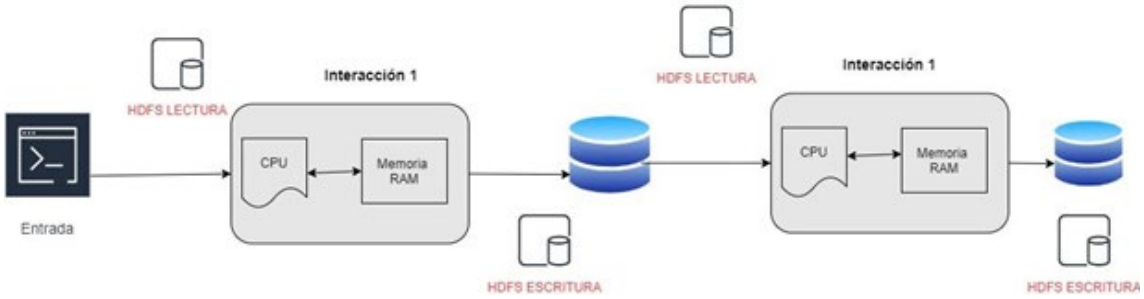


Figura 7: Esquema Apache Spark. (Apache Spark[3])

Recordemos que básicamente MapReduce funciona por medio de JOBs que deben ser programados en lenguaje Java (algo que resultaba ser bastante laborioso) o bien también podía utilizarse Apache HIVE como algo mucho más sencillo y que transformaba la consulta HIVE en un JOB de MapReduce.

Es importante mencionar que cada vez que trabajamos con MapReduce, y un JOB determinado se ejecuta, éste realiza de manera excluyente una solicitud de acceso al HDFS de Hadoop para recuperar la información. Desde el punto de vista técnico, realizar ese acceso, demanda que se utilicen recursos de hardware de la máquina tales como MEMORIA y CPU (procesador) pero no debemos perder de vista, que

[3] https://spark.apache.org/

la información almacenada en el HDFS en verdad persiste en discos y cada acceso a disco tiene cierta velocidad de respuesta, por lo que si uno, conceptualmente lo que quisiera es hacer muchas consultas (lecturas o escrituras) estaría demandando muchos recursos de Entrada y Salida a los discos (lo que técnicamente se denomina I/O – de Input – Output).

El lector debe saber que estas operaciones, por más que se realicen en discos de última generación y súper rápidos, son en verdad, en muchas ocasiones, un gran cuello de botella.

Apache Spark lo que hace es utilizar la memoria de la máquina (del servidor o de los servidores) para evitar el acceso permanente a los discos. Esta es la principal razón de mejora respecto de las prestaciones de Spark, porque desde el punto de vista técnico siempre es mucho más eficiente acceder a la memoria RAM que acceder a un disco físico.

Figura 8: Esquema Apache Spark. (Apache Spark[3'']).

Otra cuestión no menor es que debemos decir que SPARK no almacena la información. Spark utiliza y accede al HDFS de hadoop. Lo que hace que Spark tenga sentido es que PROCESA LOS DATOS MASIVAMENTE Y DE UNA FORMA MUCHO MÁS EFICIENTE.

Cuando nos referimos a los datos, debemos recordar que nos referimos a datos de todo tipo, tanto estructurados como no estructurados. Es por eso que también tiene sentido Spark en el mundo del Big Data.

Como una aplicación más del ecosistema de Hadoop, Apache Spark también tiene su propia arquitectura, donde se destacan:

- **Apache Core:** es el núcleo, el corazón de spark que apoya a los demás módulos.
- **Spark SQL:** es el módulo para procesar datos estructurados y semi estructurados.
- **MLlib:** es una librería de machine learning que cuenta con diversos algoritmos de aprendizaje automático.
- **GraphX:** módulo de procesamiento de grafos (DAG).
- **Spark Streaming:** para el procesamiento de datos en tiempo real.

[3''] https://spark.apache.org/

Para sintetizar sobre Apache Spark:

- Mejora notablemente el desempeño de aplicaciones dependientes de datos.
- Unifica algoritmos para que trabajen conjuntamente en diversas tareas.
- Integra dentro de sí el modelado analítico de datos.
- Otorga escalabilidad en su potencia al introducir más procesadores en el sistema.
- Reduce los costos ya que puede utilizarse en hardware estándar de uso común.
- Dispone de API's para Java, Phyton y Scala; también APIs para transformar y manipular datos semiestructurados. Esto es realmente muy importante para quienes manipulan datos masivos.
- Facilita la integración con sistemas de archivos como HDFS de Hadoop, Cassandra, HBase, MongoDB entre otros.
- Posee tolerancia a fallos implícita.

Para más información sobre apache Spark, recomiendo al lector visitar el sitio web oficial, donde podrán encontrar toda la documentación actualizada y las últimas versiones disponibles: https://spark.apache.org/

06

APACHE KAFKA

Apache Kafka es una plataforma distribuida de transmisión de datos que permite publicar, almacenar y procesar flujos de registros, así como suscribirse a ellos, de forma inmediata. Está diseñada para administrar los flujos de datos de varias fuentes y distribuirlos a diversos usuarios. En pocas palabras, transfiere cantidades enormes de datos, no solo desde el punto A hasta el B, sino también del punto A al Z y a cualquier otro lugar que necesite, y todo al mismo tiempo.

Apache Kafka es la alternativa a un sistema de mensajería tradicional para empresas. Comenzó como un sistema interno que LinkedIn desarrolló para gestionar 1,4 billones de mensajes por día. Ahora, es una solución open source de transmisión de datos que permite satisfacer diversas necesidades empresariales.

Apache Kafka puede gestionar millones de datos por segundo, así que es ideal para los desafíos del big data. Sin embargo, también es útil para las empresas que no necesitan manejar tantos datos en la actualidad. En muchos casos prácticos de procesamiento de datos, como el Internet de las cosas (IoT) y las redes sociales, los datos aumentan de manera exponencial y pueden sobrecargar rápidamente una aplicación diseñada en función del volumen de datos actual.

Tal como se publica en la web oficial de Kafka[11] existen diversos casos de uso donde esta tecnología se podría utilizar.

CASO DE USO: Mensajería

Kafka funciona bien como reemplazo de un intermediario de mensajes más tradicional. Los intermediarios de mensajes se utilizan por diversas razones (para desvincular el procesamiento de los productores de datos, para almacenar en búffer los mensajes no procesados, etc.). En comparación con la mayoría de los sistemas de mensajería, Kafka tiene un mejor rendimiento, partición integrada, replicación y tolerancia a fallas, lo que lo convierte en una buena solución para aplicaciones de procesamiento de mensajes a gran escala.

[11] https://kafka.apache.org/documentation/#uses

En este dominio, Kafka es comparable a los sistemas de mensajería tradicionales como ActiveMQ o RabbitMQ.

CASO DE USO: Seguimiento de la actividad del sitio web

El caso de uso original de Kafka era poder reconstruir una canalización de seguimiento de la actividad del usuario como un conjunto de fuentes de publicación y suscripción en tiempo real. Esto significa que la actividad del sitio (páginas vistas, búsquedas u otras acciones que los usuarios pueden realizar) se publica en temas centrales con un tema por tipo de actividad. Estas fuentes están disponibles para suscripción para una variedad de casos de uso que incluyen procesamiento en tiempo real, monitoreo en tiempo real y carga en Hadoop o sistemas de almacenamiento de datos fuera de línea para procesamiento e informes fuera de línea.

El seguimiento de la actividad suele ser un volumen muy alto, ya que se generan muchos mensajes de actividad para cada vista de página del usuario.

CASO DE USO: Procesamiento tipo Streaming

Muchos usuarios de Kafka procesan datos en canalizaciones de procesamiento que constan de varias etapas, donde los datos de entrada sin procesar se consumen de los temas de Kafka y luego se agregan, enriquecen o transforman en nuevos temas para su posterior consumo o procesamiento de seguimiento. Por ejemplo, una canalización de procesamiento para recomendar artículos de noticias podría rastrear el contenido de un artículo de fuentes RSS y publicarlo en un tema de *"artículos"*; el procesamiento posterior podría normalizar o de duplicar este contenido y publicar el contenido del artículo limpio en un nuevo tema; una etapa final de procesamiento podría intentar recomendar este contenido a los usuarios. Dichos canales de procesamiento crean gráficos de flujos de datos en tiempo real basados en temas individuales.

Funcionamiento general y sintético de Apache Kafka

Supongamos que nos encontramos trabajando en un proyecto donde tenemos diversos sitios web que utilizaremos como nuestras fuentes de datos. Esas fuentes de datos, serán los lugares desde los cuales deseamos incorporar información para enviarla a diversos sistemas, que pueden ser tanto internos como externos de nuestra organización.

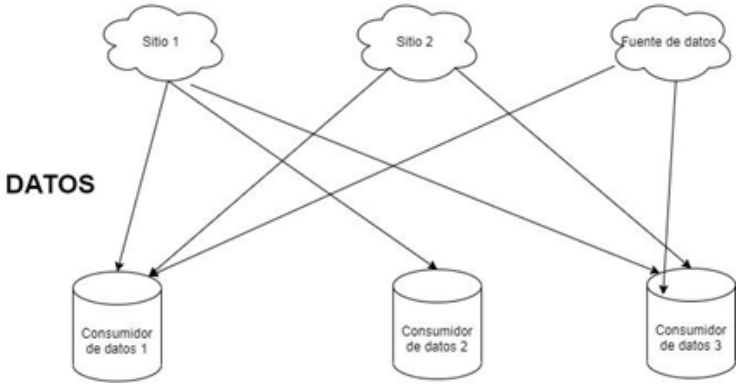


Figura 8: Funcionamiento general. (Elaboración propia).

En la figura anterior, vemos cómo con unos pocos sitios o fuentes de datos, esto podría realizarse mediante programas informáticos, integraciones puntuales, pero ¿qué sucede si la cantidad de sitios o de fuentes de datos, se incrementaran notablemente?

Apache Kafka nos permite desacoplar todas esas integraciones y nos permitirá gestionar de una manera mucho más eficiente todos los mensajes que quisiéramos recibir al mismo tiempo que podremos enviarlo a donde quisiéramos, por ejemplo, a un apache Spark o un HDFS, simplificando de esta forma, muchos problemas técnicos y ahorrando mucho tiempo.

Conceptualmente lo que haremos es tener a Kafka como el recurso que concentra, consolida y gestiona todas estas peticiones (mensajes).

La siguiente figura ilustra mejor el concepto:

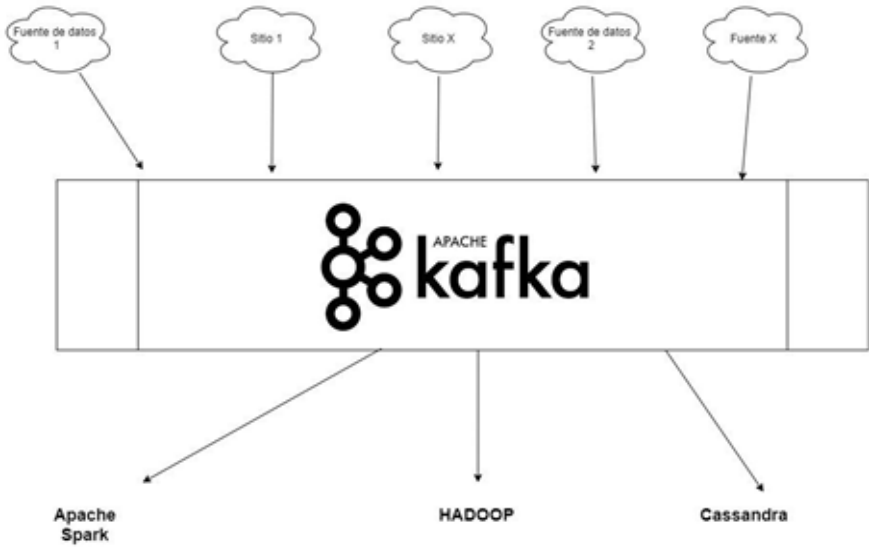


Figura 9: Apache Kafka. (Apache Kafka[12]).

Debemos considerar, que cuando nos referimos a mensajes, podríamos estar haciendo alusión a datos que provienen de un sistema de IoT (internet de las cosas), donde muchos sensores podrían estar enviando información en tiempo real a nuestro big data.

[12] https://kafka.apache.org/

EXTRA

Empresas de las más conocidas en el mundo como Netflix, Amazon, Linkedin (desde luego porque este proyecto nació allí), Spotify, entre otros, utilizan Apache Kafka como sistema de recomendación de contenidos.

No solo pueden ser datos de sensores, también podrían ser los datos de las transacciones bancarias que queremos analizar en tiempo real, la compra o la venta de productos que tenemos en un gran e-commerce, entre otros.

ARQUITECTURA DE APACHE KAFKA

Kafka tiene tres componentes fundamentales:

A. **Los denominados:** productores, los consumidores y los brokers.
B. **Los productores** son los encargados de escribir mensajes en Kafka.
C. **Los consumidores** los pueden leer y procesar.

Los brokers son los nodos que forman parte del cluster de Kafka y almacenan y distribuyen los datos.

Los mensajes de Kafka se almacenan en topics, y se reparten en particiones para hacer el sistema escalable y tolerante a fallos.

Kafka se ejecuta como un clúster (red de nodos) en uno o más servidores que pueden encontrarse en centros de datos diferentes.

Los nodos o puntos de intersección del clúster, denominados brokers, almacenan los flujos de datos entrantes, clasificándolos en los llamados topics.

Los datos se dividen en particiones y se replican y distribuyen en el clúster, recibiendo un sello de tiempo.

De esta manera, la plataforma de transmisión asegura una gran disponibilidad y un acceso de lectura rápido.

Apache Kafka distingue los temas entre *"normal topics"* y *"compacted topics"*. Los mensajes de normal topics pueden borrarse una vez se ha excedido el período o el límite de almacenamiento, mientras que las entradas de compacted topics no están sujetas a limitaciones de tiempo ni espacio.

Las aplicaciones que escriben datos en los clústeres de Kafka se denominan productores, mientras que las aplicaciones que leen los datos de un clúster se llaman consumidores.

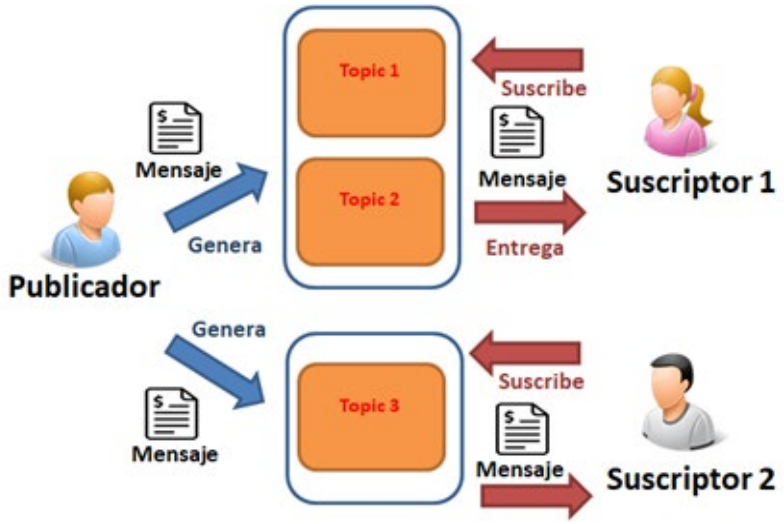


Figura 10: Funcionamiento. (Víctor Madrid[13]).

[13] https://enmilocalfunciona.io/aprendiendo-apache-kafka-parte-1/

BIBLIOGRAFÍA

ANÁLISIS DE DATOS. ARQUITECTURAS LAMBDA Y KAPPA. Recuperado el 17/01/2022 de https://analisisdedatos.net/bigData/eco/lambdaKappa.php

AGUILAR, L. BIG DATA, ANÁLISIS DE GRANDES VOLÚMENES DE DATOS EN ORGANIZACIONES. 1era edición. Ed. AlfaOmega.

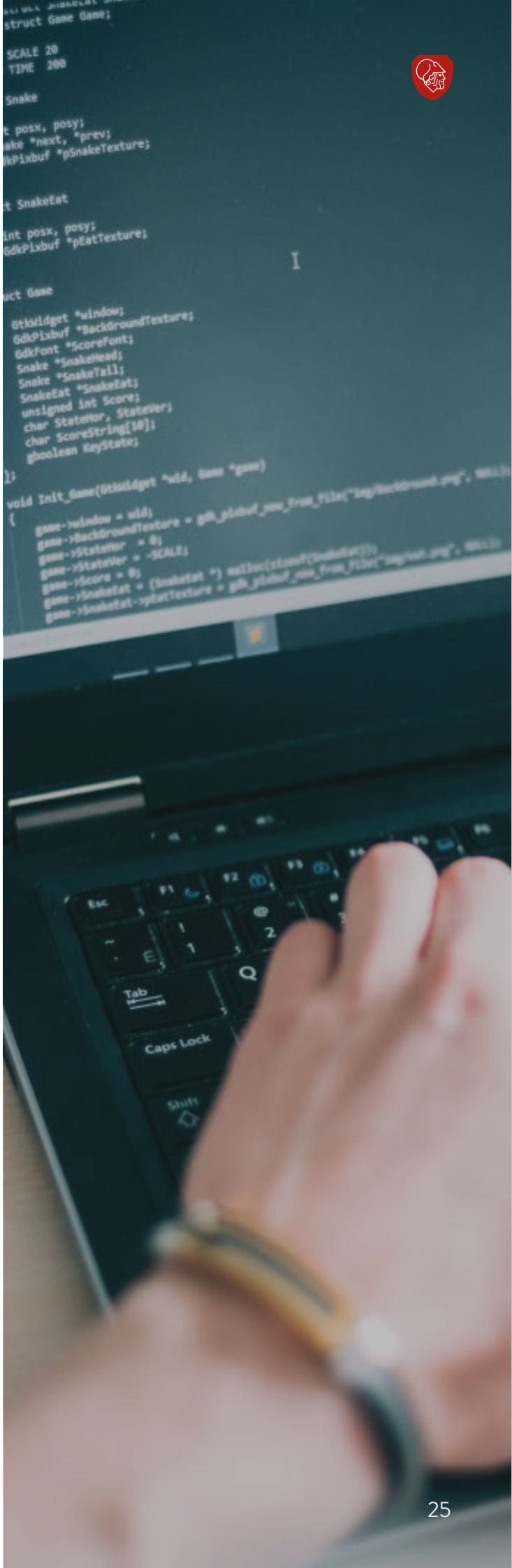
GRUPO FRACTALIA. (2015). Arquitectura Lambda y tecnologías para el diseño de soluciones Big Data. Recuperado el 17/01/2021 de https://fractaliasystems.com/arquitectura-lambda-y-tecnologias-para-el-diseno-de-soluciones-big-data-1

JAY KREPS. (2014). Questioning the Lambda Architecture https://www.oreilly.com/radar/questioning-the-lambda-architecture/

MICROSOFT. ARQUITECTURAS DE MACRODATOS. (2021). https://docs.microsoft.com/es-es/azure/architecture/data-guide/big-data/

O'REILLY. JAY KREPS. https://www.oreilly.com/people/jay-kreps/

WINGERATH, W., GESSERT, F., FRIEDRICH, S. & RITTER, N. (2016). Real-time stream processing for Big Data. it - Information Technology, 58(4), 186-194.



ADEN