Arquitectura de análisis de ventas de Inditex en tiempo real

Autor:
Santiago Hernáez Arzúa

Tutor:
Ignacio García Fernández

Resumen

En este proyecto se ha realizado la implementación de un motor de enriquecimiento y análisis en tiempo real, en un entorno Big Data, de facturas de compra basadas en la empresa Inditex. El motor de enriquecimiento se encargará de añadir a los datos recibidos los datos de negocio, mientras que el motor de análisis aplicará la inteligencia de negocio a los diferentes eventos, aportando cálculos que extraerán información de interés.

En primer lugar se ha hecho un estudio de la estructura empresarial, dividida en 8 secciones claramente diferenciadas (Zara, Oysho, Uterqüe, Stradivarius, Pull&Bear, Bershka, Massimo Dutti y Lefties), presentes en 87 países que internamente dividen en 3 zonas (Europa, Asia y Latinoamérica). Una vez conocida la estructura empresarial, se ha analizado los datos contenidos en cada evento de entrada, en este caso, un ticket de compra, para comenzar con la implementación del motor de enriquecimiento y análisis, basado en Flink. Este software permite el despliegue de un sistema distribuido que proporciona escalabilidad y alta disponibilidad. Flink posee un conjunto de interfaces para la implementación de una topología por donde transcurren los flujos de eventos, permitiendo su enriquecimiento y procesado. La construcción de la topología se ha realizado en el lenguaje de programación Scala junto con Java.

Una vez completado el motor de enriquecimiento y análisis, se ha utilizado el programa Zoomdata, un sistema de visualización interactiva, para crear dos dashboards de ejemplo de presentación de datos, uno adaptado a inteligencia de negocio y otro no interactivo para implementar en tiendas con datos particulares.

Índice

Índice de figuras

1 Introducción

1.1 Motivación

Hoy en día existe la necesidad del procesamiento de una gran cantidad de datos en tiempo real. Existen multitud de casos en el mundo real donde es necesario saber qué está ocurriendo para poder tomar decisiones de negocio basándose en la información recibida desde diferentes fuentes lo antes posible. Inditex se caracteriza por, aún siendo una empresa de moda que son tradicionalmente estacionarias, una rápida capacidad de respuesta ante los gustos cambiantes de sus clientes. En este proyecto se aplicarán estos casos de uso a datos recibidos desde los puntos de venta para intentar aportar una respuesta más precisa a los intereses de los compradores.

1.2 Objetivos

El propósito general de este proyecto es la implementación y el desarrollo de un motor de enriquecimiento y análisis de eventos, creando flujos de información y extrayendo información de negocio a partir de ellos basándose en el caso de la empresa Inditex.

1.3 Estado del arte

Actualmente las tiendas de moda, ya sea por tradición o por desconocimiento, no tienen un sistema de reporte de ventas o de stock en tiempo real. Inditex, líder en el mercado mundial, es la empresa más avanzada en el aspecto tecnológico utilizando etiquetas RFID para mantener la trazabilidad de las prendas en todo el proceso desde la fabricación a la venta y proveyendo a sus empleados con dispositivos portátiles para comprobar el stock. Pero aún así no disponen de un medio para determinar su rendimiento en tiempo real o comparado con otras tiendas de su zona, país o marca. Actualmente estos datos se obtienen de forma retardada mediante procesos batch nocturnos, con lo cual disponen de los datos al día siguiente; pero esto ya no es suficiente. Con un sistema Big Data se podrían obtener los datos en tiempo real e incluso unirlo a los sistemas de stockage, lanzando avisos cuando el stock de una prenda sea bajo o realizando pedidos a fábrica cuando la frecuencia de venta de una prenda determinada sea mayor de lo normal.

Igualmente se pueden recoger datos de prendas más vendidas, colores o tipos de prenda, que hoy en día se hace de forma "manual" mandando semanalmente los jefes de tienda un informe basado en las recomendaciones de los vendedores, que pueden ser sesgadas.

2 Arquitectura

El esquema de la topología completa es el siguiente:

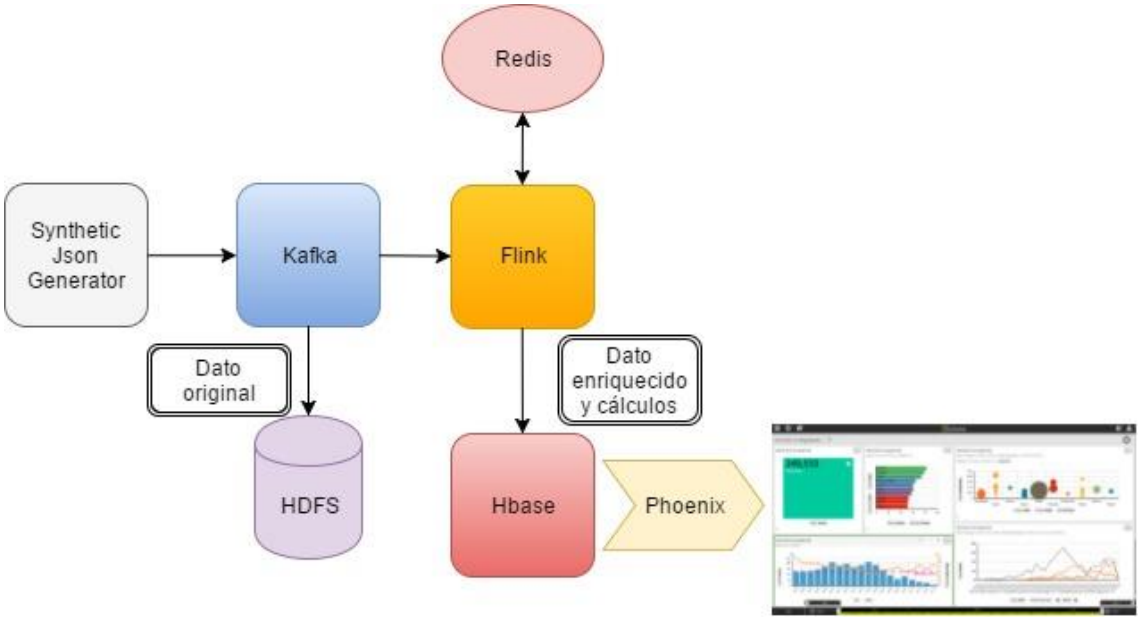

Figura 1 - Topología completa del diseño

2.1 Zookeeper

Zookeeper es un servicio distribuido de coordinación open-source para aplicaciones distribuidas. Funciona exponiendo un conjunto simple de primitivas que las aplicaciones distribuidas pueden utilizar para implementar servicios de alto nivel para sincronización, configuración de mantenimiento y agrupamientos. Está diseñado para ser sencillo de programar y utiliza un modelo de datos basado en la estructura de directorios de los sistemas de ficheros.

Zookeeper proporciona alta disponibilidad mediante redundancia.

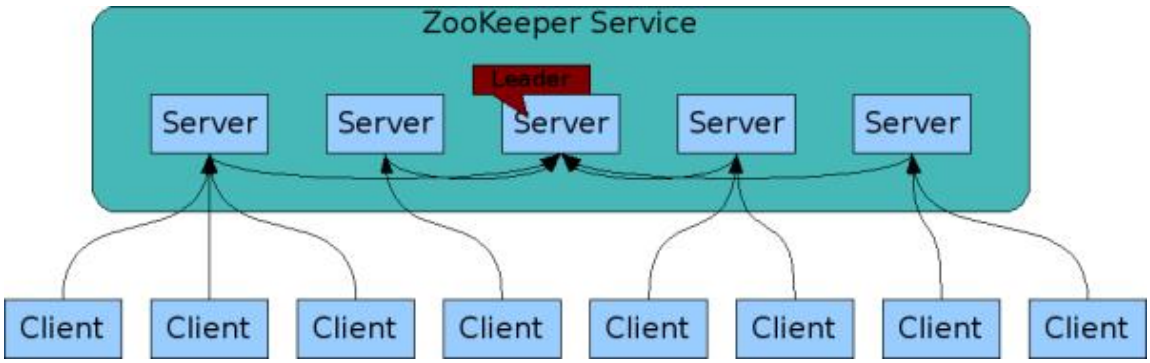

Figura 2 – Representación de un clúster Zookeeper

2.2 Apache Kafka

Apache Kafka es una plataforma distribuida de streaming, con 3 características principales:

- Permite publicar y suscribirse a streams de registros, similar a una cola de mensajes.
- Permite almacenar streams de registros de forma tolerante a fallos.
- Permite procesar streams de registros mientras ocurren.
- Es por ello que Apache Kafka es un sistema ideal para crear canales de datos en tiempo real para transportar la información entre sistemas o aplicaciones.

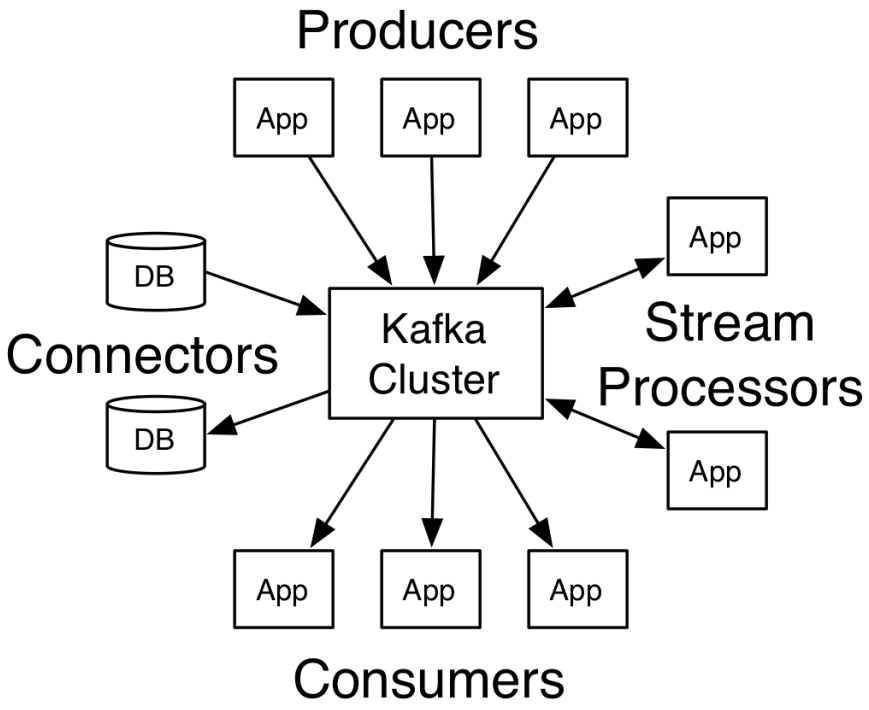

Figura 3 - Arquitectura de Apache Kafka

2.3 Redis

Redis es un almacén de datos estructurados en memoria, utilizado como base de datos, caché y bróker de mensajes. Soporta diferentes estructuras de datos como strings, hashes, lists, sets, sorted sets, bitmaps, hyperloglogs e índices geoespaciales. Redis integra replicación, Lua Scripting, transacciones y diferentes niveles de persistencia en disco. Proporciona alta disponibilidad mediante Redis Sentinel y particionamiento automático con Redis Cluster.

2.4 Apache Flink

Apache Flink es un motor de procesamiento de flujos de información para aplicaciones distribuidas, de alto rendimiento, alta disponibilidad y precisas.

Flink permite un alto procesamiento de datos con muy baja latencia y está diseñado para trabajar en clúster de alta capacidad, con varios cientos o miles de nodos. Igualmente está diseñado para trabajar con flujos continuos de datos procesando dato a dato en vez de en series de micro grupos, lo que permite asegurar un rendimiento, una resistencia a fallos y una recuperación de errores muy importante.

Flink posee librerías de proceso de eventos complejos (CEP), machine learning, procesamiento de gráficos y compatibilidad con Apache Storm.

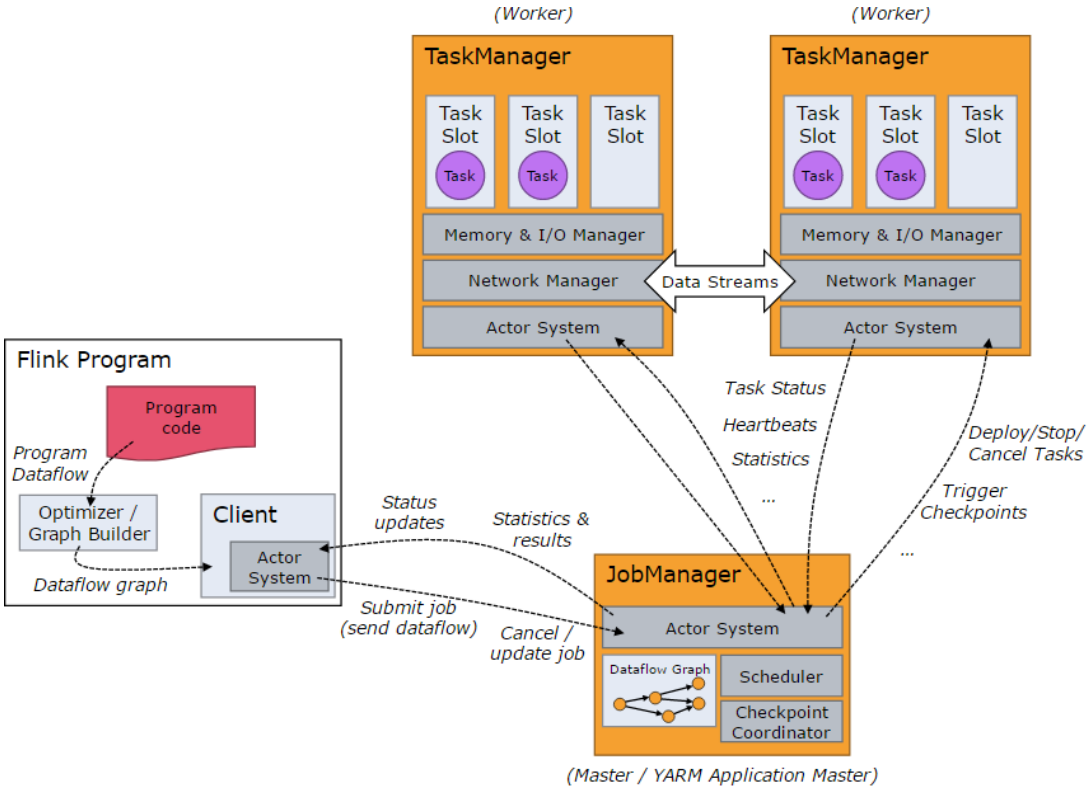

Figura 4 - Arquitectura de Apache Flink

2.5 Apache HBase

Apache HBase es una base de datos distribuida, escalable y de alto rendimiento. Apache HBase se utiliza cuando hay una necesidad de almacenar tablas muy grandes (billones de filas y millones de columnas), con acceso aleatorio de lectura y escritura en tiempo real.
Es una base de datos no relacional modelada en base a BigTable de Google y funciona sobre Hadoop, coordinándose con Zookeeper.

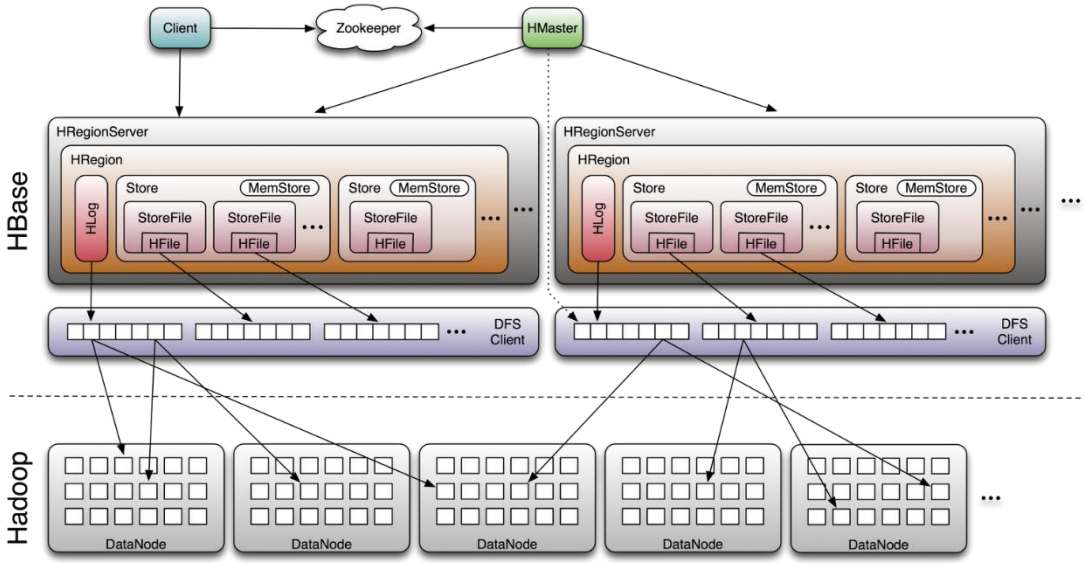

Figura 5 - Arquitectura de Apache HBase

2.6 Apache Phoenix

Apache Phoenix permite operaciones OLTP y analítica operacional en Hadoop para aplicaciones de baja latencia combinando lo mejor de ambos mundos:

- El poder del standard SQL y APIS JDBC con capacidades ACID completas
- La flexibilidad del mundo No-SQL sin esquema fijo utilizando HBase como sistema de almacenamiento.

Apache Phoenix traduce las consultas SQL y las compila en una serie de scans de HBase, orquestra la ejecución de esas órdenes y produce grupos de resultados JDBC regulares. El uso directo de la API de HBase, junto con el uso de coprocesadores y filtros resulta en un rendimiento del orden de milisegundos para consultas pequeñas o segundos en el caso de consultas sobre varios millones de filas.

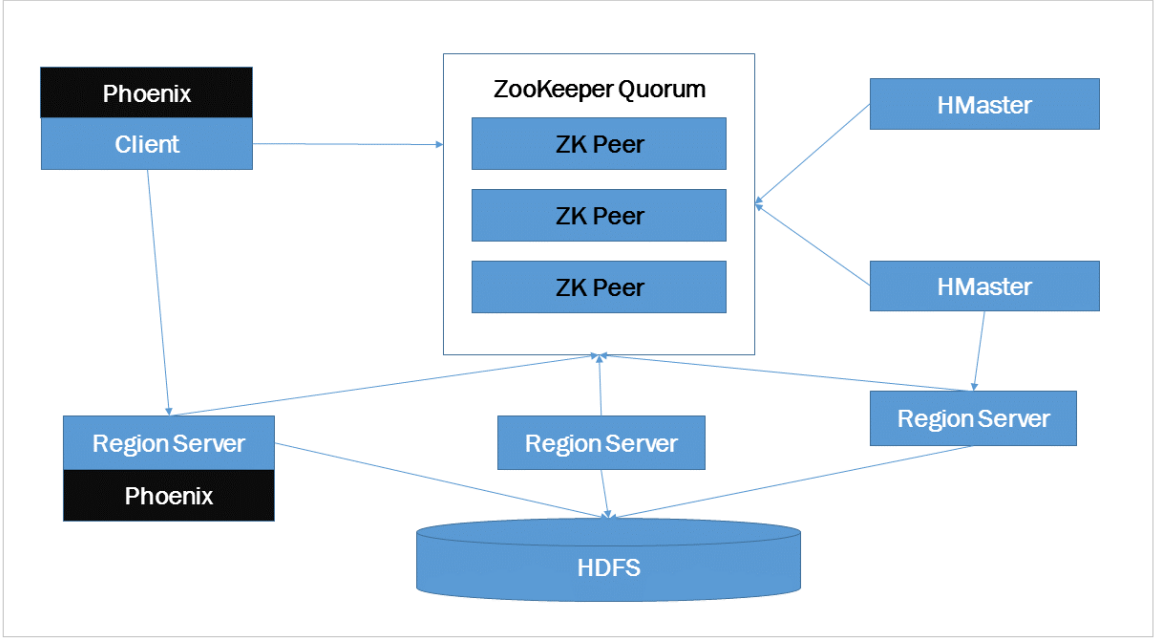

Figura 6 - Arquitectura de Apache Phoenix

2.7 Zoomdata

Zoomdata es un sistema de visualización interactivo de datos. Soporta un amplio rango de visualizaciones de datos, desde análisis exploratorio de datos hasta dashboards. Las visualizaciones interactivas permiten explorar los datos en tiempo real con una interfaz sencilla de utilizar y con una curva de aprendizaje muy corta y un diseño muy visual.

2.8 Eventos

La base fundamental de todo sistema es la información que en este caso está representada en forma de eventos de tipo JSON. Estos eventos están creados basándose en la estructura real de un ticket de venta de una tienda Zara. Estos eventos son introducidos en Apache Kafka para su lectura y posterior procesado.

Figura 7 - Ejemplo de ticket de compra

Un ejemplo de esos eventos está proporcionado en la figura siguiente:

```
{
    "id_transaccion": 12345,
    "id_tienda": 1234,
    "fecha": 1495711526,
    "metodoPago": "Efectivo",
    "prendas": ["1234:89.95", "2345:35.95"]
}
```

Figura 8 - Ejemplo de JSON generado

A continuación analizaremos los campos del evento en nuestro escenario.

- *Id_transaccion*: Número único que identifica la compra y el número de ticket.
- *Id_tienda*: Identificador de la tienda donde se ha realizado la venta.
- *Fecha*: Fecha y hora de la transacción.
- *metodoPago*: Forma de pago utilizada por el cliente. Existen definidas 5 formas de pago: Tarjeta de crédito, Efectivo, App (referente a la aplicación propia de la marca, que permite el pago desde móvil), tarjeta regalo y Affinity (tarjeta de crédito propia de la marca).
- Prendas : conjunto de valores formado por dos campos:
    - *Id de prenda*: identificador único de la prenda.
    - *Precio:* precio de la prenda en € en esa tienda en particular, habiendo aplicado descuentos, promociones y cambio de moneda.

Estos eventos serían generados en los propios puntos de venta de las tiendas, que los transmitirían directamente a un Topic de Kafka. En este proyecto, estos eventos se han generado utilizando el generador de Json: Synthetic Json generator.

2.9 Persistencia de datos:

La persistencia del dato original se obtiene mediante el guardado en HDFS de la cola Kafka original, sin modificar. Para obtener esta persistencia, se deben modificar los siguientes valores en el fichero broker.id del bróker de Kafka:

Nombre	Descripcion	Valor recomendado
delete.topic.enable	Enables delete topic. Delete topic through the admin tool will have no effect if this config is turned off	false
log.retention.bytes	The maximum size of the log before deleting it	-1
log.retention.hours	The number of hours to keep a log file before deleting it (in hours), tertiary to log.retention.ms property	-1
log.cleanup.policy	The default cleanup policy for segments beyond the retention window. A comma separated list of valid policies. Valid policies are: "delete" and "compact"	compact
compression.type	Specify the final compression type for a given topic. This configuration accepts the standard compression codecs ('gzip', 'snappy', 'lz4'). It additionally accepts 'uncompressed' which is equivalent to no compression; and 'producer' which means retain the original compression codec set by the producer.	gzip

Si el número mensual de mensajes en el Topic fuera extremadamente alto, convendría implementar algún sistema de Backup de los logs para evitar saturar los discos duros del/los brokers.

La persistencia del dato enriquecido se obtiene mediante el guardado directo en HBase en la tabla INDITEXTABLE.

3 Topología en Apache Flink

En esta sección se desarrolla la topología implementada en Apache Flink. Esta topología se encagará del enriquecimiento de los eventos a través de la base de datos contenida en Redis y realizará tareas de agregación y transformación de los flujos para guardarlos en HBase.

3.1 Funciones y utilidades:

3.1.1 Conversor de JSON a mapa de Scala

Esta función utiliza la librería de Java Jackson para convertir los eventos JSON a un mapa de Scala, que se utilizará posteriormente para crear el POJO de Scala datoOriginal.

3.1.2 Enricher

Esta función implementa RichFlatMap y recibe como parámetro un POJO datoOriginal devolviendo un POJO datoEnriquecido.
Esta función utiliza la librería de Java Jedis para acceder a Redis y extraer, mediante el ID_TIENDA y los diferentes ID_PRENDA los diferentes campos asociados a la tienda y a las prendas adquiridas.

3.1.3 Sink de HBase

Este conjunto de funciones implementa un sink de flink a HBase. Esta versión no está aún aceptada por la comunidad de Flink y no está incluida en Maven, por lo cual se ha incluido íntegra en el proyecto.

El procesamiento completo en Flink se muestra en el siguiente esquema:

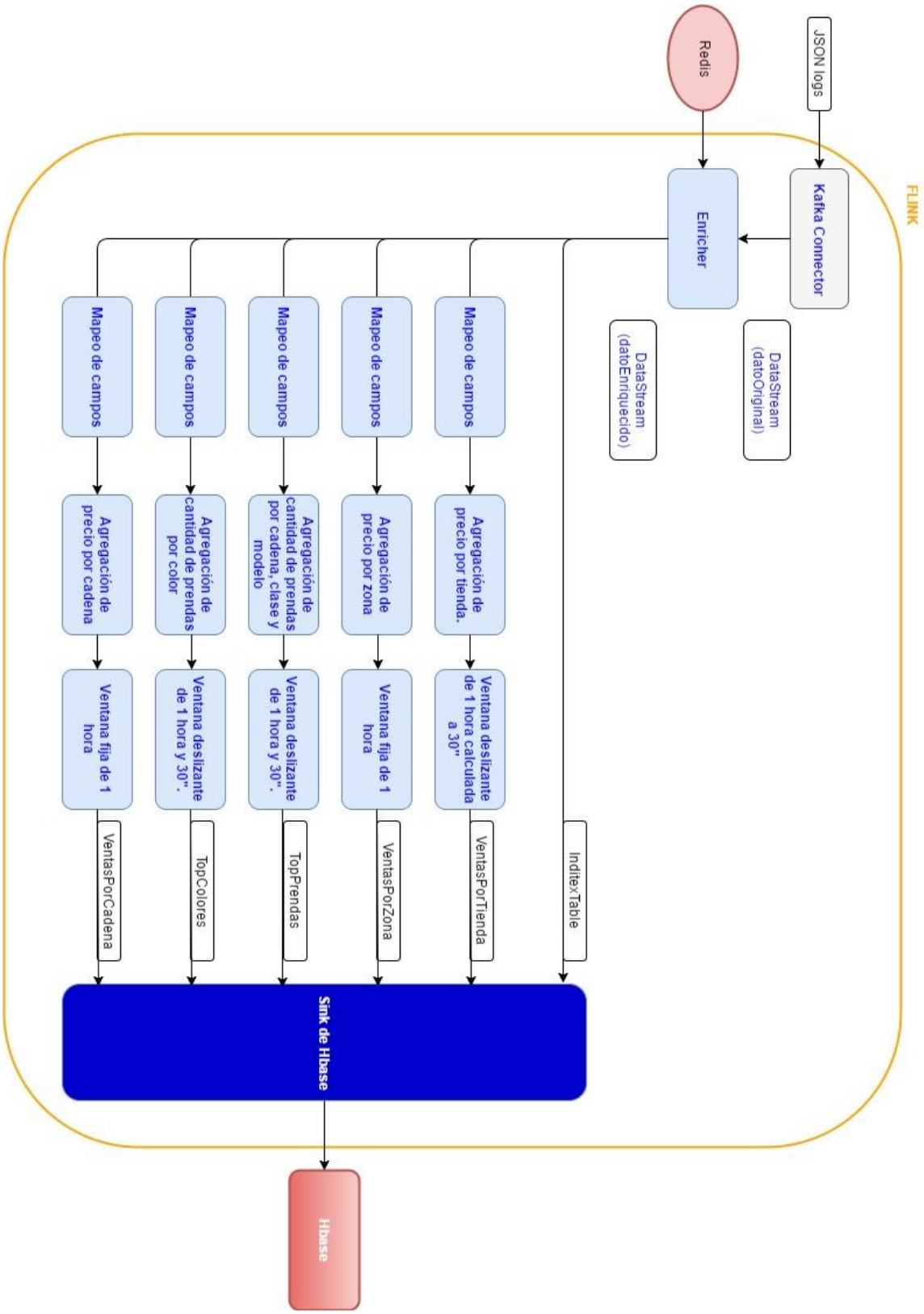

Figura 9 - Representación de la topología de procesamiento en Flink

3.2 Benchmarking Apache Flink

Esta sección trata el tema de la optimización de nuestra topología de Apache Flink. Para ello he hecho una serie de pruebas de carga sobre el equipo con los resultados siguientes:

Configuración de servidores:

Kafka: 1 broker, 1 consumer, 1 producer

Flink: Servidor en modo local

Redis: Modo local con persistencia a disco

HBase: Hmaster y HRegionServer en ejecución en modo local. Persistencia a disco (no HDFS)

Zookeeper: Modo standalone

Número de mensajes/s	CPU	RAM	CPU tras 1 h	RAM tras 1h
500	75%	64%	82%	65%
1000	87%	74%	93%	77%
1500	90%	78%	99%	89%
2000	error	error	error	Error

El equipo tras subir el número de mensajes a 1500 por segundo comienza a no responder adecuadamente y bloquearse por segundos. Con 2000 mensajes por segundo el zookeeper se bloquea por falta de RAM y se interrumpe el proceso.

Considerando que el ZoomData, sistema que utilizo para la visualización de los datos, consume más o menos un 25% de RAM (2 Gb), si lo eliminamos del sistema, se podrían manejar sin problema 3000 o 4000 mensajes por segundo, siempre que la CPU lo soporte.

Para intentar aumentar el flujo de mensajes se podría disminuir la frecuencia de cálculo de las ventanas deslizantes (de 30'' a 1') o la frecuencia de cálculo de las ventanas fijas (de 1h a 30').

El equipo en el que se han realizado las pruebas es un Intel Core i5-6300 con 8Gb de Ram.

4 Despliegue en clúster

Extrapolando datos anteriores, para manejar un volumen de 1,000,000 mensajes por segundo se necesitarían los siguientes equipos:

Zookeeper: 3-5 equipos para asegurar la alta disponibilidad del sistema.

Kafka: 5 Brokers con réplica 5, de esa forma aseguraríamos la alta disponibilidad en cualquier caso. Cada bróker tendría que gestionar sobre 200,000 mensajes por segundo.

Redis: Considerando que a Redis no se escribe de forma intensiva, sino que sólo se lee, se podría utilizar 3 de los equipos dedicados a Zookeeper para servir la BBDD de redis. El número de equipos dedicados a Redis debe ir determinado por el tamaño de la Base de datos de prendas. La Base de datos de tiendas no varía con una frecuencia tan alta y las variaciones son mínimas. El throughput de redis medio en un portátil es de aproximadamente 500,000 respuestas por segundo.(https://redis.io/topics/benchmarks)

Flink: Flink necesita para despliegue en clúster al menos un Job Manager y uno o mas Task Managers. En este caso en concreto se ha decidido implementar 3 Job Managers para asegurar la alta disponibilidad y 20 Task Managers. Considerando que los task managers dedicados pueden soportar mas flujo de mensajes que el equipo de pruebas, debería ser suficiente.

HBase: El rendimiento del HBase viene determinado por el tipo de discos duros que utilicen los equipos dedicados a éste. Siendo los SSD un 40% más rápidos que los HDD. El número de máquinas dedicadas a HBase debe ser, al menos, un Master y uno o varios RegionServers. En este caso necesitaremos 3 Masters para asegurar la alta disponibilidad y 10 region servers para conservar el flujo de mensajes.

5 Presupuesto

Si se desplegase este sistema en AWS, los costes asociados serían los siguientes:

Utilizando servidores dedicados, que sería lo normal en una operación de estas características:

Compute: Amazon EC2 Dedicated Hosts:

	Description	Number of Hosts	Usage	Type	Billing Option
⛔	Zookeeper	5	100 % Utilized/Mo	m4	1 Yr All Upfront Reservation
⛔	Kafka	5	100 % Utilized/Mo	c3	1 Yr All Upfront Reservation
⛔	Flink Job Manag	3	100 % Utilized/Mo	c4	1 Yr All Upfront Reservation
⛔	Flink Task Mana	20	100 % Utilized/Mo	c3	1 Yr All Upfront Reservation
⛔	HBase Master	3	100 % Utilized/Mo	c3	1 Yr All Upfront Reservation
⛔	HBase Region S	10	100 % Utilized/Mo	c3	1 Yr All Upfront Reservation
➕	Add New Row				

Figura 10 - Descripción de equipos presupuestados

El importe anual estimado sería el siguiente:

⊖	Amazon EC2 Service (Europe)		$ 564196.00
	Compute:	$ 0.00	
	Reserved Instances (One-time Fee):	$ 564196.00	
⊖	AWS Data Transfer In		$ 0.00
	Europe (Ireland) Region:	$ 0.00	
⊖	AWS Data Transfer Out		$ 921.51
	Europe (Ireland) Region:	$ 921.51	
⊖	AWS Support (Business)		$ 23853.49
	AWS Support Plan Minimum:	$ 100.00	
	Support for Reserved Instances (One-time Fee):	$ 23753.49	
Free Tier Discount:			$ -1.26
Total One-Time Payment:			$ 587949.49
Total Monthly Payment:			$ 1020.25

Figura 11 - Presupuesto completo en AWS

No están asignados los gastos por balanceadores de carga o IP, se han añadido 10Tb/mensuales de entrada y salida de datos.

Seleccionando instancias de EC2, el precio por mes, añadiendo costes por transferencia de datos (IN&OUT) sería de aproximadamente: 21558,87$ mensuales. (258706,44$ anuales). Con el inconveniente que las instancias de EC2 no siempre se ejecutan sobre el mismo Host, pudiendo tener problemas de transferencia de datos o persistencia de los mismos en grandes volúmenes.

6 Listado de Tablas utilizadas y características

6.1 Redis:

6.1.1 Tiendas:

Columna	Tipo
Id_tienda	Integer
Cadena	String
Sexo	String
Pais	String
Region	String
Zona	String

6.1.2 Prendas:

Columna	Tipo
Id_prenda	Integer
Nombre	String
Clase	String
Modelo	String
Color	String
Talla	String
Beneficio	Double

6.2 HBase:

6.2.1 INDITEXTABLE

Fila	Tipo
PK	String - CADENA + ID_TRANSACCION
ID_TRANSACCION	Integer
FECHA	Timestamp
METODOPAGO	String
ID_TIENDA	Integer
CADENA	String
SEXO	String
PAIS	String
REGION	String
ZONA	String
ID_PRENDA	Integer
PRECIO	Double
BENEFICIO	Double
COLOR	String
TALLA	String
NOMBRE	String
MODELO	String
CLASE	String

6.2.2 VENTASPORTIENDA

Fila	Tipo
PK	String - CADENA+ID_TIENDA
FECHA	Timestamp
ID_TIENDA	Integer
CADENA	String
SEXO	String
PAIS	String
REGION	String
ZONA	String
TOTAL	Double

6.2.3 VENTASPORZONA

Fila	Tipo
PK	String - FECHA + ZONA + PAIS
FECHA	Timestamp
ZONA	String
PAIS	String
TOTAL	Double

6.2.4 VENTASPORCADENA

Fila	Tipo
PK	String - FECHA + CADENA
FECHA	Timestamp
CADENA	String
TOTAL	Double

6.2.5 TOPPRENDAS

Fila	Tipo
PK	String - FECHA+CADENA+MODELO+CLASE
FECHA	Timestamp
CADENA	String
MODELO	String
CLASE	String
CANTIDAD	Integer

6.2.6 TOPCOLORES

Fila	Tipo
PK	String - FECHA + COLOR
FECHA	Timestamp
COLOR	String
CANTIDAD	Integer

7 Referencias

Apache foundation – Zookeeper Wiki

https://cwiki.apache.org/confluence/display/ZOOKEEPER/Index

Apache Foundation – Kafka Wiki

https://cwiki.apache.org/confluence/display/KAFKA/Index

Redis Documentation

https://redis.io/documentation

Apache Foundation – Flink Documentation

https://ci.apache.org/projects/flink/flink-docs-release-1.2/

Scala – Scala Documentation

http://www.scala-lang.org/documentation/

Apache Foundation – Hbase Documentation

http://hbase.apache.org/book.html

Apache Foundation – Phoenix

https://phoenix.apache.org/

Zoomdata: Big Data analytics and visualization

https://www.zoomdata.com/docs/2.5/

Inditex

http://www.inditex.com/es/home

Amazon AWS

http://aws.amazon.com/es/documentation/s3/

Amazon Calculator

https://calculator.s3.amazonaws.com/index.html