Universidad Autónoma de Madrid
Escuela Politécnica Superior

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

TDC (Twitter Data Collection): Creación de una gran base de datos de Tweets

Borja Gil Pérez

Tutor: Manuel García-Herranz del Olmo
Ponente: Germán Montoro Manrique

Julio 2014

Resumen

La motivación de este proyecto es crear una base de datos flexible para adquirir, almacenar, analizar y consultar tweets. Se debe llevar a cabo un trabajo de investigación sobre cuáles son las tecnologías usadas para este propósito y diseñar una arquitectura software acorde a ellas. Las investigaciones llevarán hacia el auge del Big Data, sistemas que manipulan grandes conjutos de datos. Una vez diseñada una arquitectura adecuada se procederá a comparar y elegir componentes entre el abanico software que forma todo este ecosistema. Unir todos ellos para un funcionamiento correcto y su uso adecuado es el gran reto de Twitter Data Collection, así como proporcionar ejemplos de uso relevantes.

En este documento se expondrán:

- Los motivos que llevaron al Big Data.
- La gran variedad de tecnologías usadas para lidiar con grandes volúmenes de datos.
- Conocer las distintas APIs de Twitter.
- Averiguar cuáles son los componentes que sirven para cada capa de la arquitectura software diseñada y su funcionamiento.
- Pasos necesarios para instalar un cluster con un único nodo asi como la forma de adquirir, analizar y consultar tweets con cada componente correspondiente.
- La manera de instalar un cluster con múltiples nodos aprovechando los pasos realizados para el de uno y las modificaciones necesarias para adaptar los scripts anteriormente desarrollados.

Este proyecto no solo expone cómo instalar un entorno Hadoop de una manera sencilla, también muestra un ejemplo de cómo tratar con la cantidad ingente de información que logra desbordar a las bases de datos relacionales.

Repositorio disponible en www.github.com/borjagilperez/twitter-data-collection

Palabras clave

Twitter, base de datos, tweets, Big Data, Apache, Hadoop, Pig, HBase, Flume, Hive, Maven, Cluster, Arquitectura Lambda.

Abstract

The motivation of this project is to create a flexible database to ingest, store, analyze and query tweets. Should carry out a research on what are the technologies used for this purpose and design a software architecture according to them. Researches will carry to the rise of Big Data, systems that manipulate large data sets. Once an architecture has been designed shall compare and choose between the software components that form the whole range ecosystem. Merge all for proper operation and proper use is the great challenge of Twitter Data Collection, as well as provide examples of relevant use.

This document will present:

- The reasons that led to the Big Data.
- The variety of technologies used to deal with large volumes of data.
- Know the different Twitter APIs.
- Find out what are the components used for each layer of software architecture designed and functioning.
- Steps for installing a single-node cluster so how to acquire, analyze and view tweets with each corresponding component.
- The way to install a multi-node cluster leveraging the steps taken for single-node one, and the changes needed to adapt the scripts previously developed.

This project not only shows how to install a Hadoop environment in a simple way, also shows an example of how to deal with the huge amount of information that does overwhelm the relational database.

Repository available in www.github.com/borjagilperez/twitter-data-collection

Index terms

Twitter, database, tweets, Big Data, Apache, Hadoop, Pig, HBase, Flume, Hive, Maven, Cluster, Lambda Architecture.

Agradecimientos

A mis padres, Felipe y Genoveva.

Y a todas las personas que han formado parte de esta etapa de mi vida.

Cuando los elefantes sueñan con los datos.

Contenido

TABLAS

ILUSTRACIONES

Glosario

API. *Application Programming Interface*. Conjunto de funciones y procedimientos (o métodos, en la Programación Orientada a Objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Big Data. Describen una nueva generación de tecnologías y arquitecturas, diseñadas para extraer económicamente valor de grandes volúmenes de una amplia variedad de datos, permitiendo una alta velocidad de captura, descubrimiento y/o análisis.

Bloque. Cantidad más pequeña de datos que pueden transferirse en una operación de entrada/salida entre la memoria principal de un ordenador y los dispositivos periféricos o viceversa.

Cluster. Conjuntos de computadoras que se comportan como si fuesen una única computadora.

DataNodes. Almacenan y recuperan bloques cuando se les ordena e informan al *NameNode* periódicamnete con las listas de bloques que están almacenando.

Deadlock. Bloqueo permanente de un conjunto de procesos o hilos de ejecución en un sistema concurrente que compiten por recursos del sistema o bien se comunican entre ellos.

Framework software. Abstracción en la que el software provee una funcionalidad genérica que puede cambiarse de manera selectiva por código adicional escrito por el usuario, proporcionando así el software de aplicación específico.

HDFS. *Hadoop Distributed File System*. Sistema de archivos distribuido de Hadoop que maneja el almacenamiento a través de una red de máquinas, haciendo transparente al programador las complicaciones de programación sobre la red de comunicaciones.

IDE. *Integrated Development Environment*. Programa compuesto por un conjunto de herramientas de programación.

JDBC. *Java DataBase Connectivity*. API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java.

JDK. *Java Development Kit*. Software que provee herramientas de desarrollo para la creación de programas en Java.

Jobtracker. Coordina todos los *jobs* ejecutándose en el sistema mediante la programación de tareas para ejecutarse en *tasktrackers*.

Tasktrackers. Ejecutan las tareas y envían informes de progreso al *jobtracker*. Si una tarea falla el *jobtracker* puede reprogramarlo en un *tasktracker* diferente.

JSON. *JavaScript Object Notation*. Formato ligero para el intercambio de datos. Es un subconjuto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

JVM. *Java Virtual Machine*. Máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un

código binario especial (el bytecode Java) el cual es generado por el compilador del lenguaje Java.

MapReduce. Modelo de programación para el procesamiento de datos de Hadoop.

MD5. *Message-Digest Algorithm 5*. Algoritmo de reducción criptográfico de 128 bits.

MPI. *Message Passing Interface*. Estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores.

NameNode. Maneja el espacio de nombres del sistema de archivos. Mantiene el árbol del sistema de archivos y los metadatos para todos los archivos. Conoce los *DataNodes* en los que se ubican todos los bloques que componen un archivo dado.

NoSQL. *Not Only SQL*. Amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico del sistema de gestión de bases de datos relacionales (RDBMS) en aspectos importantes.

OAuth. *Open Authorization*. Protocolo abierto que permite autorización segura de una API de modo estándar y simple para aplicaciones de escritorio, móviles y web.

Pipe. Cadena de procesos conectados de tal forma que la salida de cada elemento de la cadena es la entrada del próximo.

POSIX. *Portable Operating System Interface* (*'X'* por la identidad UNIX de la API). Familia de estándares de llamadas al sistema operativo. Su objetivo es generalizar las interfaces de los sistemas operativos para que una misma aplicación pueda ejecutarse en distintas plataformas.

RDBMS. *Relational DataBase Management System*. Sistema de gestión de bases de datos relacionales es aquel que sigue un modelo relacional.

Rowkey. Direcciona de manera única una fila en HBase.

SQL. *Structured Query Language.* Lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas.

TIC. *Tecnologías de la Información y la Comunicación*. Agrupan los elementos y las técnicas empleadas para el tratamiento de datos y la transmisión de información.

Tweet. Mensaje de Twitter enviado. Su representación interna está en formato JSON.

UDF. *User Defined Function*. Función definida por el usuario a ser ejecutada por Pig.

URI. *Uniform Resource Identifier*. Cadena de caracteres que identifica los recursos de una red de forma unívoca. La diferencia respecto a un Uniform Resource Locator (URL) es que estos últimos hacen referencia a recursos que, de manera general, pueden variar con el tiempo.

XML. *eXtensible Markup Language*. Lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible.

1. Introducción

1.1. Motivación

Recientemente Twitter se ha erigido como uno de los focos más importantes de información para el análisis de las redes sociales, tanto para la comprensión de cómo funcionamos como sociedad como para el estudio y predicción de eventos, de tendencias, de protestas y malestar social, de epidemias de gripe o de terremotos entre otros (González-Bailón, Borge-Holthoefer, Rivero, & Moreno, 2011)
(Congosto, Fernández, & Moro Egido, 2011)
(Romero, Meeder, & Kleinberg, 2011)
(Achrekar, Gandhe, Lazarus, Yu, & Liu, 2011)
(Sakaki, Okazaki, & Matsuo, 2010)

No obstante, dichos estudios requieren de la compilación previa de grandes bases de datos que, dado el tamaño actual de Twitter, no son sencillos de adquirir. Por ello muchos de los mismos se basan en unos cuantos esfuerzos de recolección que se hicieron en el pasado, como la base de datos de (Kwak, Lee, Park, & Moon, 2010) que comprende gran parte de los tweets de la segunda mitad de 2009 o pequeñas compilaciones locales solo adecuadas para estudios concretos.

Con la creciente cantidad de información que circula por Twitter, el incremento en su número de usuarios y los cambios tanto económicos como políticos que está sufriendo la sociedad se hace necesario renovar la información de que se dispone. Así el presente Proyecto de Fin de Grado propone la creación de una gran base de datos de Twitter usando la API que él mismo proporciona (Documentation, s.f.)

Concretamente en el *Ambient Intelligence Laboratory* (AmILab, s.f.) de la Escuela Politécnica Superior (EPS, s.f.) - Universidad Autónoma de Madrid (UAM, s.f.) se llevan 2 años recopilando datos de Twitter y este proyecto responde a una necesidad real de crecimiento.

1.2. Objetivos

1. Capturar el flujo constante de tweets en formato JSON (JSON, s.f.) interaccionando con Twitter API y filtrar todos aquellos que sean innecesarios para cada análisis.
2. Almacenar los tweets en una base de datos adecuada para este formato, volumen de información del orden de petabytes y con capacidad de analizarlos en un tiempo razonable.
3. Crear un sistema informático que proporcione flexibilidad a la hora de modificar análisis y consultas.

1.3. Fases de realización

1. Investigación de las diversas tecnologías con las que construir el sistema informático.
   1.1. Comparación de dichas tecnologías según sus pros y contras.
   1.2. Decisión de cuáles son las adecuadas para la gran base de datos de tweets.
   1.3. Diseño del sistema informático.
2. Construcción del sistema informático en un cluster mono-nodo.
   2.1. Instalación y configuración de cada componente.
   2.2. Diseño y implementación de las acciones que realiza cada componente.

3. Construcción del sistema informático en un cluster multi-nodo.
   3.1. Instalación y configuración de cada componente.
   3.2. Diseño y implementación de las acciones que realiza cada componente.
4. Pruebas.

1.4. Estructura del documento

Capítulo 2. *Estado del arte*.

- Visión global sobre la problemátia del escalado en los sitemas de procesamiento de datos tradicionales para hacer frente al creciente tamaño de datos a almacenar y procesar.
- Introducción a los sistemas informáticos que manipulan un volumen total de datos del orden de petabytes y su pieza fundamental dentro del entorno software que lo forma.
- Descubrimiento de una arquitectura software en auge para estos sistemas que proporciona flexibilidad en cuanto al uso que se desee hacer de ella.
- Comprensión de los beneficios e inconvenientes de cada API de Twitter. Toma de decisiones para realizar Twitter Data Collection.

Capítulo 3. *Twitter Data Collection: componentes*. Para cada tarea realizada por la gran base de datos se debe investigar cuál es el componente o componentes más adecuados. Para cada uno de los elegidos habrá una explicación para comprender con mayor detalle de qué manera pueden aprovecharse mejor sus características.

Capítulo 4. *Twitter Data Collection: cluster mono-nodo*. Cómo instalar un cluster distribuido con un único nodo incluyendo explicaciones sobre las acciones que se realizan así como el funcionamiento del proyecto desarrollado y los scripts que permiten adquirir, almacenar, analizar y consultar la información en este tipo de cluster.

Capítulo 5. *Twitter Data Collection: cluster multi-nodo*. Cómo instalar un cluster distribuido con dos nodos, extrapolable a tantos como se desee.

Capítulo 6. *Pruebas*. Ejemplos de uso con varios tipos de análisis.

Capítulo 7. *Conclusiones y trabajo futuro*. Reflexiones sobre las pruebas obtenidas y posibles ampliaciones futuras.

Referencias. Referencias a las que hace el documento.

Anexo A. *Obtención de claves en Twitter Developers*. Conectar el cluster con el API de Twitter para ingerir tweets requiere varias claves. Para su obtención habrá que realizar unos pocos pasos.

Anexo B. *Otros scripts y código fuente*. Scripts y código fuente explicados o indicados en los capítulos pero no mostrados.

2. Estado del arte

2.1. Introducción

La capacidad de almacenamiento se ha incrementado enormemente en los últimos 25 años mientras que la velocidad de acceso a los datos no ha mantenido el mismo ritmo de crecimiento. Según (IDC IVIEW, 2011) en 2010 el volumen de datos de internet sobrepasó la barrera de los zettabytes, donde 1 ZB = 10^{21} bytes, 1 trillón de gigabytes.

Los sistemas de procesamiento de datos tradicionales se han escalado para hacer frente al creciente tamaño de los datos a almacenar y procesar. Se puede realizar un escalado vertical añadiendo más recursos a un nodo particular del sistema, o realizar un escalado horizontal agregando más nodos al sistema. Las dos tienen el propósito de aumentar el rendimiento del conjunto. A continuación se expone un resumen a partir de las explicaciones proporcionadas por (Turkington, 2013) sobre las ventajas e inconvenientes de ambos escalados:

	Escalado vertical	**Escalado horizontal**
Ventajas	Aunque los componentes se cambien por unos más potentes las relaciones entre ellos siguen siendo las mimas.	La suma de los costes de cada nodo es mucho menor que la de un escalado vertical que proporcione el mismo rendimiento.
Inconvenientes de la arquitectura	Tienen un precio muy alto. Existen límites prácticos en cuanto a lo grande o potente que puede ser una sola máquina, siendo un punto crítico la conectividad entre estos grandes componentes.	Aparecen problemas de complejidad según se incrementa el número de nodos, como el mantenimiento y el balanceo de carga.
Inconvenientes en el desarrollo	Migrar software a una arquitectura donde hay cada vez más procesadores no es trivial.	Es complejo crear estrategias para dividir el procesamiento de datos entre todos los servidores.

Tabla 1 - Ventajas e inconvenientes de los escalados vertical y horizontal

2.2. Big Data y Hadoop

El término Big Data hace referencia a los sistemas informáticos que manipulan un volumen total de datos del orden de petabytes. Una magnífica definición de Big Data es la expuesta por (IDC IVIEW, 2011):

> *Las tecnologías Big Data describen una nueva generación de tecnologías y arquitecturas, diseñadas para extraer económicamente valor de grandes volúmenes de una amplia variedad de datos, permitiendo una alta velocidad de captura, descubrimiento y/o análisis.*

Para un sistema de Big Data, mezclar escalados vertical y horizontal daría como resultado algunos de los beneficios de ambos enfoques pero con la unión de los costes de fabricar arquitecturas a medida con las debilidades de ambos. Desde que el Big Data

empezó a tomar forma se ha optado por un escalado horizontal ya que el coste de los equipos es significativamente menor.

La pieza básica del entorno Big Data es Hadoop (Apache Hadoop, s.f.). Éste utiliza MapReduce, un procesador de consultas por lotes para procesar el conjunto total de datos o gran parte de éste y realizar consultas o análisis a medida en un tiempo razonable. Como sistema de archivos distribuido utiliza Hadoop Distributed File System, HDFS.

A continuación se expone un resumen a partir de las explicaciones proporcionadas por (White, 2012) sobre las comparaciones entre MapReduce y:

- Relational DataBase Management System tradicional.
- High Performance & Grid Computing.

	RDBMS tradicional	**MapReduce**
Tamaño de datos	Gigabytes.	Petabytes.
Acceso	Interactivo y en lotes. Patrón de acceso a datos gobernado por *seeks*, el cual está mejorando más lentamente que la velocidad de transferencia. Es bueno para consultas concretas donde el conjunto de datos ha sido indexado.	En lotes. Utiliza en mayor medida la velocidad de transferencia. La mejor opción cuando lo que se necesita es analizar en lote el conjunto de datos entero o gran parte de éste.
Actualizaciones	Lectura y escritura muchas veces. Actualizará rápidamente pequeñas porciones de registros donde el conjuto de datos ha sido indexado. Adecuado para bases de datos cuyos registros se están actualizando continuamente.	Escribe una vez, lee muchas veces. Más eficientes en actualizaciones que RDBMS si tiene que hacerlo sobre gran parte de los registros. Las actualizaciones deben ser las menores posibles.
Estructura	Esquema estático. Es una base de datos estructurados, organizados en entidades que tienen un formato definido.	Esquema dinámico. Funciona bien para datos semi-estructurados, aquellos que tienen un esquema que puede ser ignorado, o para datos sin estructura.
Integridad	Alta. Los datos en una base de datos relacional son normalizados a menudo para mantener integridad y eliminar redundancia.	Baja. No tiene sentido normalizar con MapReduce ya que hace la lectura de un registro procedente de una operación no local.
Escalado	No lineal.	Lineal.

	Generalmente, para consultas SQL si se dobla el tamaño de datos de entrada la ejecución de una tarea no tiene por qué ser el doble de lenta sino que podría tardar aún más.	Las funciones de Map y Reduce son independientes del tamaño de datos o el cluster ya que pueden ser ejecutadas sin cambiar su código. Si se dobla el tamaño de datos de entrada, la ejecución de una tarea será el doble de lenta, pero si se dobla el tamaño del cluster la tarea ejecutará tan rápido como la original.

Tabla 2 - Comparación entre MapReduce y RDBMS tradicional

	HPC & Grid Computing	**MapReduce**
Procesamiento de datos	Estos sistemas son efizaces para trabajos que requieren una computación de cálculo intensa. No funcionan tan bien cuando los nodos necesitan acceder a grandes volúmenes de datos, cientos de gigabytes, ya que el ancho de banda se convierte en el cuello de botella y los nodos de cómputo quedan inactivos.	Con alrededor de cientos de gigabytes de datos es donde empieza a mostrar todo su rendimiento. Trata de colocar los datos con los nodos de cómputo para mayor rapidez de acceso. Para MapReduce el ancho de banda de red es el recurso más preciado en un entorno de centro de datos.
Control del programador	Hace uso de APIs y Message Passing Interface, MPI, el cual da gran control al programador pero requiere un manejo explícito del flujo de datos.	Opera en alto nivel. El programador debe pensar sobre pares clave-valor ya que el flujo de datos está implícito. Evita al programador manejar fallos, el orden de las tareas pasa a ser manejado por MapReduce haciendo uso de otros nodos.

Tabla 3 - Comparación entre MapReduce y High Performance & Grid Computing

2.3. CDH

Cloudera Distribution including Apache Hadoop (CDH, s.f.) es una distribución de Hadoop y de proyectos relacionados 100% código abierto bajo licencia Apache.

Proporciona:

- *Online NoSQL* con HBase, una base de datos no relacional distribuida basada en almacenamiento clave-valor cuyo objetivo es construir aplicaciones de tiempo real sobre tablas de miles de millones de filas y millones de columnas, todo ello con un acceso aleatorio rápido.
- *Batch processing* también utiliza a MapReduce, Pig y Hive.
- *Analytic SQL* gracias a Impala, motor SQL masivamente paralelo construido para Hadoop.
- *Cloudera Search*, el cual permite a los usuarios buscar tal y como lo harían en un buscador web o en un comercio electrónico.

- *In-Memory Machine Learning and Stream Processing* gracias a Spark y Mahout.

Sin duda es un buen conjunto de tecnologías dedicadas a diferentes propósitos, que de primeras nos ofrece un punto de partida interesante sobre el que empezar a explorar. Puede que estos nombres en principio sean desconocidos pero muestran ejemplos de cuáles podrían ser los componentes sobre los que empezar a investigar.

2.4. Arquitectura Lambda

La Arquitectura Lambda (Marz & Warren, Big Data. Principles and best practices of scalable realtime data systems, s.f.) es el nuevo y sencillo paradigma para un sistema Big Data.

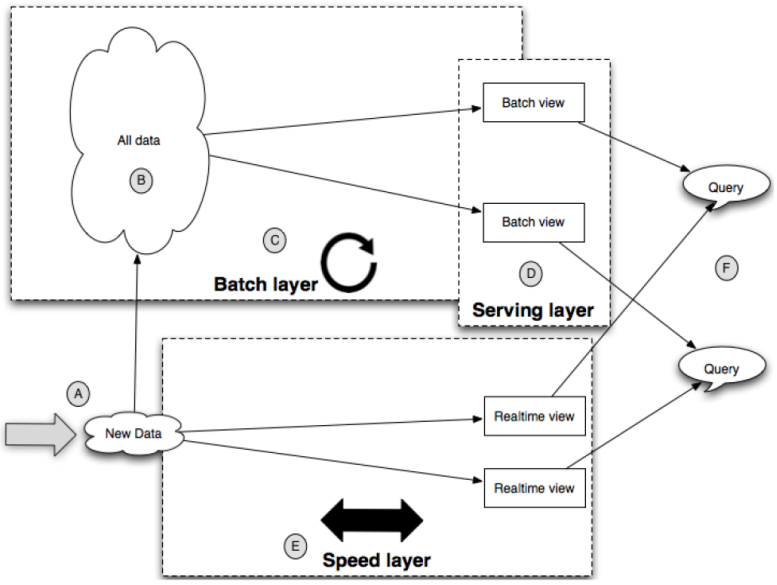


Ilustración 1 - Arquitectura Lambda (Marz & Warren, A new paradigm for Big Data, 2012)

2.4.1. Batch Layer

Almacena el conjunto total de datos y precomputa vistas que se almacenarán en Serving Layer. Éstas vistas se pueden recomputar de nuevo cada cierto tiempo tanto para corregir errores en el análisis como para obtener los nuevos datos que se han ido almacenando en el conjunto total y que no fueron incluídos en la última precomputación. Cada iteración puede tardar horas dependiendo del tamaño de conjunto de datos o del cluster (Kinley, 2013). Las vistas se pueden recomputar continuamente o no dependiendo del uso que se haga del sistema. Todos estos datos se pueden manipular con funciones definidas por el usuario.

Los sistemas de procesamiento en lote son buenos almacenando conjuntos de datos inmutables y en constante crecimiento. Hadoop es el máximo exponente de este tipo de sistemas.

2.4.2. Serving Layer

Bases de datos especializadas en donde las precomputaciones de Batch Layer se cargan e indexan para así poder ser consultadas rápida y eficazmente. Cada vez que Batch Layer recomputa las vistas, las Batch Views son actualizadas por nuevas versiones.

Las bases de datos de esta capa no requieren escrituras aleatorias, solo actualizaciones en lote y lecturas aleatorias. Las escrituras aleatorias son las responsables de la complejidad en una base de datos por lo que se podría optar por un tipo de base de datos especializada más simple.

2.4.3. Speed Layer

Los datos no representados en Batch Views son aquellos que llegan al sistema mientras se está ejecutando la precomputación de las vistas. Con Speed Layer se obtiene un sistema de datos en tiempo real. Los datos nuevos se almacenan tanto en Realtime Views como en el conjunto total de Batch Layer. Las Realtime Views se actualizan incrementalmente a diferencia de las Batch Views que sufren actualizaciones recomputadas. Cuando estos datos pasan a formar parte de Batch Views ya no son necesarios en ninguna Realtime View por lo que se eliminan.

Estas vistas requieren bases de datos que soporten lecturas y escrituras aleatorias. Por el hecho de soportar escrituras aleatorias estas bases de datos son órdenes de magnitud más complejas que las que se podrían usar en Serving Layer.

Por último solo queda unir los resultados de las consultas que se realizan a Batch Views y a Realtime Views.

2.4.4. Propiedades de un sistema Big Data

Proporcionadas por Batch & Serving Layers:

- Robusto y tolerante a fallos.
- Escalable.
- Sistema general.
- Extensible.
- Permite realizar consultas específicas.
- Mantenimiento mínimo.
- Fácilmente depurable.

Proporcionada por Speed Layer:

- Sistema en tiempo real.

Para una lectura con mayor detalle se recomienda leer (Marz & Warren, A new paradigm for Big Data, 2012)

2.5. Twitter API

La documentación de Twitter Developers (Documentation, s.f.) muestra dos tipos de API: REST API y Streaming API.

La REST API no necesita tener abierta una conexión HTTP persistente. Una aplicación web con esta API podría realizar una o más consultas.

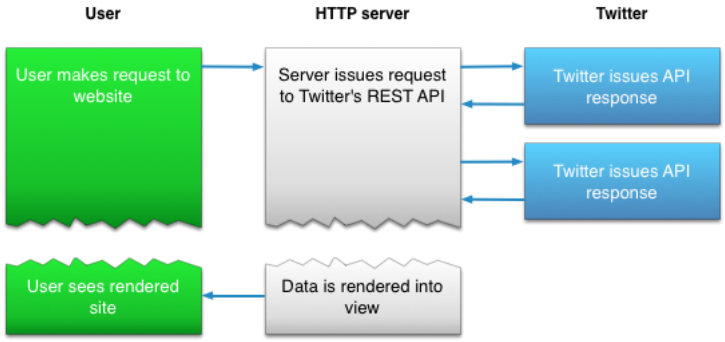


Ilustración 2 - REST API (The Streaming APIs, 2012)

Opción REST API:

- ✓ Los elementos de consulta son mayores y más específicos que en Streaming API, pudiendo realizar consultas con más detalle.
- ✓ Por cada aplicación creada puede haber varias conexiones utilizándola.
- × Tiene un límite de consultas y tiempo por usuario que el desarrollador deberá controlar, esto significa que habrá tiempos muertos en la recolección de tweets.

Con Streaming API no es posible establecer una conexión en respuesta a la consulta de un usuario. Por un lado un proceso streaming toma tweets y realiza parseo, filtrado y/o agregación antes de almacenar el resultado. Por otro lado el proceso manejador HTTP consulta la base de datos donde se almacenan los tweets en respuesta a la solicitud del usuario.

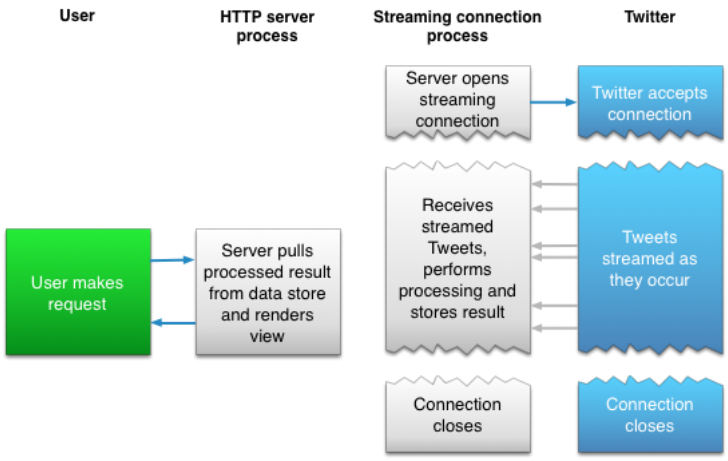


Ilustración 3 - Streaming API (The Streaming APIs, 2012)

Los tipos de Streaming API según el uso que se realice son: *Public*, *User* y *Site streams*. Mientras *User* y *Site streams* están enfocados a usuarios en particular, *Public streams* ofrece una pequeña fracción de los tweets de todo tipo que fluyen por Twitter aunque de manera constante.

Opción Streaming API:

- ✓ Proporciona un flujo constante de una gran variedad de tweets en tiempo real.
- × Los elementos de consulta sobre la propia API son menores, solo con unos pocos parámetros en común si así se desea.

- × Por cada aplicación solo puede haber una conexión.
- × El flujo de datos proporcionado tiene un límite máximo de tweets por cantidad de tiempo por lo que cuando se incrementen drásticamente los tweets publicados sobre alguna tendencia todo aquel que no entre en el *pipe* de información se perderá ya que es imposible realizar consultas que capturen tweets anteriores en el tiempo.

2.6. Conclusiones

2.6.1. Hadoop: componente básico

Hadoop será el componente básico de la aquitectura por los siguientes motivos:

- ✓ Tamaño de datos: petabytes.
- ✓ Acceso: en lotes. No se necesita realizar consultas concretas sobre la base de datos sino almacenar la mayor cantidad posible de tweets y analizar en Batch Layer el conjunto de datos entero o gran parte éste para realizar precomputaciones cuyos resultados se almacenarán en Batch Views.
- ✓ Actualizaciones: escribe una vez, lee muchas veces. Nunca se realizan actualizaciones sobre los tweets ya almacenados, solo se requiere escribirlos una vez en Batch Layer, que almacena el conjunto total de tweets. En cambio, se realizarán gran cantidad de lecturas para su análisis.
- ✓ Estructura: esquema dinámico. Los campos de cada tweet se han ido incrementando e incluso modificando con el tiempo como seguirá ocurriendo en un futuro. Por ello una base de datos semi-estructurados es la elección más conveniente.
- ✓ Escalado: horizontal y con tendencia lineal respecto al incremento de datos y al cluster. Un escalado horizontal es la opción más accesible y dado el volumen de datos a manejar el ancho de banda de red debe ser el recurso más preciado para que no se convierta en un cuello de botella.
- ✓ Control del programador: opera en alto nivel. El programador debe pensar mayormente sobre los análisis que se requieran, evitándole manejar fallos u orden explícito de tareas.

2.6.2. Elección de Twitter API

Opción REST API:

- × La base de datos no requiere almacenar tweets provenientes de consultas específicas de Twitter sino almacenar gran cantidad de éstos con solo unos pocos parámetros en común como podría ser la localización geográfica.
- × Tiene un límite de consultas y tiempo por usuario por lo que habrá tiempos muertos en la recolección de tweets. Esto no es aceptable para Twitter Data Collection ya que requiere que la información almacenada tenga continuidad en el tiempo.

Opción Streaming API:

- ✓ Es la más adecuada para la recolección constante de una gran variedad de tweets en tiempo real con algunos parámetros en común sobre los que posteriormente realizar un análisis lo más exhaustivo que se requiera. El flujo de tweets

almacenados es continuo por lo que no habrá saltos temporales en los que se pierda información que podría ser relevante para los análisis.

- ✓ Por cada aplicación solo puede haber una conexión pero esto no es un problema ya que el sistema informático podría tener únicamente una aplicación/conexión a la API y a su vez un número indefinido de usuarios realizando consultas sobre la información ya almacenada en la base de datos. Además, se pueden crear varias aplicaciones que abastezcan a la misma base de datos para así capturar mayor número y variedad de tweets.

De las dos, Streaming API será la opción más adecuada para Twitter Data Collection.

2.6.3. Arquitectura software

Podría utilizarse una arquitectura software completa, probada y tan profesional como CDH, el problema reside en que de esta manera es más difícil comprender las interrelaciones entre los componentes que la forman. Formar una arquitectura personalizada uniendo componentes necesarios para Twitter Data Collection será el modo de proceder durante este proyecto.

El modelo propuesto por la Arquitectura Lambda tiene todas las propiedades necesarias para crear la gran base de datos de tweets aunque no todas ellas serán necesarias.

Propiedades ofrecidas por Batch & Serving Layers:

- ✓ Robusto y tolerante a fallos.
- ✓ Escalable.
- ✓ Sistema general.
- ✓ Extensible.
- ✓ Permite realizar consultas específicas.
- ✓ Mantenimiento mínimo.
- ✓ Fácilmente depurable.

Propiedades ofrecidas por Speed Layer:

- × Sistema en tiempo real.

Al realizar un análisis amplio sobre la información contenida en Batch Views, los últimos tweets que se han ido almacenando en Batch Layer y que no han entrado en la última recomputación no tendrán el suficiente impacto en los resultados como para que sea necesario desarrollar la capa que proporciona los tweets más actuales. Para el propósito de Twitter Data Collection ésta capa se puede obviar ya que es aceptable un retraso de horas en la incorporación de los tweets más recientes a las Batch Views.

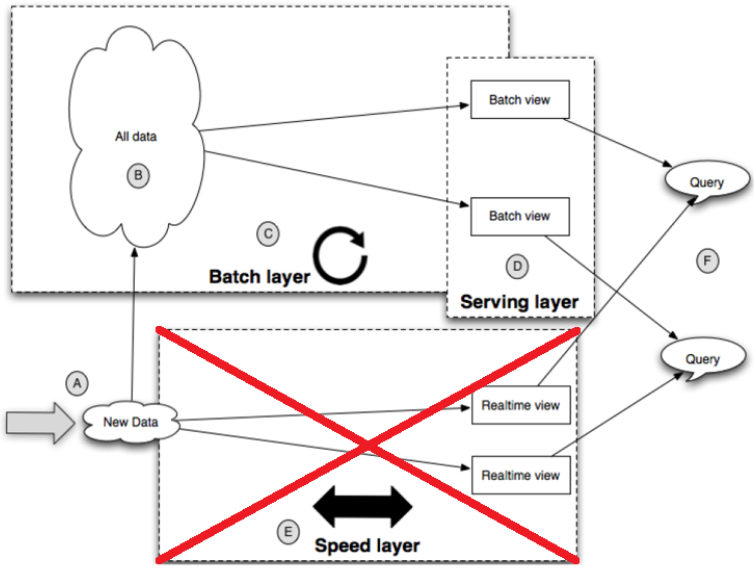


Ilustración 4 - Modificación de Arquitectura Lambda para Twitter Data Collection

3. Twitter Data Collection: componentes

Hadoop proporciona almacenamiento y framework software para procesamiento de datos pero dependiendo del propósito del sistema informático se necesitan otros componentes software que lo complementen. Twitter Data Collection además de las características proporcionadas por Hadoop necesita componentes adecuados para conectarse a Streaming API y así adquirir tweets, analizar grandes conjuntos de datos que están en bruto y realizar consultas aleatorias muy concretas.

A partir de Hadoop se desarrolla un abanico software en constante desarrollo, el ecosistema Hadoop. De entre todos los componentes existentes se deben elegir cuáles son los más adecuados para el proyecto.

3.1. Adquisición de datos

Existe gran variedad de librerías para las APIs de Twitter en diferentes lenguajes, listadas en (Twitter Libraries, 2014). La decisión de escoger Twitter4J (Twitter4J, s.f.) se debe a la familiaridad del lenguaje Java para la mayoría de los programadores. Hasta hace poco tiempo ésta era la única librería Java construída explícitamente para la plataforma Twitter y sigue siendo la más completa mientras se desarrolla este proyecto. Recientemente Twitter ha construido y mantiene su única librería, *hbc*, un cliente Java HTTP para consumir de Streaming API.

Se necesita introducir las respuestas en formato JSON que devuelve Streaming API dentro de Hadoop Distributed File System. Para elegir la opción más adecuada se puede realizar una comparación entre los componentes software más comunes para este propósito, Sqoop (Apache Sqoop, s.f.), Kafka (Apache Kafka, s.f.), Scribe (Scribe, s.f.) y Flume (Apache Flume, s.f.)

- × *Sqoop*. Librería que permite importar datos desde una base de datos estructurados, organizados en entidades que tienen un formato definido, hacia Hadoop. Del mismo modo permite la importación de datos en sentido contrario. No sirve al propósito del proyecto ya que no se desea importar datos desde una base de datos relacional.
- × *Kafka*. Su principal caso de uso es un sistema distribuido de paso de mensajes publicación-suscripción. Adecuado para sistemas altamente confiables y escalables de mensajería empresarial en los que se deben conectar múltiples sistemas, siendo uno de los ellos Hadoop.
  No es el software adecuado ya que no se necesita un sistema de paso de mensajes con varios tipos de sistemas informáticos.
- × *Scribe*. El soporte que recibe cada vez es menor.
- ✓ *Flume*. Servicio distribuido, seguro y con alta disponibilidad para una eficiente recolección, unión y movimiento de grandes cantidades no solo de log data, sino también de cantidades masivas de eventos de datos incluyendo tráfico de datos de red y datos generados por *social media* procedentes de fuentes de datos no relacionales.

Apache Flume es la opción escogida ya que maneja datos semi-estructurados como serán los tweets en JSON.

3.1.1. Librería para Twitter API: Twitter4J

Twitter4J (Twitter4J, s.f.) es una librería Java no oficial que facilita la integración con Twitter API. Nos permite gestionar tweets, usuarios, relaciones, favoritos, suscripciones, bloqueos, realizar búsquedas, etc. Incluye software de JSON.org para poder parsear las respuestas JSON procedentes de Twitter.

Respecto al lo necesario para desarrollar Twitter Data Collection, Twitter4J ofrece:

- ✓ 100% puro Java (versión Java Platform 5 o posterior)
- ✓ Cero dependencias: no se requieren jars adicionales.
- ✓ Soporte integrado OAuth (estándar abierto para autorización utilizado por Twitter API)
- ✓ 100% compatible con Twitter API 1.1 (última versión)

3.1.2. Recolección, unión y movimiento: Flume

Apache Flume (Apache Flume, s.f.) es una arquitectura simple y flexible basada en flujos de datos streaming. Es robusto y tolerante a fallos con mecanismos de fiabilidad personalizables y muchos mecanismos de fail-over y fail-back.

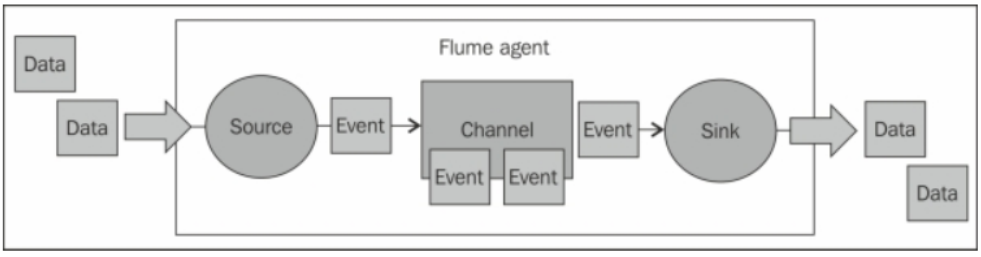


Ilustración 5 - Flume agent (Hoffman, 2013)

Todos los componentes de Flume se ejecutan en un *agent*, un proceso JVM que puede tener muchos *sources*, *channels* y *sinks*, los cuales manejan *events*.

3.1.2.1. Events

Un *event* es la carga útil básica de datos transportados por Flume. Un evento se compone por cero o más cabeceras y un cuerpo. Las cabeceras son pares clave-valor que pueden ser usadas para tomar decisiones de enrutamiento o llevar otra información estructurada. El cuerpo es un array de bytes que contienen la carga útil. Flume puede añadir cabeceras adicionales automáticamente pero el cuerpo no se toca a no ser que se modifique en ruta mediante interceptores.

Un interceptor es un punto del flujo de datos donde se pueden inspeccionar y alterar eventos. Se pueden encadenar cero o más interceptores después de que un *source* cree un evento o antes de que un *sink* envíe el evento a donde vaya a ser destinado.

3.1.2.2. Sources

Un *source* es la entidad por la que los datos entran en Flume. Escribe eventos a uno o más *channels*. Existen variedad de *sources* para recolectar variedad de datos. El source debe entender el tipo de evento que recibe.

El *source* personalizado a realizar, *FlumeTwitterSource*, utilizará la librería Twitter4J para conectarse a Streaming API y así adquirir tweets de manera continua con algunos parámetros en común como por ejemplo la localización geográfica o alguna de las palabras clave que deben contener.

3.1.2.3. Channels

Un *channel* es el área de espera de los eventos mientras pasan de un *source* a un *sink*. Este almacenamiento temporal permite a *sources* y *sinks* ejecutarse asíncronamente. Existen dos tipos de *channels*:

- *Memory channel* es un canal donde los eventos son almacenados en memoria. La memoria es normalmente órdenes de magnitud más rápida que un disco por lo que los eventos pueden ser ingeridos mucho más rápido reduciendo así costes de hardware. La pega es que un fallo en el agente propicia la pérdida de datos.
- *File channel* es un canal que almacena los eventos en el sistema de archivos local del agente. Aunque es más lento que un *memory channel* proporciona un camino de almacenamiento duradero.

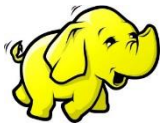
Pese a que las brechas en la adquisición de información no son deseables para Twitter Data Collection, premiará más la velocidad almacenando la cantidad enorme de datos entrantes a que se pierdan los correspondientes a un intervalo de tiempo en caso de fallo. Por este motivo se utilizarán *memory channels*.

3.1.2.4. HDFS sinks

Un *sink* recibe eventos de un único *channel*. Flume soporta más tipos de *sinks*, siendo Hadoop Distributed File System el más utilizado. El trabajo de un *HDFS sink* es abrir continuamente un archivo en HDFS, transmitir datos dentro, cerrar el archivo y empezar uno nuevo.

Tener un motón de archivos pequeños de entrada hará a los *MapReduce jobs* de Hadoop más ineficientes. Si se fueran a guardar los datos por un periodo corto de tiempo entonces se optaría por tamaños de archivo pequeños pero como se van a guardar por un periodo largo entonces se optará por tamaños de archivos grandes. Otra opción sería compactar periódicamente los archivos más pequeños en unos pocos grandes para hacerlos más agradables de cara a MapReduce.

3.2. Almacenamiento y framework para procesamiento de datos: Hadoop



La librería software Apache Hadoop (Apache Hadoop, s.f.) es un framework software que permite el procesamiento distribuido de grandes cantidades de datos a través de clusters de computadoras usando modelos de programación simples. Sus partes esenciales son su sistema de archivos distribuido Hadoop Distributed File System y su motor MapReduce.

En 2010, para clusters de más de 4000 nodos MapReduce fue insuficiente para hacer frente a los cuellos de botella en cuanto a escalabilidad por lo que se diseñó la siguiente generación de MapReduce, YARN. En este proyecto se ha utilizado una versión de Hadoop sin YARN ya que para el aprendizaje es menos complejo y con el hardware disponible para pruebas esta segunda generación de MapReduce no es esencial. Para más información consultar (White, 2012)

3.2.1. HDFS

Hadoop Distributed File System es un sistema de archivos distribuido que maneja el almacenamiento a través de una red de máquinas, haciendo transparente al programador las complicaciones de programación sobre la red de comunicaciones. Estos sistemas de archivos tienen como misión no sufir pérdidas de datos ante fallos de los nodos.

HDFS almacenará todos los archivos en bruto que contienen tweets en formato JSON.

3.2.1.1. Bloques

Al igual que en un sistema de archivos para disco, los archivos en HDFS están partidos en *block-sized chunks*. Los bloques de los sistemas de archivos (mínima cantidad de datos que puede ser leída o escrita) normalmente son de 512 bytes, sin embargo los de HDFS son de 64 megabytes por defecto. Al contrario que en un sistema de archivos para disco, un archivo en HDFS que es más pequeño que un bloque no ocupa el valor de un bloque completo de almacenamiento subyacente. Los bloques de HDFS son grandes en comparación con los bloques de disco para minimizar el coste de los *seeks*.

Beneficios de tener una abstracción de bloques para un sistema de archivos distribuido:

- Un archivo puede ser tan grande como cualquier disco de la red.
- Tener un bloque de tamaño fijo como unidad de abtracción simplifica el subsistema de almacenamiento ya que será fácil calcular cuántos pueden ser almacenados en un disco. Elimina todo lo concerniente a metadatos debido a que los bloques serán pedazos de datos a almacenar y los archivos de metadatos no necesitan ser almacenados con los bloques, por lo que otro sistema puede manejarlos.
- Los bloques encajan bien con la replicación para proporcionar tolerancia a fallos y disponibilidad. Para prevenir fallos en el sistema ante bloques y discos corruptos o fallos de máquinas, cada bloque es replicado a un número pequeño de máquinas físicamente separadas, normalmente tres.

3.2.1.2. NameNode y DataNodes

El *NameNode* (servidor maestro):

- Maneja el espacio de nombres del sistema de archivos. Mantiene el árbol del sistema de archivos y los metadatos para todos los archivos.
- Conoce los *DataNodes* en los que se ubican todos los bloques que componen un archivo dado.
- Es un púnto único de fallo (*Single Point of Failure*) ya que se requiere intervención manual si falla la máquina que lo aloja.

Existe un *SecondaryNameNode* que puede alojarse en otra máquina. Pese a su nombre, solo crea puntos de control del espacio de nombres.

Los *DataNodes* (trabajadores) almacenan y recuperan bloques cuando se les ordena e informan al *NameNode* periódicamnete con las listas de bloques que están almacenando.

3.2.1.3. Propiedades

Archivos muy grandes. De hasta de cientos de terabytes de tamaño. En HDFS un archivo existe solo como una entrada de directorio, mostrando un tamaño cero hasta que

el archivo sea cerrado, al contrario que en un sistema de archivos POSIX. Si el archivo que está siendo escrito con datos no se cierra y ocurre una desconexión de red, dejará únicamente un archivo vacío. Aún así no es conveniente utilizar tamaños de archivo pequeños ya que HDFS y MapReduce no son eficientes con ellos.

Acceso a datos constante. Patrón de acceso a datos *escribe una vez, lee muchas veces*. Cada análisis involucrará toda o gran porción del conjunto de datos.

Hardware asequible. Hadoop no requiere clusters caros con hardware de alta confianza ya que HDFS está diseñado para seguir trabajando sin interrupciones pese a fallos.

Algunos usos para los que HDFS no es la mejor opción:

- *Acceso a datos con baja latencia*. HDFS está optimizado para proporcionar un alto rendimiento en el acceso a grandes lotes de datos en lugar de a registros particulares de la base de datos o pequeños conjuntos de ellos.
- *Almacenar y manejar muchos archivos pequeños*. El *NameNode* guarda el árbol de directorios de todos los archivos en el sistema de archivos e indica dónde están guardados en el cluster por lo que cuantos más archivos se creen más memoria RAM será necesario usar. El límite del número de archivos en el sistema de archivos estará limitado por la cantidad de memoria del *NameNode*.
- *Múltiples escritores y modificaciones de archivos*. Los archivos en HDFS solo pueden ser escritos por un único *escritor* y las escrituras solo pueden hacerse al final del archivo.

3.2.2. MapReduce

Modelo de programación para el procesamiento de datos. Hadoop puede ejecutar programas MapReduce escritos en varios lenguajes, los cuales son creados para trabajar en paralelo sobre análisis de datos a gran escala. MapReduce empieza a mostrar todo su rendimiento para conjuntos de datos enormes. Es usado para análisis, problemas basados en grafos, algoritmos de aprendizaje automático, etc.

MapReduce muestra poca eficiencia con archivos diminutos. Si se tienen multitud de ellos el coste de iniciar los procesos de trabajo puede ser muy alto comparado con los datos que a procesar.

Un *MapReduce job* es una unidad de trabajo que un cliente quiere que se realice. Hadoop ejecuta el *job* dividiéndolo en tareas, de las cuales hay dos tipos: *map tasks* y *reduce tasks*. Existen dos tipos de nodos que controlan el proceso de ejecución de un *job*: un *jobtracker* y varios *tasktrackers*.

El *jobtracker* coordina todos los *jobs* ejecutándose en el sistema mediante la programación de tareas para ejecutarse en *tasktrackers*. Los *tasktrackers* ejecutan las tareas y envían informes de progreso al *jobtracker*. Si una tarea falla el *jobtracker* puede reprogramarlo en un *tasktracker* diferente.

Hadoop divide las entradas a un *MapReduce job* en piezas de tamaño fijo llamadas *input splits*. Hadoop crea un *map task* por cada *split* que ejecutará la función de map definida por el usuario para cada registro del *split*. Se procesan todos los *splits* en paralelo procurando el mejor balanceo de carga posible. Si los *splits* son demasiados pequeños, la sobrecarga de manejarlos y crear *map tasks* comienza a dominar el tiempo de

ejecución total del *job*. Para la mayoría de los *jobs* un buen tamaño de *split* tiende a ser el tamaño de un bloque HDFS, 64 megabytes por defecto.

Los *map tasks* escriben sus salidas en el disco local y no en HDFS. Esto se debe a que son salidas intermedias que serán procesadas por *reduce tasks* para producir el resultado final, y una vez el *job* se haya completado la salida del map podrá ser desechada. Si se utiliza HDFS con replicación y el nodo ejecutando el *map task* falla antes de que la salida map haya sido consumida por el *reduce task*, entonces Hadoop automáticamente volverá a ejecutar el *map task* en otro nodo para recrear la salida del map.

La siguiente ilustración porporcionada por (Gates, 2011) muestra por qué coloquialmente al flujo de datos entre los *map* y *reduce tasks* se le conoce como shuffle ya que cada *reduce task* es suministrado por muchos *map tasks*. Se trata de una aplicación MapReduce que cuenta el número de apariciones de una palabra en un texto.

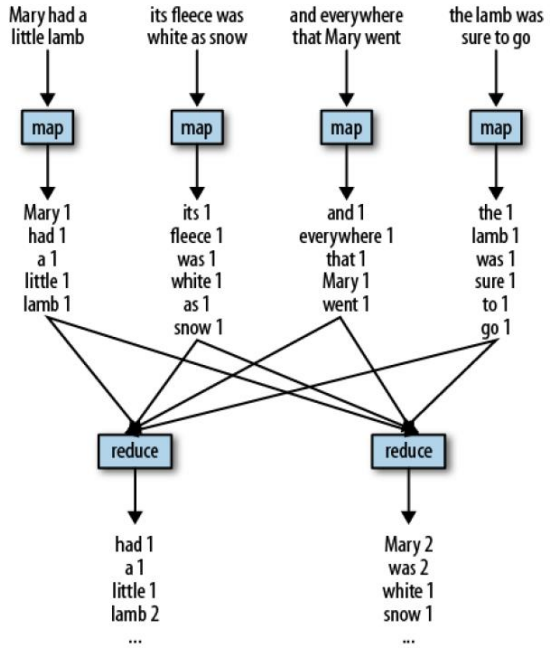


Ilustración 6 - "Hello world" de MapReduce (Gates, 2011)

La fase de map leerá cada línea del texto, de una en una. Dividirá cada palabra en una cadena de texto separada y para cada una emitirá una salida formada por la propia palabra y un '1' indicando que la palabra ha sido contada una vez.

Tras la fase de map, la fase de shuffle reunirá todos los registros con la misma palabra en el mismo *reducer*. La fase de reduce entonces sumará el número de repeticiones de cada palabra y como salida emitirá dicha palabra junto a su suma.

Para el ejemplo se asume que cada línea es enviada a un *map task* diferente y que hay dos *reducers*, el primero para las palabras que empiezan por A-L y el segundo para las que empiezan por M-Z.

En otros casos también es posible que no haya *reduce tasks*. Puede ser apropiado cuando no se necesita shuffle ya que el procesado puede ser llevado a cabo por completo en paralelo. Las únicas transferencias de datos fuera de los nodos serán las escrituras de los *map tasks* en HDFS.

3.3. Análisis de datos

Se necesita analizar gran parte o todo el conjunto de datos JSON almacenados en bruto que contiene HDFS. Para elegir la opción más adecuada se puede realizar una comparación entre los componentes software más comunes para este propósito, Hive (Apache Hive, s.f.), Cascading (Cascading, s.f.) y Pig (Apache Pig, s.f.)

- × *Hive*. Warehouse que proporciona una capa SQL por encima de Hadoop, más apropiado para análisis de datos estructurados tradicionales. Para investigación sobre datos en bruto son mejores los lenguajes procedimentales.
- ✓ *Cascading*. Framework de procesamiento de datos procedimental que permite a los usuarios contruir flujos de datos en Java sobre Hadoop. Usa MapReduce para ejecutar todo su procesamiento de datos. Proporciona una librería de operadores aunque también se pueden contruir unos propios.
- ✓ *Pig*. Motor para ejecutar flujos de datos en paralelo sobre Hadoop. Usa MapReduce para ejecutar todo su procesamiento de datos. Para expresar estos flujos de datos utiliza el lenguaje procedimental Pig Latin el cual incluye operadores tradicionales como join, sort, filter, etc. así como la posibilidad para los programadores de desarrollar sus propias funciones de lectura, procesamiento y escritura de datos.

Tanto Cascading como Pig son adecuados para Twitter Data Collection pero finalmente se ha escogido Pig ya que tiene más compatibilidades con todo el entorno Hadoop, como por ejemplo Elephant-bird y HBase, que se verán en posteriores capítulos.

3.3.1. Precomputación y recomputación: Pig

Con MapReduce el filtrado y la proyección (operador *foreach*) pueden ser implementados trivialmente en la fase de map pero otras operaciones como la de *join* no son proporcionadas y deben ser escritas por el usuario. Apache Pig (Apache Pig, s.f.) proporciona todas las operaciones de procesamiento de datos estándar y otras implementaciones complejas que no proporciona el uso directo de MapReduce. Un trabajo para el que el equipo de Pig puede llegar a tardar meses.

Pig Latin es un lenguaje de flujo de datos. Permite a los usuarios escribir cómo los datos de una o más entradas deben ser leídos, procesados y almacenados en una o más salidas en paralelo.

Pig, al igual que MapReduce, está orientado al procesamiento por lotes de grandes porciones del conjunto total de datos. Pig espera leer todos los registros de un archivo y escribir todas sus salidas secuencialmente.

A continuación se expone un resumen a partir de las explicaciones proporcionadas por (Gates, 2011) comparando el uso de Pig como capa superior por encima de MapReduce con:

- El uso directo de MapReduce.
- Structured Query Language.

	MapReduce	Pig
Optimización y depuración	El procesamiento de datos dentro de las fases de map y reduce es opaco al sistema por lo que MapReduce no tiene oportunidad para optimizar o chequear el código del usuario.	Puede analizar un script Pig Latin y entender el flujo de datos que el usuario está describiendo por lo que puede realizar un chequeo de errores temprano y optimizaciones.
Código y tiempo	Un código que ocupa unas 170 líneas y se tarda cuatro 4 horas para ponerlo en funcionamiento…	…en Pig Latin puede ocupar unas 9 líneas y se tarda alrededor de 15 minutos en escribirlo y depurarlo. Compila los scripts Pig Latin que los usuarios escriben en una serie de uno o más *MapReduce jobs* para ser ejecutados. Cuando se usa Pig no hay necesidad de estar preocupado con las fases de map, shuffle y reduce.
Tipos de algoritmos y rendimiento	Es posible desarrollar algoritmos en MapReduce que no se pueden desarrollar fácilmente en Pig. Adecuado para los algoritmos menos comunes o extremadamente sensibles al rendimiento.	El desarrollador renuncia a un nivel de control. Es mucho más fácil y rápido desarrollar algoritmos más comunes y no tan sensibles al rendimiento.

Tabla 4 - Comparación entre Pig y MapReduce

	SQL	Pig
Acceso	Permite a los usuarios describir qué preguntas quieren que sean contestadas, pero no cómo quieren que sean respondidas.	En Pig Latin el usuario describe exactamente cómo procesar los datos de entrada.
Pipeline de datos	Cuando los usuarios requieren varias operaciones de datos, lo escriben en subconsultas almacenando los datos intermedios en tablas temporales.	Diseñado con una larga serie de operaciones de datos en la que no hay necesidad de tener un conjunto de subconsultas o preocuparse por el almacenamiento de datos en tablas temporales.
Estructura e integridad	Diseñado para el entorno RDBMS, donde los datos están normalizados y se hacen cumplir los esquemas y restricciones adecuados.	Diseñado para el entorno de procesamiento de datos Hadoop, donde los esquemas son a veces desconocidos o inconsistentes. Los datos pueden no estar debidamente limitados y son raramente normalizados. Pig no requiere cargar datos en tablas, puede operar sobre ellos tan pronto como sean copiados en HDFS.

Tabla 5 - Comparación entre Pig y SQL

Usos de Pig:

- Extraer, transformar y cargar (*Extract, Transform and Load*) pipelines de datos. Un ejemplo es el de construir modelos de predicción del comportamiento.
- Investigación con datos en bruto, ya que opera en situaciones donde el esquema es desconocido, incompleto o inconsistente, y porque permite manejar fácilmente datos anidados.
- Modelos de procesamiento iterativo.

Teniedo en cuenta que 'Pig' significa 'Cerdo' en castellano, la filosofía Pig (Apache Pig Philosophy, s.f.) es:

Los cerdos comen de todo. Puede operar sobre datos tanto si tienen sus metadatos como si no. Puede operar sobre datos relacionales, anidados o sin estructura.

Los cerdos viven en cualquier lugar. Pig pretende ser un lenguaje para el procesamiento de datos en paralelo. No está atado a ningún framework.

Los cerdos son animales domésticos. Está diseñado para ser fácilmente controlado y modificado por sus usuarios. Permite integración de código que pueda ser compilado bajo Java. Además Pig tiene un optimizador que reordena algunas operaciones de scripts Pig Latin para dar mayor rendimiento, combinar *MapReduce jobs*, etc.

Los cerdos pueden volar. Pig procesa los datos rápidamente. El equipo de Pig no implementa características que hagan que el cerdo pese más y no pueda volar (sí, efectivamente es un cerdo volador)

3.3.1.1. Librería Twitter multiusos: Elephant-bird

Con Pig se puede realizar el parseo inicial de los tweets en formato JSON contenidos en HDFS separando los campos que contiene cada tweet y tiparlos. El problema reside en que es una tarea tediosa y la User Defined Function (UDF) que viene con Pig para ejecutarla requiere que los campos de cada tweet estén siempre en el mismo orden de aparición, algo que ha ido cambiando según ha evolucionado Twitter API. Para que el usuario no tenga que preocuparse por este asunto y pueda dedicar su esfuerzo en el análisis propiamente dicho se puede utilizar la librería Pig LoadFuncs de Elephant-bird, que traduce cada tweets JSON de manera anidada para que todos y cada uno de sus campos puedan ser accedidos por Pig de manera mucho más sencilla.

Elephant-bird (Elephant-bird, s.f.) es una librería Twitter de LZO, Thrift, protocolos de Hadoop buffer, Pig LoadFuncs, Hive SerDe, HBase, etc. La mayoría en producción dentro de Twitter.

Compatibilidad de versiones que afecta a Twitter Data Collection:

- Hadoop 20.2x, 1.x, 2.x
- Pig 0.8+
- Thrift 0.5.0, 0.6.0, 0.7.0

A veces al leer documentaciones de software que avanzan día a día existen detalles que no quedan del todo claros. Por ejemplo, en la sección *Version compatibility* de (Elephant-bird, s.f.) no se indica explícitamente qué software es necesario instalar si lo

que se pretende es utilizar la librería correspondiente a Pig LoadFuncs. Una posible consecuencia es no saber por qué falla el proceso de *building*.

Pregunta a través de Twitter a Kevin Weil: *Vice President of Product for Revenue at Twitter* (Kevin Weil, s.f.) y creador del repositorio de Elephant-bird en GitHub.

Ilustración 7 - Preguntando a Kevin Weil sobre Thrift (Gil Pérez & Weil, 2013)

Para instalar correctamente Elephant-bird y utilizar Pig LoadFuncs es necesario instalar Thrift (Apache Thrift, s.f.). El framework software Thrift, para el desarrollo de servicios escalables inter-lenguaje, combina una pila de software con un motor de generación de código para construir servicios que funcionan de manera eficiente y sin problemas entre varios lenguajes de programación.

3.4. Lectura aleatoria de datos en tiempo razonable

Hasta este punto, los archivos de texto en bruto almacenados en HDFS contienen tweets en formato JSON que son analizados por Pig con ayuda de Elephant-bird. Tanto MapReduce como Pig no son buenas elecciones para cargas de trabajo que requieren la lectura o escritura de un pequeño grupo de archivos o buscar muchos registros diferentes en orden aleatorio ya que estan orientados al procesamiento por lotes de grandes porciones del conjunto total de datos. Componentes que son adecuados para estos usos son las llamadas bases de datos no relacionales (NoSQL)

Los resultados que proporciona Pig se almacenarán en estas bases de datos NoSQL que guardarán la información resultante para cada análisis y así poderse realizar consultas rápida y eficazmente sobre registros de datos aleatorios.

Estas bases de datos no esperan que los datos sean normalizados y son necesarios pocos *joins* o ninguno.

Para elegir la opción más adecuada se puede realizar una comparación entre los componentes software más comunes para este propósito, ElephantDB (Marz, ElephantDB, s.f.), Voldemort (Project Voldemort, s.f.), Riak (Basho Technologies, s.f.), MongoDB (MongoDB, s.f.), Cassandra (Apache Cassandra, s.f.) y HBase (Apache HBase, s.f.)

- × *ElephantDB* y *VoldemortDB*. Actualmente no existe la posibilidad de almacenar en estas bases de datos la información precomputada por Pig.
- × *Riak*. Baja integración con Pig.
- × *MongoDB*. Los casos de uso de Hadoop y MongoDB no se ajustan del todo bien a la arquitectura elegida para Twitter Data Collection (Hadoop and MongoDB Use Cases, s.f.)
- ✓ *Cassandra*. Base de datos integrada con Pig. Posee un modelo de consistencia fuerte o eventual (los servidores pueden tener diferentes valores para mismos datos durante un periodo de tiempo), y está optimizada de cara a escrituras.
- ✓ *HBase*. Base de datos integrada con Pig, contiene una libería para realizar lecturas y escrituras masivas de Pig a HBase.

Tanto Cassandra como HBase son adecuados para Twitter Data Collection pero finalmente se ha escogido HBase ya que en este caso se le da más importancia a la optimización de lecturas que de escrituras. Para entrar en más detalle sobre la comparación ver (Gupta, 2012) y (Henschen, 2013)

3.4.1. Almacenamiento: HBase

A continuación se expone un resumen a partir de las explicaciones proporcionadas por (George, 2011) comparando Apache HBase (Apache HBase, s.f.) con un Relational DataBase Management System tradicional.

	RDBMS tradicional	**HBase**
Rendimiento	Adecuado para procesamiento transaccional.	Adecuado para el procesamiento analítico a gran escala.
Características	Las esperas y *deadlocks* se incrementan de manera no lineal con el tamaño de las transacciones y concurrencia. Los RDBMS solucionan muchos de estos problemas pero normalmente son especializados y solo cubren ciertos aspectos.	Implementa de por sí catacteristicas relacionales de redimiento permanentemente. Se puede denormalizar el modelo de datos y evitar esperas y *deadlocks* minimizando los *lockings* necesarios. Además realiza un escalado horizontal sin la necesidad de repartición según aumentan los datos.

Tabla 6 - Comparación entre HBase y RDBMS tradicional

Los valores de las columnas posteriores son almacenados contiguamente en disco. Esto difiere del enfoque orientado a filas de las bases de datos tradicionales, las cuales almacenan filas enteras de manera contigua.

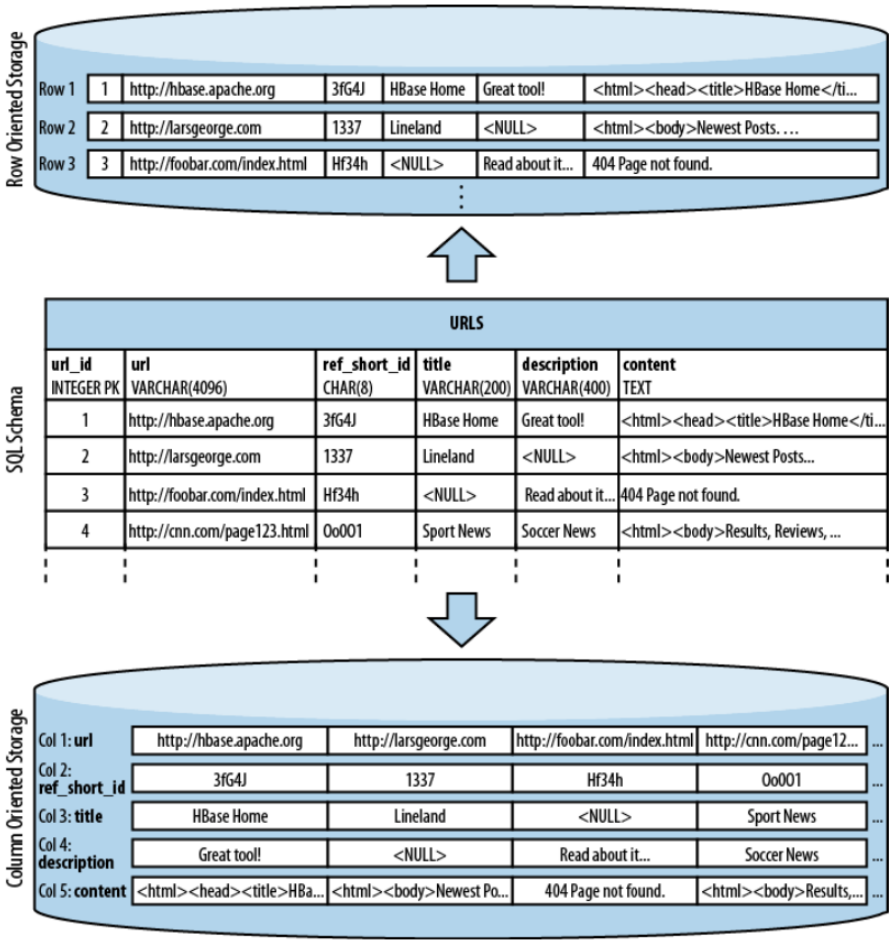


Ilustración 8 - Diseño de almacenameinto orientado a filas y a columnas (George, 2011)

La razón para almacenar valores en un diseño por columna se basa en la suposición de que, para determinadas consultas, no todos los valores son necesarios. Este es a menudo el caso de bases de datos analíticas. La reducción de I/O y mayor adecuación para la compresión son los principales estandartes para este nuevo diseño, ya que los valores de una columna son a menudo muy similares al contrario que los valores heterogéneos de un registro en una base de datos basado en filas.

3.4.1.1. Diseño de tablas

Una o más columnas forman una fila que es direccionada de manera única por una *rowkey*. Cada columna puede tener múltiples versiones, cada una con valor distinto contenido en celdas separadas. Todas las filas se ordenan siempre lexicográficamente por su *rowkey* comparada a nivel binario, byte a byte, de izquierda a derecha.

Teniendo las *rowkeys* siempre ordenadas se obtiene algo parecido al índice de clave primaria de las bases de datos RDBMS. Cada *rowkey* es única por lo que si se repitiera dicha clave lo que se estaría haciendo es actualizar la misma fila. Además HBase añade soporte para índices secundarios.

Las filas están compuestas de columnas, y éstas a su vez están agrupadas en *column families*. Todas las columnas de una familia de columnas son almacenadas juntas en el mismo archivo de almacenamiento de llamado HFile. Las *column families* necesitan ser definidas cuando se crea la tabla y no deberían ser cambiadas demasiado. En la práctica

es recomendable que haya un número de *column families* mucho menor a unas pocas decenas. No hay un límite para el número de columnas ni para la longitud de sus valores.

Filas y columnas no están organizadas como en el modelo clásico de hoja de cálculo, sino que la información está disponible bajo un tag específico. Para una base de datos con un esquema fijo hay que almacenar NULLs como valores pero en HBase simplemente se omite la columna por lo que no ocupan espacio de almacenamiento.

Cada valor de columna, cada celda, tiene un timestamp. Las celdas pueden tener múltiples versiones y diferentes columnas pueden haber sido escritan en tiempos diferentes. El API, por defecto, muestra todas las columnas seleccionando la versión más reciente de cada celda.

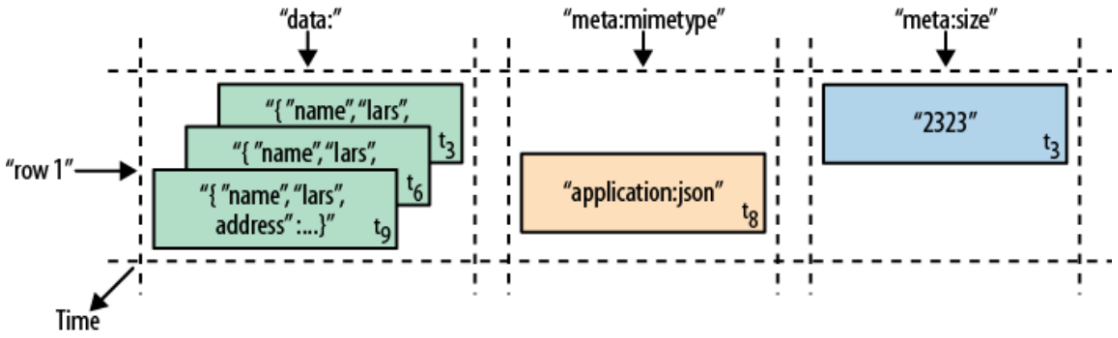


Ilustración 9 - Vista temporal de las partes de una fila (George, 2011)

El acceso a una fila es atómico. No hay más garantía transaccional que abarque múltiples filas o tablas. El acceso atómico es un factor añadido para que la arquitectura sea estrictamente consistente, ya que cada lector y escritor concurrentes pueden hacer suposiciones seguras sobre el estado de una fila.

Para el escalado a menudo se requiere un esquema de diseño diferente, un buen término para describir este principio es *Denormalization, Duplication, and Intelligent Keys*. Normalización es una técnica de las bases de datos relacionales donde cada tipo de información se almacena siempre en un único lugar específico. Así no hay que actualizar o borrar todas las copias de ese dato. Los datos se combinan al realizar las consultas mediante *joins* por lo que el rendimiento aumenta para escrituras. Denormalización es el concepto opuesto ya que los datos se repiten y almacenan en múltiples localizaciones. De este modo consultar los datos es más fácil y mucho más rápido ya que no se necesitan *joins* por lo que el rendimiento aumenta para lecturas.

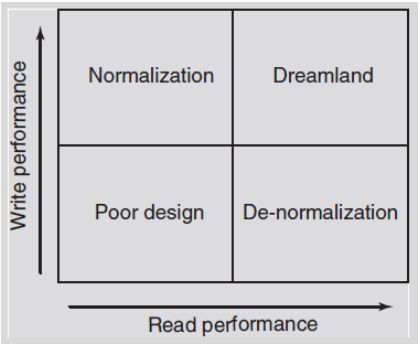


Ilustración 10 - Rendimientos de normalización y denormalización (Dimiduk & Khurana, 2013)

El diseño de tablas altas y estrechas o planas y anchas está relacionado con qué datos se quiere indexar. Las búsquedas de claves parciales son posibles, y cuando se combinan con claves compuestas, tienen las mismas propiedades que índices avanzados en RDBMS.

Utilizar el algoritmo de reducción croiptográfico MD5 para formar una *rowkey* tiene dos beneficios:

- Conseguir una longitud constante de 16 bytes proporcionando previsibilidad de rendimiento en lectura y escritura.
- No se necesita delimitar claves con caracteres como '-' para separar *rowkeys* compuestas por varios campos de longitud constante.

Además, utilizando MD5 al comienzo de una *rowkey* se consigue mejor distribución ya que se evita el hot-spotting, una alta concentración de carga en un pequeño subconjunto de regiones debido al ordenamiento que realiza HBase según las *rowkeys*. Lo deseable es que la carga esté distribuida por todo el cluster. El rendimiento total se vería afectado por el cuello de botella en los servidores que sirven estas regiones.

Dependiendo de las partes de una tabla que se especifiquen en una consulta se limitará la cantidad de datos a leer o transferir. La siguiente ilustración mezcla ilustraciones de (Dimiduk & Khurana, 2013) y (George, 2011) mostrando la relación opuesta entre rendimiento y cardinalidad.

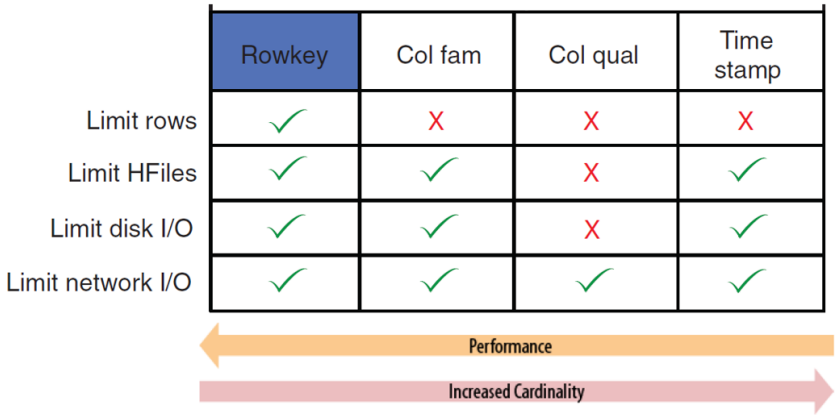


Ilustración 11 - Rendimiento y cardinalidad HBase key (Dimiduk & Khurana, 2013), (George, 2011)

3.4.1.2. Auto-Sharding, partición horizontal automática

Una región es la unidad básica de escalabilidad y balanceo de carga in HBase. Las regiones son rangos contiguos de filas almacenadas juntas. El sistema las divide dinámicamente cuando son demasiado grandes. Inicialmente solo hay una región por tabla, la cual crece hasta que llega al tamaño máximo configurado y entonces se divide por la clave central separándose en dos regiones aproximadamente iguales.

Cada región es servida por un *region server* que a su vez puede servir a muchas regiones al mismo tiempo. Para HBase, con hardware moderno se estima que el número recomendable de regiones por servidor sea de entre 10 y 1000, cada uno con un tamaño comprendido entre 1 y 2 GB.

Las regiones permiten:

- Recuperación rápida cuando un servidor falla.
- Balance de carga de grado fino gracias a que pueden moverse entre servidores cuando uno de ellos está saturado, se ha producido un fallo y queda fuera de servicio.

3.4.1.3. Coordinación distribuida altamente fiable: ZooKeeper

Desde la versión 0.20.x, HBase usa Apache ZooKeeper (Apache ZooKeeper, s.f.) como su servicio de coordinación distribuido. Ofrece acceso como si de un sistema de archivos se tratara, con directorios y archivos que los sistemas distribuidos pueden utilizar para negociar la propiedad, registrar servicios o comprobar si existen actualizaciones.

3.4.2. Consultas: Hive

Muchos programadores y responsables de *Business Intelligence* en empresas desean un lenguaje que les sea conocido para así facilitarles el acceso a la información. Apache Hive (Apache Hive, s.f.) proporciona un lenguaje SQL, el que las herramientas de BI esperan para la toma de datos.

Para Twitter Data Collection, la característica de Hive que interesa es la que permite crear consultas estilo SQL sobre HBase. De cara a un usuario final solo se deben realizar consultas a las bases de datos, que almacenan resultados de diferentes análisis, utilizando un lenguaje comúnmente usado.

Hive metastore (AdminManual MetastoreAdmin, s.f.) es una base de datos relacional donde Hive persiste esquemas de tablas y otros metadatos del sistema. Para este propósito se pueden utilizar MySQL (MySQL, s.f.) y PostgreSQL (PostgreSQL, s.f.). Para Twitter Data Collection se utilizará MySQL debido a que existe mayor documentación para su uso como Hive metastore y por el mayor uso que hacen de él las empresas.

Existen tres modos de metastore:

- *Embedded mode*. Recomendado solo para uso experimental.
- *Local mode*. Cada Hive Client abrirá una conexión a la base de datos y realizará las consultas. El servidor debe ser accesible desde las máquinas que ejecutan las

consultas. La librería del cliente JDBC debe figurar en el classpath del Hive Client.

- *Remote mode*. Todos los clientes Hive realizarán conexiones a un servidor metastore que a su vez consulta la base de datos por los metadatos. El servidor y el cliente metastore se comunican usando el protocolo Thrift.

Para más información ver (AdminManual MetastoreAdmin, s.f.) y (Configuring the Hive Metastore, s.f.)

Tanto para el cluster mono-nodo como para el cluster multi-nodo Twitter Data Collection usará Hive metastore en modo local por simplicidad, ya que el servidor es accesible desde las máquinas que ejecutan las consultas.

4. Twitter Data Collection: cluster mono-nodo

El proyecto *TwitterDataCollection* se ha desarrollado como un Project Object Model (POM Reference, s.f.). Es una representación XML de un proyecto Maven (Apache Maven, s.f.) contenido en un archivo llamado *pom.xml*. El archivo POM del proyecto contiene módulos, que son otros proyectos cuyas acciones se realizan en bloque. *TwitterDataCollection* contiene tres módulos: *FlumeTwitterSource*, *PigTwitterUDFs* y *HBaseTwitterTables*.

twitter-data-collection/pom.xml

```
    <modules>
        <module>flume-twitter-source</module>
        <module>pig-twitter-udfs</module>
        <module>hbase-twitter-tables</module>
    </modules>
```

Estas tres son aplicaciones Java, cada una basada en proyecto Maven, por lo que son aplicaciones Java que contienen cada una otro archivo *pom.xml* indicando, entre otros, dependencias software necesarias para desarrollar cada módulo sin necesidad de descargar a mano cada una de ellas.

Con el fin de realizar una instalación y puesta en marcha lo más automática posible, *TwitterDataCollection* proporciona un shell script, *tdc.sh*, a ejecutar en un sistema operativo Ubuntu. Los scripts desarrollados para cada componente, las indicaciones de los siguientes capítulos y la verificación de que el proyecto funciona se han realizado utilizando Ubuntu 12.04 LTS Precise Pangolin, desktop install Intel x86, (Ubuntu 12.04 LTS - Precise Pangolin, s.f.)

A continuación se exponen los pasos para instalar *TwitterDataCollection* en un cluster mono-nodo, es decir, un cluster que dispone únicamente de una máquina. Este modo es recomendable para empezar a desarrollar.

Todo el código fuente desarrollado puede verse en el anexo Otros scripts y código fuente.

4.1. Prerrequisitos

Colocar el directorio *twitter-data-collection* en */usr/local* y cambiar los permisos del proyecto para que todo usuario pueda manipularlo:
superuser@hostname:~$ sudo chmod -R 777 /usr/local/twitter-data-collection

Este proyecto ha sido desarrollado con NetBeans IDE (NetBeans, s.f.) y por su facilidad de uso se puede utilizar esta herramienta para ejecutar las acciones *clean and build* pulsado con el botón secundario del ratón sobre *TwitterDataCollection* y seleccionado "Clean and Build". Estas acciones se realizarán para las tres aplicaciones Java.

Comando para instalar y configurar los prerrequisitos:
superuser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh -single pres

Se instalarán:

- JDK 7, (Java Community Help Wiki, s.f.)
  (Install Oracle Java 7 in Ubuntu via PPA Repository, s.f.)
- ubuntu-restricted-extras (RestrictedFormats, s.f.)
- build-essential (CompilingEasyHowTo, s.f.)
- openssh-server (SSH/OpenSSH/InstallingConfiguringTesting, s.f.)
- maven (Apache Maven, s.f.)
- vim (VimHowTo, s.f.)
- Herramientas y librerías requeridas para construir e instalar el compilador de Apache Thrift, necesario para Elephant-bird.
- MySQL como Hive metastore, su proveedor y su conector.

MySQL pedirá responder a una serie de preguntas para la configuración inicial:
Enter current password for root (enter for none): ***<introducir contraseña MySQL>***
Change the root password? [Y/n] ***n***
Remove anonymous users? [Y/n] ***y***
Disallow root login remotely? [Y/n] ***n***
Remove test database and access to it? [Y/n] ***y***
Reload privilege tables now? [Y/n] ***y***

Además, tal y como indica (Noll, Running Hadoop on Ubuntu Linux - Single-Node Cluster, 2011) la creación de un nuevo usuario dedicado para el entorno Hadoop no es indispensable pero ayuda a separar la instalación de los componentes de otras aplicaciones software y cuentas de usuario ejecutándose en la misma máquina. Para la creación de este nuevo usuario llamado *hduser* será necesario introducir una contraseña y una serie de parámetros opcionales en el momento en el que el script ejecute los comandos oportunos. Una vez finalizada la instalación de todos los prerrequisitos se podrá acceder al nuevo usuario *hduser* de la siguiente manera.

Cambiar al nuevo usuario *hduser*:
superuser@hostname:~$ su - hduser

Volver al superusuario original:
hduser@hostname:~$ exit

Además se añadirán a */home/hduser/.bashrc* las variables de entorno necesarias, ver anexo Otros scripts y código fuente.

Por último se abrirán automáticamente 4 archivos de configuración del sistema operativo mediante el edior Vim que deberán ser modificados. Son los que se indican a continuación.

4.1.1. Loopback IP

HBase 0.94.0 y versiones anteriores esperan la dirección de loopback IP en 127.0.0.1 pero en Ubuntu por defecto es 127.0.1.1 y esto podría causar problemas. Se resuelve desde un principio para que todo el entorno Hadoop utilice esta dirección, proporcionando retrocompatibilidad y así evitando problemas.

/etc/hosts

127.0.0.1	localhost
#127.0.1.1	*hostname*
127.0.0.1	*hostname*

4.1.2. Configuración del kernel

HBase es una base de datos NoSQL corriendo sobre Hadoop y, como otras bases de datos, mantiene muchos archivos abiertos al mismo tiempo. En Linux la configuración *nofile* especifica el número por defecto de descriptores de archivo que cada proceso puede abrir, por defecto 1024.

hduser@hostname:~$ ulimit -n
1024

La configuración *nproc* especifica el máximo número de procesos que pueden existir simultáneamente para un usuario, por defecto 32072. Si fuera demasiado bajo se produciría un *OutOfMemoryError*.

hduser@hostname:~$ ulimit -u
32072

Siguiendo la recomendación de (Jiang, 2012) un buen número de descriptores de archivo por cada proceso podría ser 65535 dejando el número máximo de procesos con el mismo valor pero indicándolo de manera explícita por si en algún momento fuera necesario incrementarlo.

/etc/security/limits.conf

hduser	soft	nofile	65535
hduser	hard	nofile	65535
hduser	soft	nproc	32072
hduser	hard	nproc	32072

Para aplicar los cambios se añade la siguiente línea a */etc/pam.d/common-session*

session required	pam_limits.so

Ahora se puede comprobar que la nueva configuración ha sido aplicada.

hduser@hostname:~$ ulimit -n
65535

hduser@hostname:~$ ulimit -u
32072

Linux mueve las páginas de memoria que no han sido accedidas durante un tiempo al espacio de intercambio, *swap space*, incluso cuando hay suficiente memoria disponible. En otras circunstancias esto es necesario pero JVM no se comporta bien ante el

swapping. HBase podría dar problemas, uno típico es la expiración de sesión de ZooKeeper. Los procesos de HBase consumen gran cantidad de memoria por lo que tener un valor alto de *swapping* disminuirá drásticamente la recolección de basura.

Para evitar el *swapping* tanto como sea posible se debe indicar en */etc/sysctl.conf*

```
vm.swappiness=0
```

Es altamente recomendable reiniciar el equipo al llegar a este punto.

4.2. SSH

hduser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh -single ssh

Hadoop requiere comunicación entre múltiples procesos de uno o más hosts. Se necesita que *hduser* pueda conectarse a cada host sin necesidad de utilizar una contraseña cada vez que requiera dicha comunicación.

Para el cluster mono-nodo hay que configurar el acceso SSH a localhost para *hduser*. Se generan un par de claves RSA sin contraseña para que no tener que introducirla siempre que Hadoop interaccione con sus nodos (en este caso solo uno)

Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa): ***<Pulsar ENTER>***
Created directory '/home/hduser/.ssh'.
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
The key fingerprint is:
[…]

Posteriormente se añade también de manera automática la nueva clave pública a la lista de claves autorizadas.

Se puede hacer una prueba realizando una conexión mediante SSH a localhost confirmando que la conexión sin contraseña funciona. La primera vez aparecerá un aviso sobre si se quiere confiar en el certificado del host.

hduser@hostname:~$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is …
Are you sure you want to continue connecting (yes/no)? ***yes***
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
[…]

hduser@hostname:~$ exit
logout
Connection to localhost closed.

4.3. Instalación

superuser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh -single install

Se crearán los directorios del entorno Hadoop en */usr/local/hadoop-environment*

Los archivos de configuración originales se reemplazarán automáticamente por otros modificados para un correcto funcionamiento en un cluster mono-nodo. Las modificaciones más importantes de la versión instalada serán mostradas en el apartado correspondiente de cada componente.

En Hadoop los directorios de almacenamiento para HDFS están por defecto bajo el directorio temporal. Algunas distribuciones Linux, como Ubuntu, borran todo el contenido de esta carpeta al reiniciar el sistema operativo. Lo mismo ocurre con los directorios de almacenamiento por defecto de ZooKeeper. Por ello se crean nuevos directorios en paths persistentes donde Hadoop y ZooKeeper puedan almacenar sus ficheros.

4.4. Inicialización

hduser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh init

Acciones de inicialización:

- Hadoop: formateo de HDFS via *NameNode*.
- Creación de directorios dentro de HDFS necesarios para usar Hive.

Tras cada arranque de Hadoop puede que se deba esperar unos segundos si el *NameNode* está en *safe mode*. El mensaje de error es del estilo:
mkdir: org.apache.hadoop.hdfs.server.namenode.SafeModeException: Cannot create directory /tmp. Name node is in safe mode. […]

Arrancar Hadoop y HBase:
hduser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh start

Ejecutando Java Virtual Machine Process Status Tool se puede comprobar que los procesos necesarios para manejar el proyecto están activos.
hduser@hostname:~$ jps

```
Jps
HQuorumPeer
TaskTracker
JobTracker
HRegionServer
SecondaryNameNode
HMaster
NameNode
DataNode
```

Ilustración 12 - Procesos ejecutándose en el cluster mono-nodo

Existen múltiples interfaces web para los componentes. Las más utilizadas durante el proyecto han sido las siguientes.

Para Hadoop:

- NameNode: http://localhost:50070
- JobTracker: http://localhost:50030
- TaskTracker: http://localhost:50060

Para HBase:

- HMaster: http://localhost:60010

Cuando se desee parar HBase y Hadoop:
hduser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh stop

4.5. Batch layer

4.5.1. Hadoop 1.2.1

Los cambios en las configuraciones se realizan automáticamente gracias al script proporcionado. A continuación se exponen dichas modificaciones cuyas descripciones están dentro de cada archivo.

/usr/local/hadoop-environment/hadoop/conf/hadoop-env.sh

```
# The java implementation to use.  Required.
export JAVA_HOME=/usr/lib/jvm/java-7-oracle

# Extra Java runtime options.  Empty by default.
# export HADOOP_OPTS=-server
# Disabling IPv6 only for Hadoop.
export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true
```

/usr/local/hadoop-environment/hadoop/conf/core-site.xml

```
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/usr/local/hadoop-environment/var/hadoop</value>
        <description>A base for other temporary
directories.</description>
    </property>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://localhost:9000</value>
        <description>
            The name of the default file system.
            A URI whose scheme and authority determine the
FileSystem implementation.
            The uri's scheme determines the config property
(fs.SCHEME.impl) naming the FileSystem implementation class.
            The uri's authority is used to determine the host, port,
etc. for a filesystem.
        </description>
    </property>
```

/usr/local/hadoop-environment/hadoop/conf/mapred-site.xml

```
    <property>
        <name>mapred.job.tracker</name>
        <value>localhost:9001</value>
        <description>
            The host and port that the MapReduce job tracker runs
at.
            If "local", then jobs are run in-process as a single map
and reduce task.
        </description>
    </property>
```

/usr/local/hadoop-environment/hadoop/conf/hdfs-site.xml

```
    <property>
        <name>dfs.replication</name>
        <value>1</value>
        <description>
            Default block replication.
            The actual number of replications can be specified when
the file is created.
            The default is used if replication is not specified in
create time.
        </description>
    </property>
```

4.5.2. Flume 1.5.0 y Twitter4J 3.0.5

Los cambios en las configuraciones se realizan automáticamente gracias al script proporcionado. A continuación se exponen dichas modificaciones.

/usr/local/hadoop-environment/flume/conf/flume-env.sh

```
# Enviroment variables can be set here.
JAVA_HOME=/usr/lib/jvm/java-7-oracle

# Note that the Flume conf directory is always included in the classpath.
FLUME_CLASSPATH="/usr/local/twitter-data-collection/flume-twitter-source/target/flume-
twitter-source-1.0-SNAPSHOT.jar"
```

Es necesario que FLUME_CLASSPATH incluya el jar de *FlumeTwitterSource*, el cual utiliza Twitter4J como librería para interaccionar con Streaming API.

Partiendo de la aplicación de ejemplo (cdh-twitter-example, s.f.) para (CDH, s.f.) desarrollado por (Cloudera, s.f.) se pueden modificar los archivos del directorio *flume-sources* para lograr los propósitos deseados que requiere *TwitterDataCollection*.

Para el proyecto se han diseñado tres maneras de realizar la petición a través de Twitter4J, que a su vez Flume gestionará para introducirlos en HDFS. Se diferencian mediante la petición, adquisición y almacenamiento de tweets…

1. …geolocalizados.
2. …que contienen alguna palabra clave.
3. …geolocalizados o que contienen alguna palabra clave.

Ver anexo Otros scripts y código fuente.

Hay que conocer la heurística con la que Twitter API selecciona los tweets geoloalizados (Streaming API request parameters - locations, 2014):

1. Si el campo *coordinates* está poblado, se comprueba si sus valores están dentro de la caja de entorno geográfica indicada.
2. Si el campo *coordinates* está vacío pero *place* está poblado, se comprueba que la región definida en *place* intersecciona con la caja de entorno geográfica indicada.
3. Si en ninguno de los puntos anteriores se ha encontrado geolocalización, el tweet no posee coordenadas ya que el campo *geo* está obsoleto.

Aunque *geo* esté obsoleto es necesario tenerlo en cuenta si se van a manejar archivos con tweets antiguos.

4.5.2.1. FlumeTwitterSource: tweets geolocalizados

Para delimitar una zona del mapa de la que adquirir tweets con la dependencia *twitter4j-stream-*.jar* se deben indicar como argumentos los puntos sudoeste y noreste con los que crear la caja de entorno geográfica, y por ello se añaden en el archivo de configuración. Como se puede ver en la siguiente ilustración realizada con la ayuda de (iTouchMap, s.f.), en el rectángulo rojo que se ha creado a partir de las dos coordenadas utilizadas no solo entran España peninsular e Islas Baleares sino también la mayor parte de Portugal, el sur de Francia, un trozo de Argelia y una cantidad no despreciable de áreas marítimas/ocenánicas (los marineros también tuitean). Los tweets de coordenadas no deseadas habría que descartarlos más adelante al realizar las precomputaciones.

Ilustración 13 - Caja de entorno geográfica de la que adquirir tweets

Un *agent* se puede iniciar mediante el script proporcionado, un ejemplo es el que ingiere tweets procedentes de España peninsular e Islas Baleares:
hduser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh flume spain-pb

El comando que ejecutará el script es el siguiente:
hduser@hostname:/usr/local/hadoop-environment/flume$ bin/flume-ng agent --conf conf/ -f /usr/local/twitter-data-collection/flume-twitter-source/tdc1.conf -Dflume.root.logger=DEBUG,console -n TwitterSpainPB

Esto indica:

- --conf : directorio de archivos de configuración.
- -f : archivo de configuración específico para la ejecución.
- -n : nombre del agente.

/usr/local/twitter-data-collection/flume-twitter-source/tdc1.conf

```
# Name the components on this agent
TwitterSpainPB.sources = Twitter
TwitterSpainPB.channels = MemChannel
TwitterSpainPB.sinks = HDFS

# Describe/configure the source
TwitterSpainPB.sources.Twitter.type = source.TwitterSource
TwitterSpainPB.sources.Twitter.channels = MemChannel
TwitterSpainPB.sources.Twitter.consumerKey = <required_1>
TwitterSpainPB.sources.Twitter.consumerSecret = <required_1>
TwitterSpainPB.sources.Twitter.accessToken = <required_1>
TwitterSpainPB.sources.Twitter.accessTokenSecret = <required_1>
TwitterSpainPB.sources.Twitter.swLngLat = -9.299269, 35.999882
TwitterSpainPB.sources.Twitter.neLngLat = 4.327812, 43.79142

# Describe the sink
TwitterSpainPB.sinks.HDFS.channel = MemChannel
TwitterSpainPB.sinks.HDFS.type = hdfs
TwitterSpainPB.sinks.HDFS.hdfs.path = hdfs://localhost:9000/user/hduser/flume/twitter/spain-pb/%Y/%m/%d/%H
TwitterSpainPB.sinks.HDFS.hdfs.fileType = DataStream
TwitterSpainPB.sinks.HDFS.hdfs.writeFormat = Text
TwitterSpainPB.sinks.HDFS.hdfs.batchSize = 10000
TwitterSpainPB.sinks.HDFS.hdfs.rollSize = 0
TwitterSpainPB.sinks.HDFS.hdfs.rollCount = 1000

# Use a channel which buffers events in memory
TwitterSpainPB.channels.MemChannel.type = memory
TwitterSpainPB.channels.MemChannel.capacity = 10000
TwitterSpainPB.channels.MemChannel.transactionCapacity = 10000
```

Esta configuración define un agente llamado TwitterSpainPB que tiene:

 # Name the components on this agent

- *Sources*: un *source* llamado Twitter.
- *Sinks*: un *sink* llamado HDFS.
- *Channels*: un *channel* llamado MemChannel.

 # Describe/configure the source

- Type: nombre del tipo de componente, debe ser su Fully-Qualified Class Name.
- Channels: conectado con MemChannel.
- consumerKey, consumerSecret, accessToken y accessTokenSecret son las claves que se deben obtener de Twitter Developers. Ver anexo Obtención de claves en Twitter Developers.
- swLngLat y neLngLat: longitud y latitud (en ese orden) de los puntos sudoeste y noreste respectivamente, los cuales delimitaran la caja de entorno geográfica de la que Streaming API devolverá tweets.

 # Describe the sink

- Channel: conectado con MemChannel.
- Type: de tipo HDFS.
- Su path será *hdfs://localhost:9000/user/hduser/flume/twitter/spain-pb/%Y/%m/%d/%H* Este path se creará automáticamente siendo variables el año (%Y), el mes (%m), el día (%d) y la hora (%H) en el que se ingesta cada tweet. Para Streaming API tomar la fecha y hora en la que se ingesta el tweet es una aproximación correcta ya que el flujo de datos es en tiempo real.
- Filetype DataStream: flujo de datos contínuo en formato texto.
- rollInterval: segundos a esperar antes de cambiar de archivo en el que almacenar información entrante, por defecto 30 (0 = cambio de archivo no basado en intervalo de tiempo)
- rollSize: tamaño que debe tener el archivo para cambiar de fichero en el que almacenar información entrante (0 = cambio de archivo no basado en su tamaño)
- rollCount: número de eventos escritos en un archivo antes de cambiar a uno nuevo, por defecto 10 (0 = cambio de archivo no basado en número de eventos)
- batchSize: número de eventos escritos antes de que sean llevados a HDFS.

 # Use a channel which buffers events in memory

- Type: De tipo *memory channel*.
- Capacidad total y capacidad de transacción.

Más información en (Flume User Guide, s.f.)

Los parámetros especificados se pueden modificar y añadir en tiempo de ejecución. Una vez modificados, transcurrido un corto periodo de tiempo, automáticamente Flume actualizará el comportamiento de los *sources*, *channels* y *sinks* del agente.

Los valores numéricos de las variables de cada archivo de configuración son orientativos. Dependiendo del hardware en el que se ejecute se deben probar otros valores para evitar errores de *Java Heap Space* y o *Garbage Collector*.

4.5.2.2. FlumeTwitterSource: tweets con palabras clave

Este es un *Flume agent* que ingiere tweets devueltos por Streaming API que contienen una o varias de las palabras clave indicadas en el archivo de configuración.

El agente se puede iniciar mediante el script proporcionado:
hduser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh flume keywords

El comando que ejecutará el script es el siguiente:
hduser@hostname:/usr/local/hadoop-environment/flume$ bin/flume-ng agent --conf conf/ -f /usr/local/twitter-data-collection/flume-twitter-source/tdc4.conf -Dflume.root.logger=DEBUG,console -n TwitterKeywords

Cambios:

- Cambio de nombre del agente, ahora TwitterKeywords.

- Obtener nuevas claves de Twitter Developers para no cortar la conexión del anterior agente.
- Cambio de los puntos geográficos por una cadena de texto con palabras separadas por coma. Los tweets adquiridos contendrán alguna de éstas palabras.
- Nuevo path.

Las palabras clave pueden ser nombres de usuario, hashtags u otras palabras normales. Las tildes o mayúsculas/minúsculas no son importantes ya que Twitter se encarga de buscar dichas palabras con y sin tilde, tanto en mayúsculas como en minúsculas.

/usr/local/twitter-data-collection/flume-twitter-source/tdc4.conf

TwitterKeywords.sources.Twitter.consumerKey = <**required_2**> TwitterKeywords.sources.Twitter.consumerSecret = <**required_2**> TwitterKeywords.sources.Twitter.accessToken = <**required_2**> TwitterKeywords.sources.Twitter.accessTokenSecret = <**required_2**> **TwitterKeywords.sources.Twitter.keywords = @el_pais, @elmundoes, @abc_es, @larazon_es, @publico_es, @LaVanguardia, @elperiodico, @elconfidencial, @la_informacion, @eldiarioes, @voz_populi, @meneame_net, @El_Plural, @ElHuffPost, @elecodiario, @policia, #VOST, manifestación, abuso, 15M, corrupción, accidente, independencia, #Mundial, #MundialBrasil, #Mundial2014, #Brasil2014, #MundialBrasil2014, Mundial, fútbol**
TwitterKeywords.sinks.HDFS.hdfs.path = hdfs://localhost:9000/user/hduser/flume/twitter/**keywords**/%Y/%m/%d/%H

4.5.2.3. FlumeTwitterSource: tweets geolocalizados o con palabras clave

Solo hay que añadir al archivo de configuración con el que se ejecuta el *Flume agent* los argumentos de los dos agentes anteriores. El código fuente no hace falta que se modifique ya que tal y como explica (Streaming API request parameters - locations, 2014) hay que tener en cuenta que la petición realizada es del tipo "geolocalizados OR contienenPalabras" debido a cómo funciona Streaming API.

El agente se puede iniciar mediante el script proporcionado:
hduser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh flume spainpb-keywords

El comando que ejecutará el script es el siguiente:
hduser@hostname:/usr/local/hadoop-environment/flume$ bin/flume-ng agent --conf conf/ -f /usr/local/twitter-data-collection/flume-twitter-source/tdc5.conf -Dflume.root.logger=DEBUG,console -n TwitterSpainpbKeywords

Cambios:

- Cambio de nombre del agente, ahora TwitterSpainpbKeywords.
- Obtener nuevas claves de Twitter Developers para no cortar la conexión del anterior agente.
- Indicar tanto los puntos geográficos de la caja de entorno como la cadena de texto con las palabras clave.
- Nuevo path.

/usr/local/twitter-data-collection/flume-twitter-source/tdc5.conf

```
TwitterSpainpbKeywords.sources.Twitter.consumerKey = <required_3>
TwitterSpainpbKeywords.sources.Twitter.consumerSecret = <required_3>
TwitterSpainpbKeywords.sources.Twitter.accessToken = <required_3>
TwitterSpainpbKeywords.sources.Twitter.accessTokenSecret = <required_3>
TwitterSpainpbKeywords.sources.Twitter.swLngLat = -9.299269, 35.999882
TwitterSpainpbKeywords.sources.Twitter.neLngLat = 4.327812, 43.79142
TwitterSpainpbKeywords.sources.Twitter.keywords = @el_pais, @elmundoes, @abc_es,
@larazon_es, @publico_es, @LaVanguardia, @elperiodico, @elconfidencial, @la_informacion,
@eldiarioes, @voz_populi, @meneame_net, @El_Plural, @ElHuffPost, @elecodiario, @policia,
#VOST, manifestación, abuso, 15M, corrupción, accidente, independencia, #Mundial,
#MundialBrasil, #Mundial2014, #Brasil2014, #MundialBrasil2014, Mundial, fútbol
```

```
TwitterSpainpbKeywords.sinks.HDFS.hdfs.path =
hdfs://localhost:9000/user/hduser/flume/twitter/spainpb-keywords/%Y/%m/%d/%H
```

4.5.3. Pig 0.12.1 y Elephant-bird 4.5

Los resultados de todas las precomputaciones/recomputaciones que realice Pig deben ser almacenados en tablas HBase mediante *HBaseStorage*.

Para este proyecto *HBaseTwitterTables* creará tres tablas:

- **Tabla** de tweets: '**tweets'**. Compuesta de *column families* que corresponden a cada Twitter Platform Object (Documentation, s.f.)
  - *Column family*: 't', Twitter Platform Object: Tweets.
  - *Column family*: 'u', Twitter Platform Object: Users.
  - *Column family*: 'e', Twitter Platform Object: Entities.
  - *Column family*: 'p', Twitter Platform Object: Places.
- **Tablas** de menciones y mencionados: '**mentions**' y '**mentioned**'.
  - *Column Family*: 'f'.

Utilizando NetBeans IDE, abrir el proyecto *HBaseTwitterTables*, pulsar con el botón secundario del ratón sobre *InitTables.java* y ejecutar la acción "Run File".

Pig realiza el análisis del conjunto total de datos, precomputando/recomputando así los resultados a almacenar en las bases de datos HBase. Primeramente se deben registrar aquellos archivos JAR necesarios y, para mayor comodidad, renombrar las User Defined Functions (User Defined Functions, s.f.) desarrolladas cuyo código fuente puede verse en el anexo Otros scripts y código fuente.

hduser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh pig analysis

/usr/local/twitter-data-collection/pig-scripts/analysis.pig

```
register /usr/local/hadoop-environment/elephant-bird/hadoop-compat/target/elephant-bird-hadoop-
compat*.jar
register /usr/local/hadoop-environment/elephant-bird/pig/target/elephant-bird-pig*.jar
register /usr/local/hadoop-environment/pig/build/ivy/lib/Pig/*.jar
register /usr/local/twitter-data-collection/pig-twitter-udfs/target/pig-twitter-udfs*.jar

define UniformDate mutual.UniformDate();
define Related relationships.Related();
define Coordinates tweets.Coordinates();
define Hashtags tweets.Hashtags();
define UserMentions tweets.UserMentions();
define MD5gen util.MD5gen();
```

Existe una parte del análisis que es común. Elephant-bird convierte de manera anidada los tweets de texto en formato JSON a datos manejables por Pig (Pig Latin Basics, 2014). De todos los campos solo se escogen aquellos que interesen. Aquellos que se necesiten pero no en el formato proporcionado se pasan como argumento a una UDF. Para esta parte común del análisis se utilizarán dos UDFs:

- Coordinates: devuelve un tupla con las coordenadas de los campos *coordinates*, *geo* y *bounding_box* que podrían proporcionar la localización geográfica de los tweets. El orden de longitud y latitud será el mismo que el de cada campo.
- UniformDate: para los distintos formatos de fecha devuelve una cadena de texto que siempre sigue el formato 'yyyyMMddHHmmssZ'. Es necesario por si a parte de los tweets proporcionados por Streaming API se tienen otros de REST API, cuyo campo *created_at* tiene un formato distinto.

/usr/local/twitter-data-collection/pig-scripts/analysis.pig

```
mutual = load 'hdfs://localhost:9000/user/hduser/flume/twitter/*/*/*/*/*/*'
   using com.twitter.elephantbird.pig.load.JsonLoader('-nestedLoad=true') as (json:map[]);

mutual = foreach mutual generate
   Coordinates($0#'coordinates') as c_coordinates:tuple(float, float),
   ToDate(UniformDate((chararray)$0#'created_at'), 'yyyyMMddHHmmssZ', '+01:00') as
date_time:datetime,
   (bag{})$0#'entities'#'hashtags' as hashtags,
   (bag{})$0#'entities'#'user_mentions' as user_mentions,
   (int)$0#'favorite_count' as favorite_count,
   (chararray)$0#'filter_level' as filter_level,
   Coordinates($0#'geo') as g_coordinates:tuple(float, float),
   (long)$0#'id' as id,
   (chararray)$0#'id_str' as id_str,
   (chararray)$0#'in_reply_to_screen_name' as in_reply_to_screen_name,
   (long)$0#'in_reply_to_status_id' as in_reply_to_status_id,
   (chararray)$0#'in_reply_to_status_id_str' as in_reply_to_status_id_str,
   (long)$0#'in_reply_to_user_id' as in_reply_to_user_id,
   (chararray)$0#'in_reply_to_user_id_str' as in_reply_to_user_id_str,
   (chararray)$0#'lang' as lang,
   Coordinates($0#'place'#'bounding_box') as bb_coordinates:tuple(float, float),
   (boolean)$0#'possibly_sensitive' as possibly_sensitive,
   (int)$0#'retweet_count' as retweet_count,
   (chararray)$0#'source' as source,
   (chararray)$0#'text' as text,
   (boolean)$0#'truncated' as truncated,
   (int)$0#'user'#'followers_count' as followers_count,
   (long)$0#'user'#'id' as user_id,
   (chararray)$0#'user'#'id_str' as user_id_str,
   (chararray)$0#'user'#'screen_name' as screen_name;
```

A partir de todos esos campos se crean tres tipos de (pre/re)computaciones para rellenar las tablas NoSQL. El primer análisis, **tweets**, sigue descomponiéndolos para consultarlos posteriormente de manera más detallada. Para ello se utilizan una serie de Built In Functions que pueden explorarse en (Built In Functions, s.f.)

El fin de cada (pre/re)computación ejecutada por Pig es almacenar la información en HBase por lo que es necesario especificar las *rowkeys*. En este caso se formará mediante "MD5(id_str)", donde *id_str* es el ID del tweet como cadena de texto. De esta manera se logra:

- Una longitud de *rowkey* constante cuyo orden no es consecutivo, evitando así el hot-spotting.
- En caso de que varios *Flume agents* almacenen un mismo tweet estarían repetidos en HDFS. HBase se ocupará de dicha repetición ya que ambos poseen el mismo *id_str* así que tendrán el mismo "MD5(id_str)", es decir, la misma *rowkey*. El que fuera tomado en segundo lugar actualizará el contenido del primero en la tabla HBase, aunque es irrelevante ya que son el mismo.

Las dos UDFs desarrolladas son:

- Hashtags: al pasarle el campo *hashtags* devuelve una cadena de texto con los hashtags utilizados en el tweet separados cada uno por un espacio, y sin '#'.
- UserMentions: al pasarle el campo *user_mentions* devuelve una cadena de texto con los usuarios mencionados en el twets separados cada uno por un espacio. El formato de cada usuario mencionado es "*id_str@screen_name*". Por ejemplo si se hace mención al usuario www.twitter.com/twitterapi su mención se vería reflejada por "6253282@twitterapi".

/usr/local/twitter-data-collection/pig-scripts/analysis.pig

```
tweets = foreach mutual generate (chararray)MD5gen(id_str) as rowkey:chararray,
    c_coordinates.$0 as c_lng:float, c_coordinates.$1 as c_lat:float,
    date_time,
    GetYear(date_time) as year, GetMonth(date_time) as month, GetDay(date_time) as day,
GetWeek(date_time) as week,
    GetHour(date_time) as hour, GetMinute(date_time) as minute, GetSecond(date_time) as second,
    Hashtags(hashtags) as hashtags_str:chararray,
    UserMentions(user_mentions) as user_mentions_str:chararray,
    favorite_count,
    filter_level,
    g_coordinates.$0 as g_lat:float, g_coordinates.$1 as g_lng:float,
    id, id_str,
    in_reply_to_screen_name,
    in_reply_to_status_id, in_reply_to_status_id_str,
    in_reply_to_user_id, in_reply_to_user_id_str,
    lang,
    bb_coordinates.$0 as bb_lng:float, bb_coordinates.$1 as bb_lat:float,
    possibly_sensitive,
    retweet_count,
    source,
    text,
    truncated,
    followers_count,
    user_id, user_id_str,
    screen_name;
```

Entonces es cuando todos los tweets se filtran por una serie de parámetros que se precisa que tengan en común. Pig no maneja por el usuario el hecho de no diferenciar entre letras con y sin tilde o mayúsculas y minúsculas así que debe especificarse

mediante expresiones regulares con el formato utilizado por Java (Class Pattern, s.f.). Por ejemplo filtrar aquellos tweets que contengan alguno de los hashtags {[Ff][uú]tbol, [Mm]undial, [Mm]undialBrasil, [Mm]undial2014, [Bb]rasil2014, [Mm]undialBrasil2014} o que en el texto aparezca alguna de las palabras {[Ff][uú]tbol, [Mm]undial}.

/usr/local/twitter-data-collection/pig-scripts/analysis.pig

```
tweets = filter tweets by
   hashtags_str matches
'.*\\b([Ff][uú]tbol|[Mm]undial|[Mm]undialBrasil|[Mm]undial2014|[Bb]rasil2014|[Mm]undialBrasil201
4)\\b.*'
   or text matches '.*\\b([Ff][uú]tbol|[Mm]undial)\\b.*';
```

Ya que con Streaming API no se puede obtener la lista de usuarios que un usuario dado sigue, los dos siguientes análisis relacionan cada usuario con:

- Aquellos que ha mencionado, **mentions**. Su *rowkey* será "MD5(id_str_usuario)MD5(id_str_usuarioMencionado)"
- Aquellos que le han mencionado, **mentioned**. Su *rowkey* será "MD5(id_str_usuarioMencionado)MD5(id_str_usuario)"

A parte de las ventajas ya expuestas anteriormente de utilizar MD5, también existe la ventaja de que no se repetirá información en las tablas HBase ya que si un usuario hace mención de otro en varios tweets la concatenación de los dos MD5 será el mismo. Si lo que se hubiera pretendido es conocer el número de veces que un usuario hace mención a otro, esta rowkey no valdría y habría que diseñar otra. Todo esto ocurre de la misma manera para la tabla de mencionados. La gran ventaja de la Arquitecutra Lambda es que esto no repercute en los componentes anteriores en el flujo de datos, en esto caso Flume y Hadoop.

La UDF desarrollada es:

- Related: en el momento de realizar este proyecto si se le pasa como argumento *user_mentions* devolverá una bolsa con una serie de tuplas que contienen *user_id*, *user_id_str* y *screen_name*. Al igual que todos los demás UDFs, se podría ampliar su funcionalidad para que las relaciones no solo se basen en menciones.

Para ambos análisis los campos a almacenar son los mismos, *user_id* y *screen_name*. La diferencia será el orden igual que lo es en la *rowkey*.

/usr/local/twitter-data-collection/pig-scripts/analysis.pig

```
-- (a, {(b,c), (d,e)})
-- If we apply the expression GENERATE $0, flatten($1) to this tuple,
-- we will create new tuples: (a, b, c) and (a, d, e)

relationships = foreach mutual generate
   user_id, user_id_str, screen_name,
   flatten(Related(user_mentions)) as (r_user_id:long, r_user_id_str:chararray,
r_screen_name:chararray);

relationships = filter relationships by r_user_id is not null;
```

```
mentions = foreach relationships generate CONCAT(MD5gen(user_id_str), MD5gen(r_user_id_str))
as rowkey:chararray,
   user_id, screen_name,
   r_user_id, r_screen_name;

mentioned = foreach relationships generate CONCAT(MD5gen(r_user_id_str),
MD5gen(user_id_str)) as rowkey:chararray,
   r_user_id, r_screen_name,
   user_id, screen_name;
```

Es cierto que a partir de la tabla de "menciones a…" se pueden obtener los mismos resultados que de la tabla de "mencionados por…". El hecho de crear una tabla de mencionados reside en el aumento de rendimiento al realizar consultas a ésta, ya que HBase ordenará las filas de acuerdo a su *rowkey* por lo que todas las que empiecen por "MD5(r_user_id_str)" estarán almacenadas juntas. Esto no ocurre con la tabla de menciones ya que cada *rowkey* empezará por el MD5 del usuario que menciona, no del mencionado. Crear la tabla '**mentioned**' depende de qué se considere más importante:

- Rendimiento en lectura, creando la tabla de "mencionados por…".
- No ocupar más espacio de almacenamiento no creando la tabla de "mencionados por…".

Por ultimo, los resultados de todas las (pre/re)computación se almacenan en sus correspondientes tablas HBase mediante *HBaseStorage*. Para cada campo generado se debe indicar en qué *column family* se va a almacenar.

- Para la tabla '**tweets**', cada columna HBase corresponderá a uno de sus campos y cada columna pertenecerá a una *column family* según a que Twitter Platform Object pertenezca el campo.
- Para las tablas '**mentions**' y '**mentioned**', cada columna HBase corresponderá a uno de sus campos y todas las columnas formarán parte de la única *column family* creada para cada tabla.

/usr/local/twitter-data-collection/pig-scripts/analysis.pig

```
/**************************************************************************************
 * HBaseStorage
 * Be careful, add SPACE at the end of each field "colfam:cqual<SPACE>"
 * and they must be in the same order
 *************************************************************************************/

store tweets into 'tweets' using org.apache.pig.backend.hadoop.hbase.HBaseStorage('
   t:c_lng t:c_lat
   t:date_time
   t:year t:month t:day t:week
   t:hour t:minute t:second
   e:hashtags_str
   e:user_mentions_str
   t:favorite_count
   t:filter_level
   t:g_lat t:g_lng
   t:id t:id_str
   t:in_reply_to_screen_name
```

```
    t:in_reply_to_status_id t:in_reply_to_status_id_str
    t:in_reply_to_user_id t:in_reply_to_user_id_str
    t:lang
    p:bb_lng p:bb_lat
    t:possibly_sensitive
    t:retweet_count
    t:source
    t:text
    t:truncated
    u:followers_count
    u:user_id u:user_id_str
    u:screen_name');

store mentions into 'mentions' using org.apache.pig.backend.hadoop.hbase.HBaseStorage('
    f:user_id f:screen_name
    f:r_user_id f:r_screen_name');

store mentioned into 'mentioned' using org.apache.pig.backend.hadoop.hbase.HBaseStorage('
    f:r_user_id f:r_screen_name
    f:user_id f:screen_name');
```

4.6. Serving layer

4.6.1. HBase 0.94.20

Los cambios en las configuraciones se realizan automáticamente gracias al script proporcionado. A continuación se exponen dichas modificaciones cuyas descripciones están dentro de cada archivo.

/usr/local/hadoop-environment/hbase/conf/hbase-env.sh

```
# The java implementation to use.  Java 1.6 required.
export JAVA_HOME=/usr/lib/jvm/java-7-oracle

# Tell HBase whether it should manage it's own instance of Zookeeper or not.
export HBASE_MANAGES_ZK=true
```

/usr/local/hadoop-environment/hadoop/conf/hdfs-site.xml

```
    <property>
        <name>dfs.support.append</name>
        <value>true</value>
        <description>
            Determines whether HDFS should support the append (sync)
feature or not.
            The default value is false. It must be set to true, or
you may lose data if the region server crashes
        </description>
    </property>
    <property>
        <name>dfs.datanode.max.xcievers</name>
        <value>4096</value>
        <description>
            To have DataNode keep more threads open, to handle more
concurrent requests.
        </description>
    </property>
```

/usr/local/hadoop-environment/hbase/conf/hbase-site.xml

```
<property>
        <name>hbase.rootdir</name>
        <value>hdfs://localhost:9000/hbase</value>
        <description>
            The directory shared by RegionServers.
        </description>
    </property>
    <property>
        <name>hbase.cluster.distributed</name>
        <value>true</value>
        <description>
            The mode the cluster will be in.
            Possible values are false: standalone and pseudo-
distributed setups with managed Zookeeper
            true: fully-distributed with unmanaged Zookeeper Quorum
(see hbase-env.sh)
        </description>
    </property>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.support.append</name>
        <value>true</value>
        <description>
            Determines whether HDFS should support the append (sync)
feature or not.
            The default value is false. It must be set to true, or
you may lose data if the region server crashes.
        </description>
    </property>
    <property>
        <name>hbase.zookeeper.property.clientPort</name>
        <value>2181</value>
    </property>
    <property>
        <name>hbase.zookeeper.quorum</name>
        <value>localhost</value>
        <description>
            Comma separated list of servers in the ZooKeeper Quorum.
            For example,
"host1.mydomain.com,host2.mydomain.com,host3.mydomain.com".
            By default this is set to localhost for local and
pseudo-distributed modes of operation.
            For a fully-distributed setup, this should be set to a
full list of ZooKeeper quorum servers.
            If HBASE_MANAGES_ZK is set in hbase-env.sh this is the
list of servers which we will start/stop ZooKeeper on.
        </description>
    </property>
    <property>
        <name>hbase.zookeeper.property.dataDir</name>
        <value>/usr/local/hadoop-environment/var/zookeeper</value>
        <description>
            Property from ZooKeeper's config zoo.cfg.
            The directory where the snapshot is stored.
        </description>
    </property>
```

```
    <property>
        <name>zookeeper.znode.parent</name>
        <value>/hbase</value>
        <description>
            Root ZNode for HBase in ZooKeeper.
            All of HBase's ZooKeeper files that are configured with
a relative path will go under this node.
            By default, all of HBase's ZooKeeper file path are
configured with a relative path,
            so they will all go under this directory unless changed.
        </description>
    </property>
```

4.6.1.1. HBaseTwitterTables

Para crear o regenerar las tablas expuestas en el apartado anterior se debe ejecutar la clase *InitTables.java* y para eliminarlas *DropTables.java*. Ver anexo Otros scripts y código fuente.

4.6.2. Hive 0.13.1

Manualmente se debe configurar Hive metastore en modo local siguiendo los pasos oportunos indicados por (Configuring the Hive Metastore, s.f.):

superuser@hostname:~$ mysql -u root -p
mysql> ***CREATE DATABASE hivemetastore;***
mysql> ***USE hivemetastore;***
mysql> ***SOURCE /usr/local/hadoop-environment/hive/scripts/metastore/upgrade/mysql/hive-schema-0.13.0.mysql.sql***
mysql> ***CREATE USER 'hduser'@'localhost' IDENTIFIED BY 'tfgtdc';***
mysql> ***REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'hduser'@'localhost';***
mysql> ***GRANT SELECT,INSERT,UPDATE,DELETE,LOCK TABLES,EXECUTE ON hivemetastore.* TO 'hduser'@'localhost';***
mysql> ***FLUSH PRIVILEGES;***
mysql> ***quit;***

Los cambios en las configuraciones se realizan automáticamente gracias al script proporcionado. A continuación se exponen dichas modificaciones cuyas descripciones están dentro de cada archivo.

/usr/local/hadoop-environment/hive/conf/hive-env.sh

```
# Folder containing extra ibraries required for hive compilation/execution can be controlled by:
export HIVE_AUX_JARS_PATH=$HIVE_HOME/lib
```

/usr/local/hadoop-environment/hive/conf/hive-site.xml

```
    <property>
        <name>javax.jdo.option.ConnectionURL</name>
        <value>jdbc:mysql://localhost/hivemetastore</value>
        <description>JDBC connect string for a JDBC
metastore</description>
    </property>
```

```
    <property>
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>com.mysql.jdbc.Driver</value>
        <description>Driver class name for a JDBC
metastore</description>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionUserName</name>
        <value>hduser</value>
        <description>username to use against metastore
database</description>
    </property>

    <property>
        <name>javax.jdo.option.ConnectionPassword</name>
        <value>tfgtdc</value>
        <description>password to use against metastore
database</description>
    </property>
    <property>
        <name>datanucleus.autoCreateSchema</name>
        <value>false</value>
        <description>creates necessary schema on a startup if one
doesn't exist. set this to false, after creating it
once</description>
    </property>
```

Las llamadas tablas externas (Hive Data Definition Language - External Tables, 2014) accederán a los datos almacenados en las tablas HBase gracias a *HBaseStorageHandler* (Hive HBase Integration, s.f.). Para cada tabla externa se debe especificar el tipo de cada dato y la *column family* a la que pertenece como valor de la propiedad *hbase.columns.mapping* (Using Hive to interact with HBase, Part 1, 2013)

Comando para crear las tablas externas y así consultar las tablas HBase mediante SQL: **hduser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh hive initTables**

/usr/local/twitter-data-collection/hive-scripts/initTables.hql

```
drop table if exists tweets;
drop table if exists mentions;
drop table if exists mentioned;

-------------------------------------------------------------------------------
-- tweets
-------------------------------------------------------------------------------
create external table tweets(key string,
   c_lng float, c_lat float,
   date_time string,
   year int, month tinyint, day tinyint, week int,
   hour tinyint, minute tinyint, second tinyint,
   hashtags_str string,
   user_mentions_str string,
   favorite_count int,
   filter_level string,
   g_lat float, g_lng float,
   id bigint, id_str string,
   in_reply_to_screen_name string,
   in_reply_to_status_id bigint, in_reply_to_status_id_str string,
```

```
    in_reply_to_user_id bigint, in_reply_to_user_id_str string,
    lang string,
    bb_lng float, bb_lat float,
    possibly_sensitive boolean,
    retweet_count int,
    source string,
    text string,
    truncated boolean,
    followers_count int,
    user_id bigint, user_id_str string,
    screen_name string)
stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
with serdeproperties('hbase.columns.mapping'=
':key,t:c_lng,t:c_lat,t:date_time,t:year,t:month,t:day,t:week,t:hour,t:minute,t:second,e:hashtags_str,e:us
er_mentions_str,t:favorite_count,t:filter_level,t:g_lat,t:g_lng,t:id,t:id_str,t:in_reply_to_screen_name,t:i
n_reply_to_status_id,t:in_reply_to_status_id_str,t:in_reply_to_user_id,t:in_reply_to_user_id_str,t:lang,
p:bb_lng,p:bb_lat,t:possibly_sensitive,t:retweet_count,t:source,t:text,t:truncated,u:followers_count,u:us
er_id,u:user_id_str,u:screen_name')
tblproperties('hbase.table.name'='tweets');

-------------------------------------------------------------------------------
-- mentions
-------------------------------------------------------------------------------
create external table mentions(key string,
    user_id bigint, screen_name string,
    r_user_id bigint, r_screen_name string)
stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
with serdeproperties('hbase.columns.mapping'=
    ':key,f:user_id,f:screen_name,f:r_user_id,f:r_screen_name')
tblproperties('hbase.table.name'='mentions');

-------------------------------------------------------------------------------
-- mentioned
-------------------------------------------------------------------------------
create external table mentioned(key string,
    r_user_id bigint, r_screen_name string,
    user_id bigint, screen_name string)
stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
with serdeproperties('hbase.columns.mapping'=
    ':key,f:r_user_id,f:r_screen_name,f:user_id,f:screen_name')
tblproperties('hbase.table.name'='mentioned');
```

5. Twitter Data Collection: cluster multi-nodo

Siguiendo el modelo propuesto por (Noll, Running Hadoop on Ubuntu Linux - Multi-Node Cluster, 2011), con el fin de evitar problemas y reducir complejidad en la instalación se utilizará un cluster multi-nodo formado por la unión de varios cluster mono-nodo. Aunque para este proyecto se unirán dos cluster mono-nodo, el método de configuración es extensible a tantos nodos como se desee.

Uno de ellos será el nodo maestro, que también actuará como nodo esclavo, y el otro será solamente un nodo esclavo.

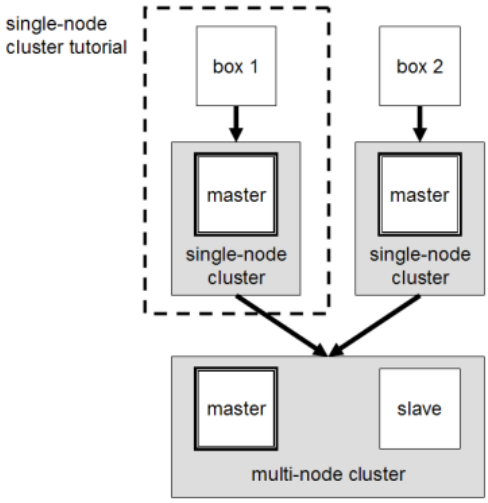


Ilustración 14 - Cluster multi-nodo (Noll, Running Hadoop on Ubuntu Linux - Multi-Node Cluster, 2011)

5.1. Prerrequisitos

La manera más fácil de que ambas máquinas logren conectarse a través de la red es mediante un hub o switch y configurar las interfaces de red para usar una red común.

Para ilustrar de manera más visual la instalación, el nodo maestro que también ejerce de escalvo se llamará **master** y el esclavo **slave**. Las IPs asignadas a los hosts serán las siguientes:

- master IP: *ipMaster*
- slave IP: *ipSlave*

El siguiente comando se deberá ejecutar tanto en el maestro como en el esclavo para modificar mediante Vim el archivo */etc/hosts*

superuser@master:~$ sh /usr/local/twitter-data-collection/tdc.sh -multi pres

superuser@slave:~$ sh /usr/local/twitter-data-collection/tdc.sh -multi pres

/etc/hosts

127.0.0.1	localhost
#127.0.1.1	*hostname*
#127.0.0.1	*hostname*
ipMaster	master
ipSlave	slave

5.2. SSH

El usuario *hduser@master* debe ser capaz de:

1. Conectar consigo mismo utilizando **ssh master** en vez de **ssh localhost** via SSH sin contraseña, algo que ya hizo el script automáticamente al añadir la clave pública SSH al archivo authorized_keys.
2. Conectar con *hduser@slave* utilizando **ssh slave**.

Se debe ejecutar:
hduser@master:~$ sh /usr/local/twitter-data-collection/tdc.sh -multi ssh <slave>
donde <slave> es el nombre del esclavo, en este caso **slave**

Es decir:
hduser@master:~$ sh /usr/local/twitter-data-collection/tdc.sh -multi ssh slave

Se puede hacer una prueba realizando una conexión mediante SSH desde *hduser@master* a *hduser@master* y *hduser@slave* confirmando que la conexión sin contraseña funciona. La primera vez aparecerá un aviso sobre si se quiere confiar en el certificado del host.

hduser@master:~$ ssh master

hduser@master:~$ ssh slave

5.3. Instalación

Los cambios en las configuraciones se realizan automáticamente gracias al script proporcionado. Vim abrirá los archivos de configuración modificados por si durante el proceso de instalación del cluster multi-nodo se ha cambiado el número de nodos o los nombres de **master** y **slave**. Por defecto los archivos modificados serán los siguientes.

5.3.1. Nodo maestro

superuser@master:~$ sh /usr/local/twitter-data-collection/tdc.sh -multi install-master

5.3.1.1. Hadoop 1.2.1

/usr/local/hadoop-environment/hadoop/conf/masters

```
master
```

/usr/local/hadoop-environment/hadoop/conf/slaves

```
master
slave
```

5.3.1.2. HBase 0.94.20

/usr/local/hadoop-environment/hbase/conf/regionservers

```
master
slave
```

5.3.2. Nodo maestro y esclavo

El siguiente comando se deberá ejecutar tanto en el nodo maestro como en el esclavo.

superuser@master:~$ sh /usr/local/twitter-data-collection/tdc.sh -multi install-all

superuser@slave:~$ sh /usr/local/twitter-data-collection/tdc.sh -multi install-all

A continuación se exponen dichas modificaciones cuyas descripciones están dentro de cada archivo.

5.3.2.1. Hadoop 1.2.1

/usr/local/hadoop-environment/hadoop/conf/core-site.xml

```
    <property>
        <name>fs.default.name</name>
        <value>hdfs://master:9000</value>
        <description>
            The name of the default file system.
            A URI whose scheme and authority determine the
FileSystem implementation.
            The uri's scheme determines the config property
(fs.SCHEME.impl) naming the FileSystem implementation class.
            The uri's authority is used to determine the host, port,
etc. for a filesystem.
        </description>
    </property>
```

/usr/local/hadoop-environment/hadoop/conf/mapred-site.xml

```
    <property>
        <name>mapred.job.tracker</name>
        <value>master:9001</value>
        <description>
            The host and port that the MapReduce job tracker runs
at.
            If "local", then jobs are run in-process as a single map
and reduce task.
        </description>
    </property>
```

/usr/local/hadoop-environment/hadoop/conf/hdfs-site.xml

```
    <property>
        <name>dfs.replication</name>
        <value>2</value>
        <description>
            Default block replication.
            The actual number of replications can be specified when
the file is created.
            The default is used if replication is not specified in
create time.
        </description>
    </property>
```

5.3.2.2. HBase 0.94.20

/usr/local/hadoop-environment/hbase/conf/hbase-site.xml

```
    <property>
        <name>hbase.rootdir</name>
        <value>hdfs://anunit:9000/hbase</value>
        <description>
            The directory shared by RegionServers.
        </description>
    </property>
```

```
    <property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
```

```
    <property>
        <name>hbase.zookeeper.quorum</name>
        <value>anunit</value>
        <description>
            Comma separated list of servers in the ZooKeeper Quorum.
            For example,
"host1.mydomain.com,host2.mydomain.com,host3.mydomain.com".
            By default this is set to localhost for local and
pseudo-distributed modes of operation.
            For a fully-distributed setup, this should be set to a
full list of ZooKeeper quorum servers.
            If HBASE_MANAGES_ZK is set in hbase-env.sh this is the
list of servers which we will start/stop ZooKeeper on.
        </description>
    </property>
```

5.4. Inicialización

Se inicializa el cluster multi-nodo de la misma manera que en la sección Inicialización del capítulo Twitter Data Collection: cluster mono-nodo.

Ejecutando Java Virtual Machine Process Status Tool se puede comprobar que los procesos necesarios para manejar el proyecto están activos.

hduser@master:~$ jps

```
HRegionServer
SecondaryNameNode
DataNode
TaskTracker
Jps
JobTracker
HMaster
HQuorumPeer
NameNode
```

Ilustración 15 - Procesos ejecutándose en el nodo maestro del cluster multi-nodo

hduser@slave:~$ jps

```
HRegionServer
Jps
TaskTracker
DataNode
```

Ilustración 16 - Procesos ejecutándose en el nodo esclavo del cluster multi-nodo

5.5. Batch Layer

5.5.1. Flume 1.5.0

Para cada uno de los archivos de configuración que describen cada *Flume agent* se debe cambiar **master** por *localhost* en el path de destino del *Flume sink*. A continuación se muestra un ejemplo.

/usr/local/twitter-data-collection/flume-twitter-source/tdc1.conf

```
TwitterSpainPB.sinks.HDFS.hdfs.path = hdfs://master:9000/user/hduser/flume/twitter/spain-pb/%Y/%m/%d/%H
```

5.5.2. Pig 0.12.1 y Elephant-bird 4.5

En el script que realiza las (pre/re)computaciones, se debe cambiar **master** por *localhost* en el path del que el JsonLoader de Elephant-bird tomará los datos.

/usr/local/twitter-data-collection/pig-scripts/analysis.pig

```
mutual = load 'hdfs://master:9000/user/hduser/flume/twitter/*/*/*/*/*/*'
   using com.twitter.elephantbird.pig.load.JsonLoader('-nestedLoad=true') as (json:map[]);
```

6. Pruebas

El cluster sobre el que se ha probado el proyecto está formado por dos servidores (Servidor HP ProLiant ML310 G4, s.f.) con 250 GB de almacenamiento y 2 GB de memoria RAM cada uno, por lo que se ha comprobado el correcto funcionamiento de Twitter Data Collection en un cluster multi-nodo pero no ha sido posible comprobar su rendimiento con un volumen de datos semejante al que cualquier grupo de investigación o empresa podrían tener.

A continuación se muestran pruebas de un caso de uso, que valen tanto para el cluster mono-nodo como para el multi-nodo.

Como ejemplo ejecutamos los *Flume agents* que toman tweets procedentes de España y aquellos en los que se buscan palabras clave, TwitterSpainPB y TwitterKeywords:
hduser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh flume spain-pb
hduser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh flume keywords

En las terminales se sucederán los tweets recogidos e información de depuración por si hubiera algún problema con alguno de los agentes.

Ilustración 17 - Flume agents: TwitterSpainPB (arriba) y TwitterKeywords (abajo)

Las (pre/re)computaciones con Pig filtraban los tweets según palabras relacionadas con la Copa Mundial de Fútbol 2014, relacionaban a los usuarios con aquellos que habían mencionado y al revés, aquellos usuarios que habían sido mencionados por otros. En el momento en el que se desee deberá ejecutarse:

hduser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh pig analysis

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
1.2.1   0.12.2-SNAPSHOT hduser  2014-07-03 01:33:18     2014-07-03 01:41:59     FILTER

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTIme      AvgMapTime      MedianMapTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime        Alias   Feature Outputs
job_201407030029_0015   4       0       313     120     221     225     n/a     n/a     n/a     n/a     mentioned,mentions,mutual,relationships,tweets  MULTI_QUERY,MAP_ONLY    tweets,mention
s,mentioned,

Input(s):
Successfully read 69204 records (219269682 bytes) from: "hdfs://localhost:9000/user/hduser/flume/twitter/*/*/*/*/*/*"

Output(s):
Successfully stored 20542 records in: "tweets"
Successfully stored 45488 records in: "mentions"
Successfully stored 45488 records in: "mentioned"

Counters:
Total records written : 111518
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_201407030029_0015


2014-07-03 01:42:00,210 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
```

Ilustración 18 - (Pre/Re)computaciones con Pig

A continuación se muestran ejemplos de consultas sobre HBase utilizando el estilo SQL de Hive cuyos resultados se guardarán en *twitter-data-collection/outs/hive* con extensión .tab (Using Hive to interact with HBase, Part 2, 2013)

hduser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh hive -tab tweets

Ejecutará el script en el que se consultará qué tweets están geolocalizados mostrando su año, mes, día, hora, minuto, segundo, longitud, latitud y texto del tweet.

/usr/local/twitter-data-collection/hive-scripts/tweets.hql

```
-- Name(Signature): COALESCE(T v1, T v2, ...)
-- Return Type: T
-- Description: Return the first v that is not NULL, or NULL if all v's are NULL

select * from (
   select year, month, day, hour, minute, second,
      coalesce(c_lng, g_lng, bb_lng) as lng, coalesce(c_lat, g_lat, bb_lat) as lat,
      text
   from tweets
) geolocated_tweets
where lng is not null and lat is not null
order by year, month, day, hour, minute, second;
```

Una muestra de resultado podría ser:

/usr/local/twitter-data-collection/outs/hive/tweets.tab

```
2014    7       2       23      44      49      3.105085        39.695393       Como se nota
cuando no hay mundial.... #FifaWorldCup #FIFAWorldCupBrazil
2014    7       2       23      45      6       -74.075836      4.598056        Polémica por
esta medida que podría tomar la FIFA en el #Mundial2014 http://t.co/ibl7O2yBfP
2014    7       3       0       14      23      -67.03475       10.342356       Disciplina,
#FIFA multa a #ALG con 56.400 $  #Brasil2014 #DIRECTVCopaMundial http://t.co/eNJu9I5ZXe
```

hduser@hostname:~$ sh /usr/local/twitter-data-collection/tdc.sh hive mentioned

Ejecutará el script en el que se consultará el *user_id* de quienes han mencionado a 'FIFAcom'.

/usr/local/twitter-data-collection/hive-scripts/mentioned.hql

```
-- Name(Signature): substr(string|binary A, int start, int len)
-- Return Type: string
-- Description: Returns the substring or slice of the byte array of A starting
--      from start position with length len
--      e.g. substr('foobar', 4, 1) results in 'b'

-- screen_name = 'FIFAcom' has been mentioned by...
select r_screen_name, user_id
from mentioned
where substr(key, 1, 32) = 'E36B945A1601DEFAE40B28565818A832';
```

Una muestra de resultado podría ser:

/usr/local/twitter-data-collection/outs/hive/mentioned.tab

FIFAcom	489826551
FIFAcom	497493324
FIFAcom	496392856
FIFAcom	2438540872
FIFAcom	348175668

7. Conclusiones y trabajo futuro

Pese a que Twitter Data Collection es un sitema Big Data para adquirir, almacenar, analizar y consultar grandes volúmenes de datos procedentes de Twitter, las tecnologías que la forman, su metodología y su arquitectura permiten que sea un sistema extensible capaz de llevarse a gran cantidad de ámbitos de las TIC.

Durante el desarrollo del proyecto se consideró recurrir a diseños construidos por profesionales y utilizar CDH (CDH, s.f.) o productos de Hortonworks (Hortonworks Sandbox Tutorials, s.f.) pero finalmente se optó por explorar las tecnologías con licencia Apache de manera libre, explorando y eligiendo cuales eran las más adecuadas y aprender sobre ellas. De esta manera, en vez de obtener un conjunto de componentes dados ya perfectamente ensamblados, se obtiene un gran aprendizaje sobre su funcionamiento así como experiencia a la hora de ensamblarlos y que funcionen correctamente.

El proyecto desarrollado puede avanzar mejorando el rendimiento de cada uno de sus componentes, por ejemplo añadiendo coprocesadores a HBase, asi como el rendimiento de la interacción entre ellos, como la de HBase y Hive.

Es un sistema que para el usuario final, alguien que simplemente quiera realizar consultas al estilo tradicional SQL, no supone un cambio de paradigma en cuanto al acceso a los datos. Pese a que no se ha realizado un API específico para acceder a los resultados de manera gráfica siempre se puede crear uno personalizado, recurrir a clientes Hive u otros componentes como Hue (Hue, s.f.)

Este es un proyecto construido a base de componentes que interactúan entre sí a los que se les puede incorporar desde compresión de datos hasta una amalgama cada día más grande de otros componentes que añaden riqueza a todo el entorno Hadoop, por ejemplo Mahout (Apache Mahout, s.f.), una librería escalable para el aprendizaje automático y la minería de datos.

En el caso de la ingesta de tweets hay que recordar que para Streaming API el flujo de datos proporcionado tiene un límite máximo de tweets por cantidad de tiempo. Esto quería decir que cuando se incrementen drásticamente los tweets sobre alguna tendencia todo aquel que no entre en el *pipe* de información se perderá ya que es imposible realizar consultas que capturen tweets anteriores en el tiempo. En un futuro sería conveniente proveer a HDFS no solo de tweets capturados mediante Streaming API sino también realizar consultas sucesivas automáticas con REST API, controlando el límite de consultas y tiempo por usuario. Así, cuando se produzcan eventos importantes se guardarán más tweets anteriores en el tiempo de los que Streaming API puede proveer.

Repositorio disponible en www.github.com/borjagilperez/twitter-data-collection

Cuando los elefantes sueñan con los datos.
En Twitter Data Collection, por supuesto.

Referencias

Achrekar, H., Gandhe, A., Lazarus, R., Yu, S.-H., & Liu, B. (2011). *Predicting Flu Trends using Twitter Data.* Retrieved from IEEE Computer Communications Workshops (INFOCOM WKSHPS), 702–707: http://cse.unl.edu/~byrav/INFOCOM2011/workshops/papers/p713-achrekar.pdf

AdminManual MetastoreAdmin. (n.d.). Retrieved from Apache Hive: https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin

AmILab. (n.d.). Retrieved from Ambient Intelligence Laboratory: http://amilab.ii.uam.es/

Apache Cassandra. (n.d.). Retrieved from Apache Cassandra: http://cassandra.apache.org/

Apache Flume. (n.d.). Retrieved from Apache Flume: http://flume.apache.org/

Apache Hadoop. (n.d.). Retrieved from Apache Hadoop: http://hadoop.apache.org/

Apache HBase. (n.d.). Retrieved from Apache HBase: https://hbase.apache.org/

Apache Hive. (n.d.). Retrieved from Apache Hive: http://hive.apache.org/

Apache Kafka. (n.d.). Retrieved from Apache Kafka: http://kafka.apache.org/

Apache Mahout. (n.d.). Retrieved from Apache Mahout: http://mahout.apache.org/

Apache Maven. (n.d.). Retrieved from Apache Maven: http://maven.apache.org/

Apache Pig. (n.d.). Retrieved from Apache Pig: https://pig.apache.org/

Apache Pig Philosophy. (n.d.). Retrieved from Apache Pig: https://pig.apache.org/philosophy.html

Apache Sqoop. (n.d.). Retrieved from Apache Sqoop: http://sqoop.apache.org/

Apache Thrift. (n.d.). Retrieved from Apache Thrift: http://thrift.apache.org/

Apache ZooKeeper. (n.d.). Retrieved from Apache ZooKeeper: http://zookeeper.apache.org/

Basho Technologies. (n.d.). Retrieved from Riak: http://basho.com/riak/

Built In Functions. (n.d.). Retrieved from Apache Pig: http://pig.apache.org/docs/r0.12.1/func.html

Cascading. (n.d.). Retrieved from Cascading: http://www.cascading.org/

CDH. (n.d.). Retrieved from Cloudera: http://www.cloudera.com/content/cloudera/en/products-and-services/cdh.html

cdh-twitter-example. (n.d.). Retrieved from GitHub: https://github.com/cloudera/cdh-twitter-example

Class Pattern. (n.d.). Retrieved from docs.oracle.com: http://docs.oracle.com/javase/1.5.0/docs/api/java/util/regex/Pattern.html

Cloudera. (n.d.). Retrieved from Cloudera: http://www.cloudera.com

CompilingEasyHowTo. (n.d.). Retrieved from Ubuntu documentation: https://help.ubuntu.com/community/CompilingEasyHowTo

Configuring the Hive Metastore. (n.d.). Retrieved from Cloudera docs: http://www.cloudera.com/content/cloudera-content/cloudera-docs/CDH4/4.2.0/CDH4-Installation-Guide/cdh4ig_topic_18_4.html

Congosto, M. L., Fernández, M., & Moro Egido, E. (2011). *TWITTER Y POLÍTICA: INFORMACIÓN, OPINIÓN Y ¿PREDICCIÓN?* Retrieved from Cuadernos de Comunicación Evoca 4: http://markov.uc3m.es/~emoro/ps/evoca.pdf

Dimiduk, N., & Khurana, A. (2013). *HBase in action.* New York: Manning.

Documentation. (n.d.). Retrieved from Twitter Developers: https://dev.twitter.com/docs/

Elephant-bird. (n.d.). Retrieved from GitHub: https://github.com/kevinweil/elephant-bird/

EPS. (n.d.). Retrieved from Escuela Politécnica Superior de la UAM: www.ii.uam.es

Flume User Guide. (n.d.). Retrieved from Apache Flume: https://flume.apache.org/FlumeUserGuide.html

Gates, A. (2011). *Programming Pig.* Sebastopol, CA: O'Reilly Media, Inc.

George, L. (2011). *HBase: The Definitive Guide.* Sebastopol, CA: O'Reilly Media, Inc.

Gil Pérez, B., & Weil, K. (2013, 11 28). *Apache Thrift doubt*. Retrieved from Twitter: https://twitter.com/borjagilperez/status/407209880282349568

González-Bailón, S., Borge-Holthoefer, J., Rivero, A., & Moreno, Y. (2011). *The Dynamics of Protest Recruitment through an Online Network.* Retrieved from Scientific reports, Nature: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3240992/

Gupta, A. (2012, 11 30). *HBase vs Cassandra*. Retrieved from BigDataNoob: http://bigdatanoob.blogspot.com.es/2012/11/hbase-vs-cassandra.html

Hadoop and MongoDB Use Cases. (n.d.). Retrieved from MongoDB: http://docs.mongodb.org/ecosystem/use-cases/hadoop/

Henschen, D. (2013, 05 08). *Big Data Debate: Will HBase Dominate NoSQL?* Retrieved from InformationWeek: http://www.informationweek.com/big-data/software-platforms/big-data-debate-will-hbase-dominate-nosql/d/d-id/1111048?

Hive Data Definition Language - External Tables. (2014, 05 13). Retrieved from LanguageManual DDL:

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-ExternalTables

Hive HBase Integration. (n.d.). Retrieved from HBaseIntegration: https://cwiki.apache.org/confluence/display/Hive/HBaseIntegration

Hoffman, S. (2013). *Apache Flume: Distributed Log Collection for Hadoop.* Birmingham: Packt Publishing Ltd.

Hortonworks Sandbox Tutorials. (n.d.). Retrieved from Hortonworks: http://hortonworks.com/tutorials/

Hue. (n.d.). Retrieved from Hue: http://gethue.com/

IDC IVIEW. (2011, June). *Extracting Value form Chaos.* Retrieved from IDC Analyze the Future: http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf

Install Oracle Java 7 in Ubuntu via PPA Repository. (n.d.). Retrieved from WEB UPD8: http://www.webupd8.org/2012/01/install-oracle-java-jdk-7-in-ubuntu-via.html

iTouchMap. (n.d.). *Latitude and Longitude of a Point*. Retrieved from iTouchMap: http://itouchmap.com/latlong.html

Java Community Help Wiki. (n.d.). Retrieved from Ubuntu documentation: https://help.ubuntu.com/community/Java

Jiang, Y. (2012). *HBase Administration Cookbook.* Birmingham: Packt Publishing Ltd.

JSON. (n.d.). *Introducing JSON*. Retrieved from http://json.org/

Kevin Weil. (n.d.). Retrieved from Linkedin: http://www.linkedin.com/in/kevinweil

Kinley, J. (2013). *The Lambda architecture: principles for architecting realtime Big Data systems*. Retrieved from jameskinley: http://jameskinley.tumblr.com/post/37398560534/the-lambda-architecture-principles-for-architecting

Kwak, H., Lee, C., Park, H., & Moon, S. (2010). *What is Twitter, a Social Network or a News Media?* Retrieved from Proceedings of the 19th international conference on World wide web, 591–600: http://snap.stanford.edu/class/cs224w-readings/kwak10twitter.pdf

Marz, N. (n.d.). *ElephantDB.* Retrieved from GitHub: https://github.com/nathanmarz/elephantdb

Marz, N., & Warren, J. (2012). *A new paradigm for Big Data.* Retrieved from Big Data. Principles and best practices of scalable realtime data systems.: www.manning.com/marz/BDmeapch1.pdf

Marz, N., & Warren, J. (n.d.). *Big Data. Principles and best practices of scalable realtime data systems*. Retrieved from Manning Publications Co.: http://www.manning.com/marz/

MongoDB. (n.d.). Retrieved from MongoDB: http://www.mongodb.org/

MySQL. (n.d.). Retrieved from MySQL: http://www.mysql.com/

NetBeans. (n.d.). Retrieved from NetBeans IDE: https://netbeans.org/

Noll, M. G. (2011, 07 17). *Running Hadoop on Ubuntu Linux - Multi-Node Cluster*. Retrieved from Michael G. Noll: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/

Noll, M. G. (2011, 07 17). *Running Hadoop on Ubuntu Linux - Single-Node Cluster*. Retrieved from Michael G. Noll: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/

Pig Latin Basics. (2014, 04 15). Retrieved from Apache Pig: http://pig.apache.org/docs/r0.12.1/basic.html

POM Reference. (n.d.). Retrieved from Apache Maven: http://maven.apache.org/pom.html#What_is_the_POM

PostgreSQL. (n.d.). Retrieved from PostgreSQL: http://www.postgresql.org/

Project Voldemort. (n.d.). Retrieved from Project Voldemort: http://www.project-voldemort.com/voldemort/

RestrictedFormats. (n.d.). Retrieved from Ubuntu documentation: https://help.ubuntu.com/community/RestrictedFormats

Romero, D. M., Meeder, B., & Kleinberg, J. (2011). *Differences in the Mechanics of Information Diffusion Across Topics: Idioms, Political Hashtags, and Complex Contagion on Twitter.* Retrieved from http://www.cs.cornell.edu/home/kleinber/www11-hashtags.pdf

Sakaki, T., Okazaki, M., & Matsuo, Y. (2010). *Earthquake Shakes Twitter Users: Real-time Event Detection by Social Sensors.* Retrieved from Proceedings of the 19th international conference on World wide web. 851-860: http://dcg.ethz.ch/lectures/fs11/seminar/paper/samuel-2-3.pdf

Scribe. (n.d.). Retrieved from GitHub: https://github.com/facebook/scribe

Servidor HP ProLiant ML310 G4. (n.d.). Retrieved from Centro de Soporte de HP: http://h20565.www2.hp.com/portal/site/hpsc/public/psi/home/?lang=es&cc=es&sp4ts.oid=3201057

SSH/OpenSSH/InstallingConfiguringTesting. (n.d.). Retrieved from Ubuntu documentation: https://help.ubuntu.com/community/SSH/OpenSSH/InstallingConfiguringTesting

Streaming API request parameters - locations. (2014, 05 01). Retrieved from Twitter Developers: https://dev.twitter.com/docs/streaming-apis/parameters#locations

The Streaming APIs. (2012, 09 24). Retrieved from Twitter Developers: https://dev.twitter.com/docs/api/streaming

Turkington, G. (2013). *Hadoop. Beginner's Guide.* Birmingham: Packt Publishing Ltd.

Twitter Developers. (n.d.). Retrieved from Twitter Developers: https://dev.twitter.com/

Twitter Libraries. (2014, 01 21). Retrieved from Twitter Developers: https://dev.twitter.com/docs/twitter-libraries

Twitter4J. (n.d.). Retrieved from Twitter4J: http://twitter4j.org

UAM. (n.d.). Retrieved from Universidad Autónoma de Madrid: www.uam.es

Ubuntu 12.04 LTS - Precise Pangolin. (n.d.). Retrieved from Ubuntu: http://releases.ubuntu.com/precise/

User Defined Functions. (n.d.). Retrieved from Apache Pig: http://pig.apache.org/docs/r0.12.1/udf.html

Using Hive to interact with HBase, Part 1. (2013, 11 11). Retrieved from Hortonworks: http://hortonworks.com/blog/hbase-via-hive-part-1/

Using Hive to interact with HBase, Part 2. (2013, 11 18). Retrieved from Hortonworks: http://hortonworks.com/blog/using-hive-to-interact-with-hbase-part-2/

VimHowTo. (n.d.). Retrieved from Ubuntu documentation: https://help.ubuntu.com/community/VimHowto

White, T. (2012). *Hadoop: The Definitive Guide.* Sebastopol, CA: O'Reilly Media, Inc.

A. Obtención de claves en Twitter Developers

Para conectarse a API de Twitter se necesitan cuatro claves por cada aplicación. Estas claves sirven para cualquiera de las APIs disponibles que se quieran utilizar.

Al utilizar Streaming API solo podrá haber una conexión haciendo uso de las claves de una aplicación y si se realizara otra conexión con las mismas claves una de las dos se cerrará, en principio la conexión más antigua. Este problema se puede solucionar creando varias aplicaciones que inserten el flujo de datos simultáneamente en la gran base de datos.

En (Twitter Developers, s.f.) se debe acceder con una cuenta de usuario de Twitter. En la esquina superior derecha elegir "*My applications*".

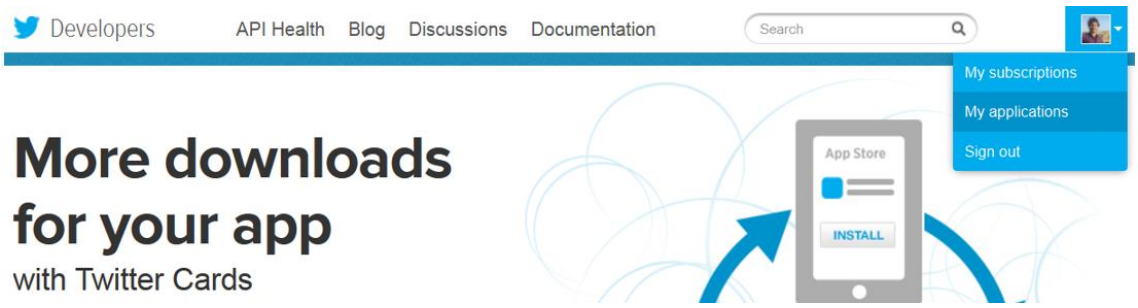


Ilustración 19 - Twitter Developers

Para crear una nueva aplicación pulsar el botón "*Create New App*".

Aparecerá una nueva página en la que se deben rellenar los datos solicitados. En la pestaña API Keys tras pulsar en el botón "*Create my Access token*" estarán disponibles las cuatro claves necesarias: API key, API secret, Access token y Access token secret.

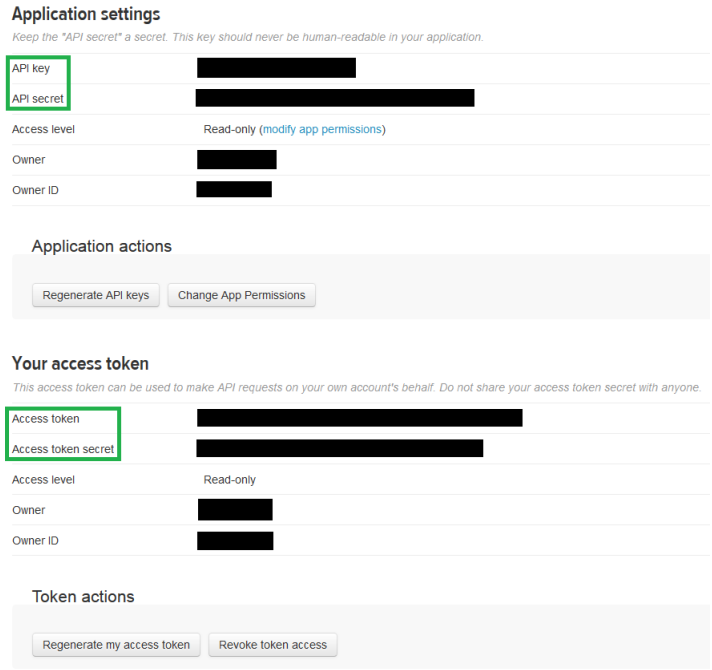


Ilustración 20 - Twitter API Keys

B. Otros scripts y código fuente

a. tdc.sh y bashrc

El shell script tdc.sh está desarrollado para automatizar todo lo possible la instalación una vez copiado el proyecto en */usr/local* y cambiado sus permisos para que todo usuario pueda modificarlo y sea más cómoda su manipulación.

twitter-data-collection/tdc.sh

```
export TDC_HOME=/usr/local/twitter-data-collection
export TDC_HDENV=/usr/local/hadoop-environment
cwd=$(pwd)

# ====================================================================
# ===== COMMON STEPS ================================================
# ====================================================================

#   superuser@hostname:~$
if [ $# -eq 2 ] && [ "$1" = "-single" ] && [ "$2" = "pres" ]; then
   echo ========================================
   echo ===== JDK 7, Java Development Kit 7 =====
   echo ========================================
   sudo add-apt-repository ppa:webupd8team/java
   sudo apt-get update
   sudo apt-get install oracle-java7-installer
   sudo apt-get install oracle-java7-set-default
   java -version
   sleep 3

   echo ================================
   echo ===== Software development =====
   echo ================================
   sudo apt-get install ubuntu-restricted-extras build-essential openssh-server maven vim

   echo
================================================================================
===============
   echo ===== Required tools and libraries to build and install the Apache Thrift compiler =====
   echo
================================================================================
===============
   sudo apt-get install libboost-dev libboost-test-dev libboost-program-options-dev libevent-dev
automake libtool flex bison pkg-config g++ libssl-dev python-dev

   echo ======================================================
   echo ===== MySQL as Hive metastore, provider, connector =====
   echo ======================================================
   sudo apt-get install mysql-server libqt4-sql-mysql libmysql-java
   sudo /usr/bin/mysql_secure_installation

   #echo ============================================================
   #echo ===== PostgreSQL as Hive metastore, provider, JDBC Driver =====
   #echo ============================================================
   #sudo apt-get install postgresql libqt4-sql-psql libpg-java libpostgresql-jdbc-java
   #sudo vim /etc/postgresql/9.1/main/postgresql.conf /etc/postgresql/9.1/main/pg_hba.conf

   echo ==========================
   echo ===== Dedicated user =====
   echo ==========================
```

```
    sudo addgroup hadoop
    sudo adduser --ingroup hadoop hduser
    sleep 3
    sudo chmod 777 /home/hduser/.bashrc
    sudo cat $TDC_HOME/confs/bashrc >> /home/hduser/.bashrc

    echo ====================================
    echo ===== Changing kernel parameters =====
    echo ====================================
    sudo vim /etc/hosts
    sudo vim /etc/security/limits.conf
    sudo vim /etc/pam.d/common-session
    sudo vim /etc/sysctl.conf
    sudo sysctl -p
fi

# ================================================================
# ===== SINGLE-NODE CLUSTER ======================================
# ================================================================

#   hduser@hostname:~$
if [ $# -eq 2 ] && [ "$1" = "-single" ] && [ "$2" = "ssh" ]; then
    echo ======================
    echo ===== ssh-keygen =====
    echo ======================
    ssh-keygen -t rsa -P ""

    echo ==========================
    echo ===== rsa-authorized =====
    echo ==========================
    cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
fi

#   superuser@hostname:~$
if [ $# -eq 2 ] && [ "$1" = "-single" ] && [ "$2" = "install" ]; then
    echo
================================================================
    echo ===== Uninstalling previous hadoop-environment version if exists =====
    echo
================================================================
    sudo rm -fr $TDC_HDENV
    sleep 3

    echo ================================================
    echo ===== Creating hadoop-environment directories =====
    echo ================================================
    sudo mkdir -p $TDC_HDENV/var
    sudo mkdir $TDC_HDENV/var/hadoop
    sudo mkdir $TDC_HDENV/var/zookeeper
    cd $TDC_HOME/software
    sudo tar -xzf *hadoop* -C $TDC_HDENV
    sudo tar -xzf *flume* -C $TDC_HDENV
    sudo tar -xzf *pig* -C $TDC_HDENV
    sudo tar -xzf *thrift* -C $TDC_HDENV
    sudo tar -xzf *elephant-bird* -C $TDC_HDENV
    sudo tar -xzf *hbase* -C $TDC_HDENV
    sudo tar -xzf *hive* -C $TDC_HDENV
    sleep 3
    cd $TDC_HDENV
    sudo mv *hadoop* hadoop
```

```
sudo mv *flume* flume
sudo mv *pig* pig
sudo mv *thrift* thrift
sudo mv *elephant-bird* elephant-bird
sudo mv *hbase* hbase
sudo mv *hive* hive
sleep 3

echo ========================
echo ===== Building Pig =====
echo ========================
cd $TDC_HDENV/pig
sudo ant
sleep 10

echo ============================
echo ===== Building Thrift =====
echo ============================
cd $TDC_HDENV/thrift
sudo bash configure --with-boost=$TDC_HDENV
sudo make install
sleep 10

echo ================================
echo ===== Building Elephant-bird =====
echo ================================
cd $TDC_HDENV/elephant-bird
sudo mvn package
sleep 10

echo ========================
echo ===== Changing JARs =====
echo ========================
cd $TDC_HDENV/pig/build/ivy/lib/Pig
sudo rm hadoop-core*.jar hadoop-test*.jar hbase*.jar zookeeper*.jar
cd $TDC_HDENV/hadoop
sudo cp hadoop-core*.jar $TDC_HDENV/pig/build/ivy/lib/Pig
sudo cp hadoop-test*.jar $TDC_HDENV/pig/build/ivy/lib/Pig
sudo cp $TDC_HDENV/hbase/hbase*.jar $TDC_HDENV/pig/build/ivy/lib/Pig
sudo cp $TDC_HDENV/hbase/lib/zookeeper*.jar $TDC_HDENV/pig/build/ivy/lib/Pig

cd $TDC_HDENV/hbase/lib
sudo rm hadoop-core*.jar libthrift*.jar
sudo cp $TDC_HDENV/hadoop/hadoop-core*.jar $TDC_HDENV/hbase/lib/
sudo cp $TDC_HDENV/hive/lib/libthrift*.jar $TDC_HDENV/hbase/lib/

echo ============================
echo ===== MySQL connector =====
echo ============================
sudo cp /usr/share/java/mysql-connector*.jar $TDC_HDENV/hive/lib/

#echo ================================
#echo ===== PostgreSQL JDBC Driver =====
#echo ================================
#sudo cp /usr/share/java/postgresql-jdbc4.jar $TDC_HDENV/hive/lib/

echo ========================================
echo ===== Modificated configuration files =====
echo ========================================
sudo cp $TDC_HOME/confs/hadoop/* $TDC_HDENV/hadoop/conf/
```

```
    sudo cp $TDC_HOME/confs/flume/* $TDC_HDENV/flume/conf/
    sudo cp $TDC_HOME/confs/hbase/* $TDC_HDENV/hbase/conf/
    sudo cp $TDC_HOME/confs/hive/* $TDC_HDENV/hive/conf/

    echo ================================
    echo ===== Changing permissions =====
    echo ================================
    sudo chmod -R 777 $TDC_HDENV/var/hadoop
    sudo chmod -R 777 $TDC_HDENV/var/zookeeper
    sudo chown -R hduser:hadoop $TDC_HDENV/hadoop
    sudo chown -R hduser:hadoop $TDC_HDENV/flume
    sudo chown -R hduser:hadoop $TDC_HDENV/pig
    sudo chown -R hduser:hadoop $TDC_HDENV/thrift
    sudo chown -R hduser:hadoop $TDC_HDENV/elephant-bird
    sudo chown -R hduser:hadoop $TDC_HDENV/hbase
    sudo chown -R hduser:hadoop $TDC_HDENV/hive
fi

# ================================================================================
# ===== MULTI-NODE CLUSTER ======================================================
# ================================================================================

#   superuser@master:~$
#   AND
#   superuser@slave:~$
if [ $# -eq 2 ] && [ "$1" = "-multi" ] && [ "$2" = "pres" ]; then
    echo ====================================
    echo ===== Changing kernel parameters =====
    echo ====================================
    sudo vim /etc/hosts
fi

#   hduser@master:~$
if [ $# -eq 3 ] && [ "$1" = "-multi" ] && [ "$2" = "ssh" ]; then
    echo ==========================
    echo ===== rsa-authorized =====
    echo ==========================
    ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduser@"$3"
fi

#   superuser@master:~$
if [ $# -eq 2 ] && [ "$1" = "-multi" ] && [ "$2" = "install-master" ]; then
    echo ========================================
    echo ===== Modificated configuration files =====
    echo ========================================
    sudo cp $TDC_HOME/confs-multi/hadoop/masters $TDC_HDENV/hadoop/conf
    sudo cp $TDC_HOME/confs-multi/hadoop/slaves $TDC_HDENV/hadoop/conf
    sudo cp $TDC_HOME/confs-multi/hbase/regionservers $TDC_HDENV/hbase/conf
    sudo vim $TDC_HDENV/hadoop/conf/masters
    sudo vim $TDC_HDENV/hadoop/conf/slaves
    sudo vim $TDC_HDENV/hbase/conf/regionservers

    echo ================================
    echo ===== Changing permissions =====
    echo ================================
    sudo chmod -R 777 $TDC_HDENV/var/hadoop
    sudo chmod -R 777 $TDC_HDENV/var/zookeeper
    sudo chown -R hduser:hadoop $TDC_HDENV/hadoop
    sudo chown -R hduser:hadoop $TDC_HDENV/hbase
fi
```

```
#   superuser@master:~$
#   AND
#   superuser@slave:~$
if [ $# -eq 2 ] && [ "$1" = "-multi" ] && [ "$2" = "install-all" ]; then
            echo ========================================
    echo ===== Modificated configuration files =====
    echo ========================================
    sudo cp $TDC_HOME/confs-multi/hadoop/core-site.xml $TDC_HDENV/hadoop/conf
    sudo cp $TDC_HOME/confs-multi/hadoop/mapred-site.xml $TDC_HDENV/hadoop/conf
    sudo cp $TDC_HOME/confs-multi/hadoop/hdfs-site.xml $TDC_HDENV/hadoop/conf
    sudo cp $TDC_HOME/confs-multi/hbase/hbase-site.xml $TDC_HDENV/hbase/conf
#sudo cp $TDC_HOME/confs-multi/hive/hive-site.xml $TDC_HDENV/hive/conf
    sudo vim $TDC_HDENV/hadoop/conf/core-site.xml
    sudo vim $TDC_HDENV/hadoop/conf/mapred-site.xml
    sudo vim $TDC_HDENV/hbase/conf/hbase-site.xml
#sudo vim $TDC_HDENV/hive/conf/hive-site.xml

    echo ================================
    echo ===== Changing permissions =====
    echo ================================
    sudo rm -fr $TDC_HDENV/var/hadoop/*
    sudo rm -fr $TDC_HDENV/var/zookeeper/*
    sudo chmod -R 777 $TDC_HDENV/var/hadoop
    sudo chmod -R 777 $TDC_HDENV/var/zookeeper
    sudo chown -R hduser:hadoop $TDC_HDENV/hadoop
    sudo chown -R hduser:hadoop $TDC_HDENV/hbase
    sudo chown -R hduser:hadoop $TDC_HDENV/hive
fi

# ======================================================================
# ===== SINGLE AND MULTI NODE CLUSTER ==================================
# ======================================================================

#   hduser@hostname:~$
if [ $# -eq 1 ] && [ "$1" = "init" ]; then
    cd $TDC_HDENV/hadoop
    bin/hadoop namenode -format
    sleep 3
    bin/start-all.sh
    sleep 60
    jps

    bin/hadoop fs -mkdir /tmp
    bin/hadoop fs -mkdir /user/hive/warehouse
    bin/hadoop fs -chmod g+w /tmp
    bin/hadoop fs -chmod g+w /user/hive/warehouse
    bin/hadoop dfs -ls "/"
    bin/hadoop dfs -ls "/user/hive"
    bin/stop-all.sh
    sleep 3
    jps
fi

#   hduser@hostname:~$
if [ $# -eq 1 ] && [ "$1" = "start" ]; then
    cd $TDC_HDENV/hadoop
    bin/start-all.sh
    sleep 3
    cd $TDC_HDENV/hbase
```

```
    bin/start-hbase.sh
    sleep 3
    jps
fi

#   hduser@hostname:~$
if [ $# -eq 1 ] && [ "$1" = "stop" ]; then
    cd $TDC_HDENV/hbase
    bin/stop-hbase.sh
    sleep 3
    cd $TDC_HDENV/hadoop
    bin/stop-all.sh
    sleep 3
    jps
fi

# ========================================================================
# ===== DATAFLOW & SCRIPTS ===============================================
# ========================================================================

#   Single-node cluster: hduser@hostname:$TDC_HOME$ sh tdc.sh flume <arg>
#   Multi-node cluster: hduser@master:$TDC_HOME$ sh tdc.sh flume <arg>
if [ $# -eq 2 ] && [ "$1" = "flume" ]; then
    cd $TDC_HDENV/flume
    case "$2" in
        "spain-pb")
            bin/flume-ng agent --conf conf/ -f $TDC_HOME/flume-twitter-source/tdc1.conf -
Dflume.root.logger=DEBUG,console -n TwitterSpainPB
            ;;
        "madrid")
            bin/flume-ng agent --conf conf/ -f $TDC_HOME/flume-twitter-source/tdc2.conf -
Dflume.root.logger=DEBUG,console -n TwitterMadrid
            ;;
        "barcelona")
            bin/flume-ng agent --conf conf/ -f $TDC_HOME/flume-twitter-source/tdc3.conf -
Dflume.root.logger=DEBUG,console -n TwitterBarcelona
            ;;
        "keywords")
            bin/flume-ng agent --conf conf/ -f $TDC_HOME/flume-twitter-source/tdc4.conf -
Dflume.root.logger=DEBUG,console -n TwitterKeywords
            ;;
        "spainpb-keywords")
            bin/flume-ng agent --conf conf/ -f $TDC_HOME/flume-twitter-source/tdc5.conf -
Dflume.root.logger=DEBUG,console -n TwitterSpainpbKeywords
            ;;
    esac
fi

#   Single-node cluster: hduser@hostname:TDC_HOME$ sh tdc.sh pig [-local|-mapreduce]
<script_name>
#   Multi-node cluster: hduser@master:TDC_HOME$ sh tdc.sh pig [-local|-mapreduce] <script_name>
if [ "$1" = "pig" ]; then
    cd $TDC_HDENV/pig
    if [ $# -eq 2 ]; then
        pig -x mapreduce $TDC_HOME/pig-scripts/"$2".pig
    else
        if [ $# -eq 3 ]; then
            case "$2" in
                "-local")
                    pig -x local $TDC_HOME/pig-scripts/"$3".pig
```

```
                ;;
            "-mapreduce")
                pig -x mapreduce $TDC_HOME/pig-scripts/"$3".pig
                ;;
        esac
    fi
  fi
fi

#   Single-node cluster: hduser@hostname:TDC_HOME$ sh tdc.sh hive [-tab] <script_name>
#   Multi-node cluster: hduser@master:TDC_HOME$ sh tdc.sh hive [-tab] <script_name>
if [ "$1" = "hive" ]; then
   cd $TDC_HDENV/hive
   if [ $# -eq 2 ]; then
      bin/hive -f $TDC_HOME/hive-scripts/"$2".hql
   else
      if [ $# -eq 3 ] && [ "$2" = "-tab" ]; then
         bin/hive -f $TDC_HOME/hive-scripts/"$3".hql > $TDC_HOME/outs/hive/"$3".tab
      fi
   fi
fi

#   hduser@hostname:TDC_HOME$ sh tdc.sh clean
if [ $# -eq 1 ] && [ "$1" = "clean" ]; then
   cd $TDC_HOME/outs/hive
   rm *
fi

# ======================================================================
cd $cwd
```

El archivo bashrc contiene todas las variables de entorno que se concatenarán a */home/hduser/.bashrc* durante la instalación.

twitter-data-collection/confs/bashrc

```
# Set JAVA_HOME
export JAVA_HOME=/usr/lib/jvm/java-7-oracle
export PATH=$JAVA_HOME/bin:$PATH

# Set TwitterDataCollection-related environment variables
export TDC_HOME=/usr/local/twitter-data-collection
export TDC_HDENV=/usr/local/hadoop-environment

# Set Hadoop-related environment variables
export HADOOP_HOME=$TDC_HDENV/hadoop
# Add Hadoop bin/ directory to PATH
export PATH=$HADOOP_HOME/bin:$PATH

# Set Flume-related environment variables
export FLUME_HOME=$TDC_HDENV/flume
# Add Flume bin/ directory to PATH
export PATH=$FLUME_HOME/bin:$PATH

# Set Pig-related environment variables
export PIG_HOME=$TDC_HDENV/pig
# Add Pig bin/ directory to PATH
```

```
export PATH=$PIG_HOME/bin:$PATH

# Set HBase-related environment variables
export HBASE_HOME=$TDC_HDENV/hbase
# Add HBase bin/ directory to PATH
export PATH=$HBASE_HOME/bin:$PATH

# Set Hive-related environment variables
export HIVE_HOME=$TDC_HDENV/hive
# Add Hive bin/ directory to PATH
export PATH=$HIVE_HOME/bin:$PATH

# HADOOP_CLASSPATH needs to be extended to include the classpath for loading HBase
HBASE_JARS=
for f in $HBASE_HOME/lib/*.jar; do
HBASE_JARS=${HBASE_JARS}:$f;
done
for f in $HBASE_HOME/*.jar; do
HBASE_JARS=${HBASE_JARS}:$f;
done
export HADOOP_CLASSPATH=$HBASE_JARS:$HBASE_HOME:$HBASE_HOME/conf

# PIG_CLASSPATH needs to be extended to include the classpath for loading HBase
export PIG_CLASSPATH=$HBASE_HOME/hbase-0.94.20.jar:$HBASE_HOME/lib/zookeeper-
3.4.5.jar:$HBASE_HOME/lib/protobuf-java-2.4.0a.jar:$PIG_CLASSPATH
```

b. FlumeTwitterSource

twitter-data-collection/flume-twitter-source/src/main/java/source/TwitterSourceConstants.java

```
package source;

public class TwitterSourceConstants {

    public static final String CONSUMER_KEY = "consumerKey";
    public static final String CONSUMER_SECRET = "consumerSecret";
    public static final String ACCESS_TOKEN = "accessToken";
    public static final String ACCESS_TOKEN_SECRET = "accessTokenSecret";

    public static final String SW_LNG_LAT = "swLngLat";
    public static final String NE_LNG_LAT = "neLngLat";
    public static final String KEYWORDS = "keywords";

}
```

twitter-data-collection/flume-twitter-source/src/main/java/source/TwitterSource.java

```
package source;

import java.util.HashMap;
import java.util.Map;
import org.apache.flume.Context;
import org.apache.flume.Event;
import org.apache.flume.EventDrivenSource;
import org.apache.flume.channel.ChannelProcessor;
import org.apache.flume.conf.Configurable;
import org.apache.flume.event.EventBuilder;
import org.apache.flume.source.AbstractSource;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import twitter4j.FilterQuery;
import twitter4j.StallWarning;
import twitter4j.Status;
import twitter4j.StatusDeletionNotice;
```

```
import twitter4j.StatusListener;
import twitter4j.TwitterStream;
import twitter4j.TwitterStreamFactory;
import twitter4j.conf.ConfigurationBuilder;
import twitter4j.json.DataObjectFactory;

/**
 * Twitter Flume Source, which pulls data from Twitter's streaming API.
 * Currently, this supports pulling from the Streaming API with two request
 * parameters: locations and keywords.
 *
 * @author Borja Gil Perez <borjagilperez at github.com>
 */
public class TwitterSource extends AbstractSource implements EventDrivenSource,
Configurable {

    private static final Logger logger = LoggerFactory.getLogger(TwitterSource.class);

    // Information necessary for accessing the Twitter API
    private String consumerKey;
    private String consumerSecret;
    private String accessToken;
    private String accessTokenSecret;

    private double[][] locations = new double[2][2];
    private String[] keywords;

    // The actual Twitter stream. It's set up to collect raw JSON data
    private TwitterStream twitterStream;

    /**
     * The initialization method for the Source. The context contains all the
     * Flume configuration info, and can be used to retrieve any configuration
     * values necessary to set up the Source.
     *
     * @param context Key-value store used to pass configuration information
     * throughout the system.
     */
    @Override
    public void configure(Context context) {

        consumerKey = context.getString(TwitterSourceConstants.CONSUMER_KEY);
        consumerSecret = context.getString(TwitterSourceConstants.CONSUMER_SECRET);
        accessToken = context.getString(TwitterSourceConstants.ACCESS_TOKEN);
        accessTokenSecret =
context.getString(TwitterSourceConstants.ACCESS_TOKEN_SECRET);

        String swString = context.getString(TwitterSourceConstants.SW_LNG_LAT);
        String neString = context.getString(TwitterSourceConstants.NE_LNG_LAT);
        if (swString != null && neString != null) {
            String[] sw = swString.split(",");
            String[] ne = neString.split(",");
            if (sw.length == 2 && ne.length == 2) {
                for (int i = 0; i < 2; i++) {
                    locations[0][i] = Double.parseDouble(sw[i].trim());
                    locations[1][i] = Double.parseDouble(ne[i].trim());
                }
            } else {
                locations = null;
            }
        } else {
            locations = null;
        }

        String keywordString = context.getString(TwitterSourceConstants.KEYWORDS);
        if (keywordString != null) {
            keywords = keywordString.split(",");
            for (int i = 0; i < keywords.length; i++) {
                keywords[i] = keywords[i].trim();
            }
        }

        ConfigurationBuilder cb = new ConfigurationBuilder();
        cb.setOAuthConsumerKey(consumerKey);
        cb.setOAuthConsumerSecret(consumerSecret);
        cb.setOAuthAccessToken(accessToken);
        cb.setOAuthAccessTokenSecret(accessTokenSecret);
```

```
        cb.setJSONStoreEnabled(true);
        cb.setIncludeEntitiesEnabled(true);
        cb.setIncludeRTsEnabled(true);

        twitterStream = new TwitterStreamFactory(cb.build()).getInstance();

    }

    /**
     * Start processing events. This uses the Twitter Streaming API to sample
     * Twitter, and process tweets.
     */
    @Override
    public void start() {
        // The channel is the piece of Flume that sits between the Source and
        // Sink, and is used to process events
        final ChannelProcessor channel = getChannelProcessor();

        final Map<String, String> headers = new HashMap<String, String>();

        // The StatusListener is a twitter4j API, which can be added to a
        // Twitter stream, and will execute methods every time a message comes
        // in through the stream
        StatusListener listener = new StatusListener() {

            // The onStatus method is executed every time a new tweet comes in
            public void onStatus(Status status) {
                // The EventBuilder is used to build an event using the headers
                // and the raw JSON of a tweet
                logger.debug(status.getUser().getScreenName() + ": " +
status.getText());

                headers.put("timestamp",
String.valueOf(status.getCreatedAt().getTime()));
                Event event =
EventBuilder.withBody(DataObjectFactory.getRawJSON(status).getBytes(), headers);

                channel.processEvent(event);
            }

            // This listener will ignore everything except for new tweets
            public void onDeletionNotice(StatusDeletionNotice statusDeletionNotice) {
            }

            public void onTrackLimitationNotice(int numberOfLimitedStatuses) {
            }

            public void onScrubGeo(long userId, long upToStatusId) {
            }

            public void onException(Exception ex) {
            }

            public void onStallWarning(StallWarning warning) {
            }

        };

        logger.debug("Setting up Twitter sample stream using consumer key {} and"
                + " access token {}", new String[]{consumerKey, accessToken});
        // Set up the stream's listener (defined above), and set any necessary
        // security information
        twitterStream.addListener(listener);

        // Set up a filter to pull out industry-relevant tweets
        logger.debug("Starting up Twitter filtering...");
        FilterQuery query = new FilterQuery().count(0);

        if (locations == null) {
            logger.debug("No locations specified");
        } else {
            String debugString = "Locations specified: ";
            debugString += "SW={" + locations[0][0] + ", " + locations[0][1] + "}, ";
            debugString += "NE={" + locations[1][0] + ", " + locations[1][1] + "}";
            logger.debug(debugString);
            query.locations(locations);
        }
```

```
        if (keywords == null) {
            logger.debug("No keywords specified");
        } else {
            String debugString = keywords.length + " keywords specified: ";
            for (int i = 0; i < keywords.length; i++) {
                debugString += keywords[i];
                if (i != keywords.length - 1) {
                    debugString += ", ";
                }
            }
            logger.debug(debugString);
            query.track(keywords);
        }

        twitterStream.filter(query);

        super.start();

    }

    /**
     * Stops the Source's event processing and shuts down the Twitter stream.
     */
    @Override
    public void stop() {
        logger.debug("Shutting down Twitter stream...");
        twitterStream.shutdown();
        super.stop();
    }

}
```

c. PigTwitterUDFs

twitter-data-collection/pig-twitter-udfs/src/main/java/mutual/UniformDate.java

```
package mutual;

import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;

/**
 * Convert 'created_at' to a uniform datetime string.
 *
 * @author Borja Gil Perez <borjagilperez at github.com>
 */
public class UniformDate extends EvalFunc<String> {

    private static final String ERROR_INPUT = "Wrong input.";
    private static final int TOKENS_LENGTH = 6;
    private static final int TIME_LENGTH = 3;

    /**
     * Method to execute.
     *
     * @param input 'created_at' field.
     * @return Uniform datetime string 'yyyyMMddHHmmssZ'
     * @throws IOException Wrong input or tokens lenght
     */
    public String exec(Tuple input) throws IOException {

        if (input == null || input.size() == 0 || input.get(0) == null) {
            throw new IOException(ERROR_INPUT);
        }

        String created_at = (String) input.get(0);
        String[] tokens = created_at.split(" ");
        if (tokens.length != TOKENS_LENGTH) {
            throw new IOException(ERROR_INPUT);
        }

        return output(tokens, this.evalType(tokens));
    }
```

```
    /**
     * Evaluate the input datetime format. REST API = "created_at":"Wed, 27 Aug
     * 2008 13:08:45 +0000". Streaming API = "created_at":"Wed Aug 27 13:08:45
     * +0000 2008"
     *
     * @param tokens Datetime splitted
     * @return 1 = REST API format, 2 = Streaming API format
     */
    private byte evalType(String[] tokens) {
        if (tokens[0].contains(",")) {
            // Wed, 27 Aug 2008 13:08:45 +0000
            return 1;
        } else {
            // Wed Aug 27 13:08:45 +0000 2008
            return 2;
        }
    }

    /**
     * Makes a datetime string with format 'yyyyMMddHHmmssZ'.
     *
     * @param tokens Datetime splitted
     * @param type 1 = REST API format, 2 = Streaming API format
     * @return Datetime string 'yyyyMMddHHmmssZ'
     * @throws IOException Wrong time lenght
     */
    private String output(String[] tokens, byte type) throws IOException {

        String day, month, year, hour, minute, second, timezone;
        String[] time;

        if (type == 1) {
            // Wed, 27 Aug 2008 13:08:45 +0000
            day = tokens[1];
            month = tokens[2];
            year = tokens[3];
            time = tokens[4].split(":");
            if (time.length != TIME_LENGTH) {
                throw new IOException(ERROR_INPUT);
            }
            hour = time[0];
            minute = time[1];
            second = time[2];
            timezone = tokens[5];
        } else {
            // Wed Aug 27 13:08:45 +0000 2008
            day = tokens[2];
            month = tokens[1];
            year = tokens[5];
            time = tokens[3].split(":");
            if (time.length != TIME_LENGTH) {
                throw new IOException(ERROR_INPUT);
            }
            hour = time[0];
            minute = time[1];
            second = time[2];
            timezone = tokens[4];
        }

        return year + this.numberMonth(month) + day + hour + minute + second +
timezone;
    }

    /**
     * Returns the string number of month.
     *
     * @param month {Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec}
     * @return String number of month
     * @throws IOException
     */
    private String numberMonth(String month) throws IOException {
        if (month.equals("Jan")) {
            return "01";
        } else if (month.equals("Feb")) {
            return "02";
        } else if (month.equals("Mar")) {
```

```
            return "03";
        } else if (month.equals("Apr")) {
            return "04";
        } else if (month.equals("May")) {
            return "05";
        } else if (month.equals("Jun")) {
            return "06";
        } else if (month.equals("Jul")) {
            return "07";
        } else if (month.equals("Aug")) {
            return "08";
        } else if (month.equals("Sep")) {
            return "09";
        } else if (month.equals("Oct")) {
            return "10";
        } else if (month.equals("Nov")) {
            return "11";
        } else if (month.equals("Dec")) {
            return "12";
        } else {
            throw new IOException(ERROR_INPUT);
        }
    }

}
```

twitter-data-collection/pig-twitter-udfs/src/main/java/relationships/Related.java

```
package relationships;

import java.io.IOException;
import java.util.Map;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.BagFactory;
import org.apache.pig.data.DataBag;
import org.apache.pig.data.Tuple;
import org.apache.pig.data.TupleFactory;

/**
 * Destinated to extract different relationships between users. Currently only
 * extract it from 'user_mentions' field.
 *
 * @author Borja Gil Perez <borjagilperez at github.com>
 */
public class Related extends EvalFunc<DataBag> {

    /**
     * Method to execute.
     *
     * @param input Field on which to extract relationships
     * @return Bag with related user tuples
     * @throws IOException
     */
    public DataBag exec(Tuple input) throws IOException {
        // "user_mentions":[{"name":"Twitter API", "indices":[4,15],
"screen_name":"twitterapi", "id":6253282, "id_str":"6253282"}]

        if (input == null || input.size() == 0 || input.get(0) == null) {
            return null;
        }

        DataBag related = BagFactory.getInstance().newDefaultBag();
        DataBag values = (DataBag) input.get(0);
        //
{([id#752911028,indices#{(4),(15)},screen_name#twitterapi,id_str#752911028,name#Twitte
r API])}

        for (Tuple t : values) {
            //
([id#6253282,indices#{(4),(15)},screen_name#twitterapi,id_str#6253282,name#Twitter
API])
            Map<String, Object> m = (Map<String, Object>) t.get(0);
            //
[id#6253282,indices#{(4),(15)},screen_name#twitterapi,id_str#6253282,name#Twitter API]
```

```
            Tuple t_aux = TupleFactory.getInstance().newTuple();
            t_aux.append((Long) Long.parseLong((String) m.get("id")));
            t_aux.append((String) m.get("id_str"));
            t_aux.append((String) m.get("screen_name"));
            related.add(t_aux);
        }

        if (related.size() == 0) {
            return null;
        } else {
            return related;
        }

    }

}
```

twitter-data-collection/pig-twitter-udfs/src/main/java/tweets/Coordinates.java

```
package tweets;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Map;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.DataBag;
import org.apache.pig.data.Tuple;
import org.apache.pig.data.TupleFactory;

/**
 * Takes the coordinates and returns them in the same API order.
 *
 * @author Borja Gil Perez <borjagilperez at github.com>
 */
public class Coordinates extends EvalFunc<Tuple> {

    private static final String ERROR_TYPE = "Unknown type.";

    /**
     * Method to execute.
     *
     * @param input 'coordinates', 'geo' or 'bounding_box' field
     * @return Tuple with coordinates in the same API order
     * @throws IOException Unknown type and get from tuple
     */
    public Tuple exec(Tuple input) throws IOException {
        // "coordinates":{"coordinates":[-75.14310264,40.05701649],"type":"Point"}
        // "geo":{"coordinates":[40.05701649,-75.14310264],"type":"Point"}
        // "bounding_box":{"coordinates":[ [ [2.2241006,48.8155414],
[2.4699099,48.8155414], [2.4699099,48.9021461], [2.2241006,48.9021461] ] ],
"type":"Polygon"}

        if (input == null || input.size() == 0 || input.get(0) == null) {
            return null;
        }

        Tuple coordinates = TupleFactory.getInstance().newTuple();
        Map<String, Object> m = (Map<String, Object>) input.get(0);
        DataBag b = (DataBag) m.get("coordinates");

        if (((String) m.get("type")).equals("Point")) {
            // {(-75.14310264),(40.05701649)}
            // {(40.05701649), (-75.14310264)}
            for (Tuple t : b) {
                coordinates.append((Float) Float.parseFloat((String) t.get(0)));
            }

            if (coordinates.size() == 0) {
                return null;
            } else {
                return coordinates;
            }
        } else if (((String) m.get("type")).equals("Polygon")) {
            ArrayList<Float> lngs_lats = new ArrayList<Float>(4 * 2);
```

```
            //
{{{{(2.2241006),(48.8155414)}),({(2.4699099),(48.8155414)}),({(2.4699099),(48.9021461
)}),({(2.2241006),(48.9021461)})})}
            for (Tuple t1 : b) {
                //
({{{(2.2241006),(48.8155414)}),({(2.4699099),(48.8155414)}),({(2.4699099),(48.9021461)
}),({(2.2241006),(48.9021461)})})
                DataBag b1 = (DataBag) t1.get(0);
                //
{{{(2.2241006),(48.8155414)}),({(2.4699099),(48.8155414)}),({(2.4699099),(48.9021461)}
),({(2.2241006),(48.9021461)})}
                for (Tuple t2 : b1) {
                    //
({(2.2241006),(48.8155414)}),({(2.4699099),(48.8155414)}),({(2.4699099),(48.9021461)})
,({(2.2241006),(48.9021461)})
                    DataBag b2 = (DataBag) t2.get(0);
                    // {(2.2241006),(48.8155414)}
                    for (Tuple t3 : b2) {
                        // (2.2241006)
                        lngs_lats.add((Float) Float.parseFloat((String) t3.get(0)));
                    }
                }
            }

            if (lngs_lats.isEmpty() || lngs_lats.size() % 2 != 0) {
                return null;
            } else {
                float sum_lngs = 0, sum_lats = 0;
                for (int i = 0; i + 2 <= lngs_lats.size(); i = i + 2) {
                    sum_lngs += lngs_lats.get(i);
                    sum_lats += lngs_lats.get(i + 1);
                }

                coordinates.append(sum_lngs / (lngs_lats.size() / 2));
                coordinates.append(sum_lats / (lngs_lats.size() / 2));
                return coordinates;
            }
        } else {
            throw new IOException(ERROR_TYPE);
        }

    }

}
```

twitter-data-collection/pig-twitter-udfs/src/main/java/tweets/Hashtags.java

```
package tweets;

import java.io.IOException;
import java.util.Map;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.DataBag;
import org.apache.pig.data.Tuple;

/**
 * Extract hashtags.
 *
 * @author Borja Gil Perez <borjagilperez at github.com>
 */
public class Hashtags extends EvalFunc<String> {

    /**
     * Method to execute.
     *
     * @param input 'hashtags' field
     * @return String hashtags separated by ' '
     * @throws IOException Get from tuple
     */
    public String exec(Tuple input) throws IOException {
        // "hashtags":[{"indices":[32,36],"text":"lol"}]

        if (input == null || input.size() == 0 || input.get(0) == null) {
```

```
            return null;
        }

        String hashtags = "";
        DataBag values = (DataBag) input.get(0);
        // {([text#lol,indices#{(32),(36)}])}

        long size = values.size(), s = 1;
        for (Tuple t : values) {
            // ([text#lol,indices#{(32),(36)}])
            Map<String, Object> m = (Map<String, Object>) t.get(0);
            hashtags += (String) m.get("text");
            if (s < size) {
                hashtags += " ";
            }
            s++;
        }

        if (hashtags.isEmpty()) {
            return null;
        } else {
            return hashtags;
        }

    }

}
```

twitter-data-collection/pig-twitter-udfs/src/main/java/tweets/UserMentions.java

```
package tweets;

import java.io.IOException;
import java.util.Map;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.DataBag;
import org.apache.pig.data.Tuple;

/**
 * Extract user mentions.
 *
 * @author Borja Gil Perez <borjagilperez at github.com>
 */
public class UserMentions extends EvalFunc<String> {

    /**
     * Method to execute.
     *
     * @param input 'user_mentions' field
     * @return String with user mentions with format 'id_str@screen_name'
     * separated by ' '
     * @throws IOException Get from tuple
     */
    public String exec(Tuple input) throws IOException {
        // "user_mentions":[{"name":"Twitter API", "indices":[4,15],
"screen_name":"twitterapi", "id":6253282, "id_str":"6253282"}]

        if (input == null || input.size() == 0 || input.get(0) == null) {
            return null;
        }

        String user_mentions = "";
        DataBag values = (DataBag) input.get(0);
        //
{([id#752911028,indices#{(4),(15)},screen_name#twitterapi,id_str#752911028,name#Twitte
r API])}

        long size = values.size(), s = 1;
        for (Tuple t : values) {
            //
([id#752911028,indices#{(4),(15)},screen_name#twitterapi,id_str#752911028,name#Twitter
API])
            Map<String, Object> m = (Map<String, Object>) t.get(0);
```

```
            //
[id#752911028,indices#{(4),(15)},screen_name#twitterapi,id_str#752911028,name#Twitter
API]
            user_mentions += (String) m.get("id_str") + "@" + (String)
m.get("screen_name");
            if (s < size) {
                user_mentions += " ";
            }
            s++;
        }

        if (user_mentions.isEmpty()) {
            return null;
        } else {
            return user_mentions;
        }

    }

}
```

twitter-data-collection/pig-twitter-udfs/src/main/java/util/MD5gen.java

```
package util;

import java.io.IOException;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;

/**
 * MD5 generator for a string.
 *
 * @author Borja Gil Perez <borjagilperez at github.com>
 */
public class MD5gen extends EvalFunc<String> {

    private static final String ERROR_INPUT = "Wrong input.";

    /**
     * Method to execute.
     *
     * @param input A tuple containing a string
     * @return MD5 string
     * @throws IOException Wrong input
     */
    public String exec(Tuple input) throws IOException {

        if (input == null || input.size() == 0 || input.get(0) == null) {
            throw new IOException(ERROR_INPUT);
        }

        MessageDigest m;
        try {
            m = MessageDigest.getInstance("MD5");
        } catch (NoSuchAlgorithmException ex) {
            throw new IOException(ex.toString());
        }
        byte[] data = ((String) input.get(0)).getBytes();
        m.update(data, 0, data.length);

        BigInteger i = new BigInteger(1, m.digest());
        return String.format("%1$032X", i);

    }

}
```

d. HBaseTwitterTables

twitter-data-collection/hbase-twitter-tables/src/main/java/admin/DropTables.java

```java
package admin;

import dao.TablesDAO;
import java.io.IOException;

/**
 * Drop all tables used for TwitterDataCollection.
 *
 * @author Borja Gil Perez <borjagilperez at github.com>
 */
public class DropTables {

    public static void main(String[] args) throws IOException, InterruptedException {
        TablesDAO dao = new TablesDAO();
        dao.dropTables();
    }

}
```

twitter-data-collection/hbase-twitter-tables/src/main/java/admin/InitTables.java

```java
package admin;

import dao.TablesDAO;
import java.io.IOException;

/**
 * Create all tables used for TwitterDataCollection.
 *
 * @author Borja Gil Perez <borjagilperez at github.com>
 */
public class InitTables {

    public static void main(String[] args) throws IOException, InterruptedException {
        TablesDAO dao = new TablesDAO();
        dao.initTables();
    }

}
```

twitter-data-collection/hbase-twitter-tables/src/main/java/dao/Table.java

```java
package dao;

/**
 * HBase table.
 *
 * @author Borja Gil Perez <borjagilperez at github.com>
 */
public class Table {

    private final String name;
    private final String[] columnFamilies;

    public Table(String name, String[] columnFamilies) {
        this.name = name;
        this.columnFamilies = columnFamilies;
    }

    public String getName() {
        return name;
    }

    public String[] getColumnFamilies() {
        return columnFamilies;
    }

}
```

twitter-data-collection/hbase-twitter-tables/src/main/java/dao/TableDAO.java

```
package dao;

import util.HBaseHelper;
import java.io.IOException;
import java.util.ArrayList;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;

/**
 * DAO tables managed.
 *
 * @author Borja Gil Perez <borjagilperez at github.com>
 */
public class TablesDAO {

    private final ArrayList<Table> tables;

    private final Configuration conf;
    private final HBaseHelper helper;

    public TablesDAO() throws IOException {
        this.tables = new ArrayList<Table>(3);
        this.tables.add(new Table("tweets", new String[]{"t", "u", "e", "p"}));
        this.tables.add(new Table("mentions", new String[]{"f"}));
        this.tables.add(new Table("mentioned", new String[]{"f"}));

        this.conf = HBaseConfiguration.create();
        this.helper = HBaseHelper.getHelper(this.conf);
    }

    public void dropTables() throws IOException {
        for (Table table : tables) {
            this.helper.dropTable(table.getName());
            System.out.println("Is the table '" + table.getName() + "' available?: " +
this.helper.isTableAvailable(table.getName()));
        }
    }

    public void initTables() throws IOException {
        for (Table table : tables) {
            this.helper.dropTable(table.getName());
            this.helper.createTable(table.getName(), table.getColumnFamilies());
            System.out.println("Is the table '" + table.getName() + "' available?: " +
this.helper.isTableAvailable(table.getName()));
        }
    }

}
```

twitter-data-collection/hbase-twitter-tables/src/main/java/util/HBaseHelper.java

```
package util;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.hbase.util.Bytes;
import java.io.IOException;

/**
 * Connection point with HBase API.
 *
 * @author Borja Gil Perez <borjagilperez at github.com>
 */
public class HBaseHelper {

    private Configuration conf = null;
    private HBaseAdmin admin = null;

    public HBaseHelper(Configuration conf) throws IOException {
        this.conf = conf;
        this.admin = new HBaseAdmin(conf);
    }
```

```
    public static HBaseHelper getHelper(Configuration conf) throws IOException {
        return new HBaseHelper(conf);
    }

    private boolean existsTable(String table) throws IOException {
        return admin.tableExists(table);
    }

    public void createTable(String table, String... colfams) throws IOException {
        createTable(table, null, colfams);
    }

    private void createTable(String table, byte[][] splitKeys, String... colfams)
            throws IOException {
        HTableDescriptor desc = new HTableDescriptor(table);
        for (String cf : colfams) {
            HColumnDescriptor coldef = new HColumnDescriptor(cf);
            desc.addFamily(coldef);
        }
        if (splitKeys != null) {
            admin.createTable(desc, splitKeys);
        } else {
            admin.createTable(desc);
        }
    }

    private void disableTable(String table) throws IOException {
        admin.disableTable(table);
    }

    public void dropTable(String table) throws IOException {
        if (existsTable(table)) {
            disableTable(table);
            admin.deleteTable(table);
        }
    }

    public boolean isTableAvailable(String table) throws IOException {
        return this.admin.isTableAvailable(Bytes.toBytes(table));
    }

}
```