



**Recomendador Musical**

**MEMORIA**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Proyecto:** | Proyecto Fin de Master | | |  |  |  |
| **Autores:** | Carlos López Pita  Jesús de Lucas Rodríguez  Javier Bermejo Izquierdo | | |  | **Código:** | RecomendadorMaster |
| **Versión:** | 1 |  |  |  | **Código interno:** | N/A |
| **Verificado:** | Carlos López Pita  Jesús de Lucas Rodríguez  Javier Bermejo Izquierdo | | |  | **Autorizado**: | N/A |
| **Modificación:** | 20/03/2017 | | |  |  |  |
|  | | | |  |  | |
| C/ Manuel Tovar 49-53  28034 Madrid  Tel: +34 917 680 490 / Fax: +34 913 833 301 | | | | | | |
| Reservados todos los derechos.  © Pragsis - Bidoop, 2015. | | | | | | |

# Control de versiones

|  |  |  |  |
| --- | --- | --- | --- |
| Versión | Fecha | Autor | Descripción |
| Versión 1 | 20/03/2017 | Carlos López Pita  Javier Bermejo Izquierdo | Versión inicial del documento |

|  |  |  |  |
| --- | --- | --- | --- |
| Versión 2 | 22/03/2017 | Jesús de Lucas Rodríguez | Versión 2 del documento |

|  |  |  |  |
| --- | --- | --- | --- |
| Versión 3 | 22/03/2017 | Carlos López Pita  Javier Bermejo Izquierdo | Versión inicial del documento |

# Contenido

Control de versiones 2

Descripción del documento 6

Requisitos 7

Esquema general 8

Procesos en batch 9

Procesos en streaming 15

Posibles mejoras 18

Conclusiones 19

# Descripción del documento

Este documento es la memoria del ejercicio práctico propuesto para el fin de master de Big Data organizado por Pragsis.

El ejercicio consiste en el diseño, implementación e implantación de un sistema de recomendación musical y envío de publicidad para una plataforma de música online.

# Requisitos

Se partirá del volumen actual de usuarios que tiene la compañía en una base de datos Oracle y de un sistema de eventos generados por los usuarios en el momento en el que reproducen una canción.

A partir de estos datos nuestro trabajo consiste en:

* Recibir los eventos del sistema de mensajería proporcionado por la empresa, en nuestro caso, a través de un servidor de eventos JMS.
* Recomendar a cada usuario un grupo cada vez que escuche una canción a través de un sistema de recomendación basado en las preferencias que han generado otros usuarios al escuchar sus canciones.
* Envío de publicidad a través de un banner cuando el usuario escuche X canciones, según unos parámetros especificados por el cliente. En este caso, según el género y la edad del usuario.
* Registrar un histórico de todas las canciones escuchadas por los usuarios durante 5 años.
* El sistema tiene que estar protegido contra fallos cuando lleguen los eventos y debe ser escalable tanto para nuevos usuarios como para nuevos eventos generados.

# Esquema general

La solución aportada tras la presentación con el cliente va a constar de dos grandes procesos en los que se enfocarán los requerimientos exigidos por la empresa para el sistema de recomendación.

* Procesos Batch:

1. Volcado de los usuarios actuales y futuros de la plataforma de música a partir de la base de datos Oracle a HDFS.
2. Calcular el modelo para que pueda ser utilizado al hacer las recomendaciones.
3. Crear y actualizar las tablas HBase necesarias (clientes, artistas y banners).
4. Crear y actualizar las tablas Hive necesarias para los procesos en streaming (para el entreno del modelo).

* Procesos en Streaming:

1. Recepción y gestión de los eventos (reproducciones).
2. Almacenamiento en HDFS de los mismos sin perder ninguno.
3. Generar la recomendación para cada usuario.
4. Generar también el banner de publicidad cuando cada usuario escuche X canciones.

Tanto los usuarios como los eventos serán almacenados en HDFS en el directorio /user/cloudera/mrec/, donde dentro del mismo estarán los directorios users, events y banners.

# Procesos en batch

## Almacenamiento de usuarios

Para la importación de los usuarios desde la base de datos relacional, vamos a utilizar Sqoop.

Primeramente, vamos a importar el histórico de usuarios que tiene el cliente en una base de datos Oracle a HDFS, para ello se ha creado el directorio /user/cloudera/mrec/users/ donde se irán almacenando los usuarios en directorios organizados por fecha.

Los usuarios van a estar ordenados por su ID, para ello vamos a transformar el String que nos viene por un Integer, eliminando la subcadena “user\_”. Con este ID vamos a poder gestionar las nuevas altas de usuarios gracias al --incremental append de Sqoop que nos permitirá ir guardando en HDFS solo los nuevos usuarios.

El job de Sqoop será lanzado con Oozie cada noche para guardar los nuevos usuarios.

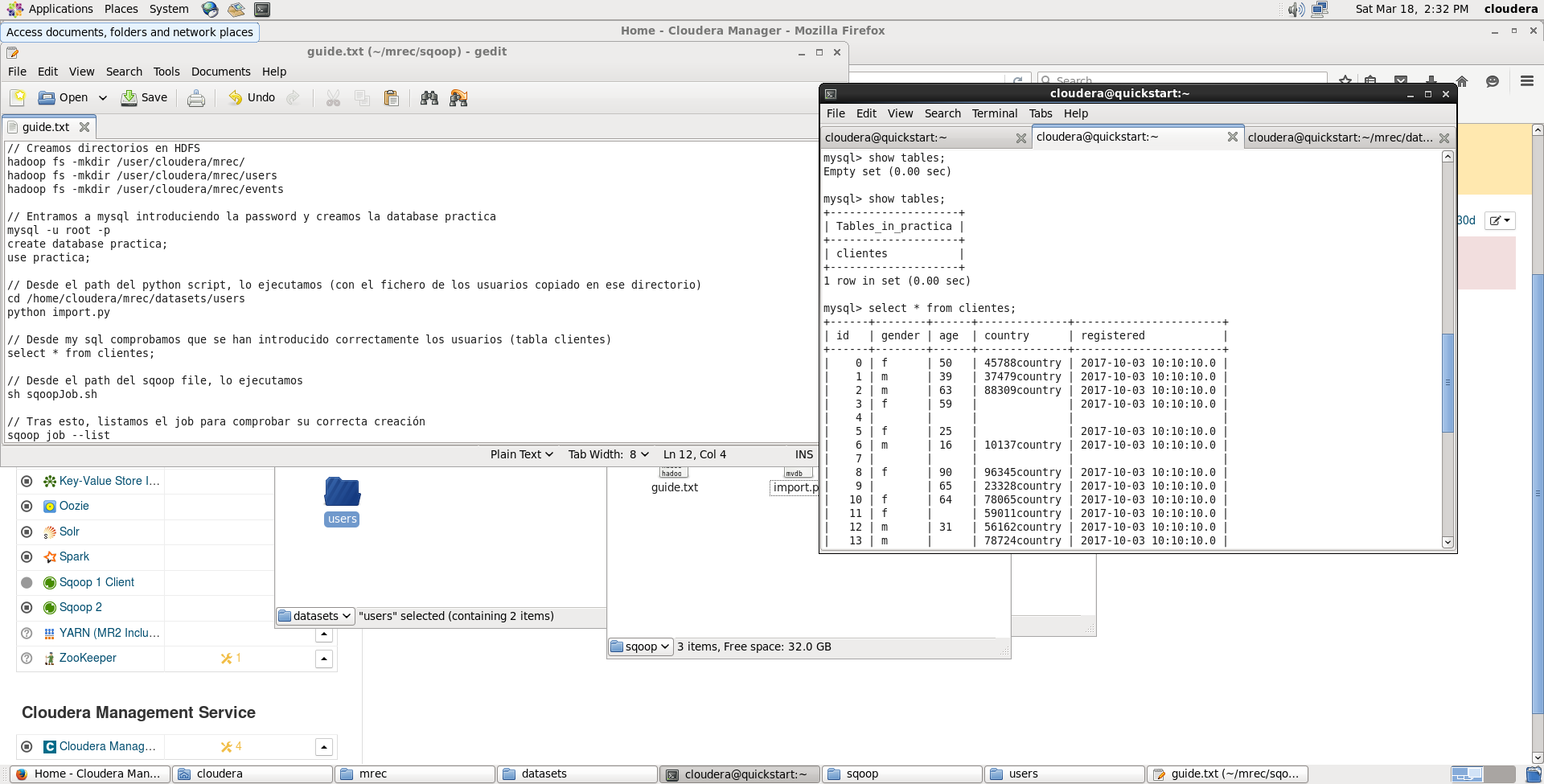


Tabla 1.1 - Visualización de los usuarios en Mysql

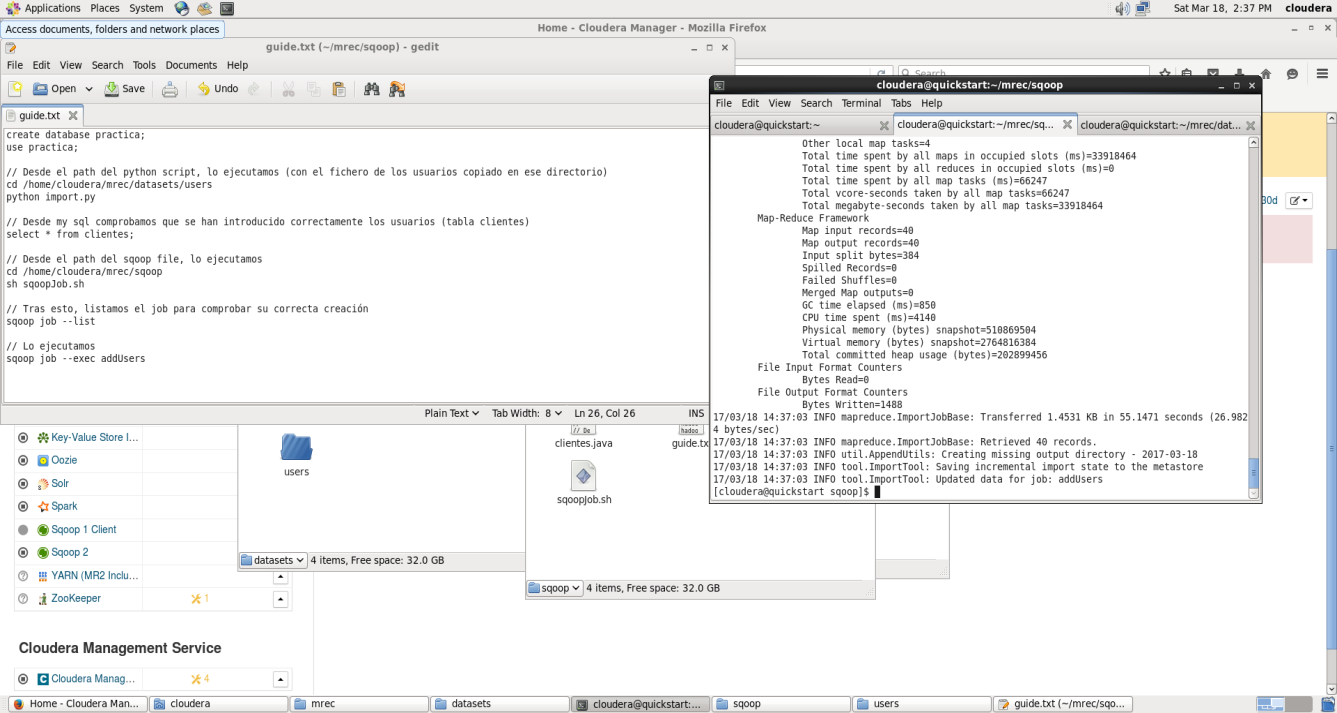


Tabla 1.2 - Volcado de los usuarios a HDFS usando Sqoop

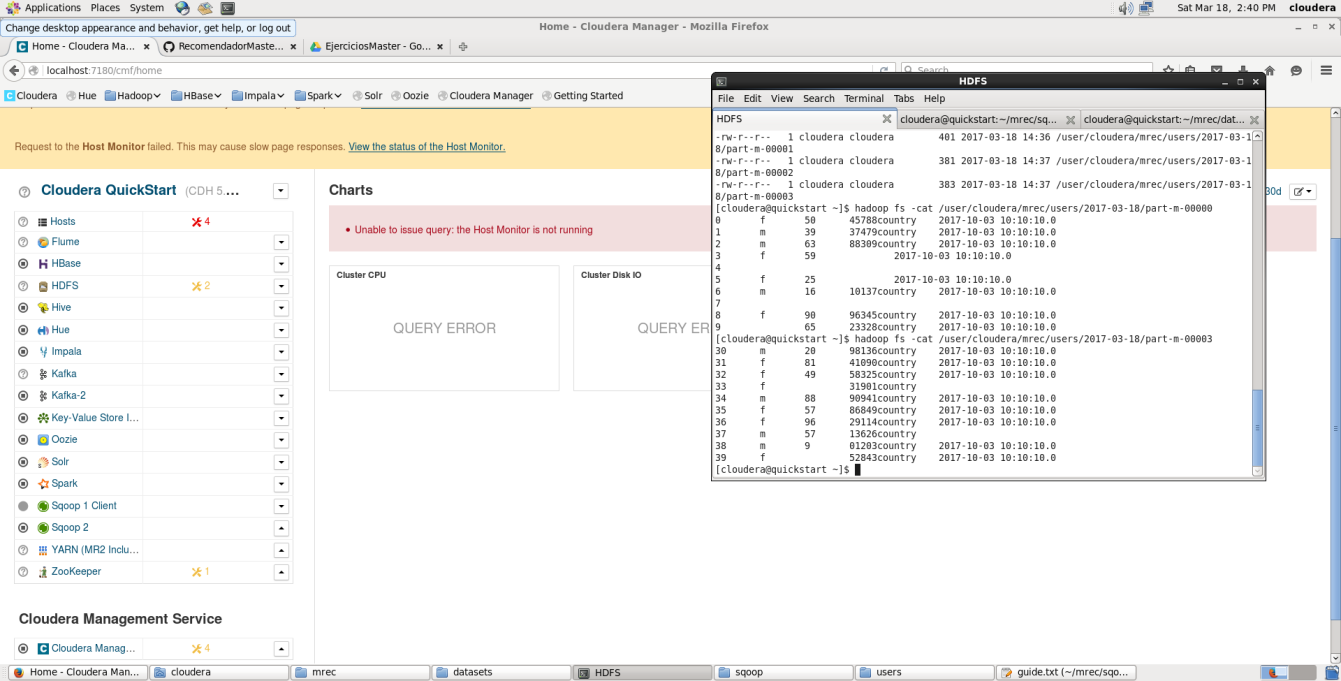


Tabla 1.3 - Listado de los usuarios en HDFS

## Creación y población de tablas HIVE para los procesos.

Vamos a crear una serie de tablas en HIVE para poder usar en el proyecto. Para ello vamos a utilizar como clave el userID y estarán particionadas por la fecha.

Las tablas son las siguientes:

* clientes: tendrá toda la información de los usuarios de la plataforma.
* eventos: tendrá la información relativa de los eventos que han llegado al sistema.
* numrepuserartist: número de reproducciones por usuario.
* artidWithNum: tabla que establece correspondencia entre el ID de artista con un ID integer y su nombre.
* ratingsperartist: ratings por cada tupla usuario y artista.
* ratingsWithArtist: ratings con el nombre del artista para hacer la recomendación.

La tabla de clientes la obtenemos de la importación de Sqoop y Kafka irá en streaming alimentando la tabla de eventos. Las tablas de clientes y eventos son externas, ya que queremos mantener los datos aunque estas se borren. El resto de tablas son datos que dependen de estas dos tablas. Para crear las tablas ejecutamos el fichero create\_tables con Hive. Luego añadimos location y las particiones con el fichero add\_partitions con Hive. Y creamos las tablas de ratings con el fichero create\_ratings con Hive también.

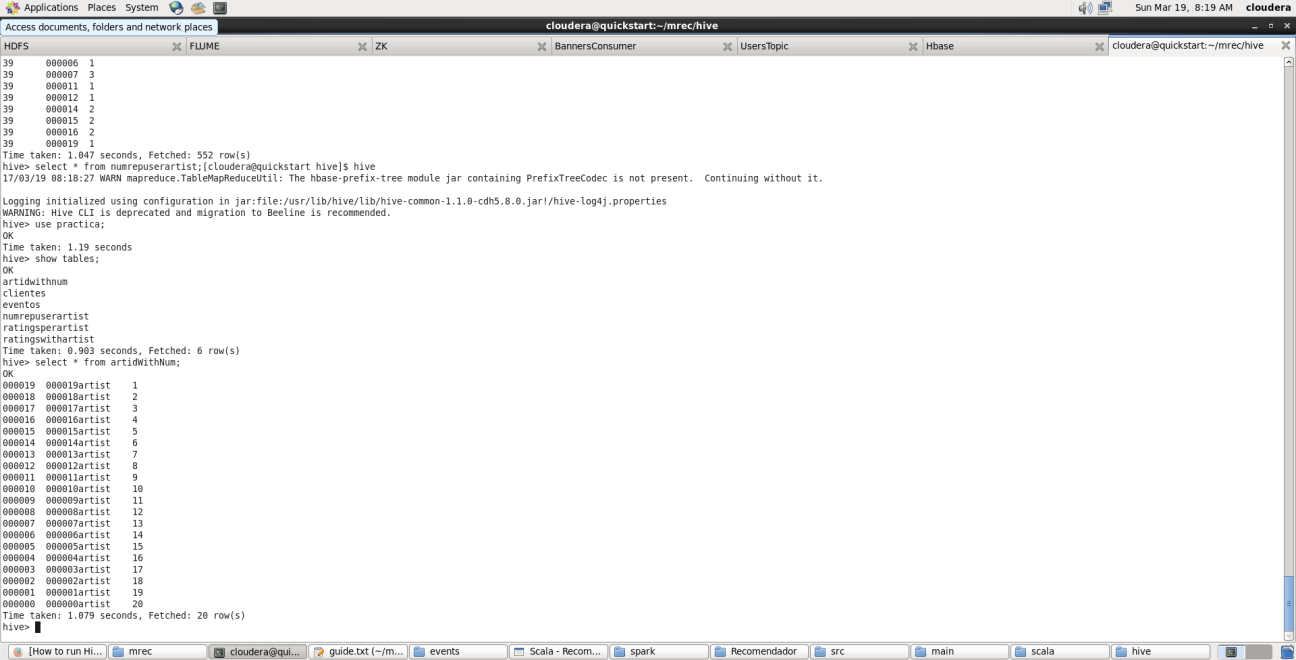


Tabla 2 – Tablas de Hive

## Banners de publicidad.

La gestión de los banners consiste en que el cliente nos proporcionará la información sobre qué tipos de banners quiere que se les muestre a los usuarios según su edad y sexo cada X reproducciones. A partir de esa información se generará una tabla que se almacenará en HDFS en el directorio /user/cloudera/mrec/banners/. Esta tabla podrá ser modificada según las necesidades del cliente en cada momento. Actualmente, la tabla tiene los campos ID (donde se contiene la edad y el género) y value (un String que sería la información del banner, el output para el usuario).

Leemos los banners de HDFS y los cargamos en tablas de HBase. Para ello vamos a utilizar Spark. Los banners son introducidos a HDFS y posteriormente a HBase con las aplicaciones Spark GenerarBannerHDFS y CargaBannerHBase, respectivamente.

Después cargaremos los clientes de HDFS a HBase ejecutando la aplicación Spark CargaClientesHBase.

Con los datos de la última tabla, dejaremos en streaming el banner en un consumidor de Kafka (bannersTopic) cuando corresponda.

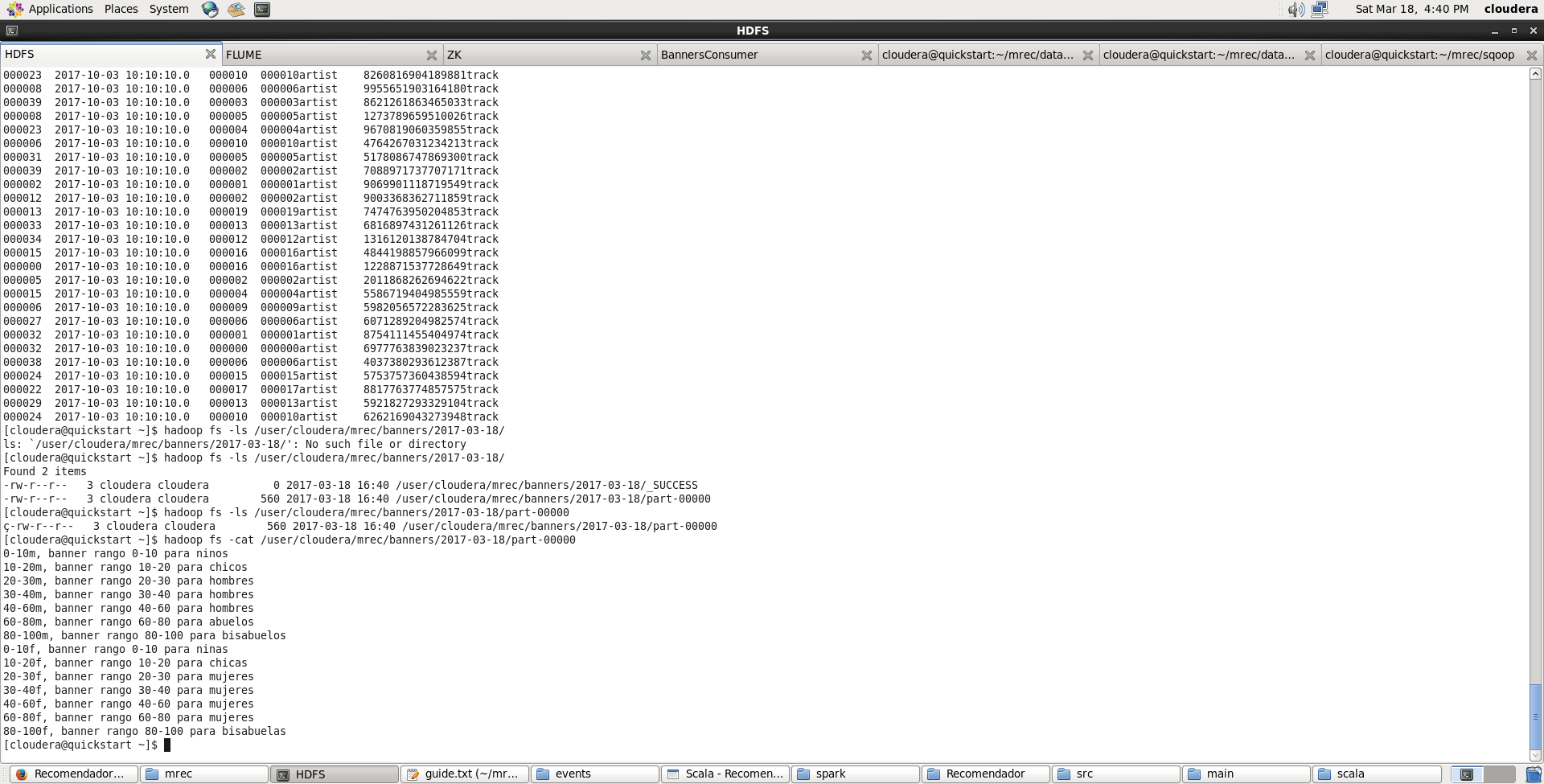


Tabla 3.1 – Directorio HDFS con banners

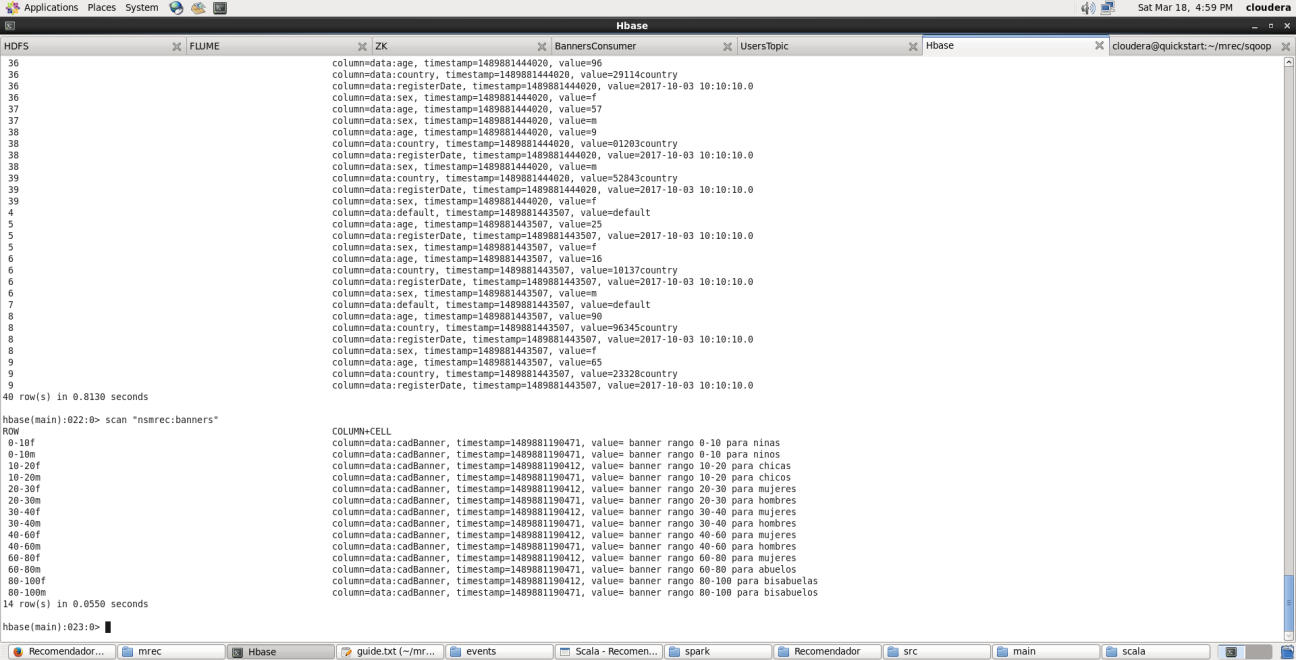


Tabla 3.2 – Importación de los banners a HBase

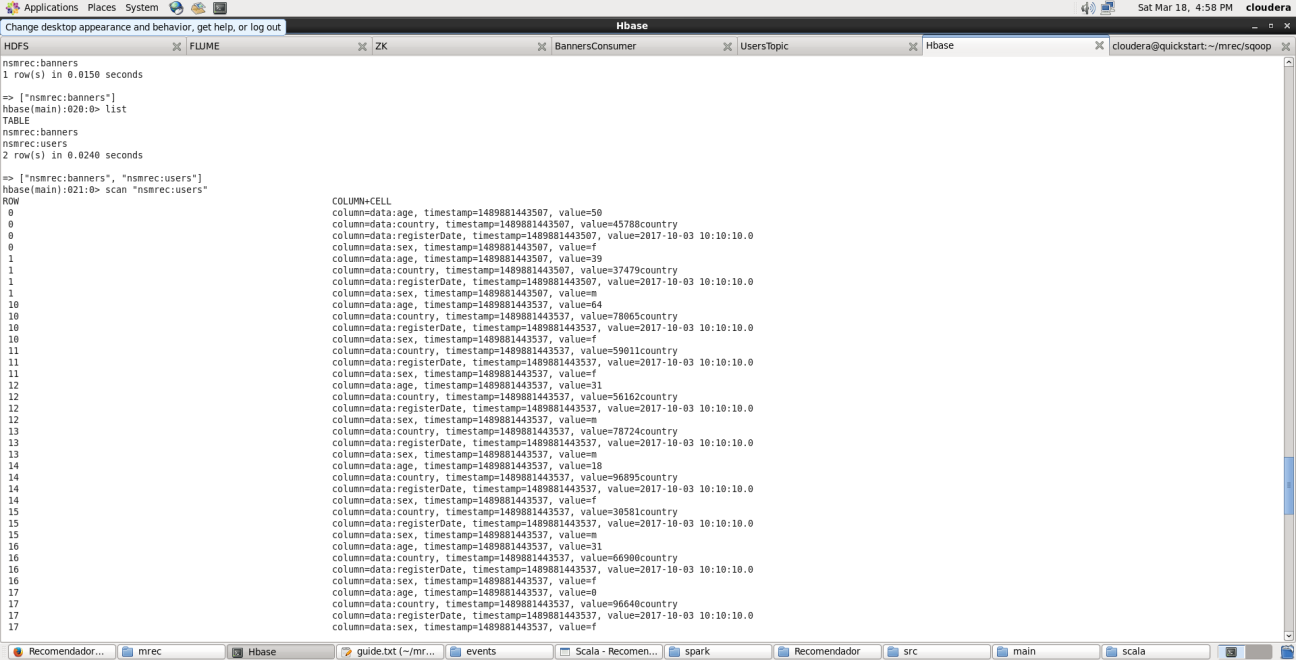


Tabla 3.3 – Importación de los usuarios a HBase

## Recomendador musical (Batch)

Para el recomendador musical vamos a utilizar el algoritmo de Machine Learning ALS (filtrado colaborativo) con el que vamos a generar un modelo usando la información histórica de eventos que nos han proporcionado y con el cual podremos hacer predicciones en streaming de los artistas que pueden gustar a los usuarios.

Antes de entrenar el modelo, lo primero que haremos es introducir en HDFS un fichero con la correspondencia de cada id de artista con su nombre de artista y un id integer (como nos pasaba también con el id de usuario) para poder procesarlo luego en ALS. Esto lo generará el programa Spark CargaDirecArtist partiendo de la tabla de Hive previamente creada.

Después con el programa Spark CargaArtistHBase cargaremos esta tabla en HBase. Esta tabla artistWithName será consultada en streaming para hacer las recomendaciones.

Con Spark ejecutamos la aplicación crearModelo en la que vamos a utilizar un contexto de Hive y hacer una consulta sobre la tabla ratingsWithArtist (creada previamente con Hive), que tiene los ratings por cada tupla usuario artista (ambos ids serían los integer procesados por nosotros y no los originales). Este modelo se generará en una ruta en local y se cargará en el proceso streaming para hacer recomendaciones.

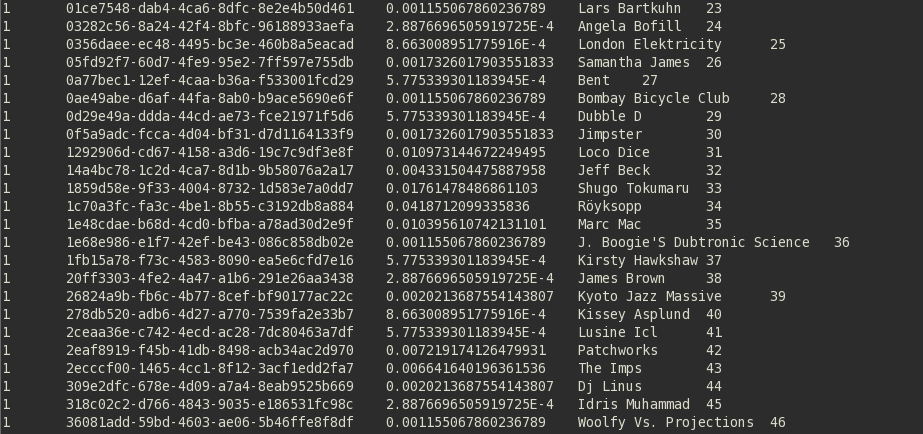


Tabla 4.1 - Tabla con los ratings que se usará para la creación del modelo

Vamos a realimentar la tabla ratingsWithArtist y ejecutar la aplicación crearModelo cada noche a través de Oozie para poder tener cada vez un modelo más preciso para realizar las recomendaciones. Este modelo se guardará en directorios guardados por fecha.

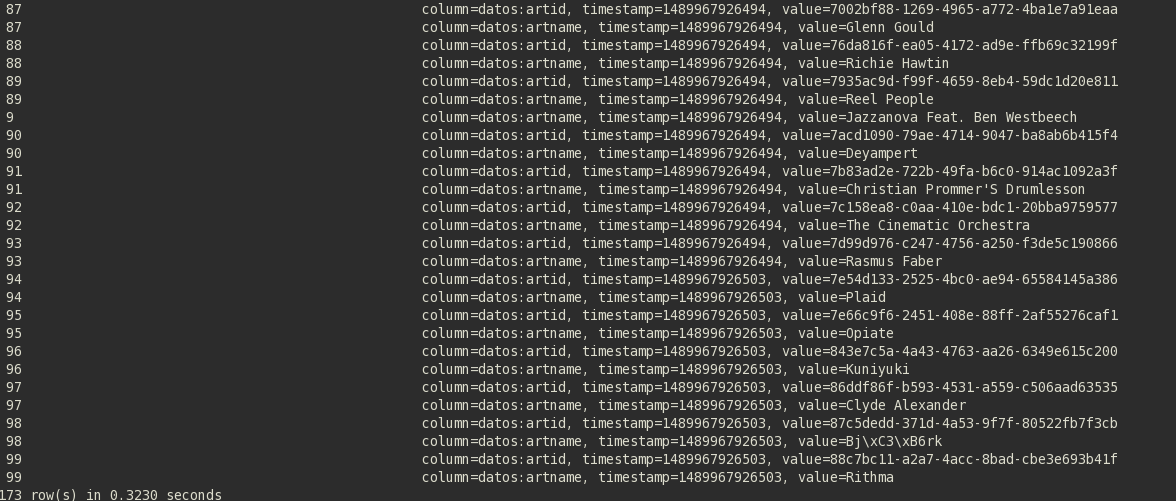


Tabla 4.2 - Tabla: artistWithName que contiene el artid y su nombre

# Procesos en streaming

## Recepción y gestión de los eventos.

Los eventos vendrán a través de un servidor JMS, pero para el proyecto utilizamos el fichero .tsv de eventos. Estos eventos contienen los siguientes campos en dicho fichero: userId, timestamp, artId, artName, trackName.

Lo primero que hacemos es crear un topic de Kafka (eventsTopic) que va a recibir y guardar los eventos y del que consumirán varios Kafka consumer (HDFS Sink y el recibidor de eventos).

Mediante Apache Flume nos creamos un agente, en el cual vamos a utilizar como source un Spooling Directory Source para leer del contenido del directorio el fichero .tsv con los eventos (en el caso del servidor JMS el source sería un JMS Source)

Como channel ponemos un Kafka Channel que es donde queremos ir almacenando los eventos. Este sería el topic de Kafka anteriormente creado. Actuaría como un Kafka producer.

Y como sink ponemos un HDFS Sink para escribir los eventos para su posterior utilización en un directorio HDFS. Actuaría como un Kafka consumer.

Una vez lanzado este agente Flume también lanzamos los consumidores de Kafka para los otros procesos de streaming, en nuestro caso el consumidor de banners (bannersTopic) para lanzar banners y el consumidor de usuarios (usersTopic) para lanzar las recomendaciones.

## Almacenamiento de eventos en HDFS.

Los eventos serán almacenados en HDFS según la fecha en la que lleguen en el directorio /user/cloudera/mrec/events/%Y-%m-%d. Son almacenados como DataStream y serán utilizados por los procesos batch para actualizar el modelo de recomendaciones y por el proceso streaming para lanzar los banners.

## Banners publicitarios.

Mediante Spark Streaming vamos a consultar los eventos según llegan al topic de eventos. Para ello ejecutamos la aplicación RecibidorEventos que contiene tanto un Spark context, un Spark Streaming context, un HBase context y un productor Kafka para devolver el banner correspondiente al usuario del evento en un Kafka consumer creado (bannersTopic).

Esta aplicación hará uso de las tablas HBase de banners y clientes para comprobar si existe el usuario en el sistema y devolver el banner indicado para el usuario.

Se creará una nueva tabla HBase con los datos de las reproducciones de cada usuario y el tipo de banner que debe recibir dicho usuario. Cuando llegue el primer evento de un usuario, se comprueba si la edad y el sexo del cliente están disponibles, y si es así, se introduce el banner correspondiente para esos datos en una nueva tabla de HBase. En el caso de que la edad o el sexo del cliente no estén disponibles o el usuario no esté importado, se introduce para ese id de usuario el banner genérico. Además, se iniciaría el número de reproducciones a 0 de ese usuario.

Los siguientes eventos incrementarán en uno las reproducciones del usuario de dicho evento.

Además, el banner generado será devuelto cada X reproducciones (consultándolo del calculado en el primer evento de cada usuario) por usuario al consumidor Kafka. En el caso de que cuando se reciba el banner sea el número indicado de reproducciones para lanzar el banner, entonces se introducirá el banner en el consumidor de Kafka bannersTopic y se reiniciará el contador de reproducciones.

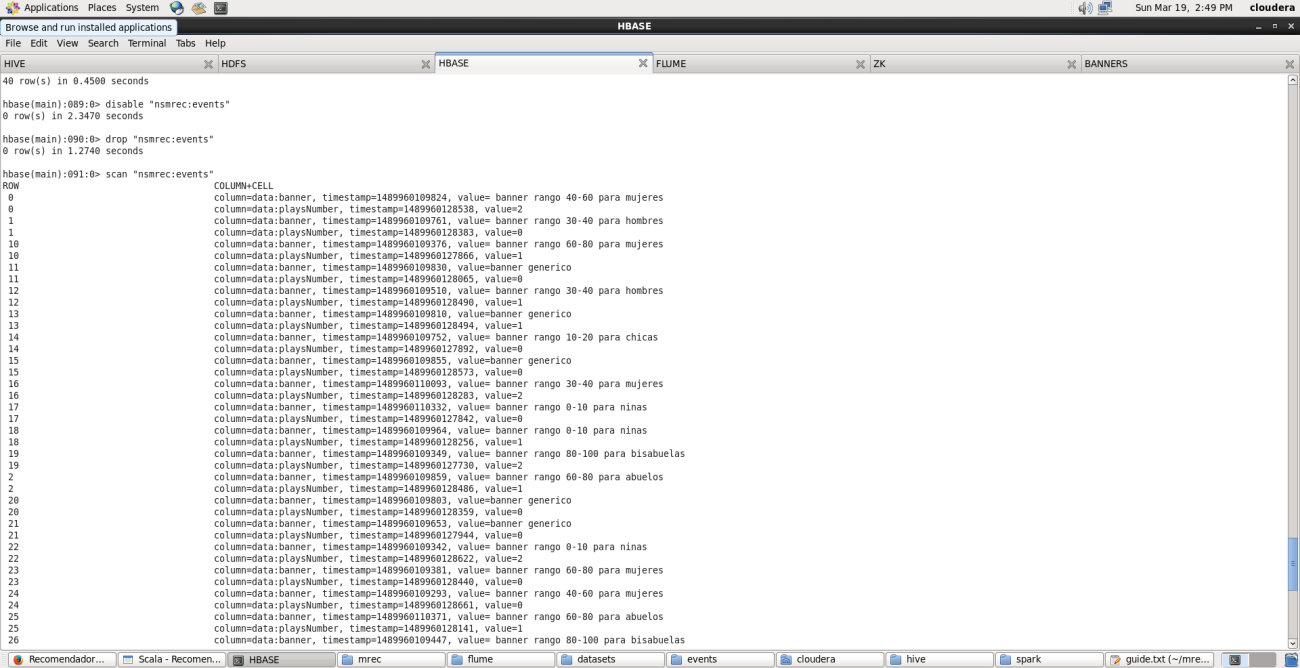


Tabla 5 – Tabla HBase con el número de reproducciones y el banner por usuario.

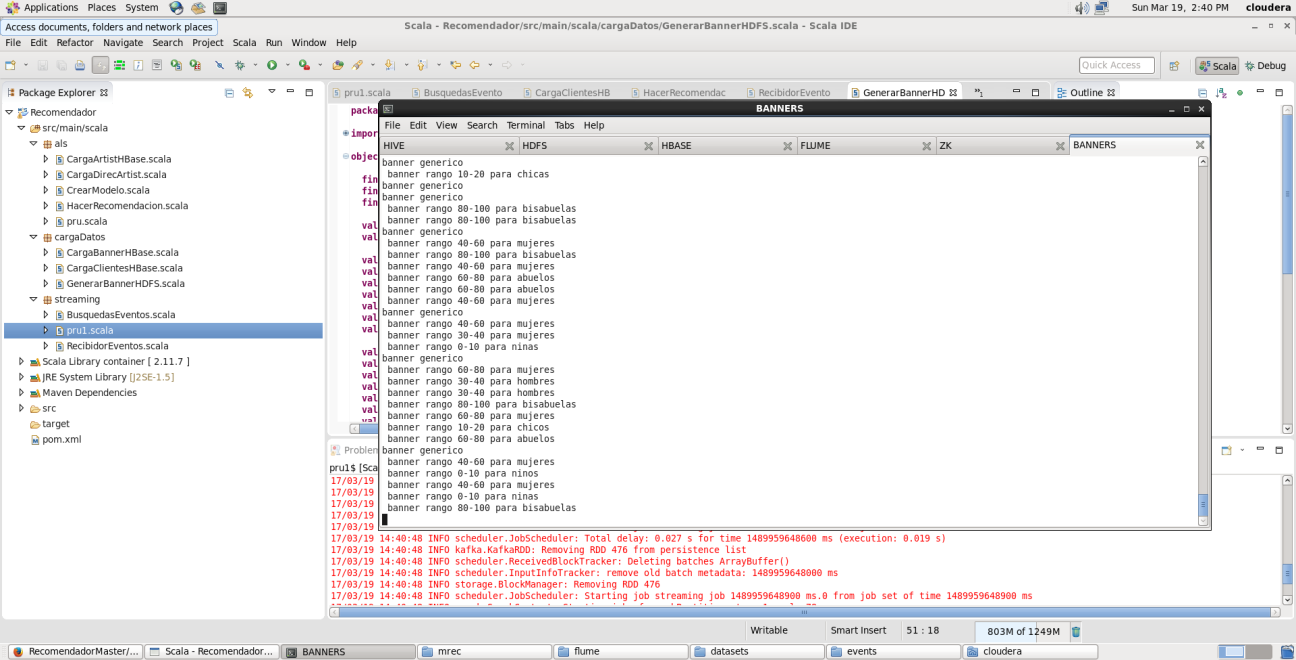


Tabla 6 – Consumer Kafka de banners

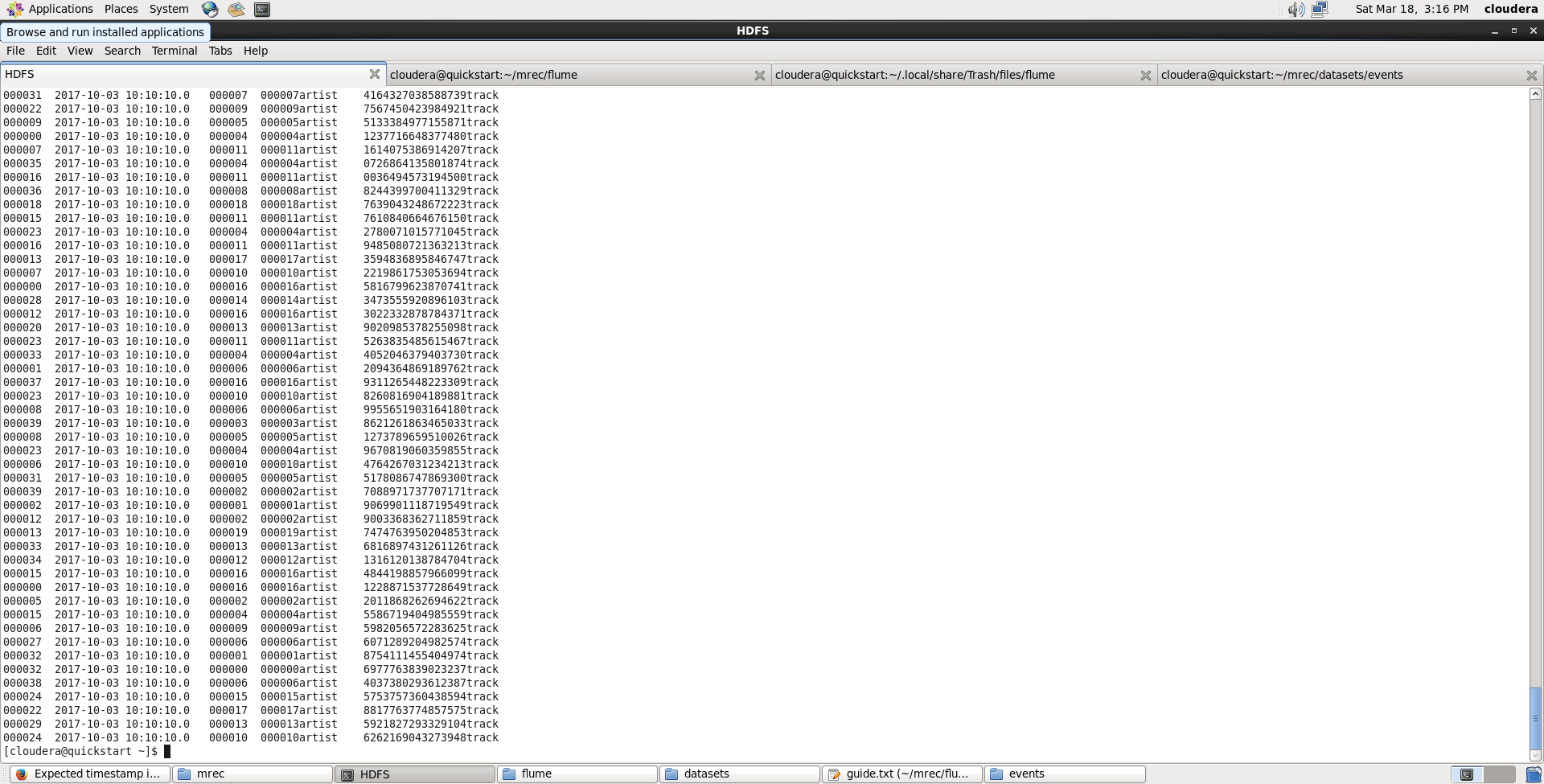


Tabla 7 – Eventos en streaming

## Generación de recomendaciones.

Mediante Spark Streaming vamos a consultar los eventos que llegan a través del topic de eventos. Para ello ejecutamos la aplicación HacerRecomendacion (que contiene un Spark context, un Spark Streaming context, un HBase context y un productor Kafka) para devolver la recomendación correspondiente al usuario del evento.

Lo primero que hacemos es cargar el modelo calculado mediante su proceso batch desde la ruta donde lo guardamos.

Por cada evento que llega se recupera el id del usuario y se genera un Dataframe para pasarle al modelo (transform) que contendrá el id de ese cliente con todos los ids de los artistas que hay en el sistema y un rating para cada uno de estos artistas. De ahí elegiremos las predicciones con ratings más altos para hacer las recomendaciones.

Esto será lo que se devolverá al usuario por el consumidor de Kafka usersTopic.

### 

Tabla 8 – Eventos de recomendación generados para usuarios (En este caso el mismo cliente)

# Workflow

Generaríamos un workflow para lanzar con Oozie que cada noche haría las siguientes acciones en paralelo:

* Importar usuarios usando Sqoop.
* Entrenar modelo y guardarlo en un directorio por fecha.

# Posibles mejoras

Se debería haber creado una tabla que calculara el banner por cada usuario (en batch) después de importar los banners y los usuarios a HDFS. Esta tabla podía haberse generado con dos campos userId y bannerId.

La tabla bannerId tiene un id (que está modelado teniendo en cuenta género y edad) del que se extrae información analizándolo y un valor que saca el mensaje. Se podía haber modelado con los campos bannerId, minAge, maxAge, gender y info, por ejemplo (más abstracción y mantenibilidad).

El modelo se podía haber guardado en HDFS y tener siempre dos modelos guardados en diferentes directorios. Con un flag, podríamos activar el modelo más recientemente insertado. Nunca nos quedaríamos sin modelo con este sistema.

# Conclusiones

Este proyecto nos ha ayudado a comprender mejor la teoría que hemos visto y a aplicarla en un problema de la actualidad.

Hoy en día hay muchas empresas tienen necesidades de este tipo, en las que hay que aplicar herramientas Big Data y técnicas de Machine Learning para desarrollar aplicaciones que ayuden en la toma de decisiones de negocio.

Hemos aprendido a manejarnos con muchas de las herramientas del ecosistema Hadoop: Flume, Kafka, Oozie, Sqoop, Hive, HBase, Spark, etc. Tanto a nivel de desarrollo como de administración. Por lo que consideramos esta práctica ha sido muy enriquecedora.