**PRÁCTICA FINAL**

**BIG DATA PROCESSING**

# **Índice de Contenidos**

1. Índice de Contenidos · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · 1
2. Datos de la Práctica · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · 2
3. Contexto · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · 2
4. Parametrización Básica y Objetivos · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · 3
5. Diagramas · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · 4
6. Descripción General del Sistema · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · 5
7. Proyecto · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · 6
   1. Fichero **build.sbt** - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - 6
   2. Ficheros de Datos de Prueba - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - 7
   3. Estructura del Proyecto - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - 7
   4. Proyecto Completo Exportado - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - 8
8. Proceso Consumidor de Topic · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · 8
   1. Fuentes de Alimentación - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - 8
   2. Script - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - 8
   3. Testeo - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - 9
9. Proceso de Tratamiento

de los Mensajes Capturados · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · 10

* 1. Fuentes de Alimentación - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - 10
  2. Script - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - 11
  3. Testeo - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · - 11

1. Agradecimientos y Comentarios Finales · - ·· - · - ·· - · - ·· - · - ·· - · - ·· - · 12

# **Datos de la Práctica**

|  |  |
| --- | --- |
| **Bootcamp IV Big Data & Machine Learning - KeepCoding** | |
| **Asignatura** | Big Data Processing |
| **Profesor** | Ricardo Jaume |
| **Alumno** | Francisco Javier Gonzálvez Chico |
| **Título** | Espías en Cloacalandia |

# **Contexto**

**Siglo XXIII**. En secreto, instituciones ocultas del gobierno de **Cloacalandia** espían desde hace tiempo a los ciudadanos de las grandes ciudades del país.

Estos ciudadanos utilizan una red social llamada **Celebram** en la que publican mensajes para sus conocidos y familiares. Los mensajes son cifrados por la red social, pero esto no es problema para los hackers del departamento del ministerio ya que han diseñado un algoritmo que es capaz de descifrar todos los mensajes. Una vez se envían, son “esnifados” por dispositivos IoT ocultos y repartidos por diversas zonas de la ciudad, generando información de forma ininterrumpida (“streaming”). Todos los mensajes, al ser interceptados, son marcados por una huella temporal (timestamp), y por la zona desde donde han sido “interceptados”.



Hemos sido contratados por el Gobierno de Cloacalandia para crear un algoritmo de proceso en Big Data cuyo fin último sea hallar, por hora (ventana temporal), las 10 palabras más usadas en los mensajes recibidos en cada ventana. Una vez realizado el proceso de los mensajes de cada ventana, en caso de que la palabra más repetida en ellos coincida con alguna de las palabras de una Lista Negra (**The BlackList)**, el sistema habrá enviar una notificación al ministro avisando de tal situación.

# **Parametrización Básica y Objetivos**

Partiremos de la base de que…

* El sistema funciona 24/7
* Los dispositivos IoT pueden dejar de funcionar, bien por estropearse, por acabarse su batería o porque hayan sido apagados en remoto. Estos IoT no deberán contabilizar para la ingesta de datos.
* El sistema de desencriptado y el de notificación serán simulados.
* La lista negra existirá realmente.
* La palabra repetida no podrá ser una preposición, ni una conjunción, ni un artículo.
* Para simular el envío de datos por parte de los IoT’s, enchufaremos ficheros de texto para que los procese **Kafka**.

Los objetivos serán:

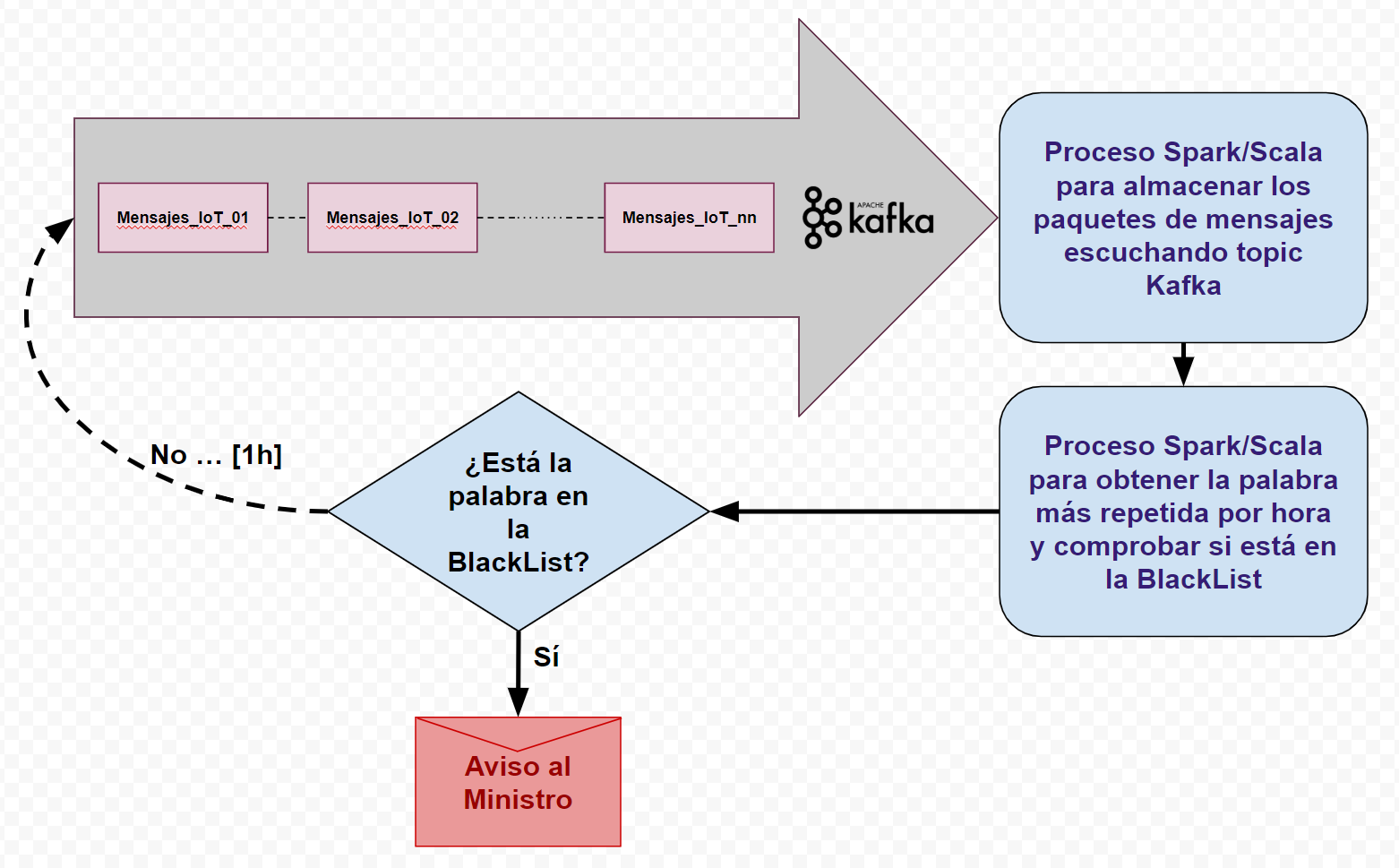
1. Crear los esquemas de los datasets necesarios.
2. Cargar los datasets con información (dummy).
3. Elaborar una fuente de información completa con la que trabajar y poder procesar en tiempo real.

Las fuentes de datos propuestas son:

|  |  |  |
| --- | --- | --- |
| **Mensajes\_Capturados** | Mensaje\_Id | *String* |
| Contenido | *String* |
| User\_Id | *String* |
| **Usuarios\_Registrados** | User\_Id | *String* |
| Nombre | *String* |
| Apellido | *String* |
| Edad | *Integer* |
| Sexo | *String* |
| **Dispositivos\_IoT** | IoT\_Id | *String* |
| Encendido | *Boolean* |
| Zona\_Id | *String* |
| **BlackList** | Palabra | *String* |

# **Diagramas**





# **Descripción General del Sistema**

Como se puede ver en el primer diagrama del párrafo anterior, nuestro sistema constará, en principio, de un número indeterminado de dispositivos IoT colocados uno en cada zona de nuestra ciudad-estado de Cloacalandia. Estos dispositivos estarán en funcionamiento indefinido 24/7 (mientras les dure su batería), salvo que se apaguen de forma remota o se estropeen.

El cometido de estos IoT no será otro que el de espiar la red social **Celebram** y capturar todos los mensajes que sus usuarios registrados envíen bajo su plataforma. Los mensajes capturados se irán guardando en los buffers de memoria de cada IoT con un identificador secuencial, un identificador de la zona que cubre el dispositivo y el timestamp de captura.

Cuando el buffer de un IoT se llene (pongamos por ejemplo que parametrizamos su capacidad para 25 mensajes), el dispositivo procederá a su descarga en forma de paquetes streaming sobre un Topic Kafka encriptado y controlado únicamente por la Agencia Federal de Inteligencia. Este proceso será continuo y todos los IoT irán volcando paulatinamente sus buffers sobre el mismo Topic.

La Agencia Federal de Inteligencia recibirá en modo “near real time” los paquetes de mensajes transmitidos por los IoT y serán desencriptados (por un proceso que no entra en nuestro scope), como se puede observar en el segundo diagrama del párrafo anterior.

Estos paquetes se irán guardando juntos en un fichero “almacén” (en formato parquet) y, una vez cada hora, otro proceso (que tampoco entra en nuestro scope) vaciará ese almacén (para que pueda seguir albergando los paquetes que se reciban durante la hora siguiente) y trasladará los mensajes, convertidos de nuevo a formato plano, que en él hubiera (de la hora pasada) para servir como fuente de alimentación del proceso batch que se ocupará de analizarlos.

El proceso batch de análisis se ocupará de descubrir cuales son las 10 palabras, con contenido semántico, más repetidas en los mensajes capturados durante la hora justo anterior y además, filtrará la más repetida de todas ellas contra la lista negra de la Agencia. En caso de que esa palabra más repetida esté entre las contenidas en The BlackList se enviará inmediatamente un mensaje de alerta al excelentísimo señor ministro.

El proceso se ha estructurado de esta forma porque, además de responder plenamente al enunciado, pensamos que realmente no tiene mucha utilidad procesar la información en “near real time”. Si debemos obtener las diez palabras más repetidas durante la ventana temporal de una hora, tendremos que esperar a que acabe la hora para poder saber cuáles son y cuál es la palabra “top” de todas ellas. Y para manejar grandes volúmenes de datos (como la gran cantidad de palabras que millones de usuarios pueden enviar durante una hora por una red social) nos parece más adecuado un proceso batch contra el cluster.

# **Proyecto**

Utilizaremos para desarrollar el proyecto el entorno IDE para Scala de **IDEA – IntelliJ** instalado en la máquina virtual **UBUNTU** proporcionada para el desarrollo de las clases.

Nuestro proyecto será del tipo Scala / sbt y lo llamaremos “**espiascloacalandia**” (evitando las mayúsculas en el nombre).

Elegiremos la versión 2.11.12 de Scala.

## Fichero **build.sbt**

Completaremos nuestro fichero build.sbt con las librerías del repositorio Maven para las siguientes dependencias:

* **Spark-Core**
* **Spark-SQL**
* **Spark-Streaming**
* **Spark-Streaming-Kafka\***
* **Spark-SQL-Kafka\***



*\*Nota: Como puede observarse, las librerías correspondientes a la aplicación Apache Kafka están comentadas en el fichero ya que no he podido trabajar con esta aplicación, no he conseguido siquiera arrancarla de forma correcta en la máquina virtual.*

## Ficheros de Datos

Se aportan a continuación los archivos csv utilizados para probar los scripts del proyecto, en formato csv.



## Estructura del Proyecto

Nuestro proyecto “**espiascloacalandia**” se estructurará con dos carpetas para albergar ficheros:

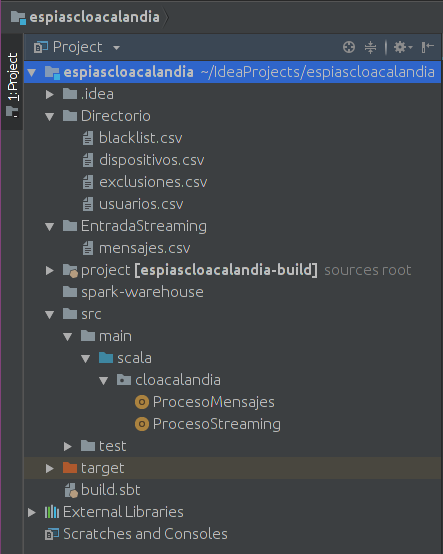
* **Directorio**

En el que estarán disponibles para el proceso los archivos csv con:

* + - La Blacklist
    - Los Dispositivos IoT
    - Los Usuarios Registrados en Celebram
    - Las palabras Excluidas en el Análisis (artículos, preposiciones y conjunciones)
* **EntradaStreaming**

En el que almacenaremos los ficheros recogidos del topic

También contendrá un Package “**cloacalandia**”, bajo ***/src/main/scala***, del que colgarán los dos procesos Spark/Scala referenciados en el segundo diagrama del apartado 5.



## Proyecto Completo Exportado

Aquí se adjunta el proyecto completo, tras los testeos de los procesos, en formato zip.



# **Proceso Consumidor de Topic**

Elegimos NetCat (ante la imposibilidad de trabajar con Kafka) como generador de paquetes streaming.

Con esta codificación conseguiremos recoger desde un topic de streaming ficheros de datos y almacenarlos en nuestro sistema a la espera de que en el siguiente proceso su contenido sea analizado.

El proceso se mantendrá ejecutado a la escucha de lo que llegue por el canal de streaming y cuando detecte información la incluirá en el almacén de salida.

## Fuentes de Alimentación

Tomaremos como fuentes de alimentación ficheros csv simulando el contenido de los buffers que los dispositivos IoT descargarían sobre el Topic. Estos ficheros contendrán 25 mensajes.

Aportamos como ejemplo 3, procedentes de los IoT de las zonas: Felgueiras, Goba y Jiamn de Cloacalandia.



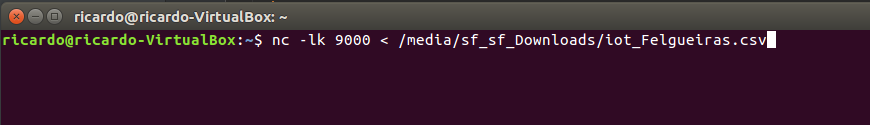
## Script

Se entrega a continuación el código de este script en un archivo .scala (que puede abrirse como texto plano). El código está completamente comentado para su mejor entendimiento.

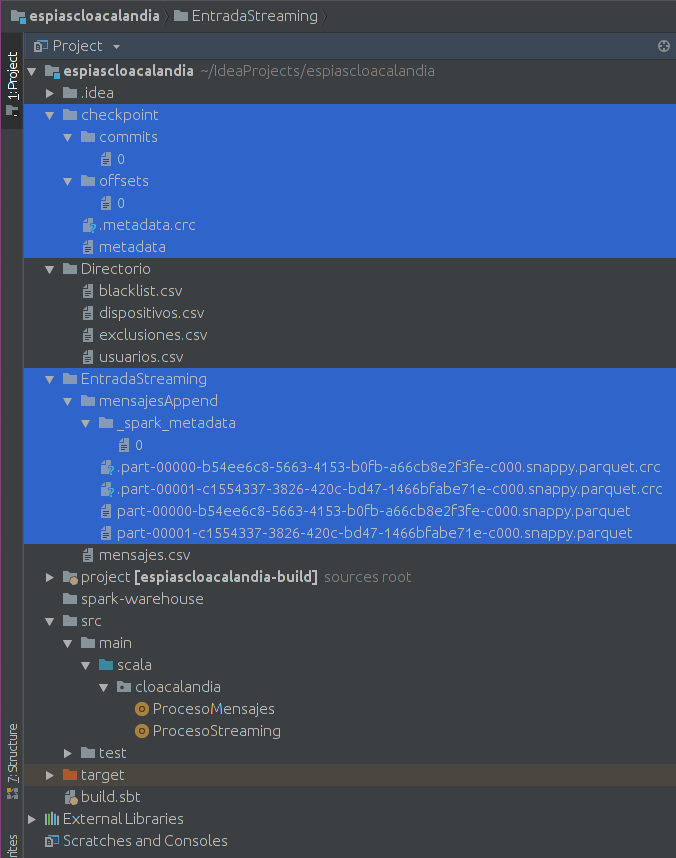


## Testeo

Para probar nuestro código abriremos primero un terminal en el que lanzaremos la aplicación NetCat a la que dirigiremos uno de los ficheros aportados en el párrafo 8.2.



Acto seguido, ejecutaremos nuestro código y la ventana de ejecución de IntelliJ debe quedarse latente (a la espera).

Pero en la estructura de nuestro proyecto aparecen dos elementos nuevos:

* La carpeta **checkpoint** y sus subcarpetas y archivos, donde se controlan posibles puntos de ruptura en la comunicación.
* La carpeta **mensajesAppend** y sus subcarpetas y archivos donde se almacenan en formato “parquet” los archivos recibidos por el canal de streaming.

La ejecución de testeo del script hay que pararla de forma forzada ya que, recordemos, es un proceso que se queda a la escucha 24/7.

# **Proceso de Tratamiento de los Mensajes Capturados**

Con esta codificación procederemos a cumplir “realmente” con los objetivos reseñados en el párrafo 4.

El proceso creará las estructuras para, tras leer los ficheros csv con los que trabajará, poder cargarlos en Datasets y así paralelizar su procesamiento utilizando el cluster.

Acto seguido cargará los datasets, leyendo los ficheros csv.

Una vez tiene sus fuentes de información completas, procede a su tratamiento para contar todas las palabras de los mensajes capturados, excluye de ese conjunto los artículos, preposiciones y conjunciones, muestrea las 10 palabras más repetidas, escoge la palabra que se repite más veces (o las palabras, si se diera la improbable casualidad de que hubiera más de una con el mismo número máximo de repeticiones), revisa si está en la BlackList y si ese fuera el caso, procede a escribir un mensaje de alerta, que asumimos a posteriori se enviará por algún otro sistema, en forma de e-mail al Ministro de Cloacalandia.

## Fuentes de Alimentación

Se tomarán, como fuentes de alimentación, los ficheros aportados en el párrafo 7.2.

Las estructuras de campos de los datasets que los albergarán son las siguientes:

***Fichero blacklist.csv***

|  |  |
| --- | --- |
| PalabraBL | *String* |

***Fichero dispositivos.csv***

|  |  |
| --- | --- |
| iotId | *String* |
| iotOnOff | *Boolean* |
| iotZona | *String* |

***Fichero exclusiones.csv***

|  |  |
| --- | --- |
| PalabraEx | *String* |

***Fichero usuarios.csv***

|  |  |
| --- | --- |
| userId | *String* |
| userNombre | *String* |
| userApellido | *String* |
| userEdad | *Integer* |
| userSexo | *String* |

***Fichero mensajes.csv***

|  |  |
| --- | --- |
| msgId | *String* |
| msgContenido | *String* |
| msgUserId | *String* |
| msgTimestamp | *String* |
| msgZona | *String* |

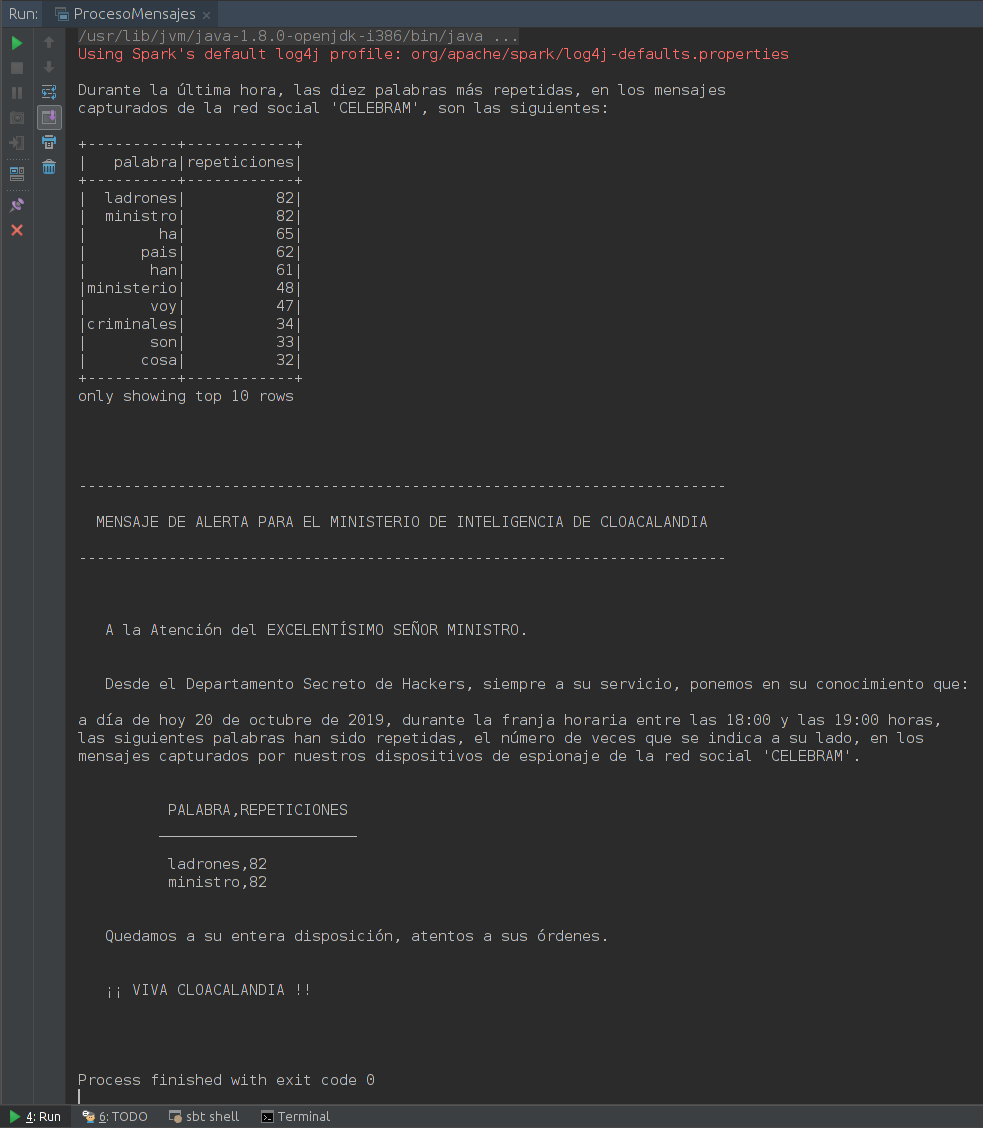
## Script

Se entrega a continuación el código de este script en un archivo .scala (que puede abrirse como texto plano). El código está completamente comentado para su mejor entendimiento.



## Testeo

En la imagen siguiente se puede apreciar el resultado de la ejecución de testeo del proceso utilizando los ficheros csv del apartado 7.2.



# **Agradecimientos y Comentarios Finales**

Como nota final de este trabajo me gustaría dar las gracias al profesor, **Ricardo Jaume**, por su trabajo, por su gran ayuda y por la atención a la hora de corregir nuestros errores y despejar nuestras dudas (en las clases y en Slack).

Reseñar de nuevo que, debido a la imposibilidad para trabajar con Apache Kafka (comentada y detallada en Slack con el profesor), he sustituido, muy a mi pesar, esta aplicación de streaming por NetCat. He de poner de manifiesto que el script del consumidor de streaming, con Kafka, hubiera sido totalmente análogo.

Comentar que la impartición de esta asignatura de Processing Big Data, a mi modo de ver, ha quedado muy corta, diría casi que demasiado para sólo seis sesiones. Scala es un lenguaje que, como bien se nos indicó, tiene una curva de aprendizaje bastante pronunciada, mayor que la de la media –entiendo que evidentemente por su gran potencia e integración nativa con Spark– y personalmente me hubiera encantado poder profundizar en él disponiendo de más tiempo para invertir en ejemplos y ejercicios. Realmente he quedado algo insatisfecho no poder descubrir, por ejemplo, las librerías MLlib y GraphX en vivo junto al profesor...

…cierto es que tenemos vídeos en off, pero la experiencia “en lata”, como se suele decir, no es comparable al directo.

Sin más, mi agradecimiento por las cosas que he aprendido, que espero se vean reflejadas en esta práctica.

**F. Javier Gonzálvez**