>> ANEXO 1. ARQUITECTURA

Procesamiento *batch* y en *streaming*

En primer lugar, vamos a recordar conceptos importantes de ambos procesamientos mediante estas dos figuras:

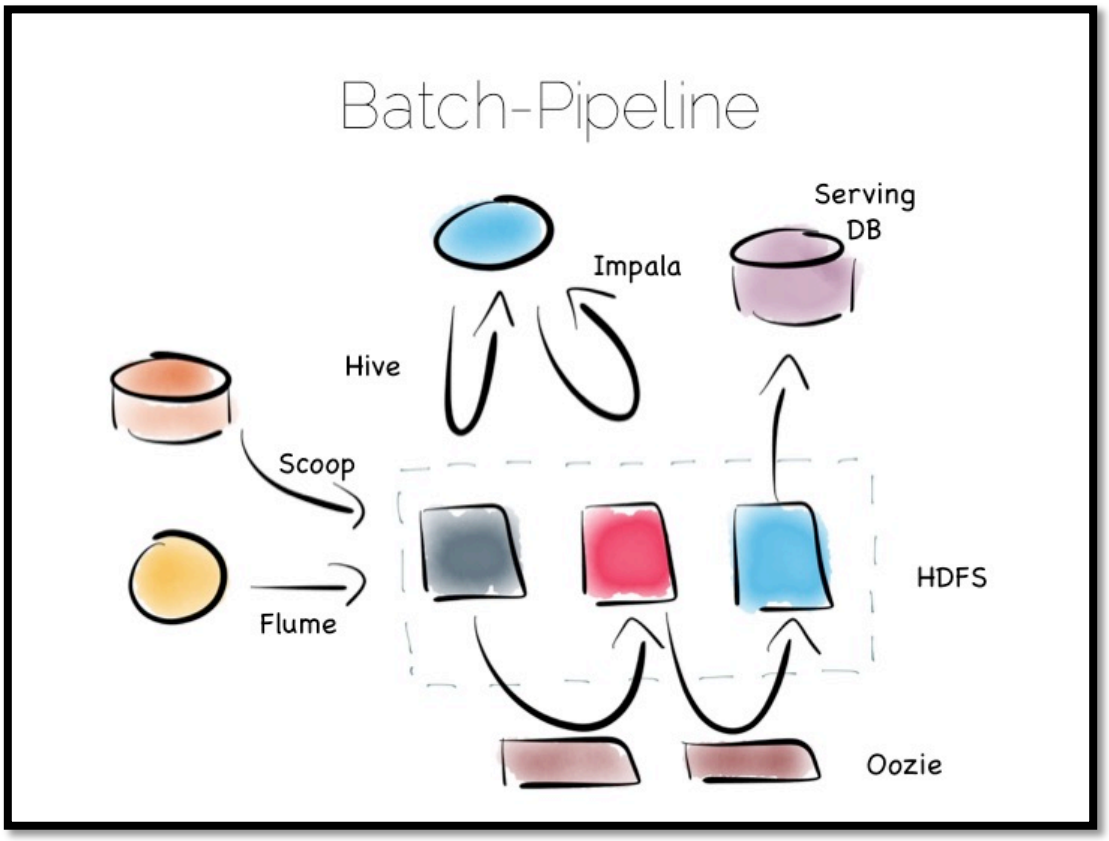

Figura 1. Procesamiento batch
Fuente: https://jaxenter.com

Procesamiento *batch* es aquel que se desarrolla normalmente en ventanas temporales y de manera desatendida para la realización de alguna tarea de procesamiento concreta. Por ejemplo, la generación de un informe durante la noche con los datos del CRM para nuestro director comercial.

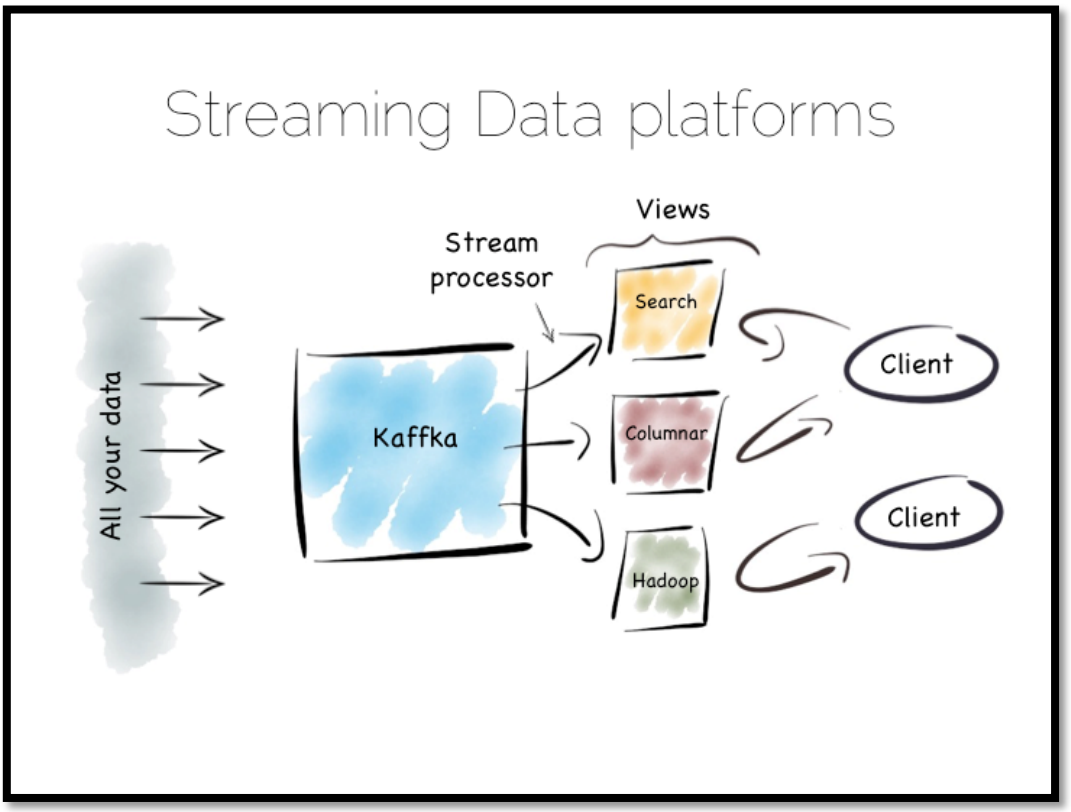

Figura 2. Procesamiento en streaming.
Fuente: https://jaxenter.com

Por el contrario, procesamiento en *streaming* es aquel que se realiza según se reciben los *streams* de datos, cercanos al tiempo real.

Arquitectura Lambda

Esta arquitectura surge en respuesta a las necesidades de las aplicaciones modernas, es decir, un mix de necesidades de procesamiento *batch* y procesamiento en *streaming*.

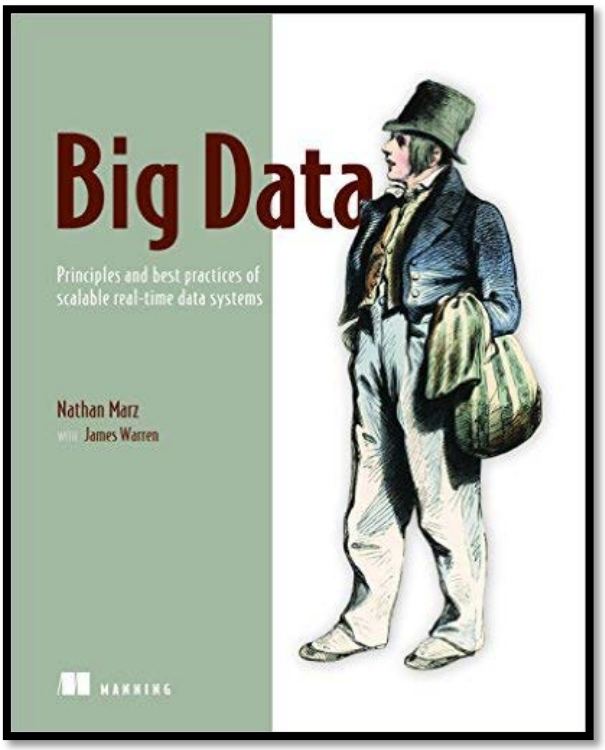

Figura 3. Libro de Nathan Marz.
Fuente: Amazon.

Obedece a un esquema en tres capas:

- Capa *batch*.
- Capa de velocidad.
- Capa de sevicio.

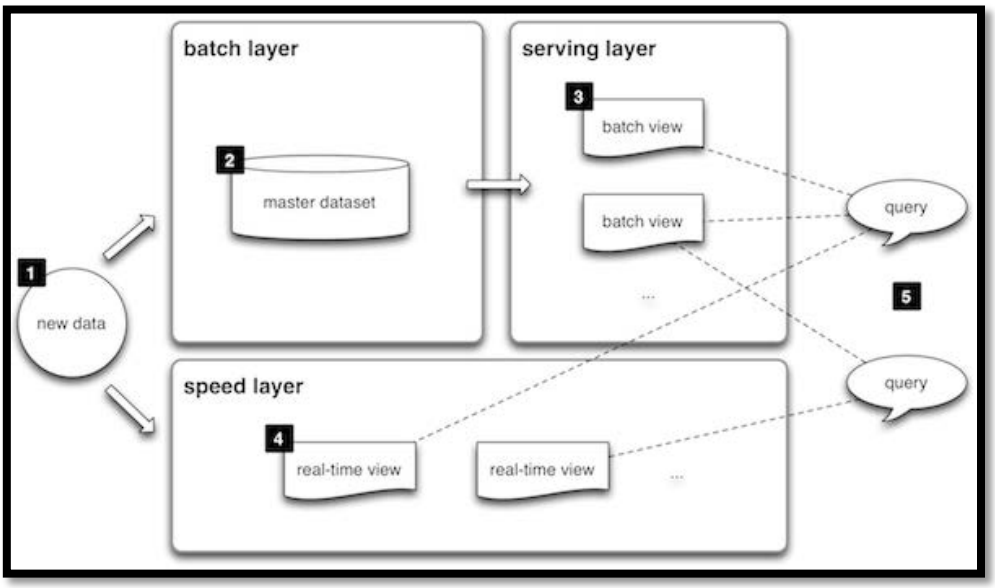

Figura 4. Demonios YARN.
Fuente: lambda-architecture.net

Asimismo, es importante señalar los siguientes aspectos:

- Todos los datos que entran en el sistema se envían tanto a la capa *batch* como a la capa *streaming* para su procesamiento.
- La capa *batch* tiene dos funciones:
  - Gestionar el conjunto de datos maestro (un conjunto inmutable y único de datos sin conexión).
  - Calcular previamente las vistas *batch*.
- La capa de servicio indexa las vistas *batch* para que puedan consultarse en modo de baja latencia y de manera *ad hoc*.
- La capa de velocidad compensa la alta latencia de actualizaciones a la capa de servicio y se ocupa solo de datos recientes.
- Cualquier consulta entrante se puede responder combinando los resultados de las vistas *batch* y las vistas en tiempo real.

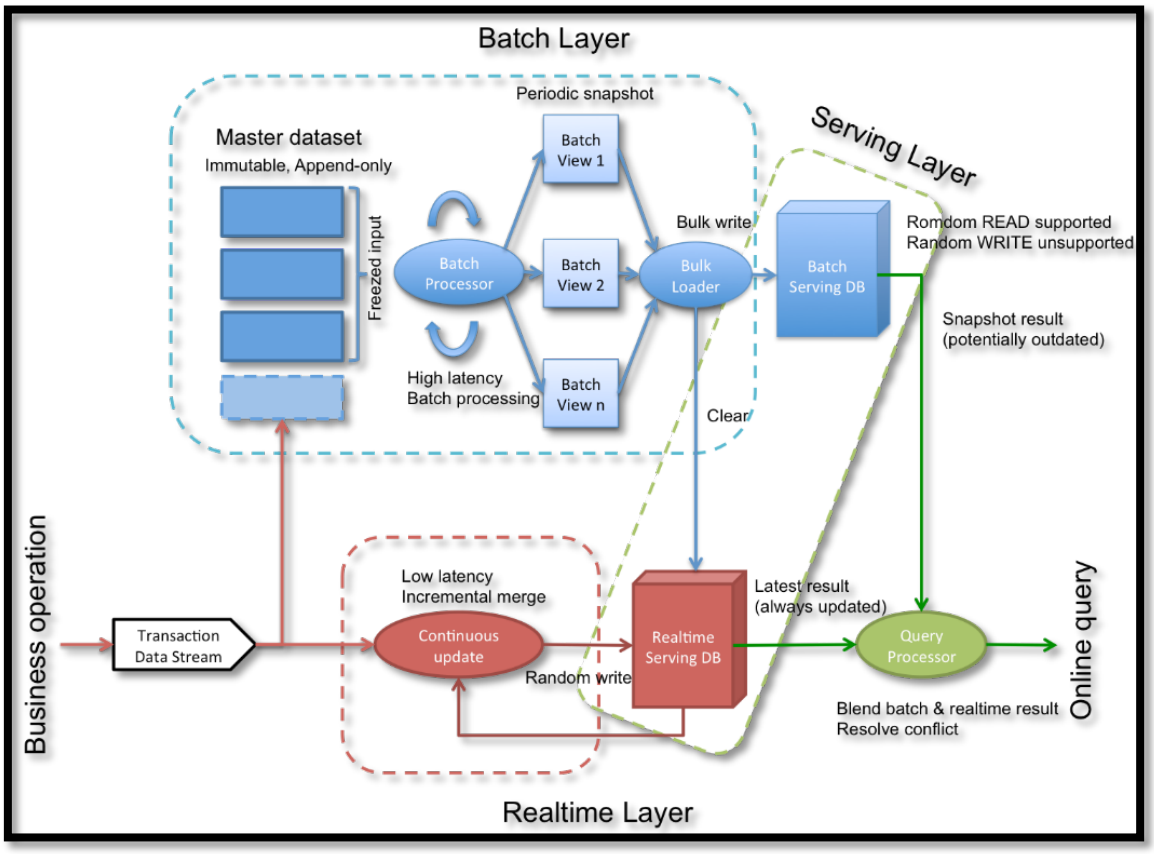

Figura 5. Arquitectura Lambda.
Fuente: http://horicky.blogspot.com.es

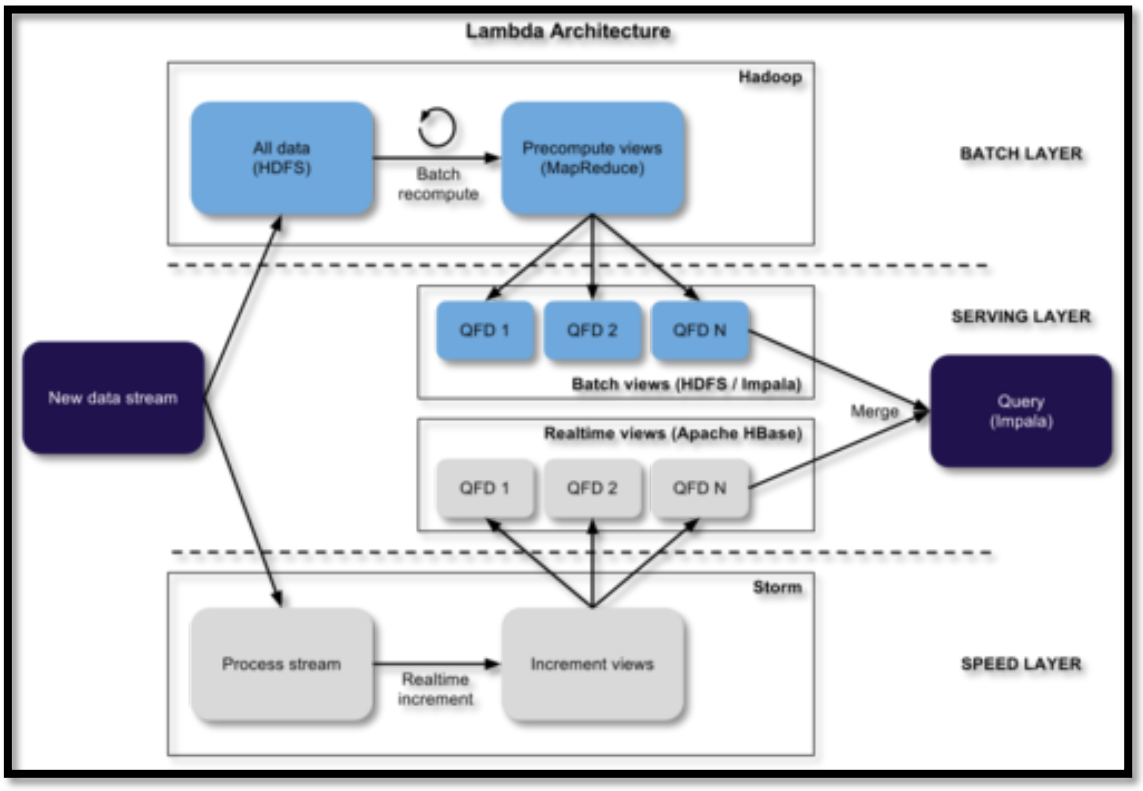

Figura 6. Arquitectura Lambda.
Fuente: http://jameskinley.tumblr.com

Ventajas:

- Mantiene los datos de entrada sin cambios.
- Tiene en cuenta el problema de reprocesamiento de datos.

Inconvenientes:

- Mantener el código que necesita para producir el mismo resultado en dos sistemas distribuidos complejos es complicado.
- No solo se trata de código diferente, sino también de diferencias en la depuración e interacción con otros productos como Hive, Oozie, Cascading, etc.
- Son dos paradigmas de programación muy distintos.

Arquitectura Kappa

En el artículo de Jay Kreps, "Questioning the Lambda Architecture", este hace la siguiente propuesta:[1]

- Utilice Kafka (u otro sistema) que le permita conservar el registro completo de los datos que necesita procesar.
- Cuando desee realizar el reprocesamiento, inicie una segunda instancia del procesamiento en *streaming* que comience a procesar desde el principio de los datos originales pero dirija la información de salida a una nueva tabla resultado.
- Cuando haya concluido el segundo proceso, cambie la aplicación para que lea la nueva tabla.
- Detenga la versión del proceso inicial y elimine la antigua tabla de salida.[2]

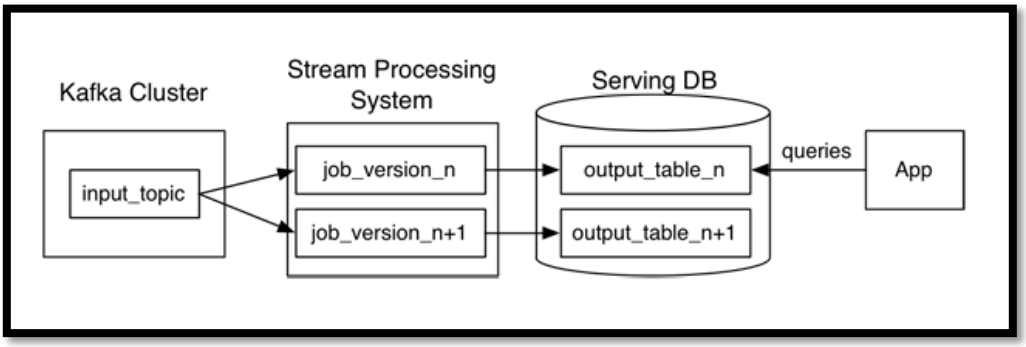

Figura 7. Arquitectura Kappa.
Fuente: https://www.zdnet.com

Ventajas:

- Facilita el reprocesamiento.
- Facilita las "marchas atrás".
- Datos originales Kafka + HDFS = escalabilidad del *data set* original.
- Solo hay un código que mantener.

[1] Según Jay Kreps: "Maybe we could call this the Kappa Architecture, though it may be too simple of an idea to merit a Greek letter" (Jay Kreps. "Questioning the Lambda Architecture". O'Reilly. 2 de julio de 2014. [En línea] URL disponible en: https://www.oreilly.com/ideas/questioning-the-lambda-architecture
[2] Ibídem.

Inconvenientes:

- Almacenamiento temporal para procesos espejo.