Ecosistema Hadoop

Índice

Ecosistema Hadoop

Los principales componentes o proyectos asociados al ecosistema Hadoop son los siguientes:

- **Componentes de acceso y procesamiento**: Apache Tez, Apache Pig, Apache Hive, Apache Impala, Apache HBase, Apache Phoenix, Apache Spark.
- **Componentes de ingesta y flujos de trabajo**: Apache Sqoop, Apache Flume, Apache Oozie.
- **Intefaces y herramientas de trabajo**: Hue, Apache Zeppelin, Apache Ambari, Cloudera Manager.
- **Procesamiento en streaming**: Apache Kafka, Apache Spark, Apache Flink (structured streaming), Apache Storm.

Apache Tez

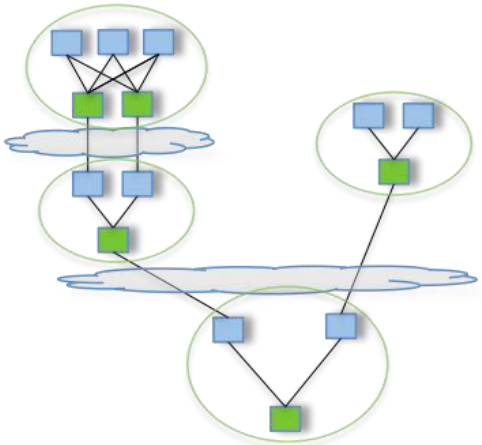
Es un **framework de ejecución** sobre YARN que ofrece un API para que las aplicaciones puedan ejecutarse sin limitarse al paradigma MapReduce, que sólo ofrece operaciones Map y Reduce, con persistencia en disco para cada fase.

Tez fue desarrollado por Hortonworks en colaboración con Microsoft, con el principal objetivo de poder acelerar el procesamiento de consultas SQL sobre Hadoop con Hive.

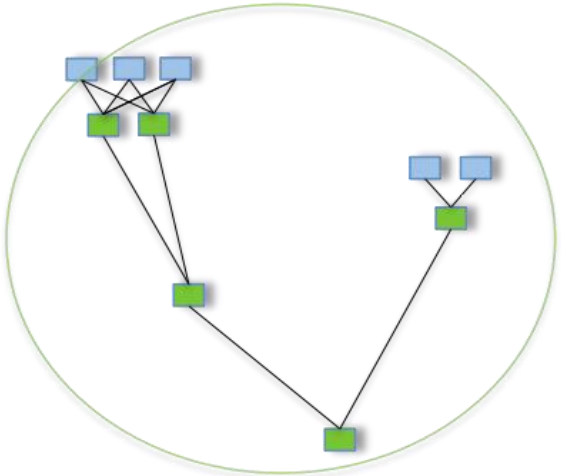
Tez utiliza un modelo de ejecución de grafo acíclico dirigido frente al modelo de fases de MapReduce, y ofrece muchas optimizaciones para poder acelerar un órden de magnitud la ejecución de muchos procesos.

Tiene como inconveniente que ofrece un nivel de fiabilidad menor.

MapReduce



Pig/Hive - MR



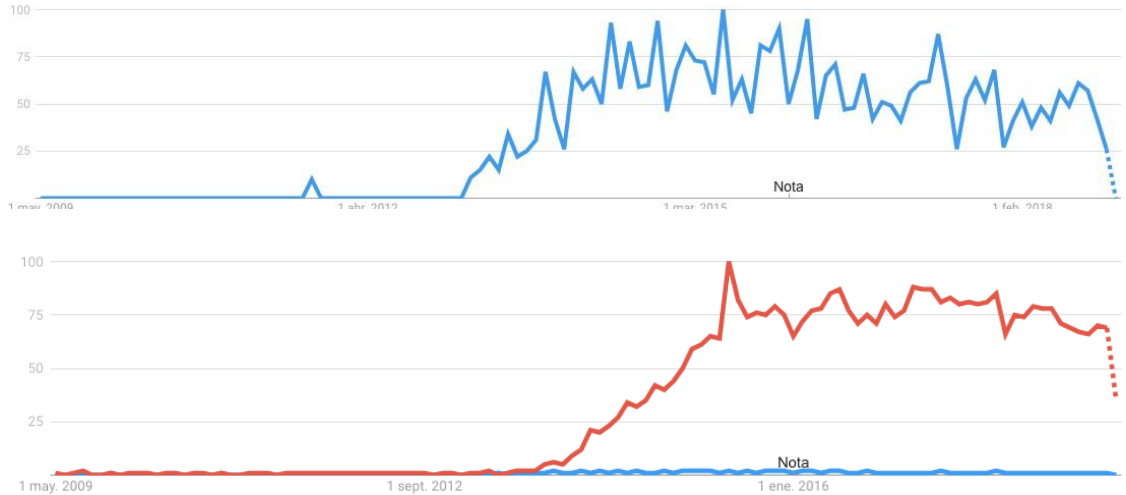
Pig/Hive - Tez

https://tez.apache.org/

MapReduce

TEZ

Pese al esfuerzo por parte de Microsoft y Hortonworks para su adopción, la realidad es que la aparición de Apache Spark ha lastrado su uso.



Apache Pig

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

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set( itr.nextToken());
        context.write(word, one);
      }
    }
  }

}
```

```
public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    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 las palabras de un fichero!

Apache Pig

Debido a la gran cantidad de código que es necesario generar, así como el nivel de especialización que requiere, la productividad de MapReduce es reducida.

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.

Apache Pig

Pig es un **motor de ejecución** sobre MapReduce (ahora también sobre Tez o Spark) que ofrece un nivel de abstracción mayor.

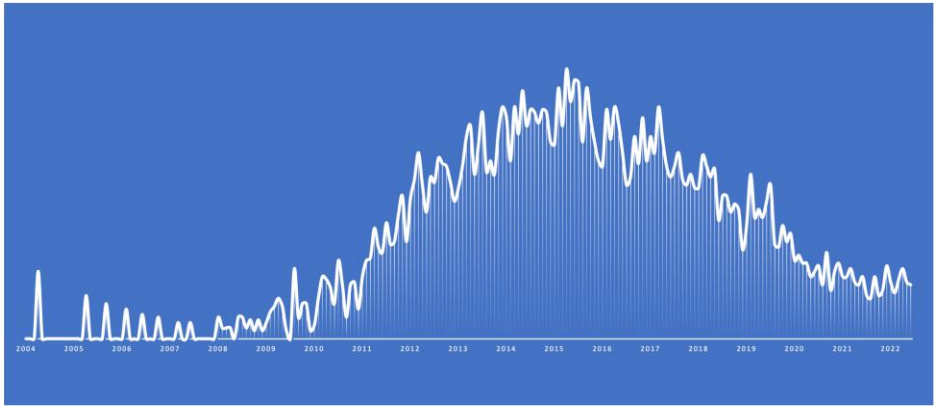
Utiliza como lenguaje Pig Latin, que tiene similitudes con SQL.

```
input_lines = LOAD '/tmp/file-to-count-words' AS (line:chararray);
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
filtered_words = FILTER words BY word MATCHES '\\w+';
word_groups = GROUP filtered_words BY word;
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count,
group AS word;
ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tmp/number-of-words-on-internet';
```

Apache Pig

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

Desde la aparición de Apache Spark, está cayendo en desuso.



Apache Hive

Hadoop presenta **limitaciones** a la hora de explotar datos en HDFS:

- Necesita **programación** de procesos **MapReduce** para manipular datos (requiere conocimientos de programación).
- Falta de **integración** con herramientas de gestión o explotación de datos.
- **Encarecimiento** de soluciones software por no poder reutilizar código MapReduce. Dificultad de industrialización.

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 incluído como proyecto Apache, lanzando su primera versión estable en 2010.

Apache Hive

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

Características y funcionalidades:

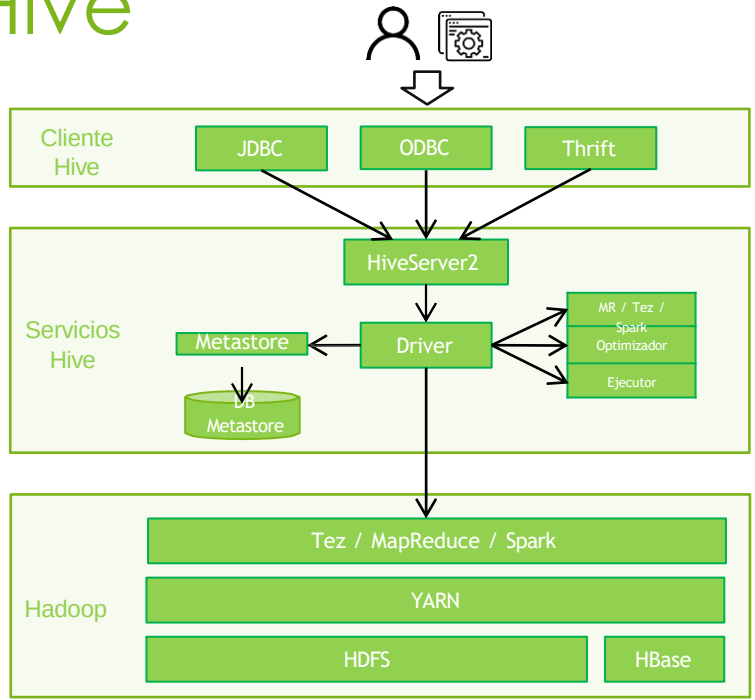
- Capacidad de definir una **estructura relacional** (tablas, campos, etc.) sobre la información “en bruto” de HDFS.
- **Lenguaje** de consultas HQL, de sintaxis muy similar a SQL incluyendo muchas de las evoluciones de SQL para analítica: SQL-2003, SQL-2011 y SQL-2016.
- **Interfaces estándar** con herramientas de terceros mediante JDBC/ODBC.

Apache Hive

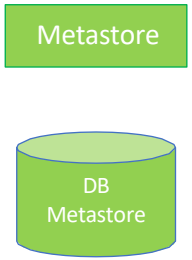
Más características:

- 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.
- Mecanismos de **seguridad** en el acceso y modificación de la información.
- Lee ficheros con diferentes **formatos**: texto plano, RF, ORC, HBase, Parquet y otros.
- **Extensible** mediante código a medida (UDF / MapReduce).
- Orientado a **consultas** aunque también ofrece operativa para creación, modificación o borrado de registros.

Apache Hive


Cliente Hive
JDBC
ODBC
Thrift
HiveServer2
Servicios Hive
Metastore
Driver
MR / Tez / Spark
Optimizador
Ejecutor
DB Metastore
Hadoop
Tez / MapReduce / Spark
YARN
HDFS
HBase

Apache Hive. HCatalog

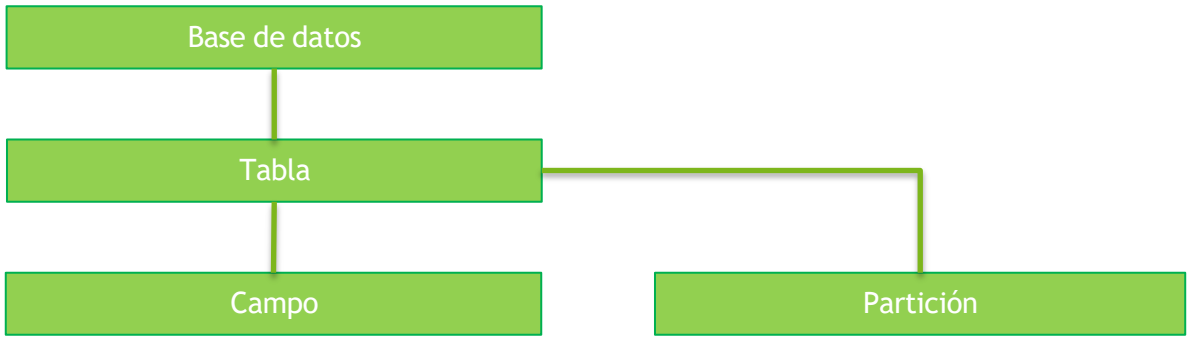


- Registro de todos los **metadatos** de Hive y otros componentes que requieren aplicar un modelo sobre datos existentes en HDFS o HBase.
- Por ejemplo, almacena la información de las **tablas** definidas, qué **campos** tienen, con qué **ficheros** se corresponden, etc.
- Permite que **aplicaciones distintas a Hive** puedan usar la definición de tablas definida en Hive, o al revés.

Apache Hive. HQL

Dos tipos de operaciones: DDL y DML.

- **DDL** (Data Definition Language): permite crear estructuras de datos.
- **DML** (Data Manipulation Language): permite alta, baja, modificación y consulta de datos.



Apache Hive. HQL

- **Tipos de datos:**
  - 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, MAP, STRUCT, UNIONTYPE.

Apache Hive. HQL

DDL

- **Bases de datos**:
  - **Creación**: CREATE DATABASE my_db
  - **Visualización**: SHOW DATABASES
  - **Uso**: USE my_db
  - **Borrado**: DROP DATABASE my_db
  - **Modificación**: ALTER DATABASE my_db SET …
  - Visualización de **detalle**: DESCRIBE DATABASE my_db

Apache Hive. HQL

DDL

- **Tablas**:
  - **Creación**: CREATE TABLE.

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

  - **Visualización**: SHOW TABLES
  - **Borrado**: DROP TABLE inmuebles

Apache Hive. HQL

- **Modificación**: ALTER TABLE my_table…

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

ALTER TABLE inmuebles RENAME TO casas
```

- Visualización de **detalle**:

```
DESCRIBE [EXTENDED|FORMATED] inmuebles
```

Apache Hive. HQL

DML

- **Consultas**: SELECT.

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

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

Apache Hive. HQL

DML

- **Consultas**: SELECT.

```
[WITH CommonTableExpression (, CommonTableExpression)*]
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]
```

```
<common_table_expression>::=
    expression_name [ ( column_name [ ,...n ] ) ]
    AS
    ( CTE_query_definition )
```

Apache Hive. HQL

DML

- **Consultas**: SELECT. Más operaciones que se permiten:
  - Subconsultas (SELECT FROM SELECT).
  - JOIN de tablas.
  - UNION de tablas.
  - Operadores (UDFs).

Apache Hive. HQL

DML

- **Cargar datos**: LOAD DATA .

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

- **Inserción múltiple**: INSERT OVERWRITE.

```
INSERT OVERWRITE TABLE inmuebles SELECT a.* FROM herencias a;

INSERT OVERWRITE LOCAL DIRECTORY '/data/local_out'
SELECT a.* FROM inmuebles a
```

Apache Hive. HQL

DML

- **Inserción simple**: INSERT.

```
INSERT INTO TABLE inmuebles
VALUES (23, ‘Calle Sol 23, 'Madrid', 'Madrid’, 234)
```

- **Actualización**: UPDATE.

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

Apache Hive. HQL

DML

- **Borrado**: DELETE.

```
DELETE FROM inmuebles WHERE provincia = 'Madrid'
```

Apache Impala

Las primeras versiones de Hive ofrecían un rendimiento pobre para consultas on-line, que requerían latencias de segundos.

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 SF** para su inclusión como proyecto, siendo elevada a proyecto top-level en noviembre de 2017.

Apache Impala

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.

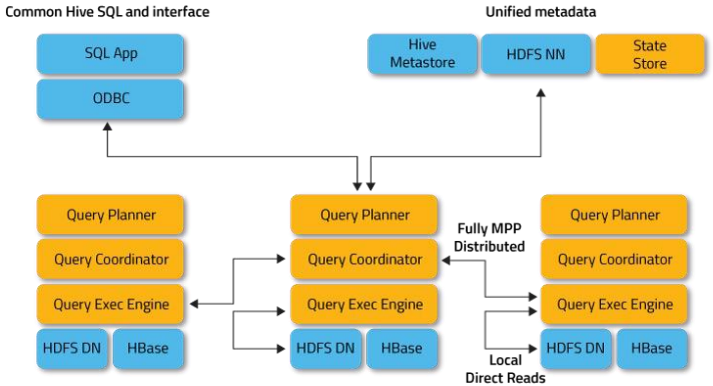
- Ofrece el mismo **interfaz** ODBC y lenguaje de consultas de Hive: **HQL**.

- Está orientado a **consultas** típicas de **BI / BA**, es decir, consultas complejas que requieren agrupaciones, subconsultas, funciones analíticas, etc.

- Ofrece el mismo nivel de **securización** de Hive, es decir, autenticación mediante Hadoop / Kerberos, y autorización mediante Sentry.

Apache Impala

- Soporta **almacenamiento en HBase y HDFS**, permitiendo para éste formatos txt, LZO, SequenceFile, Avro, RCFile y Parquet.
- Para la gestión de metadatos utiliza el mismo almacén/servicio que Hive: **HCatalog** / **Metastore**.
- Arquitectura:



Apache HBase

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

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.
- Operaciones de **actualización y borrado** de datos.

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

Apache HBase

Características:

- Modelo **no relacional** con almacenamiento **columnar**.

	Familia columnas: Datos personales		Familia columnas: 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

- Las filas tienen un **row-key** que las identifica y organiza, la definición del row-key es un aspecto clave del diseño.

Apache HBase

Características:

Sus principales operaciones son las siguientes:

Listar las tablas

- list

Crear una tabla

- Crear una tabla de nombre ‘testTable’ y familia de columnas ‘cf’

  create ‘testTable’,’cf’

Insertar datos en una tabla

- Insertar en rowA, columna ‘cf:columnName’ con valor ‘val1’

  put ‘testTable’, ’rowA’, ’cf:columnName’,’val1’

Obtener datos de una tabla

- Obtener rowA de la tabla ‘testTable

  get ‘testTable’,’rowA’

Leer el contenido de una tabla (iterar sobre la tabla)

scan ‘testTable’

Borrar una tabla

disable ‘testTable’

drop ‘testTable’

Apache HBase

Características:

- Toda la información de HBase es almacenada en forma de **array de bytes**.
- No ofrece un **lenguaje de acceso** como SQL, sino que ofrece un API, mediante Thrift y Avro o mediante un servicio HTTP RestFul.
- 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.

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.

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.

Apache 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

Apache Spark

Es una **plataforma opensource** de **computación** y un conjunto de **librerías** para **procesamiento** en **paralelo** de **datos** sobre **sistemas distribuidos**.

Apache Spark

- Es el proyecto de tecnologías Big Data con mayor número de **contribuidores**.
- Es el **estándar de facto** para procesamiento de datos en Big Data.
- 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**, aunque permite utilizar HDFS, S3 u otros sistemas de almacenamiento.
- Garantiza la **escalabilidad** en operaciones paralelizables.
- Ofrece **tolerancia a fallos**.

Apache Spark. Historia.

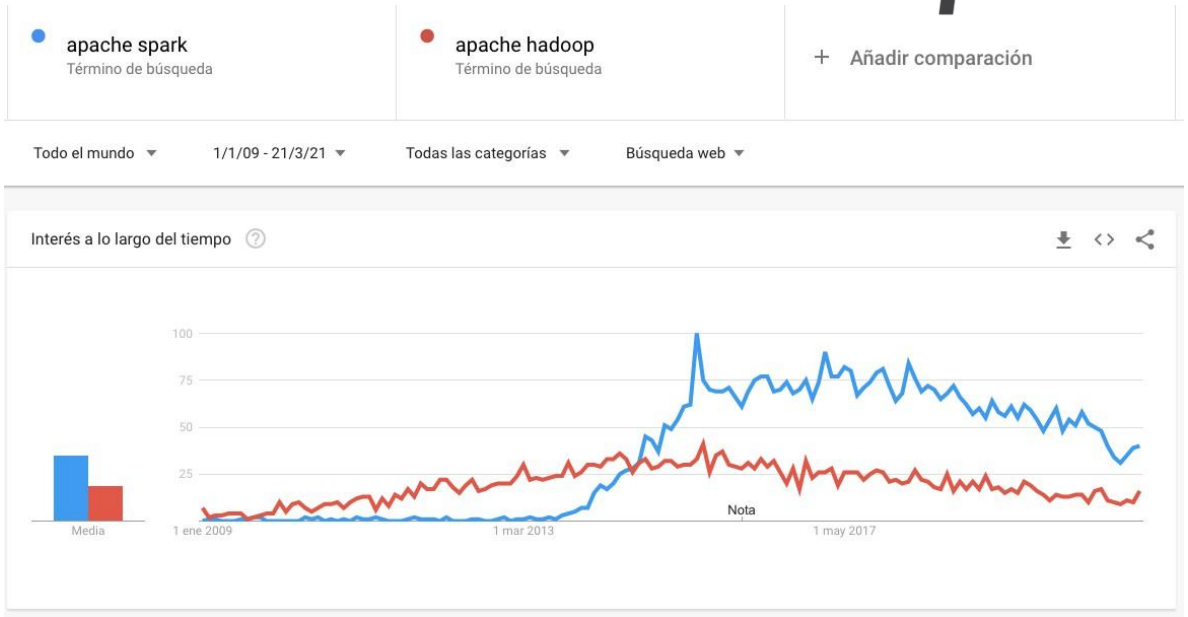
2009
AMPLab @ UC Berkeley
Primeros desarrollos dentro de un proyecto del Proyecto Spark Research por Matei Zaharia.

2013
Apache Software Foundation
Donado a la Apache Software Foundation

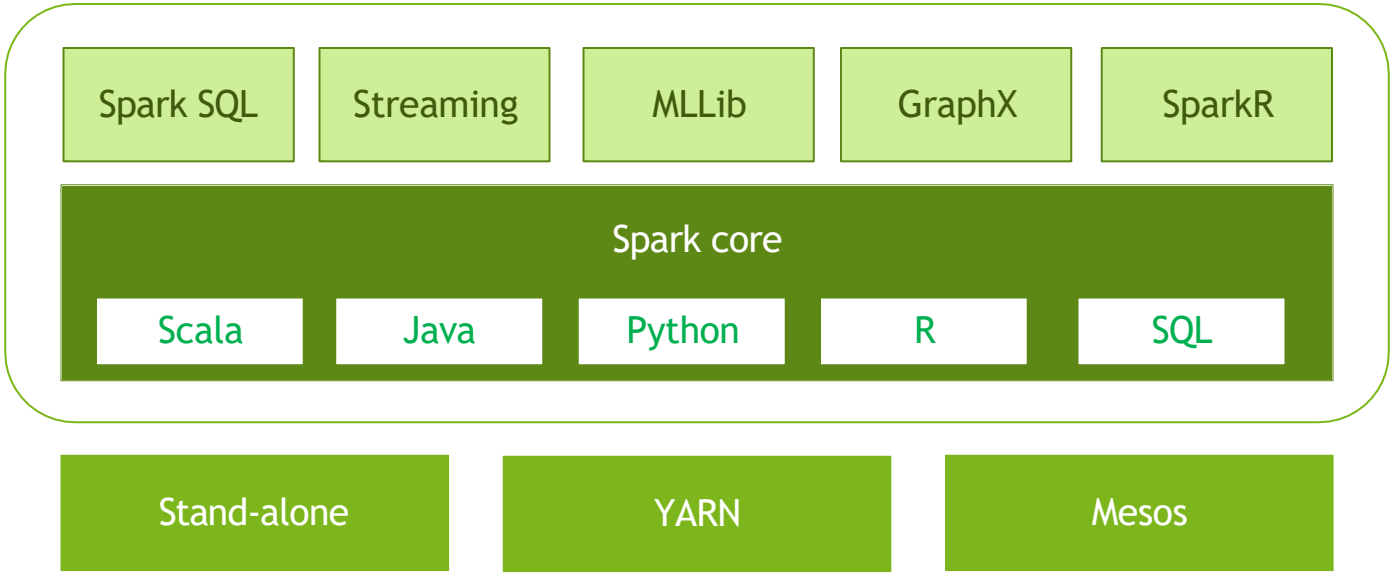
2014
Databricks
Matei Zaharia y otras 6 personas fundan Databricks, una compañía que ofrece soporte y un entorno cloud para Spark.

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

Apache Spark. Evolución.

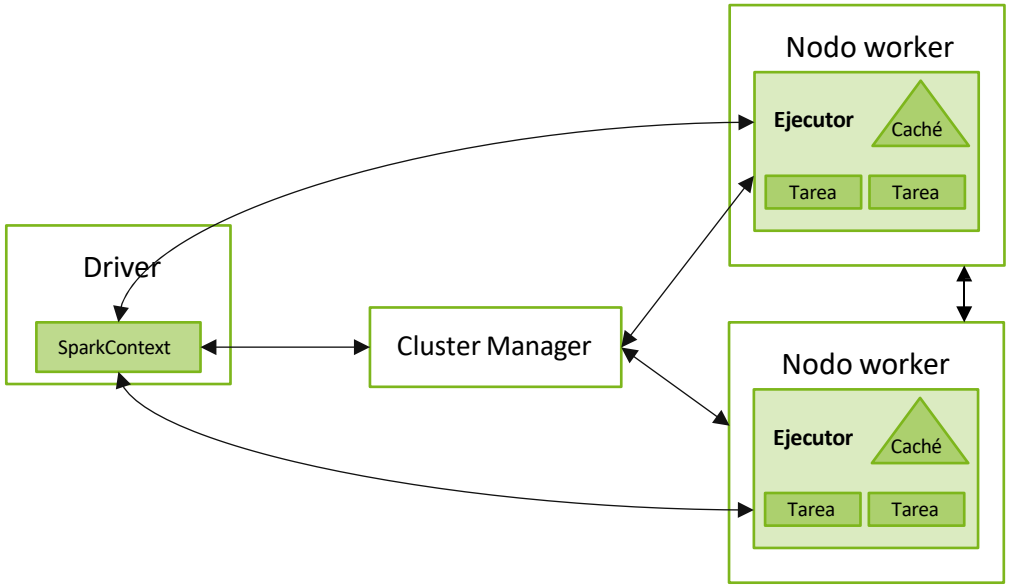


Apache Spark. Componentes.



Apache Spark. Arquitectura.

Una aplicación Spark se compone de un proceso Driver y de un conjunto de ejecutores.



Apache Spark. Arquitectura.

- 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.

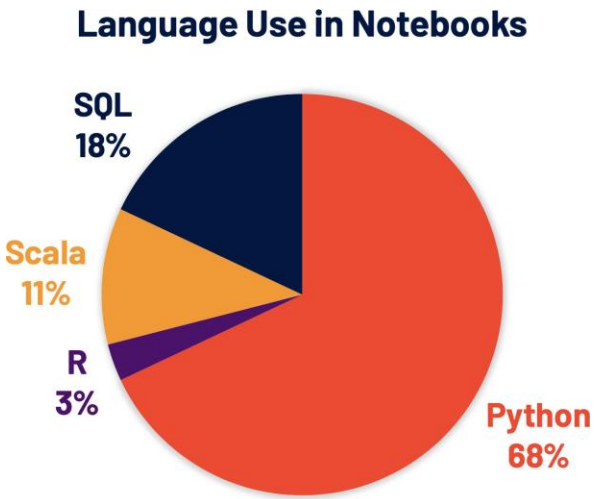
- Los **ejecutores** se ocupan de la ejecución de las tareas / trabajos que el driver les asigna y de informar al driver del estado de estos trabajos.

- 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.

Apache Spark. Lenguajes.

Spark ofrece APIs en los siguientes lenguajes:

- **Scala**: Spark está escrito principalmente en Scala, siendo éste el lenguaje “por defecto”.
- Java.
- Python.
- SQL.
- R.



https://databricks.com/blog/2020/06/18/introducing-apache-spark-3-0-now-available-in-databricks-runtime-7-0.html

Apache Spark. Ejecución.

Mediante el ejecutable spark-submit se pueden lanzar programas Spark:

```
spark-submit --help
  Usage: spark-submit [options] <app jar | python file> [app arguments]
  [...]
  Options:
    --master MASTER_URL         spark://host:port, mesos://host:port, yarn, or local.
    --deploy-mode DEPLOY_MODE   Whether to launch the driver program locally ("client") or on one of
                                the worker machines inside the cluster ("cluster") (Default: client).
    --class CLASS_NAME          Your application's main class (for Java / Scala apps).
    --name NAME                 A name of your application.
    --jars JARS                 Comma-separated list of local jars to include on the driver and executor
                                classpaths.
    --packages                  Comma-separated list of maven coordinates of jars to include on the driver
                                and executor classpaths. [...] groupId:artifactId:version.
    --exclude-packages,  --repositories      [...]
    --py-files PY_FILES         Comma-separated list of .zip, .egg, or .py files to place on the PYTHONPATH
                                for Python apps.
    --files FILES               Comma-separated list of files to be placed in the working directory of each
                                executor.
```

Apache Spark. Ejecución.


```
  --conf PROP=VALUE           Arbitrary Spark configuration property.
  --properties-file FILE      Path to a file from which to load extra properties. [ ... ]

  --driver-memory MEM         Memory for driver (e.g. 1000M, 2G) (Default: 1024M).
  --driver-java-options       Extra Java options to pass to the driver.
  --driver-library-path       Extra library path entries to pass to the driver.
  --driver-class-path         Extra class path entries to pass to the driver.
  --executor-memory MEM       Memory per executor (e.g. 1000M, 2G) (Default: 1G).

  [ ... ]

Spark standalone with cluster deploy mode only:
  --driver-cores NUM          Cores for driver (Default: 1).
 Spark standalone or Mesos with cluster deploy mode only:
  --supervise                 If given, restarts the driver on failure.
  --kill SUBMISSION_ID        If given, kills the driver specified.
  --status SUBMISSION_ID      If given, requests the status of the driver specified.
```

Apache Spark. Ejecución.


```
Spark standalone and Mesos only:
 --total-executor-cores NUM  Total cores for all executors.

Spark standalone and YARN only:
 --executor-cores NUM        Number of cores per executor. (Default: 1 in YARN mode,
                             or all available cores on the worker in standalone mode)

YARN-only:
 --driver-cores NUM          Number of cores used by the driver, only in cluster mode
                             (Default: 1).
 --queue QUEUE_NAME          The YARN queue to submit to (Default: "default").
 --num-executors NUM         Number of executors to launch (Default: 2).
 --archives ARCHIVES         Comma separated list of archives to be extracted into the
                             working directory of each executor.
[ ... ]
```

Apache Spark. Ejecución.

Ejemplos:

Ejecución local del programa SparkPi con 8 cores asignados:

```
spark-submit
    --class org.apache.spark.examples.SparkPi
    --master local[8]
    /path/to/examples.jar
    100
```

Apache Spark. Ejecución.

Ejemplos:

Ejecución sobre YARN del programa SparkPi:

```
export HADOOP_CONF_DIR=XXX
spark-submit
  --class org.apache.spark.examples.SparkPi
  --master yarn
  --deploy-mode cluster
  --executor-memory 20G
  --num-executors 50
  /path/to/examples.jar
  1000
```

Apache Spark. Beneficios.

- 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**.
- +1000 **desarrolladores** contribuyendo en el proyecto.

Apache Spark. Dificultades.

- Está orientado a **perfiles muy técnicos**, con conocimientos sólidos de programación.
- Es **difícil de optimizar** en producción.
- 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.

Apache 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 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.

Apache Spark. Core.

Existen dos tipos de operaciones sobre 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.

- **Acciones:** persisten o re-distribuyen un RDD. 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.

Apache Spark. Core.

Principales transformaciones:

- **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.

Apache Spark. Core.

Principales acciones:

- **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.)

Apache Spark. SparkSQL.

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

Ofrece:

- Un nivel de abstracción de datos estructurados o semi-estructurados superior a 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.

Apache Spark. SparkSQL.

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).

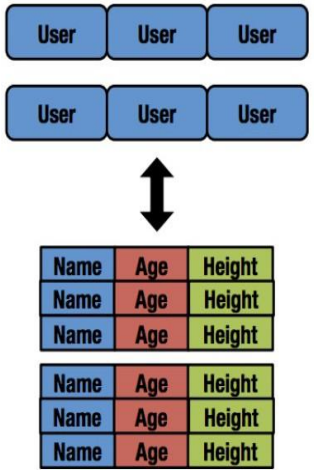
Al igual que RDD, los DataFrames y DataSets son inmutables.

Apache Spark. SparkSQL.

Con RDD los objetos son manejados por los programas Spark de forma opaca, sin una estructura concreta (que sólo conoce el desarrollador).

Con DataFrames, se define un esquema y se permite el uso de lenguaje estándar para el acceso o manejo de los datos.

DataFrame = RDD + esquema.



Apache Spark. SparkSQL.

SparkSQL permite trabajar con las siguientes fuentes de datos:

Apache Spark. SparkSQL. Ejemplo

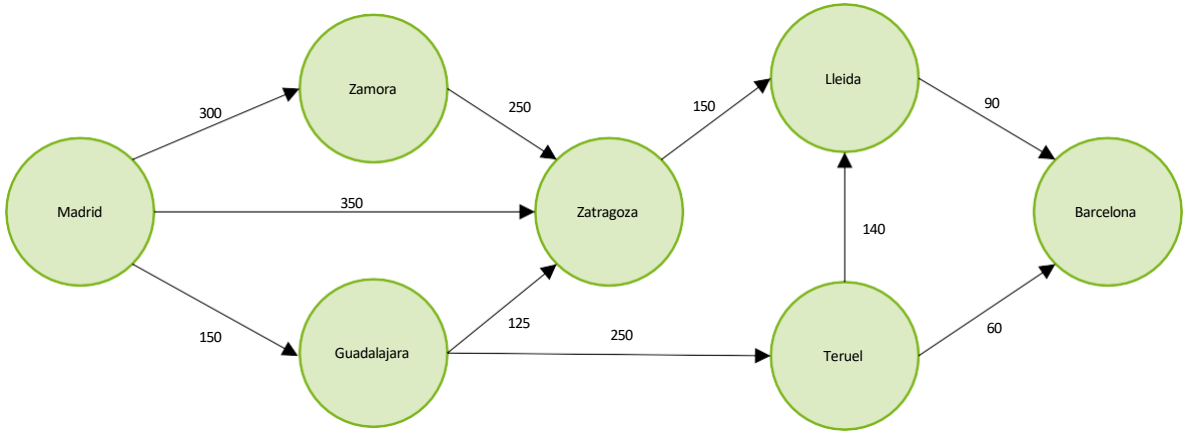
https://github.com/databricks/Spark-The-Definitive-Guide/blob/master/data/flight-data/csv/2015-summary.csv

```
DEST_COUNTRY_NAME,ORIGIN_COUNTRY_NAME,count
United States,Romania,15
United States,Croatia,1
United States,Ireland,344
Egypt,United States,15
United States,India,62
United States,Singapore,1
United States,Grenada,62
Costa Rica,United States,588
…
```

Apache Spark. GraphX.

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.



Apache Spark. GraphX.

El modelo de abstracción 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 subpragh, joinVertices, mapReduceTriplets, etc.

Apache Spark. MLLib

Es una librería de alto nivel de Spark que ofrece funcionalidades 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.

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.

Apache Sqoop

Algunos comandos:

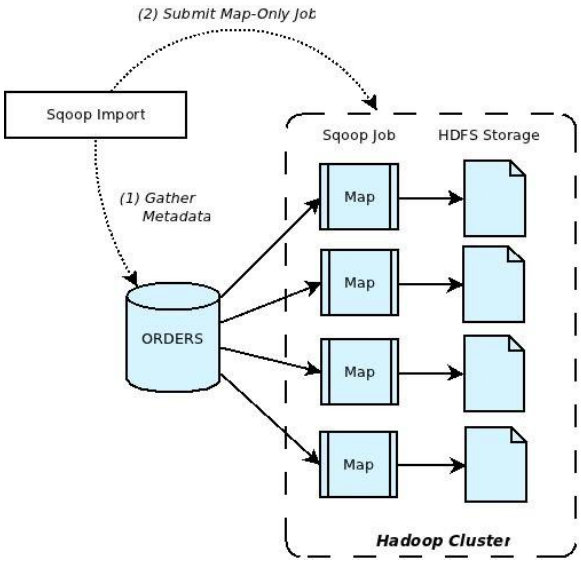
- **sqoop-import**: importa una tabla de una base de datos relacional en HDFS.

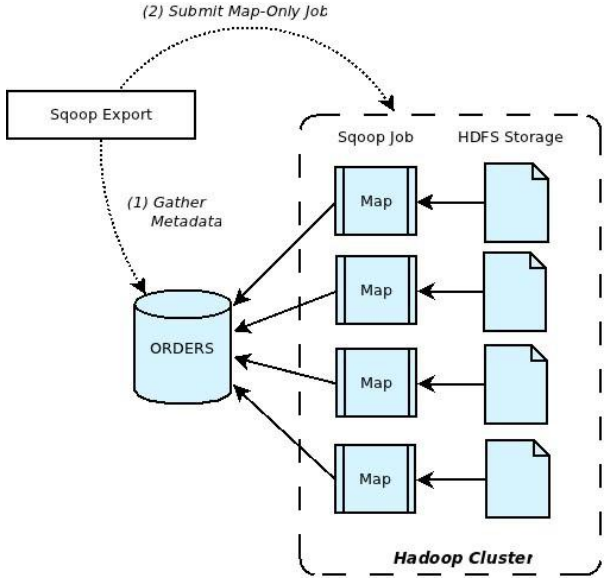
```
sqoop import --connect jdbc:mysql://db.foo.com/corp
             --table EMPLOYEES --hive-import
```

- **sqoop-export:** exporta un conjunto de ficheros de HDFS a una base de datos relacional.

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

Apache Sqoop





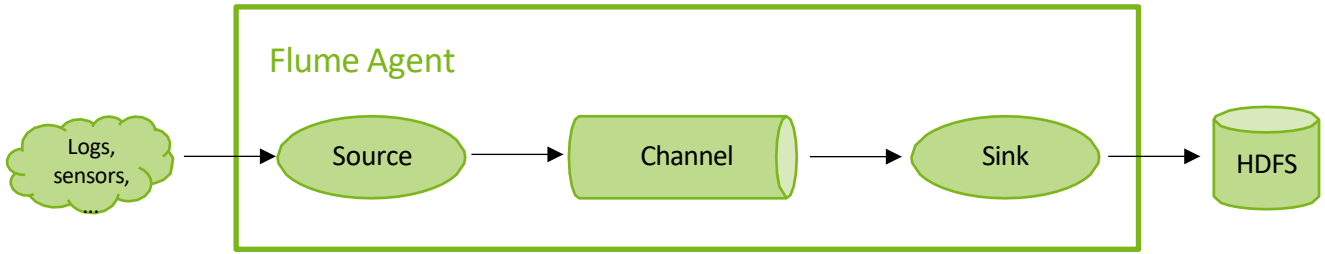
https://www.researchgate.net/figure/Apache-SQOOP-data-import-architecture_fig2_331908752

Apache Flume

Flume es un servicio distribuido y confiable para **recoger, agregar y mover datos** generados de forma contínua 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.

Su arquitectura es la siguiente:



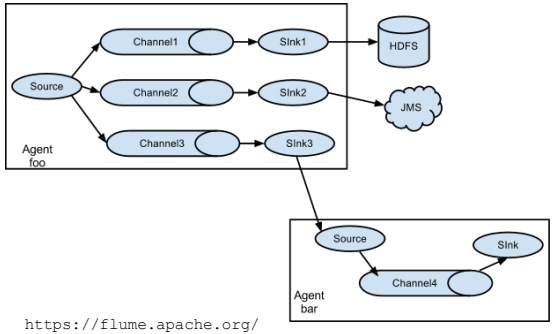
Apache Flume

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**: 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.



https://flume.apache.org/

Apache Flume

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.

Apache Oozie

Facilita el lanzamiento de flujos de trabajo (workflows) cuando se cumplen determinadas condiciones. 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.

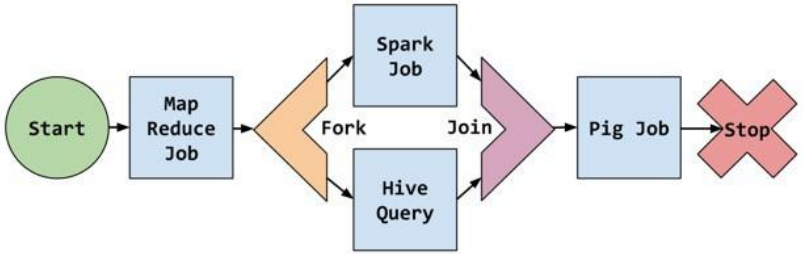
* Los pasos 3 y 4 serían controlados por Oozie.

Apache Oozie

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

- Condiciones de **inicio**: temporal o existencia de datos.
- Mecanismos de **control**: bifurcaciones, decisiones, uniones.
- **Acciones**: MapReduce, Hive, Sqoop, DistCp, Spark, Pig, SSH, email.
- Estados de **fin y error**.

El diseño de los workflows se hace mediante XML, y la gestión de flujos es realizada mediante una aplicación web que arranca Oozie.



https://oozie.apache.org/

Apache Airflow

Apache Airflow es otra herramienta popular para la orquestación de flujos de trabajo:

1. Permite programar, monitorear y gestionar flujos de trabajo definidos en código Python.
2. Diseñado para procesamiento por lotes, pero también puede manejar tareas en tiempo real.
3. Los flujos de trabajo en Airflow se definen como código Python, permitiendo usar estructuras de control de flujo de Python para definir dependencias y condiciones.
4. Interfaz web robusta para monitorear y gestionar los flujos de trabajo, lo que facilita la visualización del estado y los logs de las tareas.
5. Extensible, con muchos operadores y sensores predefinidos para integrarse con diversas tecnologías y servicios en la nube.
6. Casos de Uso: orquestación de flujos de trabajo complejos.

Apache Airflow

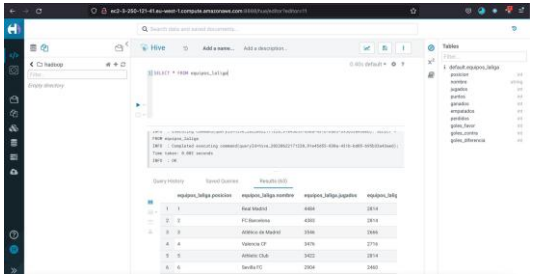

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.operators.hive_operator import HiveOperator
from airflow.operators.spark_submit_operator import SparkSubmitOperator
from datetime import datetime, timedelta

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 12, 16),
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(
    'example_dag',
    default_args=default_args,
    description='An example DAG with Hive and Spark',
    schedule_interval=timedelta(days=1),
)
```

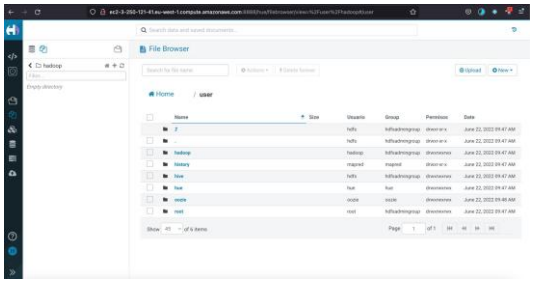
```
hive_task = HiveOperator(
    task_id='run_hive_query',
    hql='SELECT * FROM my_table',
    hive_cli_conn_id='my_hive_conn',
    dag=dag,
)

spark_task = SparkSubmitOperator(
    task_id='run_spark_job',
    application='/path/to/my_spark_job.py',
    conn_id='my_spark_conn',
    dag=dag,
)

hive_task >> spark_task
```

Hue

Hue 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.





Hue

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).

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.

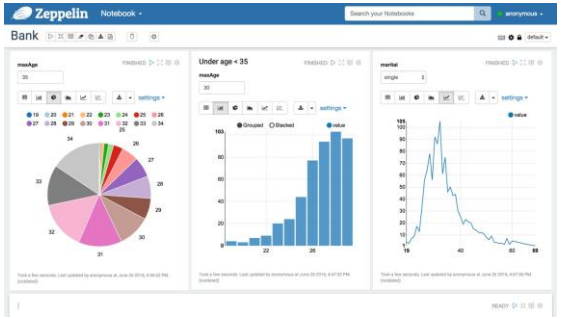
Apache Zeppelin permite cubrir ese gap, ya que es una **herramienta web** para **notebooks**, es decir, una herramienta web que permite **interactuar con los datos** mediante la creación de historias mediante **código** (Spark) o **consultas** (Hive), generando **gráficos**, **tablas**, elementos **interactivos**,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

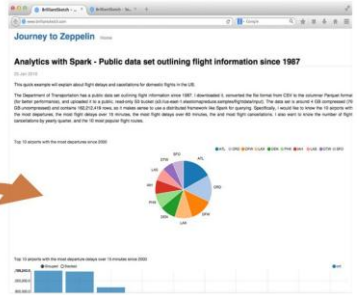
Zeppelin suele ser la primera herramienta de uso por parte de los Data Scientist, aunque los proyectos requieren fases posteriores para **industrializar** estos desarrollos.

Apache Zeppelin

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.





Apache Ambari

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, etcc.
- **Monitorizar** el cluster: recogida y visualización de métricas, definición y getión de alarmas, etc.

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

Cloudera Manager

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

Cloudera Manager es **muy similar a Ambari** en cuanto a 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.

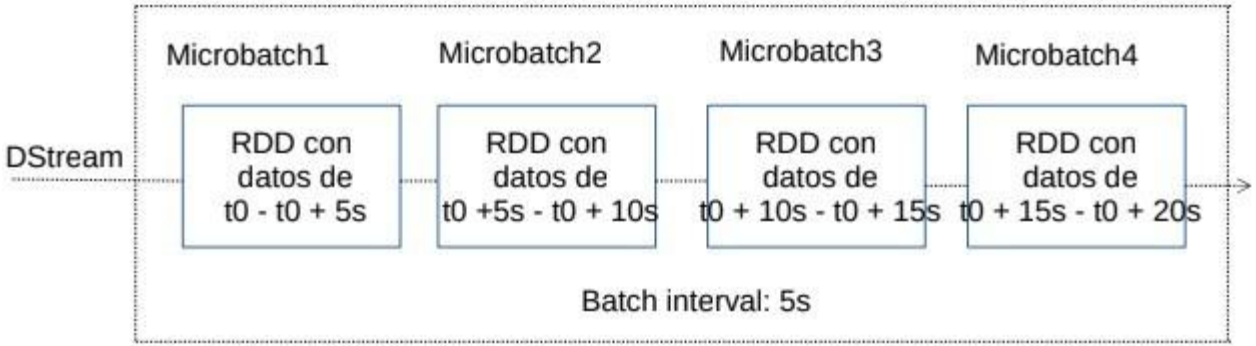
Apache Spark. Streaming.

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.

Se basa en **DStream**, una abstracción construida sobre RDDs que representa un flujo de datos continuo obtenido a partir del stream de entrada o de aplicar transformaciones sobre otros Dstreams.

Structured Streaming. Modelo de ejecución



Apache Storm

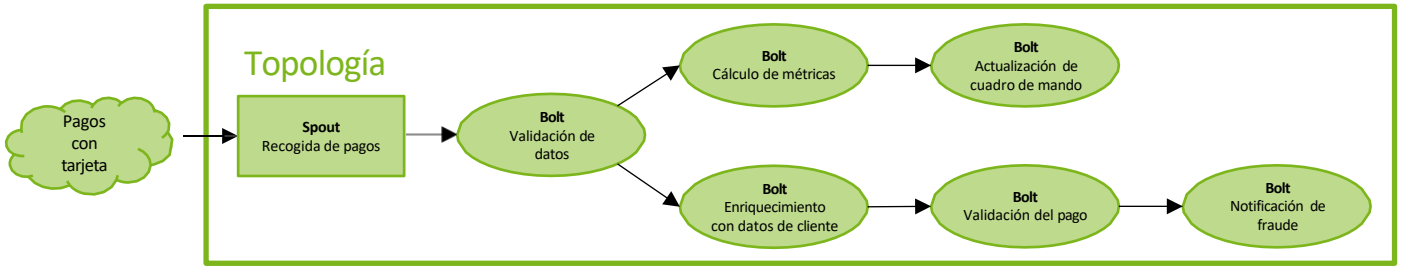
Framework de computación distribuida:

- Ofrece tiempos de latencia de ms, al realizar el procesamiento de cada evento de forma individual y en el momento de su ocurrencia (**one-at-a-time**).
- Garantía "**at least once**". Con requerimientos "exactly once" requiere utilizar Trident API.
- **Escalable y elástico**: se pueden añadir nodos durante la ejecución sin pérdida de servicio salvo en Zookeeper.
- Permite utilizar distintos **lenguajes** de programación, no sólo Java.
- Puede ser ejecutado sobre **YARN**, y es ofrecido "de caja" por alguna distribución de Hadoop.

Apache Storm

Se basa en la definición de Spouts y Bolts para definir topologías de procesamiento de streams de tuplas.

Requiere programar cada Bolt y Spout.



Apache Flink

Framework de procesamiento distribuido orientado a streaming:

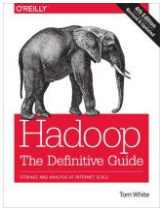
- Ofrece distintos modos de procesamiento: batch, streaming, procesos iterativos, etc.
- El modo de procesamiento es **event-at-a-time**.
- Ofrece un **gran throughput** y **baja latencia**.
- Válido para procesamiento de eventos complejos (**CEP**), que requieren mantenimiento del estado y persistencia (no ofrece repositorio, pero permite conectarlo a Kafka, HDFS, Cassandra, etc.).
- **Tolerante a fallos** y con garantía **exactly-once**.
- Soporta **lenguajes** Java, Scala, Python y SQL.

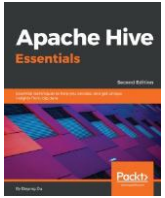
Bibliografía

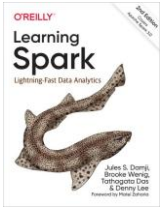
Hadoop The Definitive Guide, 4th Edition.
Tom White. Ed. O'Reilly
https://learning.oreilly.com/library/view/hadoop-the-definitive/9781491901687/

Apache Hive Essentials. Dayong Du. Packt Publishing
https://www.packtpub.com/product/apache-hive-essentials-second-edition/9781788995092

Learning Spark: Lightning-fast Data Analytics. Jules S. Damji. Ed. O'Reilly
https://learning.oreilly.com/library/view/learning-spark-2nd/9781492050032/







Apache Kafka

Apache Kafka es un sistema de mensajería distribuido de **alto rendimiento** que se utiliza a menudo como **fuente de datos en tiempo real** para frameworks de procesamiento como Spark Streaming.

Características clave:

- **Publicación y suscripción a flujos de datos:** Similar a una cola de mensajes, Kafka permite a las aplicaciones publicar y suscribirse a flujos de registros.
- **Almacenamiento duradero y tolerante a fallos:** Los registros se almacenan de forma persistente en un clúster de brokers.
- **Escalabilidad horizontal:** se puede escalar fácilmente agregando más brokers al clúster.
- **Alto rendimiento:** diseñado para manejar miles de mensajes por segundo.
- **Modelo de consumidor basado en grupos:** para compartir la carga de trabajo de procesar los mensajes de un tema, garantizando que cada mensaje se procesa una sola vez por el grupo.
- **Registro de confirmación distribuido: registro de transacciones** y eventos, lo que permite la reproducción y la reconstrucción del estado del sistema.

Apache Kafka

Los componentes clave de la arquitectura de Kafka incluyen:

- **Brokers:** Servidores individuales que forman un clúster de Kafka.
- **Productores:** Aplicaciones que publican mensajes en temas de Kafka.
- **Consumidores:** Aplicaciones que se suscriben a temas de Kafka y procesan los mensajes.
- **Temas:** Categorías en las que se organizan los mensajes.
- **Particiones:** Subdivisiones de los temas que permiten la distribución y el paralelismo.

En resumen, Apache Kafka es una plataforma de mensajería robusta y escalable que ofrece un rendimiento excepcional, almacenamiento duradero y un modelo de consumidor flexible. Estas características lo convierten en una opción popular para una variedad de casos de uso de big data, incluida la ingesta de datos en tiempo real, el procesamiento de flujos y la creación de canalizaciones de datos.