Universidad ORT

Facultad de Ingeniería
Posgrado de analítica en Big Data

Taller de analítica en tiempo real
Obligatorio

Monitoreo en tiempo real de calificaciones de productos

Docentes: Pedro Bonillo / Nicolás Martínez

Enrique Peirano (252190)
Alfredo Rodríguez (5515)

Montevideo, Diciembre 2021

Indice

1 Resumen 3

2 Objetivos 3

2.1 Generales 3

2.2 Específicos 3

3 Metodología 4

4 Justificación y Alcance 4

5 Limitaciones 5

6 Obtención y tratamiento de datos 5

7 Propuesta y Arquitectura seleccionada 6

7.1 Etapa de Edge: generación de datos 6

7.2 Gateway de Ingesta de datos: Broker Mosquitto 6

7.3 Administración de flujos de streaming: Apache NiFi 7

7.4 Data Syndication: Apache Kafka 9

7.5 Procesamiento y análisis del flujo de streaming: Apache Flink 10

7.6 Visualización resultados 14

8 Conclusiones 15

9 Recomendaciones 16

10 Referencias 16

11 Anexos 17

1 Resumen

El procesamiento de streaming de datos en tiempo real o casi tiempo real (*near real time*), permite obtener *insights* en el mismo momento en que están arribando, lo que posibilita tomar decisiones rápidamente a partir del objetivo definido. El uso de un sistema distribuido de procesamiento es clave en aplicaciones de Big Data dados por la variabilidad, velocidad de generación, y grandes volúmenes de datos (las 3 "V" de Big Data).
El presente documento pretende implementar una solución para monitorear en tiempo real el *rating* promedio de distintos productos comercializados por la plataforma de *e-commerce* de Amazon. Se describe la implementación de un sistema de streaming basado en una arquitectura de referencia para IoT, utilizando un Broker MQTT para IoT como es Mosquitto, y herramientas de software libre dentro del ecosistema Apache.
A los efectos del caso de uso seleccionado, los datos provienen del dataset de calificaciones de productos en la plataforma *e-commerce* de Amazon.
Si bien no se trata de una aplicación que implique la generación de streaming de datos a partir de dispositivos IoT, desde el punto de vista académico el modelo es totalmente válido. La etapa de Edge (fuera del contexto de nuestro sistema), en vez de ser un dispositivo IoT, será un *script* de Python que simula la generación de los datos de streaming a partir de un archivo csv con los datos de calificaciones de productos de Amazon.

2 Objetivos

2.1 Generales

Como objetivo general se propone la implementación de una arquitectura de referencia para IoT para el procesamiento de streaming de datos y la obtención de *insights* o información valiosa en tiempo real.

Estos *insights* en tiempo real pueden ser detección de anomalías, elaboración de dashboards para visualización y monitoreo, detección de patrones de comportamiento de los clientes, entre otros.

2.2 Específicos

Para el caso de uso particular seleccionado, se busca obtener diferentes indicadores en tiempo real, a partir de un flujo de datos que contiene las calificaciones de usuarios correspondientes a varios productos, ofrecidos en la plataforma de e-Commerce de Amazon. En particular, las categorías *Home and Kitchen* y *Movies and TV*.
El indicador obtenido corresponde al promedio de las calificaciones de cada producto cada cierto tiempo (ventana de tiempo configurable).

Por otro lado, como objetivo académico, poder aplicar una arquitectura de referencia de tiempo real en un entorno *cloud* integrando varias herramientas de código abierto vistas en el curso.

3 Metodología

El trabajo realizado se basó en colaboración remota entre los integrantes del equipo y los docentes. Para el ambiente de trabajo, se utilizó una Virtual Machine (VM) creada en Microsoft Azure con las siguientes características:

- Imagen SO: CentOS-based 7.9 – Gen2
- Tamaño: Standard D2as_v4 (2 vcpus, 8 GB de memoria)
- Puertos públicos de entrada: SSH

En esta VM se instalan las siguientes herramientas: Mosquitto, Jupyter Notebook, Zookeeper, Apache NiFi, Apache Kafka y Apache Flink. Jupyter Notebook, Apache NiFi y Apache Flink, se utilizaron a través de su interfaz web, por medio de los puertos 8888, 8443 y 8081 respectivamente.

Estos servicios se levantan automáticamente una vez encendida la VM.

4 Justificación y Alcance

Para el caso de ventas retail, contar con indicadores en tiempo real permite tomar decisiones o acciones en el instante en que ocurren, mejorando la experiencia de usuario y evitando situaciones desfavorables para los objetivos del negocio.

Algunos de los beneficios que se identifican en un caso de uso como éste son:

- Control de stock en tiempo real
- Rápida detección de problemas (tanto con los productos, como con la plataforma de *e-commerce*).
- Identificar comportamientos de usuarios de modo de minimizar el *churning* de clientes y realizar acciones en el momento.
- Identificar situaciones favorables y poder realizar un análisis de contexto en tiempo real, sin necesidad de almacenar y analizar datos históricos.

El alcance del presente trabajo, incluye la recolección e ingesta de los datos, la transformación, estructuración y distribución, redireccionamiento del stream de datos, procesamiento y análisis en tiempo real de modo de poder obtener en consola los *insights* definidos.

Se toma una arquitectura ya definida, y por tanto no se realiza una justificación ni un comparativo con otras herramientas para su diseño.

Si bien el *framework* utilizado, soporta procesamiento distribuido, no se realizaron pruebas en clusters de más de un nodo. Entendemos que con el caso de uso planteado, no justifica el uso

de una estructura más compleja. De todas formas, como se mencionó la arquitectura es escalable de modo de requerirlo.

Los datos analizados son archivos csv extraídos de datasets de reseñas de clientes de Amazon en el año 2018 [1], para las categorías “Home and Kitchen” y “Movies and TV” con aproximadamente 4.5 millones de filas cada uno.

5 Limitaciones

Las limitaciones en el alcance de este trabajo, básicamente se derivan del corto plazo con que se contó para su realización y la falta de experiencia y conocimiento de algunas herramientas por parte de los integrantes del grupo

Si bien la estructura está diseñada para una ingesta de *streaming* de datos para IoT, usando el protocolo MQTT a través de Mosquitto, la generación de datos se realiza a través de archivos csv y no de una conexión real.

Una limitación que se encontró fue que las aplicaciones de Flink están realizadas en Java, por lo que, debido a la falta de experiencia en la programación en este lenguaje, no se pudo aprovechar la flexibilidad que brinda la API de DataStream de esta herramienta para modificar y adaptar las aplicaciones usadas en clase.

Queda fuera del alcance, la implementación de una etapa posterior en el pipeline, por ejemplo de visualización implementada con SOLr o ElasticSearch por ejemplo.
El desarrollo se limitó al análisis de dos categorías de productos, y a la categorización de acuerdo a sus calificaciones. No se realizaron otras implementaciones con el dataset de entrada, como podría ser el análisis de sentimiento o recomendación de productos a usuarios mediante modelos del tipo filtro colaborativo o basados en contenido.

6 Obtención y tratamiento de datos

En el dataset utilizado, cada fila es una revisión o calificación, conteniendo los siguientes campos:

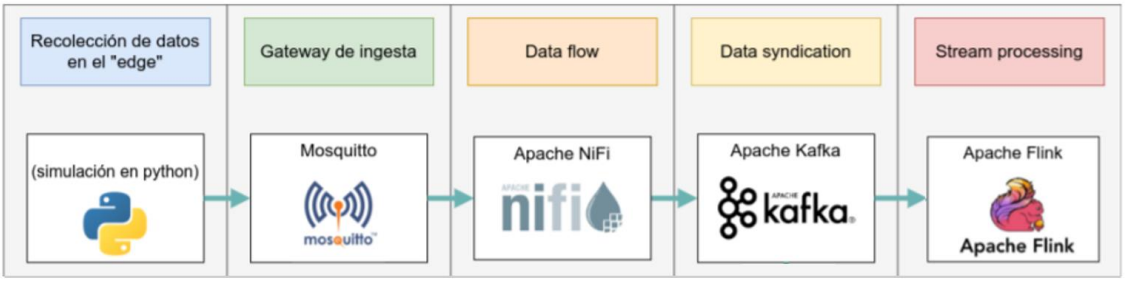
- reviewerID: ID del usuario que realizó la calificación, por ejemplo A2SUAM1J3GNN3B
- asin: ID del producto calificado, por ejemplo 0000013714
- overall: rating del producto
- unixReviewTime: hora de la calificación (unix time)

Los *datasets* considerados contienen más de cuatro millones de calificaciones cada uno. Por lo que cumple con el requisito dado para el proyecto, de contener más de un millón de registros.

El único pre-procesamiento realizado en el dataset, fue convertir el tipo de dato de rating de flotante a entero mediante código Python *(preprocesamiento.ipnyb)*, dado que la aplicación Java en Flink espera el dato como entero. Tal como se comentó anteriormente, la modificación del programa en java y su complicación en Flink quedó por fuera del alcance del trabajo.

7 Propuesta y Arquitectura seleccionada

Tal como fue mencionado, la arquitectura utilizada consta de varias etapas, y en cada una de ellas se emplean herramientas Open Source con un objetivo definido. Si bien para nuestro caso de uso no se justifica plenamente, esta arquitectura constituye un esquema sumamente escalable tanto a nivel de procesamiento (soporta procesamiento distribuido en cada etapa), de ingesta (el volumen de los datos puede ser muy grande), y de programación (se pueden reutilizar bloques principalmente en la etapa de gestión de flujos con NiFi).



7.1 Etapa de Edge: generación de datos

En esta primera etapa de nuestro *pipepline*, se tiene un módulo de Edge (es decir, que está fuera del contexto del sistema de streaming), que simula un dispositivo IoT. Esto es implementado mediante un script de python (*reproductor_ratings.py*) que toma los datos de los archivos csv de calificaciones de Amazon, ubicados en un directorio predeterminado.

7.2 Gateway de Ingesta de datos: Broker Mosquitto

En una segunda etapa, tenemos un Gateway de ingesta de datos conformado por un broker Mosquitto, el cual maneja el protocolo MQTT, muy usado en sistemas IoT.
El *Broker* se denomina al servidor que acepta mensajes publicados por clientes y los difunde entre los clientes suscritos. Los mensajes son etiquetados con un *tópico* o tema.

Mosquitto es un broker de mensajes de código abierto que implementa el protocolo MQTT, liviano y adecuado para su uso en todo tipo de dispositivos.
El protocolo MQTT proporciona un método ligero para intercambiar mensajes mediante un modelo de *publish/subscribe* donde se transmiten los datos en forma asíncrona y donde no se define el consumidor. Esto lo hace adecuado para la mensajería de IoT, como en el uso de sensores de baja potencia o dispositivos móviles. El proyecto Mosquitto también proporciona una librería en C para implementar clientes MQTT y la línea de comando mosquitto_pub y mosquitto_sub [2].

Una posible alternativa a Mosquitto en esta etapa podría haber sido Kafka, plataforma que será analizada más adelante.

En nuestro caso, el broker Mosquitto es también implementado por el script de python *reproductor_ratings.py.* Este script, toma los archivos csv correspondientes a las revisiones de clientes de Amazon, almacenados en el directorio “simulacion”, los convierte en formato json, arma el stream de datos y los envía al puerto 1883 dentro del tópico “amazon”.

7.3 Administración de flujos de streaming: Apache NiFi

Apache NiFi es una herramienta que permite administrar/orquestar flujos de datos utilizando una interfaz gráfica a través de un diseño modular. Una herramienta similar para este caso podría ser Flum.

En la interfaz gráfica de NiFi se definen flujos con Processors que pueden leer/escribir/procesar Flowfiles (la mínima unidad de datos en NiFI) desde distintas fuentes. Los Processors definidos en el pipeline, están conectados mediante Conexiones, salvo aquellos que terminan el flujo (con conexiones auto-terminadas).

En las propiedades de los Processors se define el tratamiento de los Flowfiles que realiza. Esto puede realizarse usando los NiFI Expression Language (EL) que permite acceder y realizar tareas con los atributos de los Flowfiles. En la pestaña Scheduling de cada Processors se puede agendar para que las tareas se ejecuten en batch mediante CRON, o bien con Timer Driven para la ejecución de tareas en tiempo real (definiendo Run Schedule = 0 seg) como es nuestro caso.

En las Conexiones se pueden definir políticas de priorización y encolamiento, definiendo umbrales de modo de avisar al Processor generador que no envíe más datos. También se pueden definir esquemas de balanceo de carga.

En una arquitectura de procesamiento distribuido, NiFi necesita sincronizarse con Zookeeper para autodescubrir el cluster.

En el presente proyecto se definen tres Processors en Nifi:

- **ConsumeMQTT:** es donde comienza el flujo NiFi. Este Processor toma los mensajes JSON de MQTT. Aquí se define dónde está el broker Mosquitto (puerto 1883 del localhost), y el tópico (amazon).
- **ConvertRecord:** convierte los Flowfiles en formato JSON a CSV. Los esquemas o formatos del JSON y los CSV están dados por los Controller Services. Estos formatos para los Flowfiles se definen internamente en NiFi mediante archivos del tipo avro. El Controller Service JSONTreeReader indica el formato con que lee el JSON. En nuestro caso simplemente se indica que infiera el esquema de los mensajes JSON que le llegan.
  Mientras que el Controller Service CSVRecordSetWriter indica el formato del CSV que escribe. Este esquema se define en un JSON en el archivo avro:

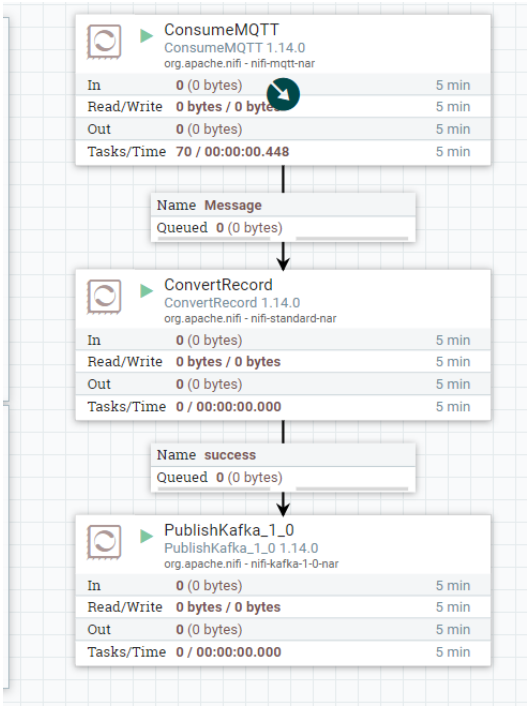
```
{
"name": "MyClass",
"type": "record",
"namespace": "com.acme.avro",
"fields": [
{
"name": "timestamp",
"type": "int"
},
```

```
{
"name": "reviewerID",
"type": "string"
},
{
"name": "dummy",
"type": "string"
},
{
"name": "rating",
"type": "int"
},
{
"name": "productID",
"type": "int"
}
]
}
```


La variable “dummy” se define simplemente a efectos de mantener la misma cantidad de variables que espera el programa de Flink utilizado en clase. Tal como se explicó dentro de las limitaciones del alcance de este trabajo, el programa Java no fue modificado.

- **PublishKafka:** finalmente este Processor envía el flujo de datos a Kafka bajo el tópico *amazon.*

A continuación se muestra el flujo descrito más arriba:



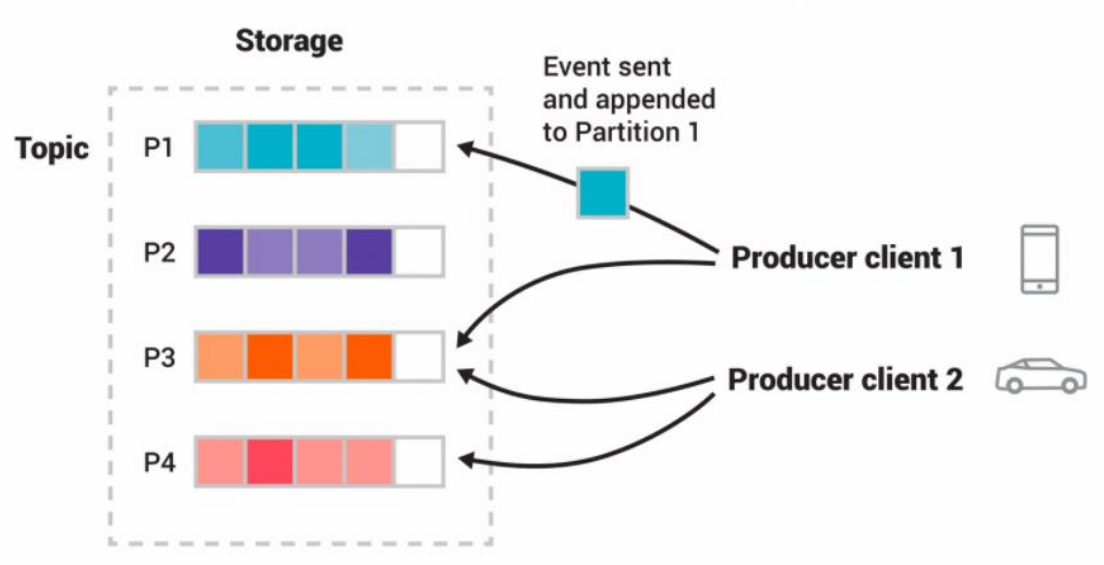
7.4 Data Syndication: Apache Kafka

En nuestra arquitectura, luego que NiFi consume los datos del broker Mosquitto y los procesa, los envía a Kafka para su distribución.

En la versión que usamos de Kafka, ésta necesita a Zookeeper para descubrir los clusters. A partir de la versión 2.8 Kafka ya cuenta con su propio motor de manejo de procesamiento distribuido.

Kafka en esta etapa puede realizar una bifurcación del flujo de datos, si por ejemplo se quisiera implementar una arquitectura Lambda, con una rama para procesamiento en batch, y otra para procesamiento en tiempo real, copiando el stream de salida en ambas ramas (en este caso habría dos suscriptores). En nuestro caso solamente será implementado el envío del stream para su procesamiento en tiempo real.

Los eventos en Kafka se organizan y almacenan en tópicos. Los tópicos en Kafka pueden ser de múltiples productores y múltiples suscriptores. Los eventos de un tópico se pueden leer con la frecuencia necesaria; a diferencia de los sistemas de mensajería tradicionales, los eventos no se eliminan después del consumo. La persistencia de estos eventos es configurable dentro de Kafka.
En un esquema distribuido, los tópicos están particionados, lo que significa que un tópico se distribuye en varios "buckets" ubicados en diferentes brokers de Kafka. Cuando se publica un nuevo evento en un tópico, en realidad se agrega a una de las particiones del tema. En la siguiente figura, se muestra el ejemplo de un tópico con 4 particiones (P1-P4) con dos productores [4].



En Kafka se crea el tópico “amazon” para que permita consumir el flujo de datos que viene desde el NiFi.

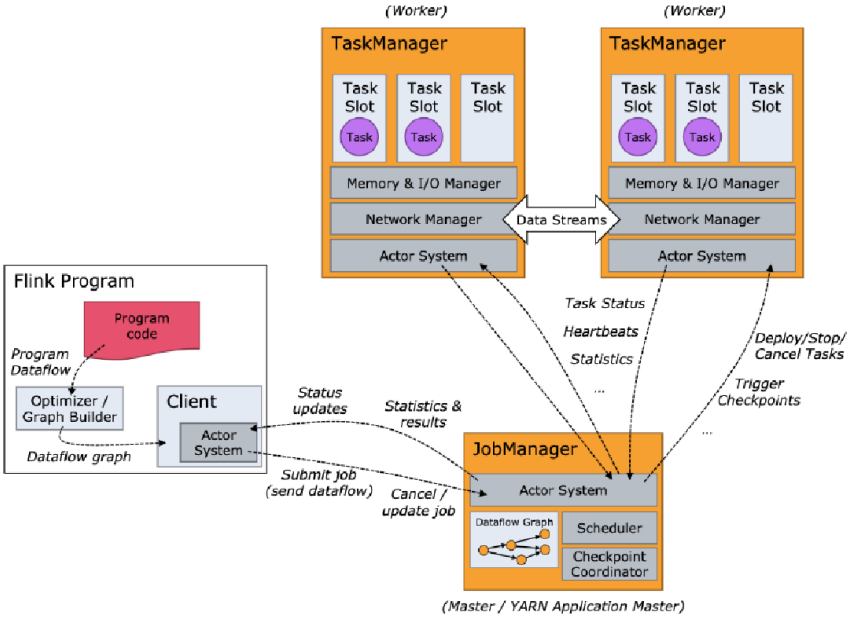
7.5 Procesamiento y análisis del flujo de streaming: Apache Flink

Apache Flink está en la misma categoría de Spark Streaming, pero a diferencia de éste que está orientado a near real time, Flink funciona muy bien en real time (aunque también puede ser usado en batch). Actualmente es la herramienta Open Source de procesamiento en tiempo real mejor conceptuada, a tal punto que está desplazando a Storm en las soluciones ofrecidas por la empresa Cloudera.

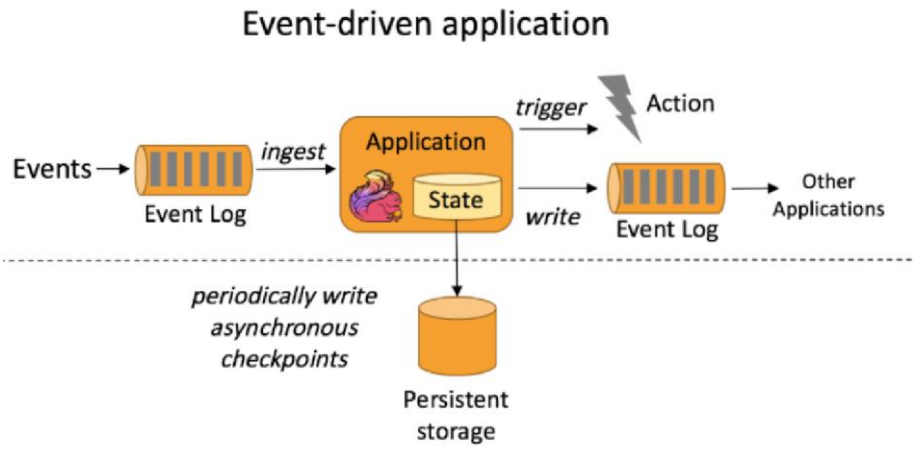
Apache Flink por un lado es un framework, es decir que permite desarrollar aplicaciones (en java), y por otro es un motor de procesamiento, dado que cuenta con su propia orquestación de cómputo distribuido, a diferencia de la versión de Kafka que usamos.

Algunas de las principales características de Flink:

- Puede trabajar con streams *unbounded* (para aplicaciones en tiempo real y casi tiempo real) y *bounded* (para procesamiento en batch, por ejemplo aplicaciones de machine learning).
- Maneja estados, lo que permite la funcionalidad de *recovery* frente a fallas.
- La arquitectura distribuida se basa en un Job Manager (master) y uno o varios Task Managers (nodos *workers* para el escalado horizontal).



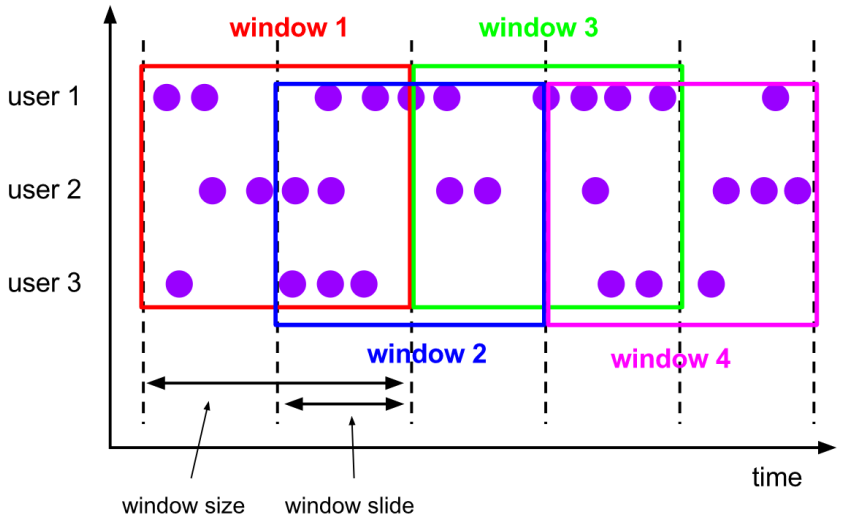
Uno de los casos de uso de Apache Flink es una aplicación orientada a eventos, tal como es nuestro proyecto. La figura siguiente muestra el esquema de este tipo de aplicación.



Una aplicación orientada a eventos es una aplicación con estado que ingesta eventos de un o más streams de eventos y reacciona a los eventos entrantes activando cálculos, actualizaciones de estado o acciones externas. Son una evolución del diseño de aplicaciones tradicionales donde las aplicaciones leen datos y los conservan en una base de datos transaccional remota.
Por el contrario, las aplicaciones orientadas a eventos se basan en aplicaciones de procesamiento de flujo con estado. En este diseño, los datos y el cálculo se ubican conjuntamente, lo que proporciona acceso a datos locales (en memoria o en disco). La tolerancia a fallas se logra escribiendo periódicamente puntos de control en un almacenamiento persistente remoto.

En nuestro caso el Event Log de entrada es Kafka que es manejado a través de un tópico. La aplicación de Flink toma los datos, los procesa en tiempo real, y dispara una acción.
Cada proceso de Flink en nuestro proyecto, tiene dos tareas cuya ejecución se puede comprobar en la interfaz web de la herramienta. Estas dos tareas son:

- Agrupamiento y mapeo de los datos: la aplicación hace un map formando una tupla con *productID* y *rating* y luego los agrupa por *productID* mediante la función *KeyBy*, para formar el DataStream.
- Procesamiento del DataStream en tiempo real utilizando ventanas del tipo *sliding* (con solapamiento) tal como se muestra en la siguiente figura [3].
-



En esta implementación se definen dos tipos de ventanas: window size y window slide (30 y 5 segundos respectivamente en nuestro caso). La primera define cuánto tiempo mira hacia atrás para asignar los elementos a ser procesados, y la segunda ventana controla cuán frecuentemente comienza una nueva ventana deslizante. Se da un solapamiento cuando la primera es mayor que la segunda, en este caso los mismos elementos pueden ser asignados en varias ventanas.
En nuestro caso tenemos una ventana de 30 segundos que se desplaza cada 5 segundos. De esta forma se obtienen cada 5 segundos, resultados parciales con el procesamiento de los eventos arribados en los últimos 30 segundos.

Este procesamiento es implementado mediante funciones *map* y *reduce*, donde obtiene el promedio de las calificaciones de un mismo producto con los datos que se encuentran en la ventana de 30 segundos. Luego se van actualizando estos datos cada 5 segundos, como se explicó anteriormente.

Finalmente, en la etapa de *sink* o de la salida de la aplicación, imprime los resultados en un archivo. Otras opciones en esta etapa podrían ser: enviar a Kafka, enviarlo a una Base de Datos NoSQL de tipo clave-valor como Redis, o enviarlo a Elastic Search y visualizar los resultados en un dashboard Kibana.

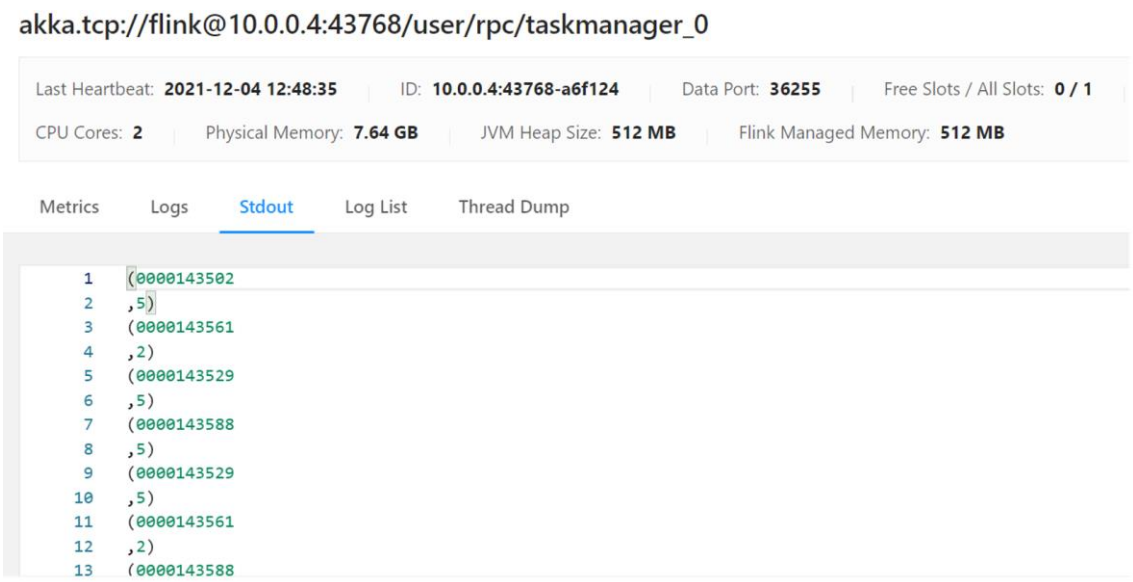
Los parámetros para ejecutar Flink son el Broker de Kafka ubicado en el puerto 9092 de nuestra VM, y el tópico.

La figura siguiente muestra los datos procesados en las primeras 6 ventanas. Se muestra la tupla (productID, rating) calculada manualmente para cada ventana. Los datos son generados en el MQTT Mosquitto cada un segundo.

productID	rating	window 1	window 2	window 3	window 4	window 5	window 6
143502	5	143502, 5	143502, 5	143502, 5	143502, 5	143502, 5	143502, 5
143529	5	143529, 5	143529, 5	143529, 5	143529, 5	143529, 5	143529, 5
143561	2	14561, 2	14561, 2	14561, 2	14561, 2	14561, 2	14561, 2
143588	5	143588, 5	143588, 5	143588, 5	143588, 5	143588, 5	143588, 5
143588	5		589012, 3.8	589012, 4	589012, 4	589012, 4.25	589012, 4.12
589012	5						
589012	2						
589012	5						
589012	2						
589012	5						
589012	5						
589012	5						
589012	5						
589012	5						
589012	1						
589012	4						
589012	5						
589012	1						
589012	5						
589012	5						
589012	5						
589012	5						
589012	5						
589012	5						
589012	5						
589012	2						
589012	1						
589012	5						
589012	5						

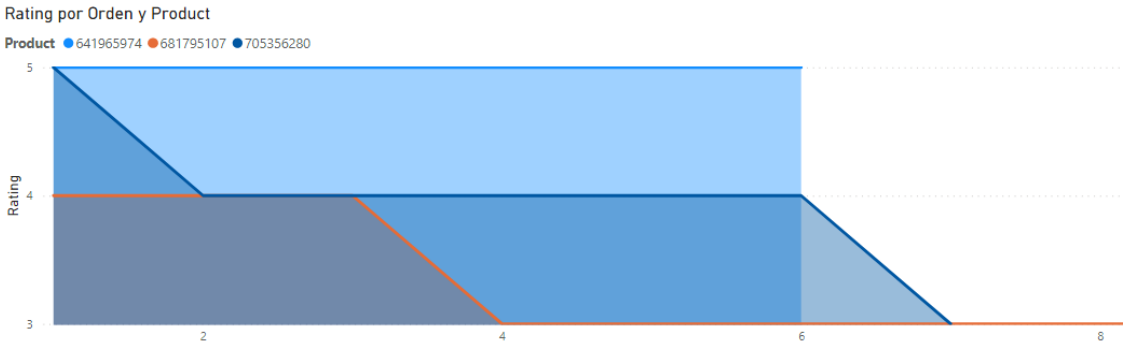
Usando la interfaz web de Flink se puede ver el log completo de salida del proceso (como se muestra en la figura siguiente), pudiéndose visualizar el promedio de cada producto para las distintas ventanas. Se comprueba que está funcionando bien, aunque por el tipo de datos de rating que es *integer,* trunca el resultado y no se obtiene el dato con mucha precisión. Esto fue una de las limitaciones que nos encontramos dado que no pudimos adaptar la aplicación de Flink para nuestro caso (por ejemplo tomando el dato de rating como tipo flotante).

Se adjunta como anexo la validación de los resultados para los primeros datos de la categoría *Movies and TV (*validación_ventanas.xlsx)



7.6 Visualización resultados

Si bien no se implementó una visualización en tiempo real, con la finalidad de mostrar gráficamente la salida de la aplicación Flink, se realizó un dashboard en Power BI, donde se muestra para 3 productos seleccionados de la categoría *Home and Kitchen*, la evolución del rating promedio de dichos productos a lo largo del tiempo. Por ejemplo, para el producto número 705356280 se puede apreciar que comienza el stream con un rating promedio de 5 para luego bajar a 4 y finalmente terminar con rating de 3.



También se muestra un dashboard obtenido con Tableau, donde se pude ver la distribución de algunos productos de acuerdo al recuento del promedio de las calificaciones para la categoría *Movies and TV*.

8 Conclusiones

Se logró implementar satisfactoriamente la arquitectura de referencia vista en clase para IoT en un ambiente *cloud* utilizando una VM creada en Microsoft Azure, para el caso de uso planteado.

Se utilizó un *dataset* público correspondiente a calificaciones de usuarios de Amazon, los cuales fueron transmitidos cada un segundo a través de un reproductor implementado en un script en lenguaje Python. La ingesta de dichos datos fue realizada por un Broker Mosquitto el cual maneja el protocolo MQTT, en un esquema publish/subscribe. Luego, se utilizó Apache Nifi donde a través de su interfaz gráfica se definió un flujo para consumir los datos de Mosquitto, convertirlos de formato JSON a CSV, y publicarlos en Kafka.

La aplicación Flink tomó los datos de Kafka y los procesó calculando el Rating promedio por artículo con una ventana de 30 segundos y cálculos parciales cada 5 segundos.

Por último, se comprobó que los resultados generados por Flink tuvieran correlación con los enviados por el productor y las ventanas definidas (*size window & slide window*). Los mismos fueron graficados con Power BI y Tableau. Como anexo se incluye una tabla de validación de los primeros datos (mostrando por un lado el csv original consumido, y el log de salida de Flink), y los dashboards obtenidos en Power BI y Tableau.

Otro de los resultados de destaque fue el hecho de poder diseñar e implementar un pipeline continuo para el stream de datos integrando varias herramientas del ecosistema Apache.

9 Recomendaciones

Como ya fue explicado anteriormente, varias de las alternativas y potencialidades de esta aplicación, no fueron incluidas en el alcance del proyecto.

Sin embargo y a efectos de posibles trabajos, los pasos sugeridos de modo de poder continuar con el desarrollo del proyecto y potenciar su aplicación son:

- Ingestar a través de Mosquitto datos generados en tiempo real a través de una API por ejemplo o en una aplicación IoT real a través del protocolo MQTT.
- Probar la performance de las diferentes herramientas y del sistema global en un cluster con varios nodos *workers.*
- Modificación de las aplicaciones de Java en Flink: de modo de poder variar las ventanas deslizantes para el procesamiento de los datos y poder realizar otras operaciones además del promedio de las calificaciones.
- Ingestar el stream de datos de salida, a Elastic Search y ensayar diferentes visualizaciones y dashboards en tiempo real a través de Kibana.
- Implementar una arquitectura lambda agregando una rama de procesamiento batch en Kafka y entrenar un algoritmo de Machine Learning a los efectos de realizar alguna predicción mediante MLIB de Spark Streaming o en el propio Flink con la librería FlinkML.

10 Referencias

[1] https://nijianmo.github.io/amazon/index.html

[2] https://mosquitto.org/

[3]https://nightlies.apache.org/flink/flink-docs-master/docs/dev/datastream/operators/windows/

[4] https://kafka.apache.org/intro

11 Anexos

A11-1: Jupyter Notebook con pre-procesamiento de los CSV de ratings originales *(preprocesamiento.ipynb)*

A11-2: Script Python con el generador del stream y Broker Mosquitto *(reproductor_ratings.py)*

A11-3: Template de NiFi con el Process Group que incluye el flujo implementado (*Amazon_NiFi.xml)*

A11-4: Log de salida de Flink para la categoría *Movies and TV (taskmanager_10.0.0).*

A11-5: Excel con validación de los resultados para los primeros datos del CSV *(validación ventanas.xlsx)*

A11-6: Dashboard en Power BI para los resultados de la categoría *Home and Kitchen (visualizacion Home&Kitchen.pbix)*

A11-7: Dashboard en Tableau para los resultados de la categoría *Movies and TV (dashboard_Tableau_1.pdf y dashboard_Tableau_2.pdf).*