

# Ecosistema Hadoop.

---

En la unidad anterior se explicó la base de Hadoop, que se conoce como Hadoop Core, y que está formado por:

- ✓ HDFS, que es la capa de almacenamiento.
- ✓ YARN, que es el gestor de los procesos que se ejecutan en el clúster.
- ✓ MapReduce, que es un modelo de programación para desarrollar tareas de procesamiento de datos.



Íñigo Sanz (Dominio público)

Sólo con estas piezas, Hadoop no ofrece mucha funcionalidad, ya que para cualquier análisis se requiere que un equipo de expertos en Big Data programe procesos en MapReduce, que no son sencillos de implementar, y peor aún, más difíciles de mantener en caso de que los datos cambien, o se quiera obtener algún dato adicional como resultado.

Además, tanto las ingestas como cualquier operación con la plataforma, se realiza mediante consolas y comandos difíciles de utilizar para cualquier persona que no sea un experto en tecnología.

Para corregir estos problemas y hacer la plataforma mucho más usable, surgen los componentes del ecosistema Hadoop. Estos componentes ofrecen funcionalidades específicas con las que Hadoop puede convertirse en una plataforma de datos global para toda la empresa, y con las que no sólo los expertos técnicos pueden utilizarla.

En esta unidad vamos a conocer todas estas funcionalidades:

- ✓ En primer lugar, conoceremos las funcionalidades asociadas con el almacenamiento y el procesamiento de datos.
- ✓ A continuación, conoceremos las funcionalidades para poder ingestar datos o automatizar procesos de trabajo.
- ✓ Después conoceremos otros interfaces que abren Hadoop a un uso más amplio, sin requerir grandes conocimientos técnicos.
- ✓ Por último, veremos las herramientas para procesamiento en streaming o tiempo real.

# 1.- Introducción al ecosistema Hadoop.

Como se vio en la unidad anterior, el core de Hadoop está formado por HDFS YARN y MapReduce. Con estos tres únicos componentes, Hadoop no ofrece apenas funcionalidad más que para almacenar datos y desarrollar programas de cierta complejidad para analizar dichos datos.

Para dotar a Hadoop de funcionalidades necesarias en proyectos Big Data empresariales, como puede ser la ingesta de información, el acceso a datos con lenguajes estándar, o las capacidades de administración y monitorización, existen otros componentes, que suelen ser proyectos opensource de Apache, que junto con Hadoop core se denomina **ecosistema Hadoop**.

Recuerda que cada componente es un proyecto Apache independiente, con su propia política de versionado (periodicidad, identificación, ...), dependencias, roadmap y estrategia de proyecto.

Los componentes que forman parte del ecosistema Hadoop son muy variados y, si bien no hay una "lista oficial", los principales componentes del ecosistema son los siguientes:

- ✓ Apache Pig.
- ✓ Apache Hive.
- ✓ Apache Impala.
- ✓ Apache HBase.
- ✓ Apache Phoenix.
- ✓ Apache Sqoop.
- ✓ Apache Flume.
- ✓ Apache Oozie.
- ✓ Apache Hue.
- ✓ Apache Zeppelin.
- ✓ Apache Ambari.
- ✓ Cloudera Manager.
- ✓ Apache Storm.
- ✓ Apache Flink.
- ✓ Apache Accumulo.
- ✓ Apache Zookeeper.

Todos estos componentes son proyectos opensource, que añaden funcionalidades distintas a Hadoop.

Para simplificar el entendimiento de estos componentes, se han clasificado en cuatro grandes grupos:

- ✓ **Componentes de acceso y procesamiento de datos:** son aquellos que ofrecen, o una forma de almacenar o acceder a los datos diferente de HDFS, por ejemplo, permitiendo el acceso a datos aleatorios y unitarios, o mecanismos para poder procesar los datos de HDFS mediante lenguajes de consulta o mediante la programación de trabajos. Los componentes de este tipo son:
  - ◆ Apache Pig.
  - ◆ Apache Hive.
  - ◆ Apache Impala.
  - ◆ Apache HBase.
  - ◆ Apache Phoenix.
  - ◆ Apache Spark.

- ✓ **Componentes de ingesta y flujos de trabajo:** son herramientas que permiten intercambiar datos con el exterior, es decir, tanto obtener datos del exterior, como poder exportar datos de Hadoop hacia otros sistemas, o herramientas que permiten automatizar los procesos o flujos de trabajo cuando se incorporan datos, por ejemplo, programando acciones que se ejecutan con cierta periodicidad. Los componentes de este tipo son:
  - ◆ Apache Sqoop.
  - ◆ Apache Flume.
  - ◆ Apache Oozie.
- ✓ **Interfaces y herramientas de trabajo:** facilitan el trabajo con Hadoop ofreciendo interfaces visuales para poder acceder a los datos, utilizar funcionalidades o administrar el sistema. Los componentes de este tipo son:
  - ◆ Apache Hue.
  - ◆ Apache Zeppelin.
  - ◆ Apache Ambari.
  - ◆ Cloudera Manager.
- ✓ **Procesamiento en streaming:** como tipo especial, estos componentes permiten resolver los casos de uso de análisis o procesamiento de datos en tiempo real, es decir, cuando los datos no están en reposo, sino que se van incorporando en Hadoop y hay que realizar un análisis de los mismos "al vuelto". Los componentes de este tipo son:
  - ◆ Apache Kafka.
  - ◆ Apache Spark.
  - ◆ Apache Flink (structured streaming).
  - ◆ Apache Storm.

## 2.- Componentes de acceso y procesamiento de datos.

---

Dentro de esta clasificación, los componentes de acceso y procesamiento de datos son aquellos que ofrecen funcionalidades para:

- ✓ Poder **acceder a los datos y analizarlos** de una forma más sencilla que mediante MapReduce. Recuerda que MapReduce simplificaba la forma en la que realizar tareas de procesamiento de datos mediante un modelo en el que se desarrollaba una fase de mapeo y otra de reducción. Este framework ayudaba para que el programador no tuviera que preocuparse de los detalles de concurrencia, control de errores, etc. pero requería programar en Java programas que, pese a todo, podían ser bastante complejos. Existen otros componentes, como Hive, que permiten hacer consultas o análisis sobre los datos sin programar, por ejemplo, utilizando un lenguaje de consultas como SQL.
- ✓ Poder almacenar los datos de una forma diferente a HDFS, para **optimizar el acceso**, por ejemplo, cuando se quiere obtener los datos aleatorios. HDFS, está optimizado para lecturas de ficheros muy voluminosos, y tenía un tamaño mínimo de lectura de 128 megabytes por defecto, lo que le hacía muy lento para acceder a datos aislados y pequeños. Algunos componentes, como HBase, permiten almacenar los datos sobre HDFS pero de una manera con la que sea muy rápido acceder a los datos.

Vamos a ver estas herramientas, que por cierto, son las más utilizadas en Hadoop, principalmente por los ingenieros de datos.

Hay muchos tipos de roles que trabajan dentro de una plataforma Big Data, los más importantes son los siguientes:

El ingeniero de datos o **data engineer** es el que hace las ingestas de los datos en crudo, tal cual están en los sistemas origen, y los procesa para que puedan ser datos utilizados en los análisis.

El científico de datos o **data scientists** es un analista que utilizando técnicas basadas en inteligencia artificial y otras técnicas de análisis de datos, puede investigar y solucionar preguntas del negocio que suelen requerir modelos predictivos, prescriptivos o incluso cognitivos.

Los analistas de negocio o **business analyst** son personas que, teniendo un conocimiento alto del negocio, pero un conocimiento menor de la tecnología, son capaces de resolver preguntas de negocio, pero utilizando herramientas más sencillas, que no requieren desarrollar algoritmos ni programas complejos

## 2.1.- Apache Pig.

---

MapReduce es un framework que requiere mucho desarrollo de código de bajo nivel para poder implementar aplicaciones de procesamiento de datos.

Por ejemplo, el código MapReduce que cuenta el número de ocurrencias de cada palabra en un fichero (por ejemplo, todos los artículos de la Wikipedia) sería el siguiente:



Apache Pig

[Apache Software Foundation](#) (Apache License)

```
1  import java io IOException
2  import java util StringTokenizer
3
4  import org apache hadoop conf Configuration
5  import org apache hadoop fs Path
6  import org apache hadoop io IntWritable
7  import org apache hadoop io Text
8  import org apache hadoop mapreduce Job
9  import org apache hadoop mapreduce Mapper
10 import org apache hadoop mapreduce Reducer
11 import org apache hadoop mapreduce lib input FileInputFormat
12 import org apache hadoop mapreduce lib output FileOutputFormat
13
14 public class WordCount {
15
16     public static class TokenizerMapper
17         extends Mapper<Object Text Text IntWritable {
18
19         private final static IntWritable one = new IntWritable(1)
20         private Text word = new Text()
21
22         public void map(Object key Text value Context context
23             ) throws IOException InterruptedException {
24             StringTokenizer itr = new StringTokenizer(value toString())
25             while (itr hasMoreTokens()) {
26                 word set(itr nextToken())
27                 context write(word one)
28             }
29         }
30     }
31
32
33     public static class IntSumReducer
34         extends Reducer<Text IntWritable Text IntWritable> {
35         private IntWritable result = new IntWritable()
36
37         public void reduce(Text key Iterable<IntWritable values
38             Context context
39             ) throws IOException InterruptedException {
40             int sum = 0
41             for (IntWritable val values) {
42                 sum += val get()
43             }
```

```

        result.set(sum)
        context.write(key, result)
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration()
    Job job = Job.getInstance(conf, "word count")
    job.setJarByClass(WordCount.class)
    job.setMapperClass(TokenizerMapper.class)
    job.setCombinerClass(IntSumReducer.class)
    job.setReducerClass(IntSumReducer.class)
    job.setOutputKeyClass(Text.class)
    job.setOutputValueClass(IntWritable.class)
    FileInputFormat.addInputPath(job, new Path(args[0]))
    FileOutputFormat.setOutputPath(job, new Path(args[1]))
    System.exit(job.waitForCompletion(true) ? 0 : 1)
}
}

```

¡Todo esto para contar cuántas veces aparece cada palabra en un texto!

Imagina que queremos hacer el siguiente análisis de los ficheros de estadísticas que tienen una línea por cada pase, gol, falta, etc. que hace un jugador en cada partido: obtener el ranking de los jugadores de la liga española que tienen una mayor efectividad de pases siempre y cuando hayan jugado más de 20 partidos en la temporada y que hayan jugado el último partido. ¿Cuántos miles de líneas tendría el programa MapReduce?

Y, por supuesto, este análisis sólo lo podrían llevar a cabo expertos que conozcan bien Hadoop, MapReduce y el lenguaje Java.

Con el objetivo de poder simplificar la implementación de programas de procesamiento de datos, Yahoo desarrolló en 2006 Pig como lenguaje de mayor nivel que pudiera ser utilizado por analistas que no disponen de grandes conocimientos de programación. El objetivo era poder proporcionar un lenguaje más sencillo de utilizar y que internamente se tradujera en código MapReduce.

Pig es un motor de ejecución sobre MapReduce (ahora también sobre Tez o Spark) que ofrece un nivel de abstracción mayor, y que utiliza como lenguaje Pig Latin, que tiene similitudes con SQL.

Veamos el mismo ejemplo de conteo de palabras con Pig Latin:

```

1 | input_lines  LOAD '/tmp/file-to-count-words' AS (line:chararray)
2 | words       FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word
3 | filtered_words  FILTER words BY word MATCHES '\\w+'
4 | word_groups   GROUP filtered_words BY word
5 | word_count    FOREACH word_groups GENERATE COUNT(filtered_words) AS count group
6 | ordered_word_count  ORDER word_count BY count DESC
7 | STORE ordered_word_count INTO '/tmp/ranking-of-words'

```

Como puedes observar, se ha reducido el número de líneas de 62 a 7.

Analizemos lo que significa cada línea:

Se abre el fichero sobre el que se quieren contar las ocurrencias de cada palabra.

- 2 Se separa cada línea en palabras.
- 3 Se filtran las palabras para eliminar signos o blancos.
- 4 Se crea un grupo para cada palabra.
- 5 Cuenta las ocurrencias de cada grupo (de cada palabra).
- 6 Se ordenan las palabras por ocurrencias.
- 7 Se almacena la lista de palabras y sus ocurrencias en un fichero.

Un programa Pig Latin se compone de una serie de operaciones, o transformaciones, que se aplican a los datos de entrada para producir resultados. En su conjunto, las operaciones describen un flujo de datos, que el entorno de ejecución de Pig traduce en una representación ejecutable y luego ejecuta. Pig convierte las transformaciones en una serie de trabajos de MapReduce que son transparentes para el programador o el usuario de Pig, lo que le permite concentrarse en los datos en lugar de la naturaleza de la ejecución.

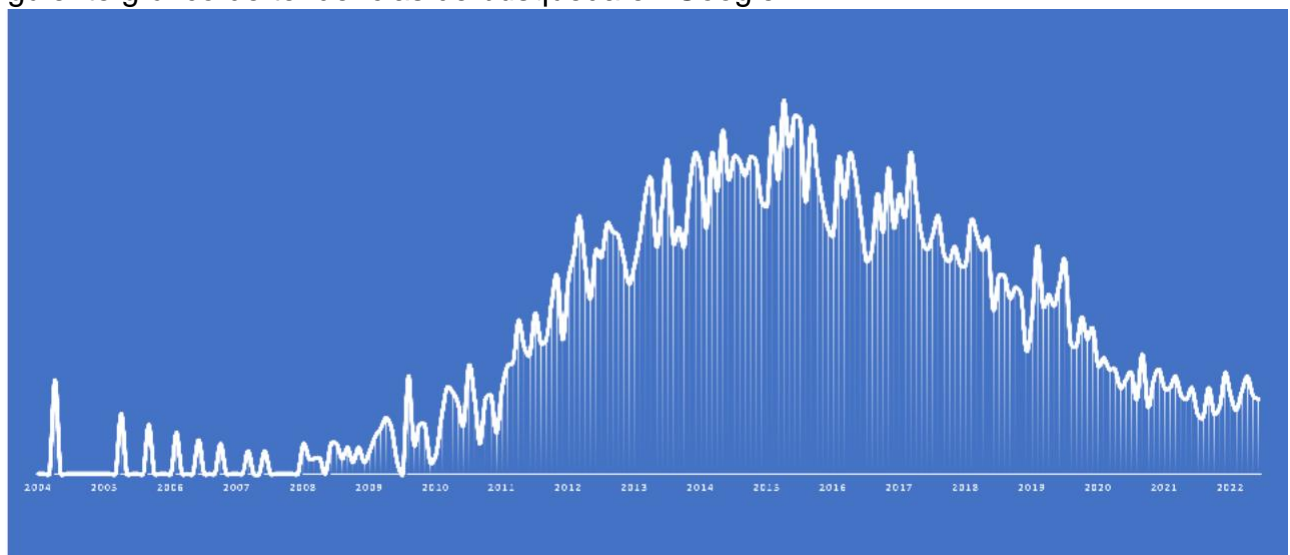
Pig ofrece comandos para filtrar, agrupar, leer, cargar, guardar o unir datos, y además, ofrece funcionalidades para revisar las estructuras de datos en un programa a medida que se escribe, o permite realizar ejecuciones sobre un subconjunto de datos de entrada, para que comprobar si hay errores en el procesamiento antes de liberarlo en el conjunto de datos completo.

Pig fue diseñado para ser extensible. Prácticamente todas las partes de un programa son personalizables: la carga, el almacenamiento, el filtrado, la agrupación y la unión pueden modificarse mediante funciones definidas por el usuario (UDF).

En ocasiones, Pig no tiene un rendimiento tan bueno como los programas escritos en MapReduce, pero suele ser una buena solución para la mayoría de casos por el tiempo de desarrollo que ahorra.

Pig se utiliza principalmente para movimiento o transformación de datos (ETL).

Desde la aparición de Apache Spark, está cayendo en desuso, como puedes ver en el siguiente gráfico de tendencias de búsqueda en Google:



## 2.2.- Apache Hive.

---

Apache Hive es sin lugar a dudas uno de los componentes más utilizados en el ecosistema Hadoop porque permite que usuarios no técnicos puedan acceder a los datos almacenados en el clúster, así como la integración de herramientas de terceros, como pueden ser herramientas de visualización de datos, con los datos de HDFS.

Y no sólo eso, además ofrece un lenguaje de consultas muy rico, potente y sencillo para los programadores, sustituyendo a MapReduce, que como has visto, requiere el desarrollo de programas complejos en lenguaje Java.

Hive, y el lenguaje de consultas que ofrece, denominado HQL, se ha convertido en un estándar de facto del mundo Big Data, así que te recomiendo que prestes mucha atención a este capítulo.

### 2.2.1.- Conceptos generales.

---

Hadoop presenta limitaciones a la hora de explotar datos en HDFS.

**Necesita programación** de procesos MapReduce para manipular datos (requiere conocimientos de programación). Incluso utilizando Pig, el lenguaje Pig Latin no es fácil de aprender y no está al alcance de analistas de negocio sin conocimientos de programación.

**Falta de integración** con herramientas de gestión o explotación de datos. La mayor parte de las herramientas que se utilizan en las empresas para analizar o visualizar datos, como Excel, las herramientas de Business Intelligence o de visualización, como PowerBI, Tableau, Microstrategy o Cognos, no pueden acceder a los datos de HDFS porque no permiten desarrollar programas MapReduce, ya que el único lenguaje que permiten utilizar para consulta de datos es SQL.

**Encarecimiento de las soluciones software.** MapReduce requiere programar soluciones para las que es difícil reutilizar código, por ejemplo, si queremos obtener una lista de jugadores ordenada por minutos de juego, y posteriormente queremos una lista ordenada de jugadores por pases realizados, apenas podremos reutilizar el código de la primera para realizar la segunda. Además, MapReduce presenta dificultades para industrializar el desarrollo, es decir, estandarizarlo, automatizar los despliegues, etc.

Ante estas dificultades, un grupo de ingenieros de Facebook desarrolló Hive como una herramienta que permite simplificar las tareas de analítica con ficheros de HDFS, **utilizando un lenguaje similar a SQL para su acceso**

Hive fue incluido como proyecto Apache, lanzando su primera versión estable en 2010.

Hive ofrece una visión de los datos en HDFS como un Datawarehouse, o una base de datos, que permite manejar grandes volúmenes de información de forma simple.

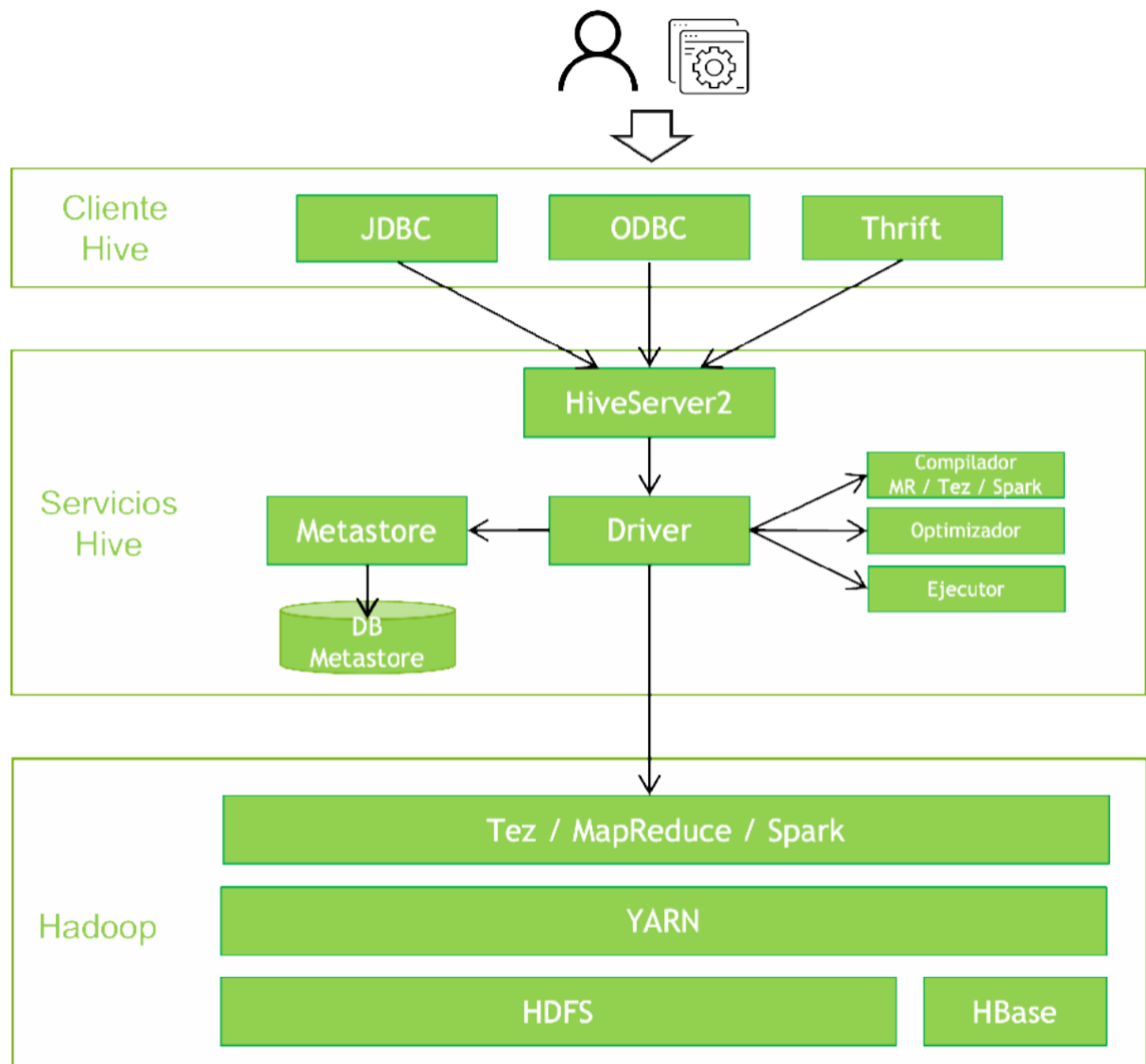


# Características y funcionalidades de Hive:

- ✓ Permite definir una **estructura relacional** (tablas, campos, etc.) sobre la información “en bruto” de HDFS. Es decir, sobre ficheros de HDFS que tengan una estructura, por ejemplo, ficheros CSV, se puede definir una o varias tablas con esos datos.
- ✓ Ofrece un **lenguaje de consultas, denominado HQL**, de sintaxis muy similar a SQL incluyendo muchas de las evoluciones de SQL para analítica: SQL-2003, SQL-2011 y SQL-2016. Esto permite que los analistas de negocio, o en general, personas sin mucho conocimiento técnico o de programación, puedan acceder a los datos de HDFS y consultarlos con un lenguaje muy rico y estándar, que ofrece operaciones para hacer cálculos, agregaciones, filtrado, etc.
- ✓ Interfaces estándar con **herramientas de terceros mediante JDBC/ODBC** es decir, permite que aplicaciones externas como Excel o PowerBI puedan acceder a los datos de HDFS para poder extraerlos o analizarlos.
- ✓ Utiliza **motores de procesamiento de Hadoop estándar** para la ejecución de consultas: MapReduce, Spark o Tez, realizando la traducción automática entre el lenguaje de consultas (HQL) y el código de los programas a ejecutar. Es decir, Hive traduce las consultas en lenguaje HQL en programas MapReduce, Tez o Spark, y las ejecuta utilizando esos motores. Esto permite aprovechar las optimizaciones que estos motores van ofreciendo durante el paso del tiempo y, ya que estos motores se ejecutan utilizando YARN como gestor, las consultas Hive conviven con el resto de aplicaciones que se ejecutan sobre Hadoop.
- ✓ **Mecanismos de seguridad** en el acceso y modificación de la información: Hive permite definir permisos a nivel de tabla o de operación (consulta, modificación, borrado, ...) con una sintaxis similar a la de las bases de datos relacionales.
- ✓ Lee ficheros con **diferentes formatos**: texto plano, ORC, HBase, Parquet y otros.
- ✓ Extensible mediante **código a medida (UDF)**: en el caso de que el lenguaje HQL no ofrezca una funcionalidad que se requiere para una consulta, se puede implementar una función a medida, denominada UDF, que consiste en un código Java que se añade a la consulta y que permite realizar operaciones no estándar, como podría ser realizar una transformación de textos o eliminar palabras u ofuscar cantidades en una consulta.
- ✓ **Orientado a consultas** aunque también ofrece operativa para creación, modificación o borrado de registros: aunque Hive tiene operaciones de inserción, borrado o modificación de registros, no está optimizado para ello, sino para consultas, especialmente para consultas complejas sobre un volumen de datos elevado. Por ejemplo, en una tabla de 5 millones de registros, obtener un sólo registro es una operación que puede tardar 3 ó 4 segundos, mientras que en una base de datos relacional estándar, es una operación que se resuelve en menos de 1 segundo. Sin embargo, cuando la consulta es compleja, por ejemplo, agregando los datos por un campo, realizando cálculos o filtrados complejos, su rendimiento es claramente superior al de las bases de datos tradicionales. Lo mismo ocurre con las inserciones o modificaciones, ya que una modificación de un campo es una operación que puede llevar un orden de magnitud más que la misma operación en una base de datos relacional.

## 2.2.2.- Arquitectura.

La arquitectura de Hive dispone de los siguientes componentes:



Íñigo Sanz (Dominio público)

### Cliente

Hive permite escribir aplicaciones en varios lenguajes, incluidos Java, Python y C++. Dispone de tres tipos de cliente:

- ✓ **Thrift Server:** Apache Thrift es un protocolo e implementación para publicar y consumir servicios binarios similar a RPC que puede ser utilizado por cualquier lenguaje de programación. Hive es capaz de ofrecer este protocolo, por lo que aplicaciones que utilicen Thrift pueden utilizar este cliente para invocar a Hive.
- ✓ **Cliente JDBC:** Hive permite que las aplicaciones Java se conecten mediante el controlador JDBC. El controlador JDBC usa Thrift para comunicarse con el servidor Hive.
- ✓ **Cliente ODBC:** el controlador ODBC de Hive permite que las aplicaciones basadas

en el protocolo ODBC se conecten a Hive. La mayoría de aplicaciones que se integran con Hive utilizan este cliente (Excel, herramientas de Business Inteligente, etc.). Al igual que el controlador JDBC, el controlador ODBC usa Thrift para comunicarse con el servidor Hive.

Los clientes se ejecutarán en la máquina que invoca a Hive, incluyendo el paquete de software de cliente de la distribución de Hive.

## Servicios Hive

Estos servicios se suelen ejecutar en un nodo frontera, donde se instala todo el software que recibe las peticiones de los clientes, las traduce a trabajos MapReduce, Tez o Spark, las ejecuta en el clúster Hadoop, y devuelve los resultados al cliente.

Los servicios principales son los siguientes:

- ✓ **HiveServer:** recibe las peticiones de los clientes e inicia su resolución. Permite peticiones concurrentes y desde interfaces variados como JDBC, ODBC o Thrift.
- ✓ **Driver:** este componente gobierna toda la ejecución de la petición, utilizando el resto de servicios como Metastore, el compilador, etc.
- ✓ **Metastore:** contiene el registro de todos los metadatos de Hive y otros componentes que requieren aplicar un modelo sobre datos existentes en HDFS o HBase. Por ejemplo, almacena la información de las tablas definidas, qué campos tienen, con qué ficheros se corresponden, etc. Permite que aplicaciones distintas a Hive puedan usar la definición de tablas definida en Hive, o al revés.
- ✓ **Compilador:** analiza la consulta. Realiza análisis semánticos y verificación de tipos en los diferentes bloques de consulta y expresiones de consulta utilizando los metadatos almacenados en metastore y genera un plan de ejecución. El plan de ejecución creado por el compilador es el DAG (Gráfico acíclico dirigido), donde cada etapa es un trabajo de mapeo/reducción, operación en HDFS, una operación de metadatos. Es decir, este componente realiza la traducción de HQL a un programa MapReduce, Tez o Spark.
- ✓ **Optimizador:** realiza las operaciones de transformación en el plan de ejecución y divide la tarea para mejorar la eficiencia y la escalabilidad.
- ✓ **Ejecutor:** ejecuta el plan de ejecución creado por el compilador y mejorado por el optimizador en el orden de sus dependencias usando Hadoop.

## Hadoop

El clúster Hadoop ejecuta los trabajos resultantes de las consultas como cualquier otra tarea YARN, es decir, asignándole una prioridad, un ApplicationMaster, etc. Por lo tanto, el desempeño de las consultas Hive dependerá en gran medida de la capacidad de ejecución del clúster Hadoop.

## 2.2.3.- HQL.

Hive permite el acceso a muchos tipos de clientes para realizar consultas, y el lenguaje que ofrece para poder realizar esas consultas se denomina HQL, o HiveQL.

Es un lenguaje con una sintaxis similar a SQL. En las primeras versiones de Hive, HQL no incluía muchas de las sentencias o funcionalidades de SQL, pero con el tiempo ha ido ganando en funcionalidad hasta llegar a ser un lenguaje prácticamente idéntico a SQL.

En primer lugar, es preciso conocer qué modelo conceptual utiliza Hive para tratar con los datos:



Íñigo Sanz (Dominio público)

Hive tiene diferentes niveles de entidades:

- ✓ El concepto general es una **tabla**, que contiene información de una misma entidad, por ejemplo, la tabla ventas, empleados o tiendas.
- ✓ La tabla tiene **campos**, que es cada una de las propiedades que pueden tener todos los elementos de una tabla, los campos de la tabla ventas podrían ser el importe, la fecha, la tienda, etc. Cada campo tiene un tipo (numérico, alfanumérico, fecha, etc.).
- ✓ Las tablas contienen **registros** o **filas**, que serían cada una de las ventas que hay en la tabla
- ✓ Las tablas se pueden dividir en **particiones**. Una partición es una división vertical de la tabla, es decir, imagina que la tabla de ventas tiene 50.000.000.000 registros porque incluye las ventas de todas las tiendas de todos los países. La mayor parte de las consultas que se realizan son para obtener diferentes métricas de una tienda (ingresos, número de ventas, ticket medio, etc.). Sin particionar, significa que para cada consulta en la que se quiera calcular una métrica, es necesario leer todos los registros de la tabla. Particionar permite dividir la tabla, por ejemplo, teniendo una partición por cada tienda y año, es decir, tendríamos una partición para la tienda 1 y el año 2022, otra para la tienda 1 y año 2021, otra para la tienda 2 y año 2022, etc. Cada partición se va a almacenar en ficheros o directorios distintos. Ahora, si queremos hacer una consulta para obtener el total de ventas de la tienda 1 en el año 2022, sólo hay que leer los ficheros asociados a esa partición, en lugar de todos los ficheros de la tabla. Por lo tanto, las particiones es un mecanismo para hacer eficientes las operaciones de Hive.
- ✓ Las **tablas** se agrupan en bases de datos, teniendo, por ejemplo, una base de datos de ventas, otra base de datos de recursos humanos, etc. La división en bases de

datos permite ajustar los permisos así como no tener una multitud de tablas gestionadas por Hive que es difícil de manejar.

En cuanto a los tipos de datos que pueden contener las columnas, Hive permite los siguientes:

- ✓ Enteros: TINYINT, SMALLINT, INT/INTEGER, BIGINT
- ✓ Decimales: FLOAT, DOUBLE, DECIMAL, NUMERIC.
- ✓ Fecha/hora: TIMESTAMP, DATE, INTERVAL.
- ✓ Cadenas: STRING, VARCHAR, CHAR.
- ✓ Otros: BOOLEAN, BINARY.
- ✓ Compuestos:
  - ◆ ARRAY: agrupa elementos del mismo tipo, es decir, cada campo almacena una lista de elementos, por ejemplo, en una tabla de ventas, la lista de artículos comprados: ["toalla", "camisa", "pantalón].
  - ◆ MAP: define parejas de clave-valor, por ejemplo, para la tabla de ventas podría haber un campo con los artículos vendidos con su referencia: {1121: "toalla", 3324: "camisa"}
  - ◆ STRUCT: define estructuras con propiedades, por ejemplo, para cada venta: {"IP origen": "127.0.0.1", "Tiempo de compra": 13123, "Tipo de cliente": "VIP"}.

Como se ha comentado anteriormente, Hive permite definir tablas, campos, etc. sobre ficheros que existen en HDFS. Por ejemplo, imagina el fichero CSV que vimos en el ejemplo de MapReduce, con las cotizaciones de todas las empresas del mundo, donde para cada línea, tenemos la fecha y hora, la empresa, el valor de cotización actual y el valor de cotización anterior:

20/01/2021 11:54:34;SANTANDER;4,54;4,49

14/05/1995 09:54;TELEFONICA;11,90;12,01

01/01/1997 08:03:21;SANTANDER;11,24;11,49

19/06/2022 11:54:22;APPLE;111,25;114,89

23/04/2003 16:32:11;ALPHABET;34,49;36,44;

21/12/2020 10:10:56;TELEFONICA;14,31;14,29

26/02/1995 14:09:40;MICROSOFT;132,29;133,95

04/05/1999 11:05:34;WALLMART;34,98;35,05

Recuerda que era un fichero CSV con miles de millones de líneas.

Hive permitiría definir una tabla, por ejemplo, denominada cotizaciones, que tendría 4 campos: la fecha, la empresa, el valor de cotización actual y el valor de cotización anterior.

Una vez definida la tabla, podríamos lanzar consultas del tipo:

- ✓ ¿Cuál ha sido la cotización media anual de cada empresa desde el año 2000?

- ✓ ¿Cuál ha sido el máximo y mínimo por año de cada empresa desde el año 2000?
- ✓ ¿Qué porcentaje de veces la cotización de cada compañía ha subido?

Para lanzar estas consultas o para definir la tabla, se utiliza el lenguaje HQL.

Hay dos tipos de consultas en HQL:

- ✓ **DDL** (Data Definition Language): permite crear, consultar o modificar estructuras de datos (tablas, bases de datos, etc.)
- ✓ **DML** (Data Manipulation Language): permite alta, baja, modificación y consulta de datos.

## Sentencias DDL

Son las sentencias con la que se ve o manipula la definición de las bases de datos, tablas, particiones, etc.

## Bases de datos

Sentencias para crear o modificar bases de datos

- ✓ Creación de una base de datos: **CREATE DATABASE my\_db**
- ✓ Visualización de las bases de datos existentes: **SHOW DATABASES**
- ✓ Uso de una base de datos para las consultas que se van a lanzar a continuación:  
**USE my\_db**
- ✓ Borrado de una base de datos: **DROP DATABASE my\_db**
- ✓ Modificación de las propiedades de una base de datos: **ALTER DATABASE my\_db SET ...**
- ✓ Visualización de detalle de una base de datos: **DESCRIBE DATABASE my\_db**

## Tablas

- ✓ Creación de una table: **CREATE TABLE**. Ejemplo para crear una tabla de inmuebles en la que existe un identificador, una dirección, la provincia, superficie y precio a partir de un fichero de texto en el que los campos están separados por tabuladores:

```
CREATE TABLE inmuebles (
  id INT,
  direccion STRING,
  provincia STRING,
  superficie INT
  precio DOUBLE)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;
```

- ✓ Visualización de las tablas que hay en una base de datos: **SHOW TABLES**
- ✓ Borrado: **DROP TABLE inmuebles**
- ✓ Modificación: **ALTER TABLE**. Ejemplo para añadir un campo país a la tabla anterior de inmuebles:

```
ALTER TABLE inmuebles
ADD COLUMNS (pais STRING COMMENT 'Nombre del país')
```

O un ejemplo para renombrar la tabla:

```
ALTER TABLE inmuebles RENAME TO casas
```

- ✓ Visualización del detalle de una tabla, es decir, para conocer los campos que tiene, formato, etc `DESCRIBE [EXTENDED|FORMATED] inmuebles`

## Sentencias DML

Son las sentencias con las que se puede ver o modificar los datos de las tablas.

## Consulta

La sentencia `SELECT` se utiliza para consultar los datos de una o varias tablas, aplicando filtros, haciendo agregaciones, uniendo tablas, etc

Es una sentencia muy potente ya que permite consultar los datos y responder a preguntas complejas sobre ellos. Su sintaxis es muy parecida a la sentencia `SELECT` de `SQL`, por lo que es recomendable que revise esta sentencia de `SQL` para poder entender bien todo lo que se puede hacer para el caso de `Hive`.

La sintaxis de `SELECT` es la siguiente:

```
SELECT [ALL    DISTINCT] select_expr select_expr
      FROM table_reference
      [WHERE where_condition]
      [GROUP BY col_list]
      [ORDER BY col_list]
      [CLUSTER BY col_list
        [DISTRIBUTE BY col_list] [SORT BY col_list]
      ]
      [LIMIT [offset ] rows]
```

Para entender su sintaxis, vemos algunos ejemplos:

- ✓ Consulta para obtener la dirección de todos los inmuebles de Madrid ordenados por superficie:

```
SELECT id, direccion FROM inmuebles
WHERE provincia = 'Madrid'
ORDER BY superficie
```

- ✓ Consulta para obtener cuántos inmuebles de un tamaño mayor a 100 metros hay para cada provincia:

```
SELECT provincia, count(*) FROM inmuebles
WHERE superficie > 100
GROUP BY provincia
```

La sentencia `SELECT` es muy rica, ofreciendo una gran cantidad de posibilidades, por ejemplo:

- ✓ Subconsultas (`SELECT FROM SELECT`): permite anidar varias consultas, tomando una de ellas los datos de la otra.
- ✓ JOIN de tablas: permite unir dos tablas para procesarlas, por ejemplo, si hay una tabla de ventas y una tabla de tiendas, se podrían unir las dos tablas para obtener un listado de la dirección de la tienda junto con el total de sus ventas mensuales.
- ✓ Operadores (UDFs): en caso de que se quiera realizar una operación que no está soportada por HQL, se puede definir una `UDF` que implemente la operación, e integrarla dentro de una consulta.

## Carga de datos en una tabla

Para cargar datos en una tabla existen tres posibilidades:

- ✓ Carga de datos en la tabla desde un fichero existente en `HDFS`.

Para este caso, se utiliza la sentencia `LOAD DATA`, por ejemplo, para cargar la tabla inmuebles con los datos de un fichero que está en el disco local:

```
LOAD DATA LOCAL INPATH './data/inmuebles.csv'  
OVERWRITE INTO TABLE inmuebles
```

- ✓ Carga de datos en una tabla del resultado de una consulta. Por ejemplo, para insertar en la tabla inmuebles el resultado de una consulta a una tabla del Catastro:

```
INSERT OVERWRITE TABLE inmuebles SELECT a.direccion, a.provincia, a.superficie FROM catastro a;
```

- ✓ Inserción de registros fila a fila mediante la sentencia `INSERT`, por ejemplo:

```
INSERT INTO TABLE inmuebles  
VALUES ('Calle Sol 23', 'Madrid', 'Madrid', 95, 234452)
```

## Modificación o borrado de registros

Si se desea modificar los valores de algunas columnas de los registros de una tabla, se utiliza la sentencia `UPDATE`

Por ejemplo, para modificar el campo provincia, cambiando el literal "La Coruña" por "A Coruña", se utilizaría la siguiente sentencia:

```
UPDATE inmuebles SET provincia = 'A Coruña' WHERE provincia = 'La Coruña'
```

Para borrar los registros de una tabla que cumplan una condición, se utiliza la sentencia `DELETE`, a la que se puede añadir condiciones de los registros a borrar.

Por ejemplo, para borrar todos los registros de los inmuebles de la provincia de Madrid con una superficie mayor a 500 metros cuadrados, se utilizaría la siguiente sentencia:

```
DELETE FROM inmuebles WHERE provincia = 'Madrid' AND superficie > 500
```



## Ejercicio Resuelto

Imagina que tenemos una tabla con todas las empresas del mundo, en la que, por ejemplo, estos serían los datos que contiene (una parte):

| Nombre de la empresa              | Facturación (M€) | Empleados | Beneficio (M€) |
|-----------------------------------|------------------|-----------|----------------|
| COMPAÑIA ESPAÑOLA DE PETROLEOS SA | 23.857           | 700       | 833            |
| MERCADONA SA                      | 23.343           | 80.000    | 622            |
| NATURGY ENERGY GROUP SA           | 23.035           | 3.000     | 1.796          |
| REPSOL PETROLEO SA                | 21.270           | 2.500     | 502            |
| ENDESA SA                         | 19.258           | 3.000     | 180            |
| INDUSTRIA DE DISEÑO TEXTIL SA     | 18.261           | 60.000    | 10.418         |

Íñigo Sanz (Dominio público)

La tabla se creó en Hive con la siguiente sentencia:

```
CREATE TABLE empresas (nombre STRING,facturación DOUBLE,empleados INT,beneficios  
DOUBLE)ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'STORED AS TEXTFILE;
```

¿Con qué sentencia se obtendría la lista de empresas de más de 1.000 empleados ordenadas de mayor a menor margen? Necesitamos obtener el listado con el nombre de la empresa y su margen. El margen se calcularía dividiendo los beneficios entre la facturación y multiplicando el resultado por 100.

Mostrar retroalimentación

```
SELECT nombre, (beneficios/facturación)*100 AS margen  
FROM empresas  
WHERE empleados > 1000  
ORDER BY margen DESC
```

## 2.3.- Apache Impala.

Las primeras versiones de Hive ofrecían un **rendimiento pobre** para consultas on-line, es decir, las consultas que requieren una respuesta en pocos segundos, por ejemplo, para actualizar un cuadro de mando interactivo, o para dar respuesta a una consulta de un analista de negocio que requiere una solución inmediata.

Impala fue desarrollado inicialmente por Cloudera como alternativa o complemento a

Hive, permitiendo prácticamente la **misma funcionalidad**, es decir, ofrecer un lenguaje SQL-like (HQL) sobre datos almacenados en HDFS o HBase

La primera versión beta fue lanzada en octubre de 2012 y era incluida en la distribución comercial de Cloudera, denominada Cloudera CDH.

En 2015 fue donada a Apache Software Foundation para su inclusión como proyecto, siendo elevada a proyecto top-level en noviembre de 2017.



[Apache Software Foundation](https://www.apache.org/licenses/)  
(Apache License)

## Características y funcionalidades

- ✓ El **rendimiento** de Impala se debe a que está implementada en código de más bajo nivel de abstracción que Hive (C++), y que dispone de un amplio conjunto de optimizaciones en la ejecución de consultas.
- ✓ Ofrece el mismo **interfaz ODBC** y lenguaje de consultas de Hive: HQL.
- ✓ Está **orientado a consultas** típicas de Business Intelligence o Business Analytics, es decir, consultas complejas que requieren agrupaciones, subconsultas, funciones analíticas, etc
- ✓ Ofrece el mismo nivel de **segurización** de Hive, es decir, autenticación mediante Hadoop / Kerberos, y autorización mediante Sentry.
- ✓ Soporta **almacenamiento en HBase y HDFS**, permitiendo para éste formatos txt, SequenceFile, Avro, RCFile y Parquet.
- ✓ Para la gestión de **metadatos** utiliza el mismo almacén/servicio que Hive: Metastore.

## Problemática

Impala tuvo un gran auge entre los años 2014 y 2020 por tener un gran incremento en el desempeño de las consultas SQL sobre Hadoop.

Sin embargo, tenía una problemática asociada a su arquitectura, y es que estaba basado en procesos en C++ que se ejecutaban en los nodos worker fuera de YARN, es decir, sin el control de YARN, por lo que en el mismo nodo se ejecutan procesos de YARN y de Impala que compiten por los recursos del sistema. YARN es un sistema optimizado para asignarle todos los recursos de un nodo, por lo que la convivencia genera muchos problemas por procesos que se quedan bloqueados, o porque el rendimiento de YARN o Impala se reduce sin tener una única herramienta donde poder monitorizar el consumo de recursos y poder gestionar este tipo de contingencias.

Asimismo, Hive ha ido ganando rendimiento con el paso de los años, especialmente tras la incorporación de Spark como motor de procesamiento, por lo que a día de hoy, la diferencia de rendimiento entre Hive e Impala es muy baja, lo que está haciendo que muchas compañías abandonen el uso de Impala y lo sustituyan por Hive.

## 2.4.- Apache HBase.

---

Dos de las principales características de HDFS (sistema de almacenamiento de Hadoop) son la inmutabilidad y el gran tamaño de bloques:

- La inmutabilidad significa que un bloque no puede modificarse, sino que en caso de querer realizar una modificación hay que descartar el bloque y escribir uno con las modificaciones realizadas.
- El gran tamaño de bloques implica que cualquier operación debe leer o escribir un número de bytes muy grande, como sabes, 128 megabytes por defecto.

Estas dos características dan a HDFS un gran ancho de banda en lecturas o escrituras masivas, pero dificultan las operaciones CRUD de datos atómicos o de pequeño tamaño, que son las operaciones que habitualmente se demandan para dar servicio a las aplicaciones operacionales de cualquier compañía.

HBase surge para dar soporte (escalable) a estas operaciones:

- ✓ Acceso aleatorio a la información, es decir, poder acceder a un dato que se encuentra en cualquier lugar de un bloque, por ejemplo, los datos de un registro concreto.
- ✓ Operaciones de actualización y borrado de datos.

## HBase es una base de datos NoSQL de modelo clave-valor sobre Hadoop.

### Características de HBase

- ✓ Toda la información de HBase es almacenada en forma de **array de bytes** en HDFS, es decir, no es un formato que pueda leerse por humanos.
- ✓ **No ofrece un lenguaje de acceso** como SQL, sino que ofrece un API, mediante Thrift y Avro o mediante un servicio HTTP RestFul, con un conjunto de operaciones bastante simple. No permite, por lo tanto, toda la riqueza de SQL para hacer agregaciones, filtrados, uniones, etc.
- ✓ Su modelo de escalado se basa en **sharding**: trocear las tablas y almacenar cada fragmento por separado en HDFS.
- ✓ **Tolerante a fallos** mediante replicación a nivel de HDFS, alta disponibilidad, consistencia a nivel de operaciones sobre filas.
- ✓ Usa **memoria para cachear** bloques y de esta forma no requerir leer de HDFS para operaciones habituales. HBase va guardando en memoria la información y sólo leerá o escribirá los bloques en HDFS cada cierto tiempo.

### Modelo de representación de datos

HBase utiliza un modelo **no relacional** con almacenamiento columnar. Tiene el concepto de tabla, siendo una tabla de la siguiente manera:

|              | Familia columnas:<br>Datos personales |              | Familia columnas:<br>Ubicación |       |
|--------------|---------------------------------------|--------------|--------------------------------|-------|
| ID (Row key) | Nombre                                | Apellidos    | Provincia                      | CP    |
| 12345678Z    | José Luis                             | Pérez García | Sevilla                        | 41001 |
| 33434123E    | María                                 | Sol Gómez    | Córdoba                        | 14002 |
| 87676545S    | Esteban                               | Pino López   | Madrid                         | 28003 |

Íñigo Sanz (Dominio público)

Las filas tienen un row-key que las identifica y organiza, la definición del row-key es un aspecto clave del diseño, ya que los accesos podrán ser de dos tipos: o directamente por row-key (se le da el row-key y HBase accede directamente al registro) o la lectura total de la tabla.

En lugar de tener el concepto column, se tiene el concepto familia de columnas: al definir una tabla, se debe indicar qué familia de columnas tiene una tabla. Posteriormente, cuando se inserta cada registro, se indica para ese registro qué columnas concretas tiene para cada familia de columnas. De esta manera, puede haber registros con columnas diferentes a otros registros.

## Operaciones

Como se ha indicado, HBase ofrece un conjunto de operaciones bastante simple:

- ✓ Listar todas las tablas de la base de datos: `list`
- ✓ Crear una tabla: `create`. Ejemplo:
  - ✦ Crear una tabla de nombre 'inmuebles' y familia de columnas 'datosinmueble'
 

```
create 'inmuebles','datosinmueble'
```
- ✓ Insertar datos en una tabla: `put`. Ejemplo:
  - ✦ Insertar en la fila con identificador "IB002221", la columna 'datosinmueble:precioventa' el valor '221221':
 

```
put 'inmuebles', 'IB002221', 'datosinmueble:precioventa','221221'
```
- ✓ Obtener una fila de una tabla: `get`. Ejemplo:
  - ✦ Obtener los datos del registro con identificador 'IB002221':
 

```
get 'inmuebles','IB002221'
```
- ✓ Leer todo el contenido de una tabla (iterar sobre la tabla): `scan`
  - ✦ Ejemplo: leer toda la tabla de inmuebles:
 

```
scan 'inmuebles'
```
- ✓ Borrar una tabla: `drop`
  - ✦ Ejemplo: borrar la tabla de inmuebles:
 

```
drop 'inmuebles'
```

## 2.5.- Apache Phoenix.

---

HBase, que podría considerarse la base de datos operacional sobre Hadoop, tiene como una de sus principales desventajas el interfaz y lenguaje de acceso, que lo hacen difícil de integrar con soluciones de terceros. Phoenix es una capa superior a HBase que permite ofrecer un interfaz SQL mediante JDBC sobre esta. Es decir, Phoenix ofrece un interfaz JDBC, con el que poder conectar múltiples aplicaciones, especialmente aplicaciones Java, y permite realizar consultas SQL sobre los datos de HBase, y con el rendimiento que ofrece, es decir, permite dar soporte a aplicaciones transaccionales con un lenguaje fácil de usar como SQL.

Phoenix traduce las sentencias SQL en operaciones de escaneo en HBase con filtros y coprocesadores, por lo que la mayor parte del procesamiento es llevado a cabo dentro de HBase, no en Phoenix como componente. En cualquier caso, cómo resuelve Phoenix las consultas es transparente para los clientes.

Phoenix permite los siguientes comandos SQL:

- ✓ SELECT
- ✓ UPSERT VALUES
- ✓ UPSERT SELECT
- ✓ DELETE
- ✓ DECLARE CURSOR
- ✓ OPEN CURSOR
- ✓ FETCH NEXT
- ✓ CLOSE
- ✓ CREATE TABLE
- ✓ DROP TABLE
- ✓ CREATE FUNCTION
- ✓ DROP FUNCTION
- ✓ CREATE VIEW
- ✓ DROP VIEW
- ✓ CREATE SEQUENCE
- ✓ DROP SEQUENCE
- ✓ ALTER
- ✓ CREATE INDEX
- ✓ DROP INDEX
- ✓ ALTER INDEX
- ✓ EXPLAIN
- ✓ UPDATE STATISTICS
- ✓ CREATE SCHEMA
- ✓ USE
- ✓ DROP SCHEMA
- ✓ GRANT
- ✓ REVOKE

## 2.6.- Apache Spark.

Apache Spark una plataforma opensource de computación y un conjunto de librerías para procesamiento en paralelo de datos sobre sistemas distribuidos. Es decir: Es el proyecto opensource de tecnologías Big Data con mayor número de contribuidores o desarrolladores.

- ✓ Es el estándar de facto para procesamiento de datos en Big Data, es decir, para implementar procesos complejos como procesos de transformación de datos (ETL), procesos de machine learning, etc.
- ✓ Ofrece una plataforma unificada y de propósito general para procesamiento Big Data, tanto batch como streaming, ofreciendo funcionalidades de carga de datos, exploración, explotación, transformaciones, etc.
- ✓ No ofrece funcionalidades de persistencia durable, es decir, Spark no tiene un sistema de almacenamiento propio, aunque permite utilizar HDFS, S3 u otros sistemas de almacenamiento.
- ✓ Garantiza la escalabilidad en operaciones paralelizables. La mayoría de operaciones son paralelizables (una suma, por ejemplo), pero hay algunas operaciones que no se pueden paralelizar y se deben hacer en un único proceso (un promedio, por ejemplo).
- ✓ Ofrece tolerancia a fallos, de manera que si se cae un nodo durante la ejecución de una tarea, esta tarea se reinicia en otro nodo y el procesamiento puede continuar.

### Origen de Apache Spark



2009

AMPLab @ UC Berkeley

Primeros desarrollos dentro de un proyecto del Proyecto Spark Research por Matei Zaharia.



2013

Apache Software Foundation

Donado a la Apache Software Foundation



2014

Databricks

Matei Zaharia y otras 6 personas fundan Databricks, una compañía que ofrece soporte y un entorno cloud para Spark.



2014

Apache Top-level

En febrero de 2014 se convirtió a Proyecto Top-Level de Apache.  
Se libera la versión 1.0

Spark surge en la Universidad de Berkeley como un proyecto de investigación de Matei Zaharia. La idea inicial de Spark era hacer un motor que aprovechando la memoria de los

nodos, pudiera acelerar los procesos de computación distribuida que hasta entonces se realizaban con MapReduce, que no era muy eficiente al guardar en HDFS los resultados parciales en cada paso.

El desarrollo inicial fue ganando funcionalidad hasta que en 2013 fue donado a la comunidad Apache.

Matei Zaharia y otras 6 personas fundan la empresa Databricks, que ofrece una solución de Spark opensource pero añadiendo soporte empresarial al igual que unos años antes, empresas como Cloudera o Hortonworks, habían realizado con Hadoop.

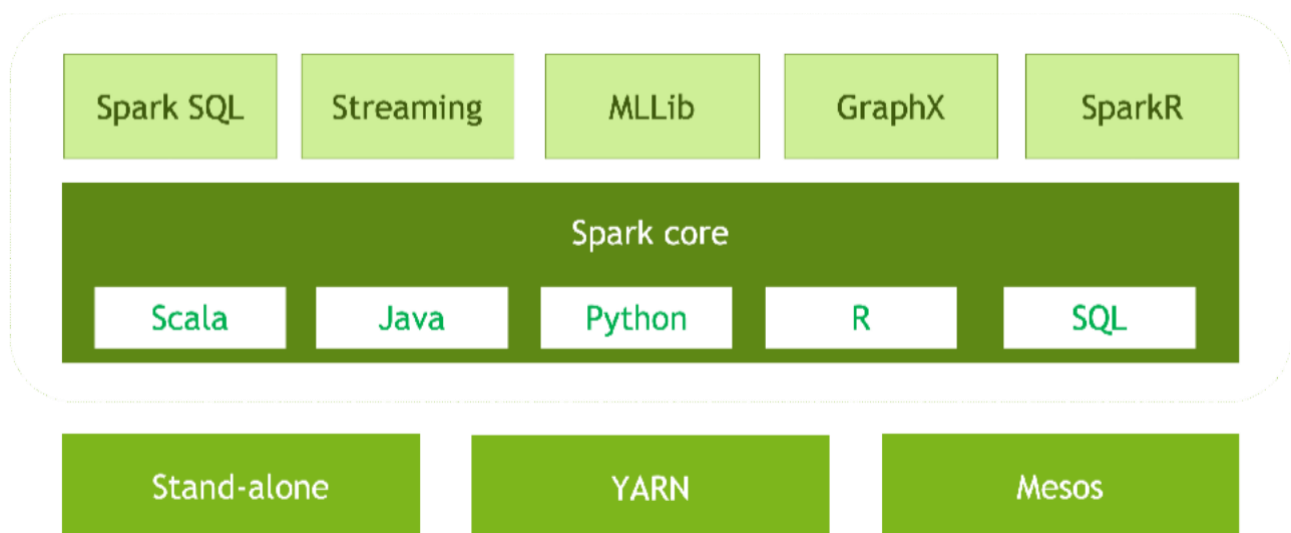
Desde entonces, Spark no ha parado de ganar tracción y funcionalidades, y es por ello que actualmente es la principal herramienta de ingeniería de datos de gran volumen.

## 2.6.1.- Arquitectura y componentes.

Spark es una plataforma, es decir, ofrece un marco de ejecución de procesos de computación distribuida con el que podemos crear nuestras propias aplicaciones, y además ofrece diferentes funcionalidades.

Es importante resaltar que Spark no reemplaza a Hadoop, sino que lo complementa, ya que en la mayoría de las ocasiones, Spark se ejecuta sobre YARN, es decir, sobre clústers Hadoop.

Los componentes de Spark son los siguientes:



Íñigo Sanz (Dominio público)

Spark puede ejecutarse en tres entornos de ejecución diferentes:

- ✓ Sobre YARN, es decir, sobre un clúster Hadoop, que es el escenario más habitual.
- ✓ Sobre Mesos, que es otro gestor de recursos parecido a YARN pero con una concepción más escalable y más orientado a entornos híbridos.
- ✓ En modo stand-alone, es decir, utilizando un motor de ejecución propio. Este modo es el habitual para despliegues pequeños o en una única máquina.

El core de Spark es un conjunto de librerías que permiten implementar programas de computación distribuida, donde el programador no tiene que preocuparse de los aspectos como la concurrencia, gestión de recursos, la sincronización, etc. al estilo de

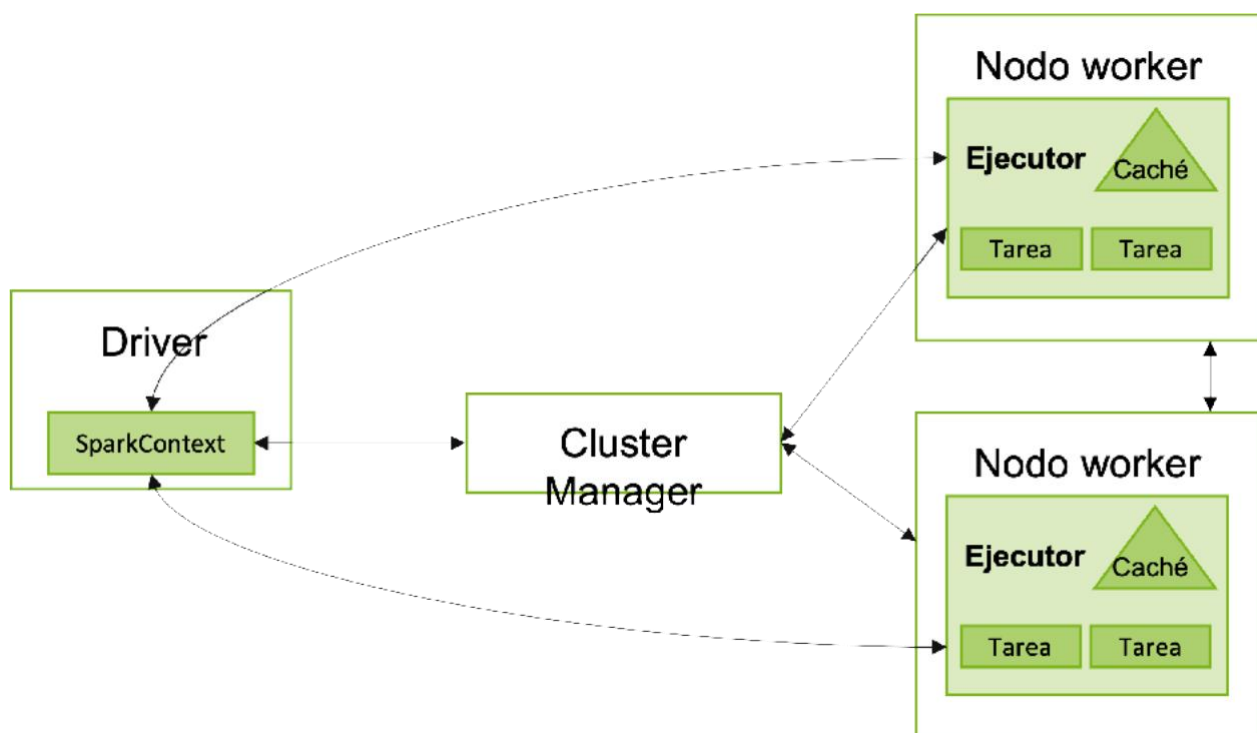


MapReduce, pero ofreciendo un conjunto de funcionalidades más numeroso, no sólo funciones de map y de reduce (se verán a continuación). El core de Spark puede utilizarse con lenguajes Scala, que es el nativo de Spark, Java, Python, o incluso R o SQL para algunas funcionalidades.

Sobre este core, Spark ofrece una serie de librerías para facilitar la construcción de aplicaciones distribuidas para propósitos más concretos:

- ✓ **Spark SQL:** librerías para tratamiento de datos utilizando un lenguaje SQL, con las operaciones que SQL ofrece en cuanto a agregaciones, filtrado, etc.
- ✓ **Structured Streaming:** librerías para realizar programas de procesamiento de datos en tiempo real.
- ✓ **MLlib:** librerías para la construcción de modelos de machine learning.
- ✓ **GraphX:** librerías para la construcción de modelos basados en grafos, con algoritmos para facilitar sus operaciones (distancias entre nodos, caminos más cortos, etc.).
- ✓ **SparkR:** una librería para poder conectar programas escritos en lenguaje R con procesamiento en Spark.

En cuanto a su arquitectura de ejecución, es la siguiente:



Íñigo Sanz (Dominio público)

El Driver ejecuta la función main (función de arranque) y se encarga de tres tareas:

- ✓ Mantener la información/estado de la aplicación.
- ✓ Responder a las entradas de datos del usuario.
- ✓ Analizar, distribuir y planificar el trabajo de los ejecutores.

Los ejecutores se ocupan de la ejecución de las tareas / trabajos que el driver les asigna y de informar al driver del estado de estos trabajos. Los ejecutores se ejecutan en diferentes nodos del clúster.

El gestor del cluster controla el entorno de ejecución donde en el que se ejecutan las

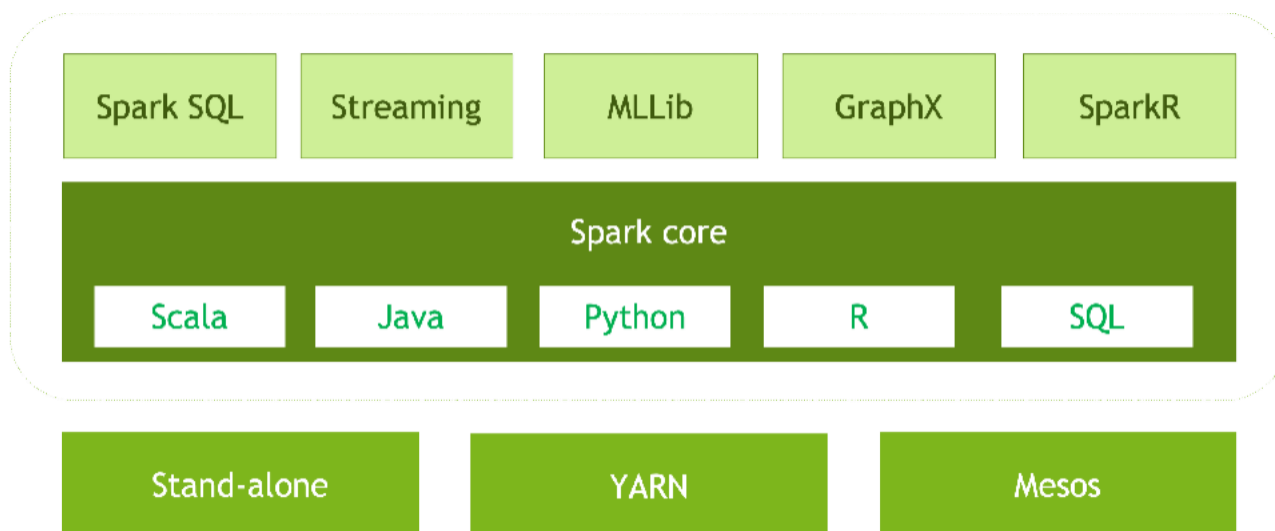


aplicaciones, disponibilizando recursos para los ejecutores, mediante su conexión con el sistema de procesamiento, que puede ser YARN, Mesos o en modos stand-alone y local.

## 2.6.2.- Detalle de los componentes de Apache Spark.

---

Recuerda los principales componentes de Apache Spark:



Íñigo Sanz (Dominio público)

### Spark Core

Spark Core **es la base de Spark**. Ofrece funcionalidades para gestión de tareas distribuidas, programación, orquestación, así como funcionalidades básicas de entrada y salida de datos mediante un API en varios lenguajes.

**Resilient Distributed Dataset (RDD)** es la principal abstracción de datos de Spark Core: es una representación simbólica de una colección de elementos con los que se puede trabajar en paralelo.

Los RDDs son inmutables, es decir, no se pueden modificar sus elementos, sino que se crean nuevas copias si es necesario alguna modificación.

Existen dos tipos de operaciones sobre los RDDs:

- ✓ **Transformaciones:** permiten transformar un RDD para generar otro. Las transformaciones no se ejecutan de forma inmediata, sino que se van concatenando para su ejecución lo más tarde posible (al ejecutarse acciones). Por ejemplo, 3 filtrados seguidos de un RDD pueden unirse. Suelen ejecutarse en paralelo.
- ✓ **Acciones:** persisten o re-distribuyen un RDD y **no se ejecutan en paralelo**. Se ejecutan de forma inmediata. Hay 4 tipos:
  - ✦ Ver datos en consola.
  - ✦ Redistribuir los datos en los nodos de computación.
  - ✦ Recoger datos para mapearlos en objetos nativos cada lenguaje.
  - ✦ Escribir en repositorios de datos.

Las principales transformaciones que ofrece Spark son las siguientes:

- ✓ **map(func)**: aplica una transformación (func) a todos los elementos del RDD generando un nuevo RDD.
- ✓ **flatMap(func)**: aplica una transformación a todos los elementos del RDD pudiendo resultar más de un elemento por cada elemento original del RDD.
- ✓ **filter(func)**: genera un nuevo RDD resultante de aplicar el filtro func a todos los elementos del RDD original.
- ✓ **distinct()**: genera un nuevo RDD eliminando duplicados del original.
- ✓ **union(rdd)**: genera un nuevo RDD que es la unión del original con el enviado como parámetro.

Y las principales acciones son las siguientes:

- ✓ **count()**: devuelve el número de elementos del RDD.
- ✓ **reduce(func)**: aplica una función de agregación sobre los elementos del RDD.
- ✓ **take(n)**: devuelve los n primeros elementos del RDD.
- ✓ **collect()**: devuelve todos los elementos del RDD (al driver).
- ✓ **saveAsTextFile(path)**: escribe los elementos de un RDD en un repositorio (filesystem, HDFS, S3, etc.)

## SparkSQL

Es un componente sobre Spark Core para el procesamiento de datos estructurados o semiestructurados.

Ofrece:

- ✓ Un nivel de abstracción de datos estructurados o semi-estructurados superior a los RDD: **DataSet y DataFrame**
- ✓ **Funcionalidades** para acceder o manejar este tipo de información.
- ✓ Soporte a **lenguaje SQL** así como interfaces de acceso desde herramientas relacionales.

Las principales ventajas de SparkSQL es su facilidad de uso y productividad en la construcción de procesos de transformación o consultas, ya que es más sencillo utilizar lenguaje SQL para transformar o procesar datos que utilizar las transformaciones y acciones que se ofrecen en Spark Core.

SparkSQL permite trabajar con las siguientes fuentes de datos:

- ✓ Documentos JSON
- ✓ Bases de datos relacionales mediante conexión JDBC
- ✓ Ficheros con formato Apache Parquet (formato columnar comprimido).
- ✓ Apache Hive.
- ✓ Bases de datos MySQL mediante conector nativo.
- ✓ Bases de datos PostgreSQL mediante conector nativo.
- ✓ Ficheros en HDFS.
- ✓ Ficheros en Amazon S3.
- ✓ Ficheros en formato Apache Avro.
- ✓ Ficheros CSV

- ✓ Bases de datos Apache HBase.
- ✓ Bases de datos Elasticsearch.
- ✓ Bases de datos Apache Cassandra.
- ✓ Bases de datos Amazon RedShift.

## DataFrame y DataSet

**DataFrame** es una abstracción que representa tablas de datos (filas y columnas), sobre las que Spark ofrece un API para su manejo y transformación, así como la capacidad de uso de SQL como lenguaje de acceso.

**DataSet** es una abstracción similar a DataFrame aunque con la característica de que las filas tienen unos atributos de tipo establecido (las columnas tienen un tipo previamente definido).

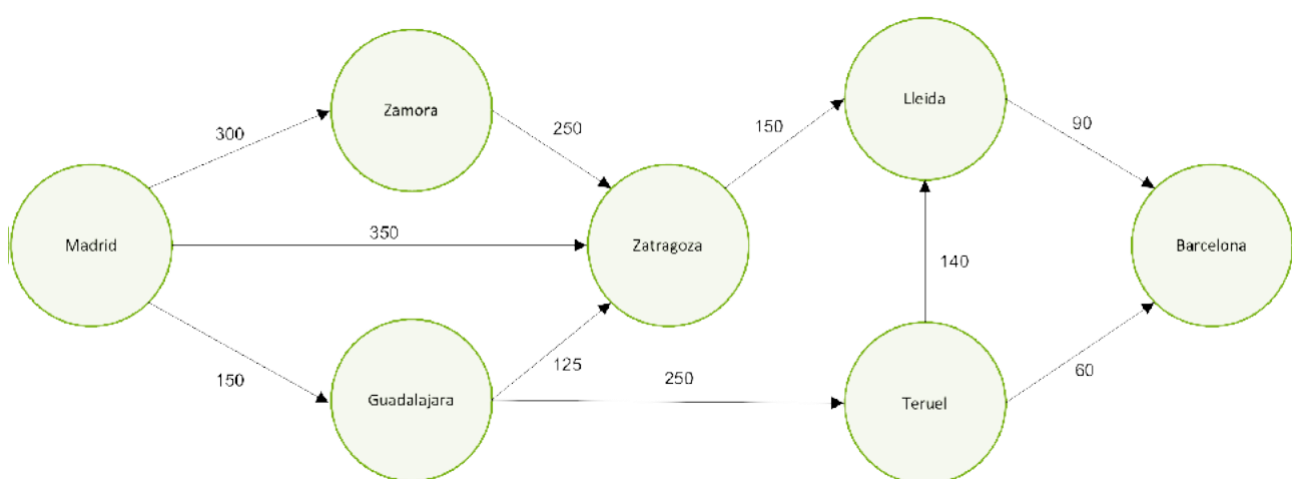
Al igual que RDD, los DataFrames y DataSets son inmutables.

Con RDD los objetos son manejados por los programas Spark de forma opaca, sin una estructura concreta (que sólo conoce el desarrollador), sin embargo, con DataFrames, se define un esquema y se permite el uso de lenguaje estándar para el acceso o manejo de los datos. Así que se puede decir que un DataFrame es un RDD con un esquema.

## GraphX

GraphX es un componente de Spark que ofrece un API para manejo de datos como grafos así como funcionalidades de procesamiento de grafos en paralelo.

Un grafo (dirigido) es una estructura que contiene vértices y aristas, como en el siguiente ejemplo:



Íñigo Sanz (Dominio público)

Para el ejemplo, se podrían plantear un caso de uso que podría ser calcular la ruta más corta entre dos puntos.

GraphX extiende RDD con un Resilient Distributed Property Graph: un grafo dirigido, formado por nodos y aristas, con propiedades en ambos elementos.

Las funcionalidades que ofrece son `subgraph` `joinVertices` `mapReduceTriplets` etc. que permiten manipular y analizar el grafo con diferentes algoritmos.

## MLLib

La librería MLLib ofrece funcionalidades de para facilitar el diseño, el entrenamiento y la construcción de modelos de Machine Learning:

- ✓ **Algoritmos:** clasificación, clustering, regresión y filtros colaborativos.
- ✓ **Obtención de features:** extracción, transformación, reducción de dimensionalidad y selección.
- ✓ **Generación de modelos:** entrenamiento, evaluación, aplicación.
- ✓ **Persistencia:** almacenamiento y carga de modelos.
- ✓ **Utilidades:** funciones de álgebra lineal, estadística, etc.

### 2.6.3.- Ventajas y desventajas.

---

Spark tiene una serie de ventajas frente a otras tecnologías de Big Data como Hive o MapReduce:

- ✓ Utiliza memoria para persistencia efímera de datos, lo que incrementa en órdenes de magnitud su **velocidad en procesos iterativos** frente a frameworks como MapReduce.
- ✓ Permite utilizar un **mismo paradigma** y modelo para distintos tipos de procesamiento (batch y streaming).
- ✓ Se **integra** con distintos sistemas de persistencia y motores de gestión de datos como Hive, HDFS etc
- ✓ Ofrece un **API muy rica**: procesamiento de datos, transformaciones, machine learning, grafos, etc
- ✓ Frente a otros paradigmas como MapReduce requiere desarrollar una **cantidad de código muy inferior**
- ✓ Tiene una **comunidad** de más de 1000 desarrolladores contribuyendo en el proyecto, por lo que su evolución está siendo muy rápida y está incorporando muchas funcionalidades.

Sin embargo, también tiene una serie de dificultades o desventajas que es necesario tener en cuenta

- ✓ Está orientado a **perfiles muy técnicos**, con conocimientos sólidos de programación. Suelen ser perfiles con salarios altos y con niveles de rotación elevados.
- ✓ Es **difícil de optimizar** o depurar en producción. A veces, un programa desarrollado con Spark no funciona correctamente en producción, por ejemplo, porque consume demasiados recursos o porque se ralentiza en exceso. Depurar este tipo de casuística suele ser más difícil que en otras tecnologías como MapReduce.
- ✓ Aunque es una plataforma con un nivel de madurez alto está sufriendo **bastantes cambios** lo que dificulta el manejo de distintas versiones, nuevas funcionalidades, etc

## 3.- Componentes de ingesta de datos y flujos de trabajo.

### 3.1.- Apache Sqoop.

---

Apache Sqoop es una herramienta diseñada para transferir datos entre Hadoop y repositorios relacionales, como bases de datos o sistemas mainframe. Permite, por ejemplo, importar datos existentes en una tabla de Oracle y almacenarlos en HDFS en un fichero CSV para poder procesarlo y exportar su resultado para su explotación en Oracle.

Consiste en un programa de línea de comandos que traduce las órdenes en programas MapReduce que son lanzados en el clúster.

Algunos comandos que ofrece:

- ✓ **sqoop-import**: importa una tabla de una base de datos relacional en HDFS. Ejemplo: importar una tabla de empleados de una base de datos externa en Hive:

```
sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES --hive-import
```

- ✓ **sqoop-export**: exporta un conjunto de ficheros de HDFS a una base de datos relacional. Ejemplo: exportar los ficheros de Hive a una tabla de una base de datos externa:

```
sqoop export --connect jdbc:mysql://db.example.com/foo  
--table bar  
--export-dir /results/bar_data
```

Es importante entender su modelo de funcionamiento: en Sqoop, se genera un proceso MapReduce que se ejecutará en varios nodos del clúster, y cada uno de ellos abrirá una conexión con la base de datos de origen o destino. Por este motivo, es importante revisar que la base de datos vaya a soportar tal número de conexiones concurrentes. En caso de que exista un límite, se puede indicar a Sqoop cuál es el límite de conexiones.

### 3.2.- Apache Flume.

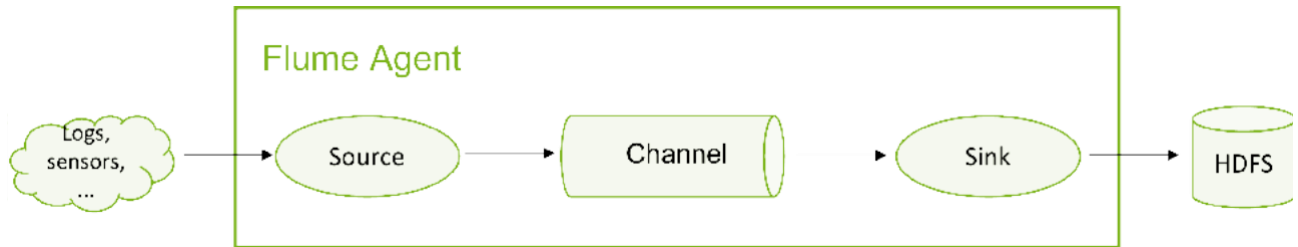
---

Flume es un servicio distribuido y confiable para recoger, agregar y mover datos generados de forma continua y atómica, como es el caso de los logs.

Normalmente se usa para recoger y almacenar en Hadoop datos provenientes de sistemas de log, social media, IoT, emails, etc. que generan sus datos como un stream de datos, es decir, una fuente de datos continua.

Su arquitectura es la siguiente:





Íñigo Sanz (Dominio público)

Sus componentes principales son:

- ✓ **Fuentes / Sources:** consume datos de una fuente de datos externa. Al recibir un evento/dato, el source lo almacena en uno o varios canales.
- ✓ **Canal / Channel:** almacena los datos hasta que otro agente de Flume lo consume.
- ✓ **Sumidero / Sink:** recoge los eventos del canal, los procesa y los envía a un repositorio externo, como puede ser HDFS

Flume garantiza la entrega de todos los datos recibidos por las fuentes, y ofrece capacidad de recuperación ante caídas.

Los flujos de Flume pueden configurarse, pudiendo montar topologías complejas, por ejemplo, volcando los datos de una fuente en dos canales diferentes, o de un sumidero en dos destinos distintos, o también, anidando varias topologías de source-channel-sink > source-channel-sink

Algunos tipos de fuente, canales y sumideros ofrecidos por Flume “de caja”:

- ✓ **Fuentes:** Avro, Thrift, Exec, Spooling directory, Taildir, Twitter, Kafka, NetCat, Syslog, HTTP o Custom.
- ✓ **Canales:** memoria, JDBC, Kafka, File o Custom.
- ✓ **Sumideros:** HDFS, Hive, Logger, Avro, Thrift, IRC, Hbase, Kafka, HTTP, File o Custom.

## 3.3.- Apache Oozie.

---

Apache Oozie facilita el lanzamiento de flujos de trabajo (workflows) cuando se cumplen determinadas condiciones. Por ejemplo:

1. Un sistema X genera un fichero CSV con los datos de los pedidos del día anterior.
2. El fichero es enviado a Hadoop.
3. Al llegar el fichero, es necesario guardarlo para su historificación, descargar de la base de datos de CRM el maestro de clientes y generar un tablón resumido con las ventas por categorías, códigos postales, etc.
4. Al terminar el proceso, es necesario enviar un email al director de operaciones para notificarle de que ya puede acceder a la herramienta de informes para ver las ventas del día anterior.

\* Los pasos 3 y 4 serían controlados por Oozie.

Oozie permite diseñar flujos de trabajo. Un flujo de trabajo está formado por:

- ✓ **Condiciones de inicio:** puede ser una condición temporal (por ejemplo, todos los días a las 14:00) o existencia de datos (por ejemplo, cuando en el directorio /temp haya un fichero nuevo).
- ✓ **Mecanismos de control:** bifurcaciones, decisiones, uniones.
- ✓ **Acciones:** MapReduce, Hive, Sqoop, DistCp, Spark, Pig, shell, Sqoop, SSH, email o acciones a medida.
- ✓ **Estados de fin y error.**

Las definiciones de los flujos de trabajo de Oozie están escritas en hPDL (un lenguaje de definición de procesos XML similar a JBOSS JBPM jPDL).

Los flujos de trabajo de Oozie se pueden parametrizar (usando variables como \${inputDir} dentro de la definición del flujo de trabajo). Al enviar un trabajo de flujo de trabajo, se deben proporcionar valores para los parámetros. Si se parametriza correctamente (es decir, se utilizan diferentes directorios de salida), varios trabajos de flujo de trabajo idénticos pueden realizarse al mismo tiempo.

Asimismo, en un flujo de control, una acción no puede ejecutarse hasta que la acción anterior se haya completado.

La gestión de flujos es realizada mediante una aplicación web que arranca Oozie.

## 4.- Interfaces y herramientas de trabajo.

---

### 4.1.- Hue

Hue, que es uno de los pocos componentes que **no es gobernado por Apache**, aunque sí es un proyecto **opensource**, es un interfaz **web** que permite trabajar con **Hadoop de un modo sencillo** para navegar por el espacio de nombres de **HDFS**, lanzar consultas a Hive/Impala, *etc.* visualizar las tareas en ejecución, *etc.*

Todas las distribuciones de Hadoop incluyen este componente, ya que facilita enormemente el uso de Hadoop por parte de usuarios de negocio o técnicos. Siendo un componente sencillo, es de los más utilizados ya que elimina la necesidad de acceder a Hadoop por consola o tener que conocer los comandos de HDFS para crear directorios, copiar ficheros, *etc.*

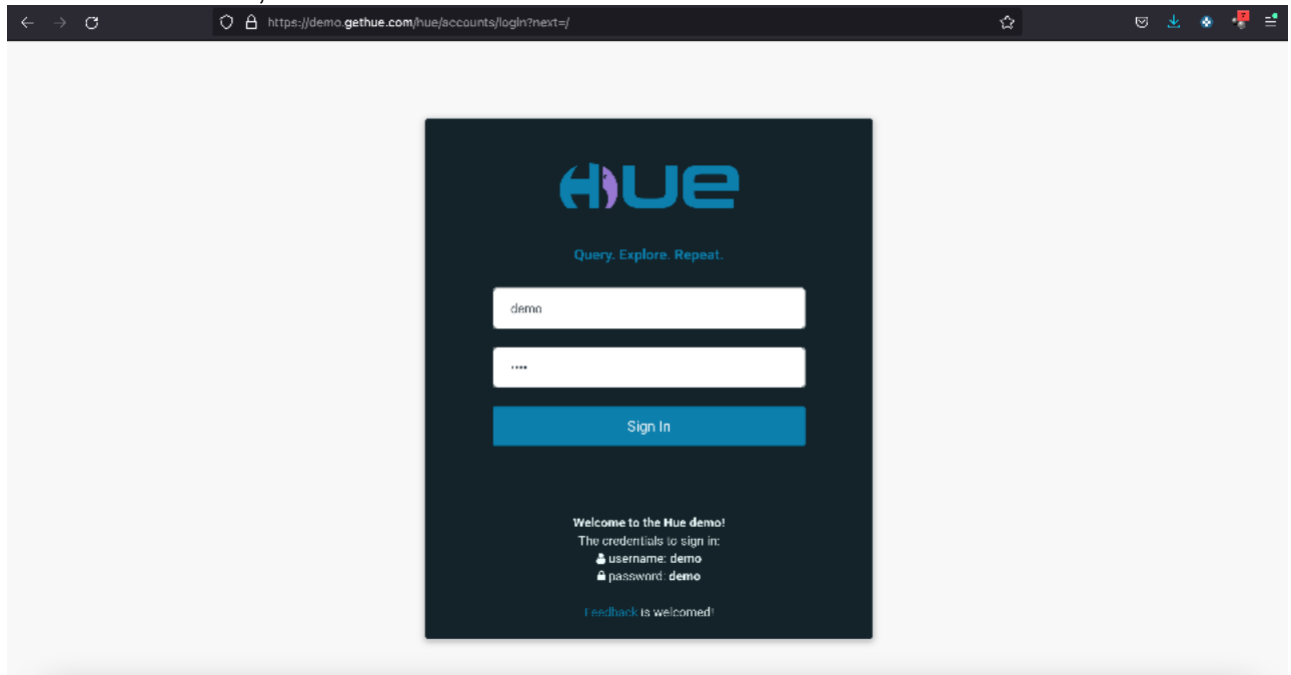
Ofrece, entre otras, las siguientes funcionalidades y capacidades:

- ✓ **Acceso securizado** mediante usuario y contraseña.
- ✓ **Editor SQL/HQL** para Hive e Impala, además de para cualquier base de datos SQL (MySQL, Oracle, Phoenix, *etc.*).
- ✓ **Visualización** de tablas, bases de datos.
- ✓ Ejecución de **consultas** y visualización del histórico de consultas.
- ✓ Visualización de **resultados**
- ✓ **Dashboards** para visualización de datos tomando como fuentes las consultas creadas en los editores.
- ✓ **Explorador de datos** que permite navegar por los sistemas de ficheros de HDFS, Amazon S3, así como las tablas y bases de datos en Hive o Hbase.
- ✓ Explorador de **tareas** y **Jobs** lanzados.
- ✓ Diseño/definición de **flujos de trabajo** (Oozie).



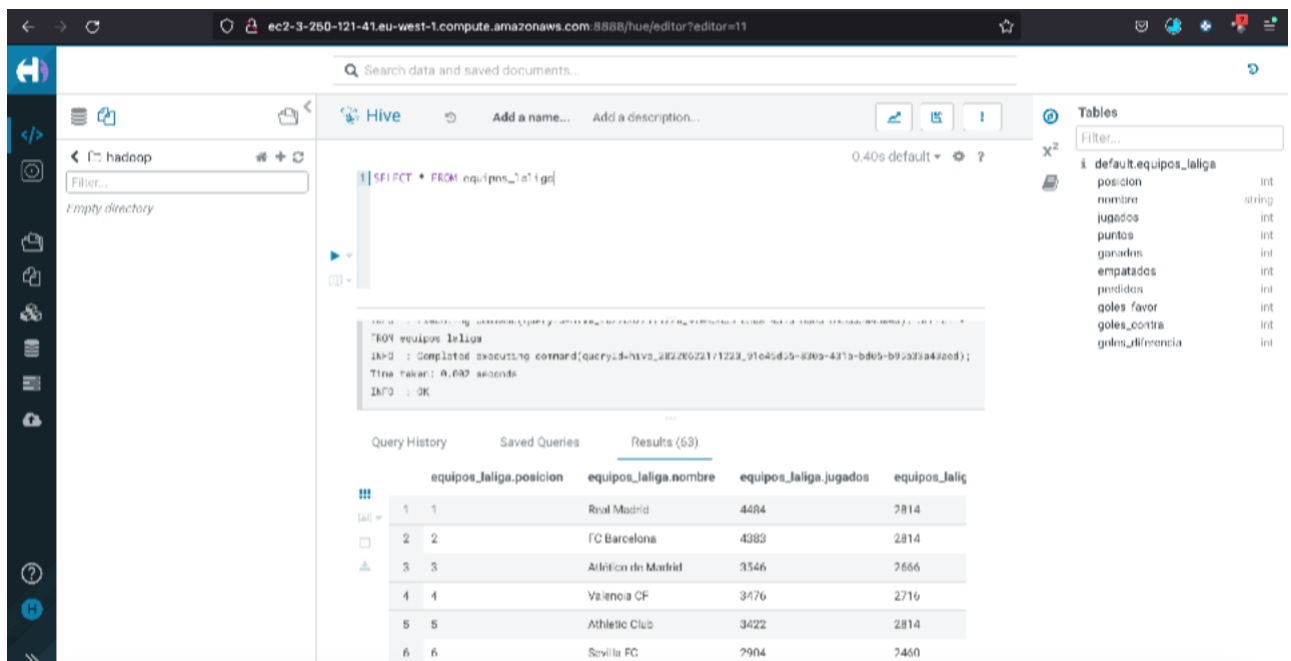
# Algunos ejemplos de pantallas de Hue:

Pantalla de acceso, con identificación de usuario:



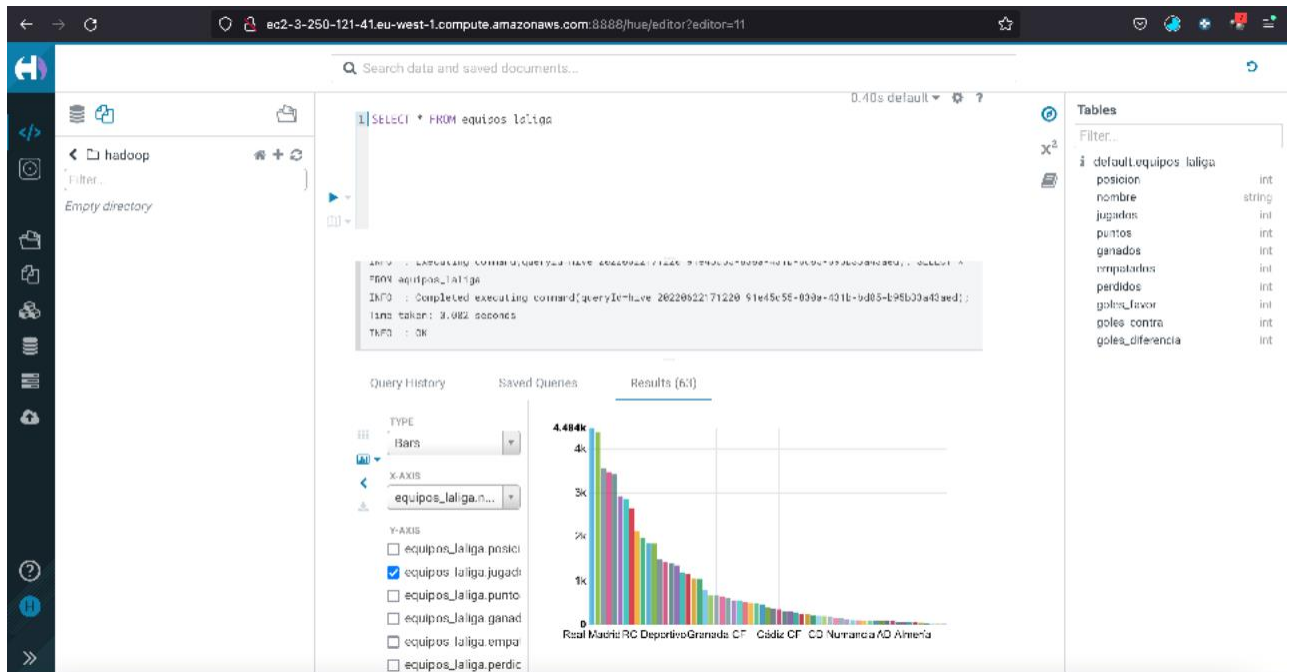
Íñigo Sanz (Dominio público)

Editor de consultas Hive y ejecución de las mismas:



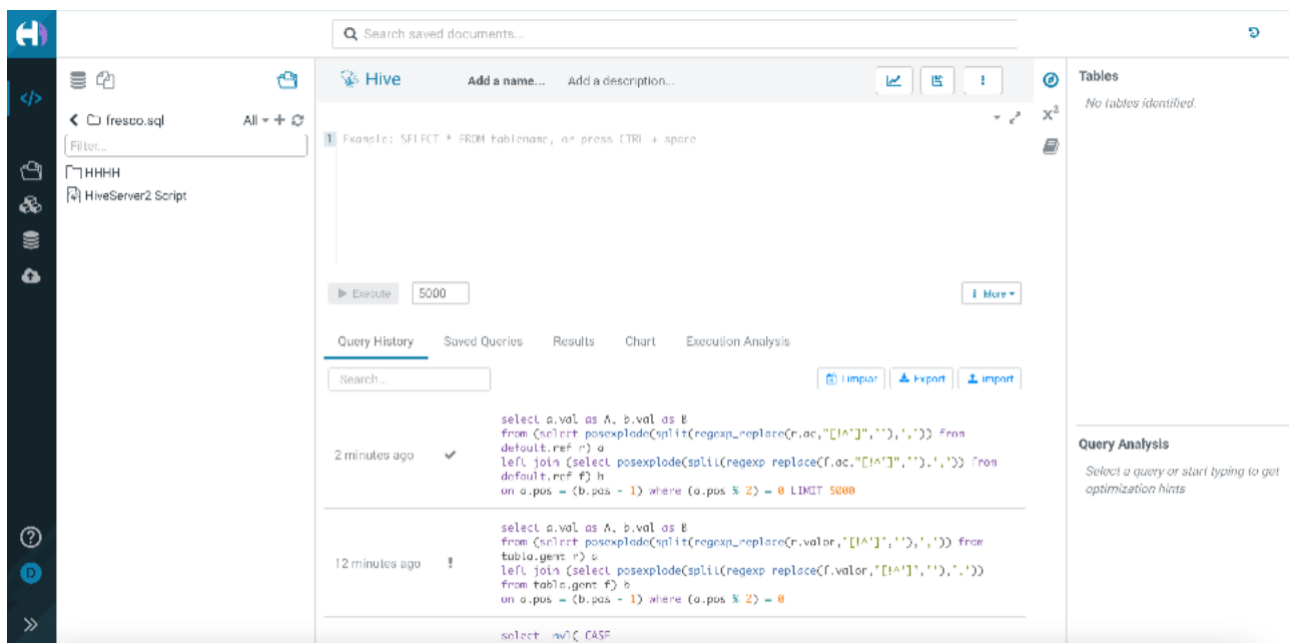
Íñigo Sanz (Dominio público)

Visualización de resultados de las consultas en modo gráfico:



Íñigo Sanz (Dominio público)

Visualización del histórico de consultas en Hive:



Íñigo Sanz (Dominio público)

Listado de bases de datos y tablas:





---



  Hive



**Databases**



(6)  


Filter databases...

-  cdm
-  default
-  hello1
-  primo
-  sports
-  vin\_emp








 default


Tables (8) + 


Filter...


 emp  
id (int)  
name (string)  
deptid (int)


 emp\_global


 employee\_txt

 medals

 ref

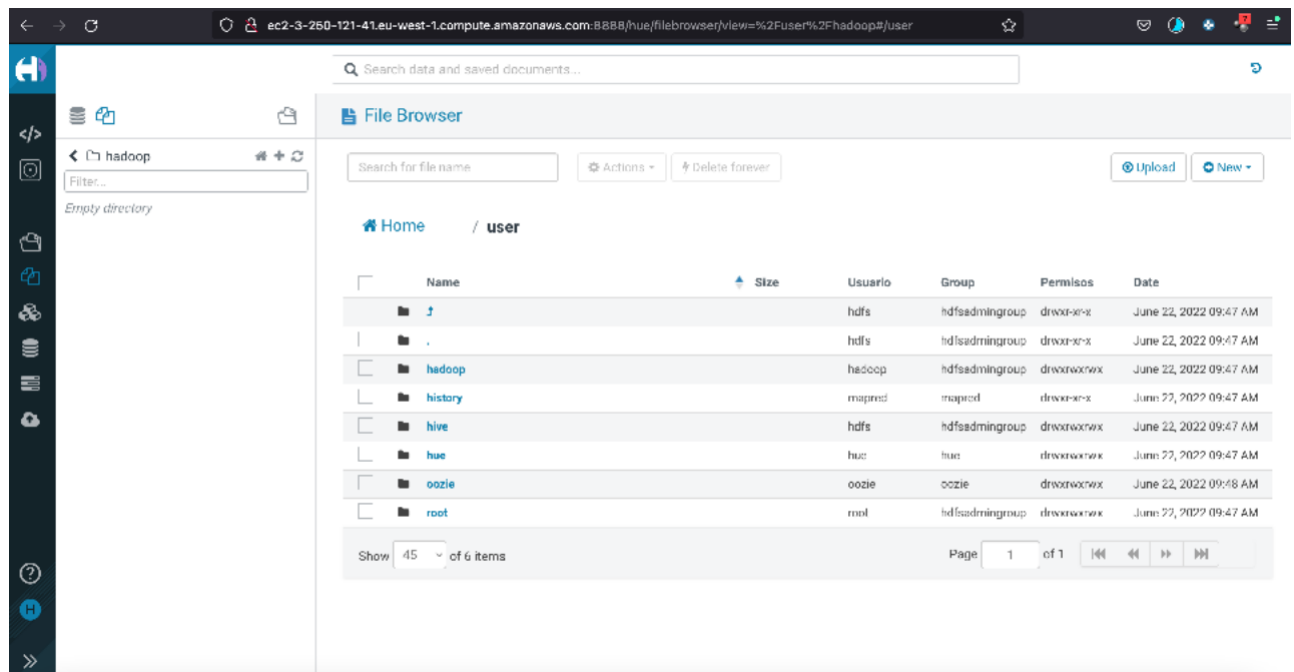
 values\_\_tmp\_\_table\_\_1

 values\_\_tmp\_\_table\_\_2

 values\_\_tmp\_\_table\_\_3

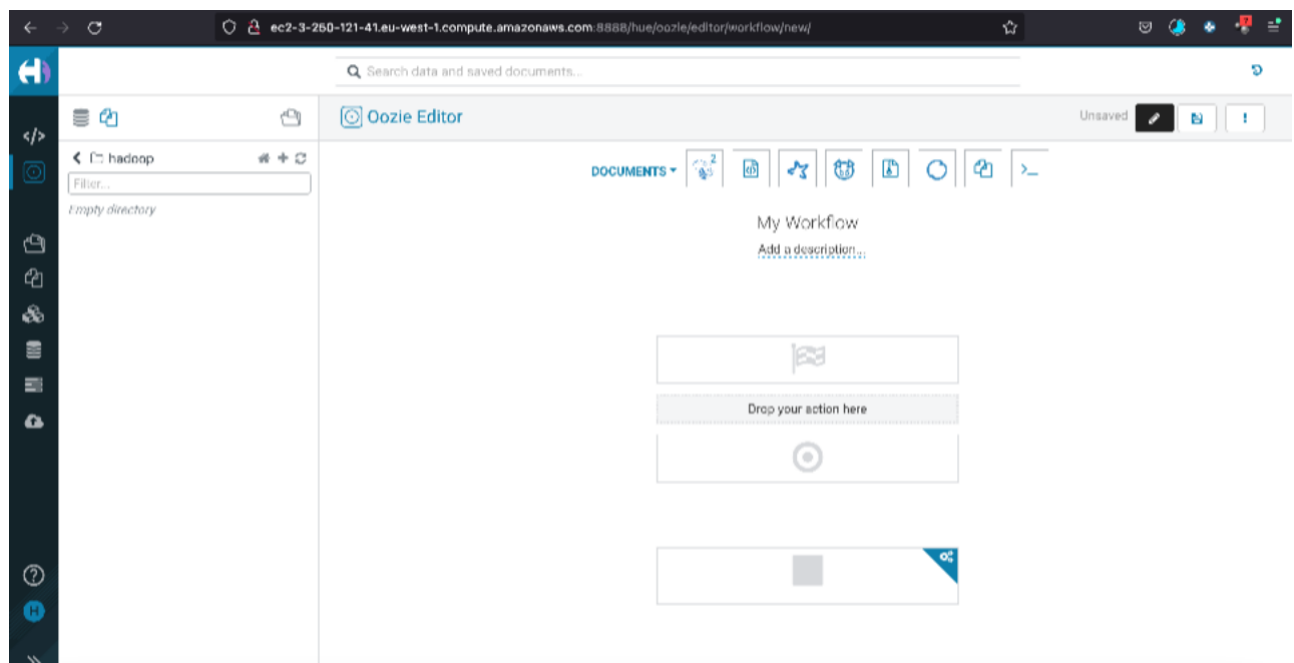
Íñigo Sanz (Dominio público)

Explorador de ficheros:



Íñigo Sanz (Dominio público)

Editor de workflows de Oozie:



Íñigo Sanz (Dominio público)

## 4.2.- Apache Zeppelin.

Hue es un interfaz sencillo para poder realizar consultas, navegar por los datos, etc. Es un interfaz útil, pero no ofrece toda la potencia que un Data Scientist requiere para su trabajo.

Apache Zeppelin permite cubrir ese gap, ya que es una **herramienta web para notebooks**, es decir, una herramienta web que permite interactuar con los datos mediante la creación de historias con código (Spark) o consultas (Hive), generando gráficos, tablas, elementos interactivos, que además pueden ser publicados, compartidos o desarrollados en modo colaborativo.



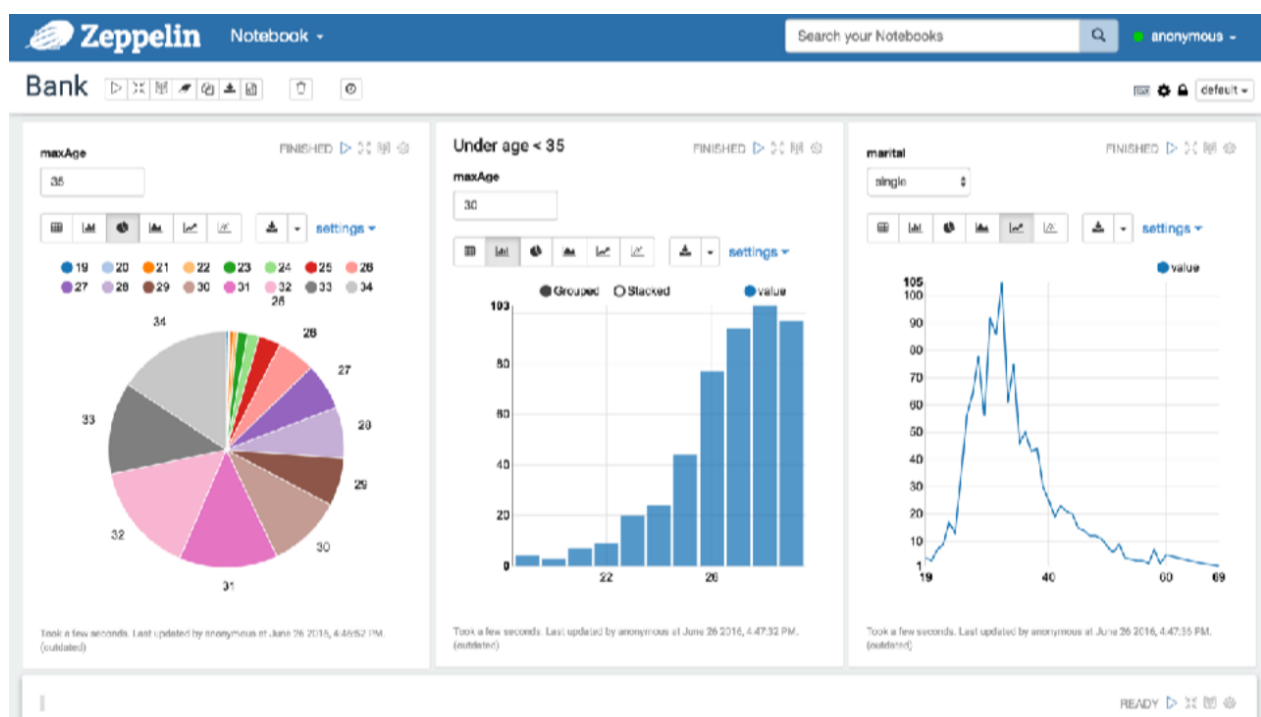
[Apache Software Foundation](#) (Apache License)

Zeppelin suele ser la primera herramienta de uso por parte de los Data Scientist, aunque los proyectos requieren fases posteriores para industrializar estos desarrollos.

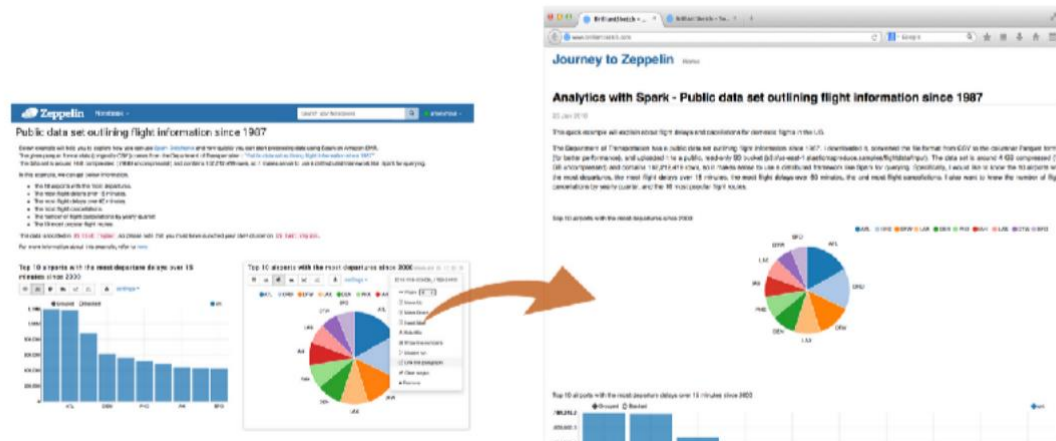
Permite incorporar intérpretes para distintos lenguajes (R, Scala, etc.), incorporando “de caja” los siguientes:

- ✓ Lenguaje Python.
- ✓ Apache Spark.
- ✓ Apache Flink.
- ✓ SQL.
- ✓ Hive / Impala.
- ✓ R.
- ✓ Shell.

Zeppelin permite visualizar los resultados de la ejecución de diferentes sentencias con distintos modos de gráfico, e incluso permite hacer que los gráficos sean interactivos, pudiendo añadir filtros o campos sobre los que pivotar los resultados.



Además, Zeppelin permite guardar los trabajos que se están realizando, compartirlos con otros usuarios, permitiendo incluso la colaboración de varios desarrolladores en un mismo trabajo. Y no sólo eso, también permite que un trabajo se pueda publicar como una página web independiente:



Apache (Dominio público)

## 4.3.- Apache Ambari y Cloudera Manager.

Hadoop, como plataforma distribuida, tiene como una de sus principales dificultades la gestión o monitorización de la cantidad de servicios que se ejecutan en un cluster formado por distintos servidores físicos.

Ambari es una herramienta que permite:

- ✓ Instalar un cluster Hadoop mediante un wizard, seleccionando la topología del cluster, los servicios a instalar, su configuración, etc
- ✓ Gestionar los servicios de un cluster: arrancar, parar o reconfigurar.
- ✓ Gestionar la seguridad: políticas de autorización, autenticación, etc.
- ✓ Monitorizar el cluster: recogida y visualización de métricas, definición y gestión de alarmas, etc.

La gestión de un cluster Hadoop sin una herramienta como Ambari es inmanejable.

Cloudera desarrolló una herramienta de gestión y administración de Hadoop que fue incorporada a sus distribuciones.

En cuanto a Cloudera Manager, es muy similar a Ambari en funcionalidades y manejo. De hecho, Cloudera Manager fue desarrollada 3 años antes que Ambari, aunque su principal limitación es que no está liberada como código libre, siendo una herramienta propietaria de Cloudera, a diferencia de Ambari.

## 5.- Procesamiento en streaming

---

El concepto Big Data suele hacer referencia principalmente a datos de gran volumen que están en reposo, y que son procesados utilizando diferentes técnicas de forma masiva, normalmente en una ventana temporal como puede ser todas las noches o cada cierto periodo de tiempo. Hadoop suele asociarse normalmente a este tipo de casos de uso, donde se hace uso de HDFS para almacenar los datos, y tecnologías como Hive o Spark para procesarlos o analizarlos.

Sin embargo, desde hace unos años, el concepto Fast Data ha adquirido una mayor importancia, al descubrirse que aporta más valor incluso a los negocios. El concepto Fast Data suele hacer referencia a datos en movimiento, es decir, datos que se generan a gran rapidez, de forma continua y uno a uno, es decir, no se generan por bloques de datos sino lo que se denomina "gota a gota".

Fast Data, también llamados datos en streaming, es una tipología de datos que, normamente, si se procesa en tiempo real, es decir, nada más se genera el dato, puede ofrecer a las empresas beneficios enormes.

Dentro del ecosistema Hadoop hay diferentes herramientas que permiten analizar los datos en tiempo real. Las tres herramientas principales son:

- ✓ Apache Storm.
- ✓ Apache Flink.
- ✓ Apache Spark

Entre ellas hay diferencias, y normalmente hay que decidir cuál utilizar para abordar los casos de uso de streaming.

Antes de verlas, es necesario conocer algunos conceptos:

### Garantía de procesamiento: at least once vs exactly

Cualquier tecnología de streaming, especialmente cuando se trata de tecnologías distribuidas, tiene un modo de funcionamiento por el que garantiza que todos los datos que llegan se procesan. Ahora bien, hay dos tipos de garantía:

- ✓ **Exactly once (exactamente una vez):** indica que todos los datos se procesan una vez, aunque se produzca una caída en el sistema. Este nivel de garantía es el más difícil de conseguir en los sistemas distribuidos, pero es importante saber qué exigencia tiene nuestro caso de uso para determinar si el nivel de garantía que ofrece la herramienta elegida será suficiente. Veamos dos ejemplos para entenderlo:
  - ✦ Para el caso de uso en el que se pretende procesar los pagos con tarjeta según llegan a nuestro sistema, necesitaríamos que se procesen todos, pero sólo una vez. En caso de procesarse más de una vez, podríamos estar cobrando dos veces a un cliente por una única transacción.
  - ✦ Para el caso de uso en el que se reciben medidas de los sensores de las máquinas de una planta de fabricación y se va generando un cuadro de mando con la situación actual, sacando las medidas de las medidas de los sensores (por ejemplo, la temperatura), que un dato se procese dos veces, si bien no es un escenario deseado, no supondría un problema de gran entidad.
- ✓ **At-least-once (al menos una vez):** en este nivel de garantía, la herramienta



garantiza que todos los datos se procesan, pero podría ocurrir, en casos poco probables pero posibles, que un dato se procesara dos veces, por ejemplo, si hay una caída en uno de los servidores que procesan los datos en tiempo real.

## Procesamiento evento a evento (one-at-a-time) o en microbatch

Algunas herramientas procesan cada evento por separado según llega al sistema, mientras que otras, lo que hacen es agrupar los eventos que han llegado en una ventana temporal pequeña (por ejemplo, todos los eventos del último segundo), y los procesan todos juntos a la vez. Esta última técnica se conoce como micro-batch.

Normalmente el procesamiento evento a evento es tiene menos latencia, es decir, el tiempo desde que se recibe un evento hasta que se termina de procesar es más bajo. Sin embargo, este tipo de procesamiento suele tener peor ancho de banda (throughput), es decir, permite procesar menos eventos por unidad de tiempo.

El procesamiento en microbatch tiene mayor latencia, ya que si por ejemplo la duración de la ventana es 1 segundo, habrá eventos que tengan que esperar un segundo completo antes de empezar a procesarse. En cambio, suelen ofrecer un ancho de banda superior.

Al igual que la garantía de procesamiento, dependiendo del caso de uso, se requerirá un procesamiento evento a evento o podrá admitir escenarios micro-batch. Por ejemplo, en sistemas donde hay que dar una respuesta con una latencia mínima, no servirá un escenario de micro-batch. Un ejemplo de este tipo de casos de uso podría ser un sistema para detectar fraude en pagos por tarjeta de crédito, para los que es necesario denegar la operación en menos de 100 o 200 milisegundos. Un ejemplo para el que un escenario de micro-batch puede ser válido sería el caso en el que se pretende generar un cuadro de mando con las métricas que se están recibiendo de los sensores instalados en las máquinas de una fábrica.

A continuación vamos a conocer los tres principales componentes del ecosistema Hadoop que permiten hacer procesamiento en tiempo real sobre datos en streaming:

## Apache Storm

Apache Storm fue el primer framework de computación distribuida sobre Hadoop.

Tiene las siguientes características:

- ✓ Realiza un procesamiento **evento a evento** (one-at-a-time) y en el momento de su ocurrencia. Por este motivo, ofrece tiempos de latencia de milisegundos.



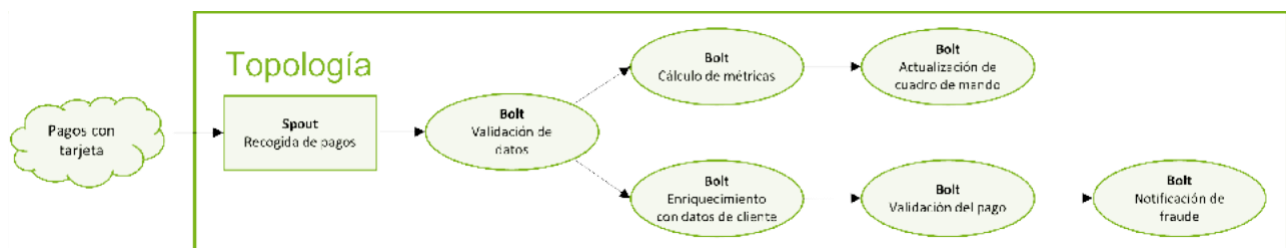
[Apache Software Foundation](https://www.apache.org/licenses/)  
(Apache License)

- ✓ **Garantía “at least once”**: los eventos se procesan al menos una vez, pero no hay garantía de que se procesen sólo una vez, salvo que se utilice un framework que tiene incorporado, denominado **Trident**, que a cambio de ofrecer mayor latencia y peor ancho de banda, sí ofrece garantía de procesamiento exactly-once.
- ✓ **Escalable y elástico**: se pueden añadir nodos durante la ejecución sin pérdida de servicio.
- ✓ Permite utilizar distintos **lenguajes** de programación, aunque normalmente se desarrolla en **Java**
- ✓ Puede ser ejecutado sobre **YARN**, y es ofrecido “de caja” por alguna distribución de Hadoop.

Para desarrollar un flujo de procesamiento de eventos, o topología, en primer lugar es necesario separar el procesamiento en diferentes bloques, denominados Bolts. Cada Bolt se programa por separado.

Asimismo, es preciso definir las piezas que recogerán los eventos desde el origen y los enviarán a la topología para su procesamiento. Los Spouts es la otra pieza que es necesario programar siendo el framework de Storm.

Ejemplo de una topología:



Íñigo Sanz (Dominio público)

El ejemplo podría tratarse de un sistema para validar los pagos con tarjeta que recibe un banco:

- ✓ Un Spout es el encargado de leer los pagos del sistema de origen y los emite a la topología.
- ✓ En primer lugar, un Bolt valida los datos, es decir, descarta los eventos que estén en blanco, o con datos corruptos.
- ✓ Una vez los datos se han validado, se lanzan dos flujos en paralelo:
  - ◆ Por un lado, se pretende actualizar un cuadro de mando con datos en tiempo real, para lo que en primer lugar se calculan las métricas con cada nuevo evento, y posteriormente se actualiza el cuadro de mando.
  - ◆ Por otro lado, se pretende validar los pagos, notificando los casos en los que se ha detectado una transacción fraudulenta.

## Apache Spark

Structured Streaming es el componente de Spark que permite procesar en tiempo near-realtime flujos de datos.

Realiza procesamiento de los datos en **microbatches**, es decir, toma datos durante un periodo de tiempo (batch interval) y lanza un procesamiento en paralelo sobre los datos

recogidos. Por este motivo, la latencia mínima para el procesamiento de un evento,

desde que se genera hasta que termina de procesarse, será, como mínimo, el tiempo de la ventana definida como batch interval. Esta característica le permite, sin embargo, tener un **ancho de banda** muy alto.

La principal ventaja de Spark en el procesamiento en streaming es que permite **reutilizar el código** de procesamiento batch para el procesamiento de eventos ya que, simplificando, lo que realiza Structured Streaming es un proceso similar, pero con el conjunto de eventos/datos que ha llegado durante la última ventana. Para entender la importancia de la reutilización del código, imagina el ejemplo de la validación de pagos con tarjeta. Implementando el proceso en Spark Structured Streaming permitiría, con el mismo código, hacer la validación de eventos en tiempo real, y por ejemplo, todas las noches validar los pagos del día, a lo mejor, tomando una información adicional para el cálculo, como podría ser los saldos medios, o cualquier otra métrica que es difícil utilizar en el procesamiento en tiempo real.

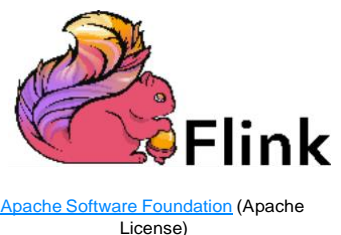
En cuanto a la garantía de procesamiento, Spark ofrece un nivel **exactly-once**, es decir, garantiza que cada evento se procesa una única vez, por lo que es válido para los casos de uso en los que se requiere no duplicar el procesamiento de eventos.

## Apache Flink

Apache Flink es uno de los framework de procesamiento distribuido orientado a streaming más nuevos, es decir, ha sido uno de los últimos en aparecer.

Tiene las siguientes características:

Ofrece distintos modos de procesamiento: batch, streaming, procesos iterativos, etc.



- ✓ El modo de procesamiento es **one-at-a-time**, es decir, los eventos se procesan uno a uno, según se generan, ofreciendo una latencia muy baja.
- ✓ Ofrece un **gran ancho de banda**, al nivel de Apache Spark.
- ✓ Válido para procesamiento de eventos complejos (CEP), que requieren mantenimiento del estado y persistencia (no ofrece repositorio, pero permite conectarlo a Kafka, HDFS, Cassandra, etc.).
- ✓ Tolerante a fallos, es decir, en caso de que uno de los servidores donde se procesan los eventos se caiga, el sistema sigue procesando los eventos.
- ✓ Garantía **exactly-once**
- ✓ Soporta lenguajes Java, Scala, Python y SQL.