

**CIENCIA DE DATOS:**  
**APRENDE LOS FUNDAMENTOS**  
**DE MANERA PRÁCTICA**



**SESION 04**  
**Apache Spark**

**Juan Chipoco**

[mindquasar@gmail.com](mailto:mindquasar@gmail.com)

# ÍNDICE

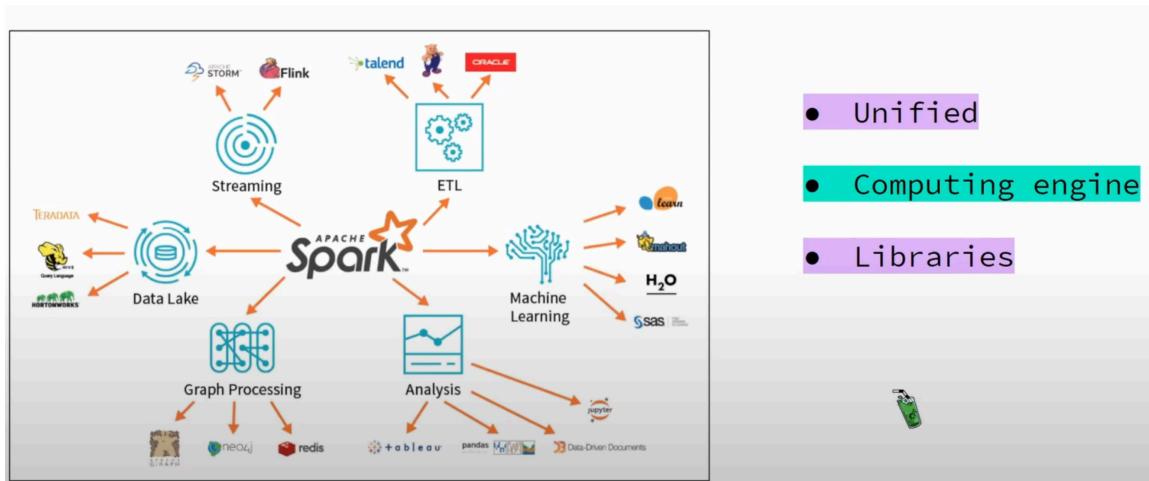
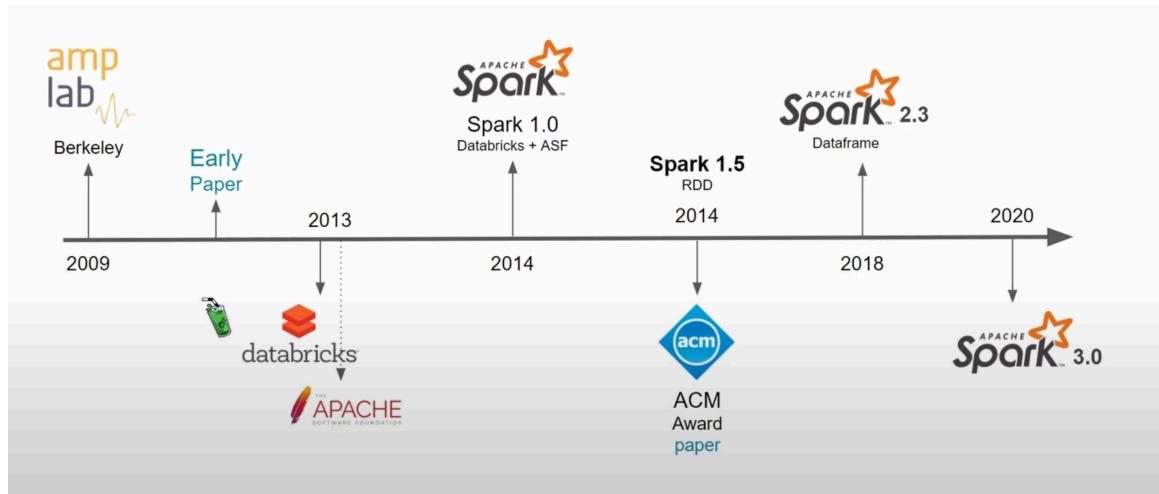
OBJETIVO .....	4
HADOOP & APACHE SPARK .....	6
COMPONENTES DE APACHE SPARK .....	10
FEATURES.....	17
ARQUITECTURA.....	19

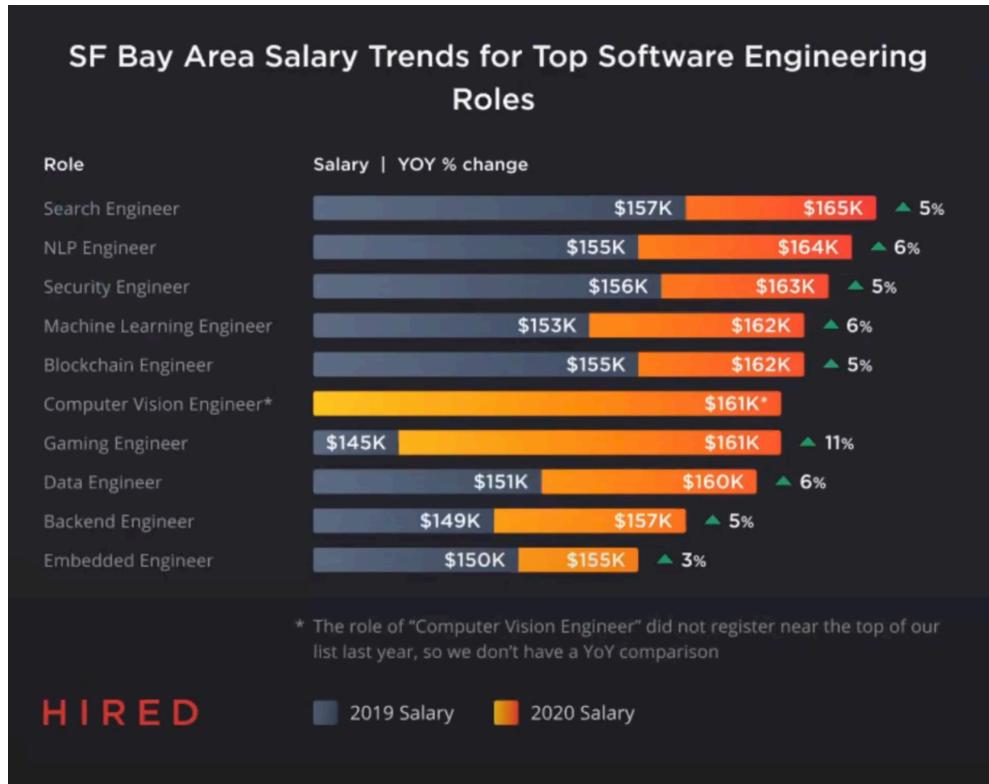
## OBJETIVO



En esta sesión, aprenderás Spark desde cero, comenzando con su historia antes de crear una aplicación de análisis de Wikipedia como uno de los medios para aprender una amplia gama de su API principal.

Ese conocimiento central facilitará la búsqueda de otras bibliotecas de Spark, como las API de transmisión y SQL. Finalmente, aprenderás cómo evitar algunos aspectos ásperos de Spark que se encuentran comúnmente.





## Hadoop & Apache Spark

- Hadoop y Spark: frameworks para el **procesamiento Big Data** con **arquitectura en clúster** (múltiples nodos)
- Ambos son **escalables** y **tolerantes a fallos**
- El uso de Spark presenta **ventajas** en
  - Forma de procesar datos – Spark es **más rápido**
  - Estilo de programación – **APIs más sencillas** de usar
  - Disponibilidad de **componentes específicos** (Mlib, GraphX, Spark Streaming, Spark SQL)
- Se pueden combinar: **HDFS** como sistema de **almacenamiento** de ficheros y **Spark** para el **procesamiento**

- Surgimiento de Spark: **problemas velocidad** paradigma **MapReduce** (Apache Hadoop)
- Casos con **muchas operaciones de escritura y lectura** de datos en disco<sup>1</sup>

<sup>1</sup> Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. **Spark: cluster computing with working sets**. In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (HotCloud'10). USENIX Association, Berkeley, CA, USA, 10-10.

- Principales casos de uso con limitaciones de MapReduce de Hadoop:

➤ **Procesos iterativos**, requieren almacenamiento tras cada iteración  
➤ **Procesos intensivos en consultas**, con varios pasos **intermedios** hasta resultados finales. Si resultados intermedios en memoria, operaciones sucesivas sobre ellos más rápidamente

- Spark usa memoria resultados intermedios (no disco)
- Las claves de la eficiencia de Spark
  - Tipo de datos utiliza su núcleo (**Spark core**): conjunto de datos distribuidos resistentes o **RDD** (*Resilient Distributed Datasets*)
  - Planificador de los grafos, **DAG Scheduler**
- RDD **divide datos en particiones en nodos** del clúster, permitiendo **operaciones en paralelo** (posteriormente agrega resultados)
- **Grafo de etapas** de trabajo a ejecutar en cada clúster a partir de tareas submitidas a spark

- Grafos **denominados DAG** (*Directed Acyclic Graph*, Grafos Acíclicos Dirigidos)
- Componente de Spark DAG Scheduler genera plan de ejecución con **secuencia operaciones que más rápidamente pueden ejecutarse** sobre los RDDs

- **Velocidad** operación transformación de RDD: **afecta a 1 partición o varias**

- **2 tipos** de transformaciones
  - *Narrow transformation* (“transformación estrecha”): **no es necesario mezclar datos** de diferentes particiones. (ej: filtrado valores cumplen condición; transformando otra escala)
  - *Wide transformation* (“transformación ancha”): **es necesario mezclar datos** de diferentes particiones. (ej: *groupBy* en un *join*)

- Op. más costosa: mezclar datos particiones → **planificador minimiza cantidad de datos a mezclar**

- Se sigue la secuencia

➤ **Primero transformaciones narrow** sobre cada RDD

➤ **Después transformaciones wide** sobre conjunto de RDDs

- **Ejemplo:** resultado de filtrar 2 RDDs

➤ `SELECT RDD1 JOIN RDD2 WHERE RDD1>100 AND RDD2>1000`

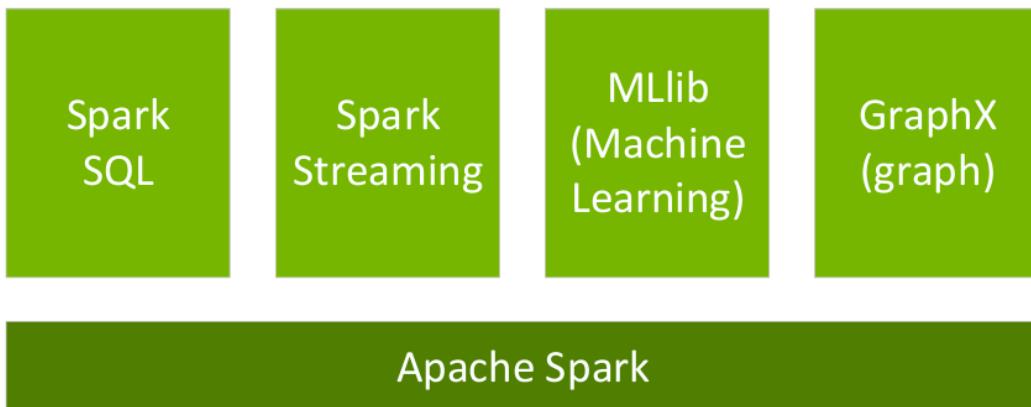
- **2 alternativas**

1. **Filtrar** RDD1 y RDD2, **luego unir** conjuntos filtrados

2. **Unir** los RDD y **luego filtrar**

- **1 más rápida**, (*join*, más lenta, sobre conjuntos más pequeños)

## Componentes de Apache Spark



### ¿Qué es Apache Spark?

Spark es un proyecto de Apache anunciado como "computación de clúster ultrarrápida". Tiene una próspera comunidad de código abierto y es el proyecto Apache más activo en este momento.

Spark proporciona una plataforma de procesamiento de datos más rápida y general. Spark le permite ejecutar programas hasta 100 veces más rápido en la memoria o 10 veces más rápido en el disco que Hadoop. El año pasado por ejemplo, Spark superó a Hadoop al completar el concurso Daytona GraySort de 100 TB 3 veces más rápido en una décima parte de la cantidad de máquinas y también se convirtió en el motor de código abierto más rápido para clasificar un petabyte.

Spark también permite escribir código más rápidamente, ya que tiene a su disposición más de 80 operadores de alto nivel. Para demostrar esto, echemos un vistazo a "¡Hola mundo!" de BigData: el ejemplo de Word Count. Escrito en Java para MapReduce, tiene alrededor de 50 líneas de código, mientras que en Spark (y Scala) puede hacerlo de la siguiente manera:

```
sparkContext.textFile("hdfs://...")  
    .flatMap(line => line.split(" "))  
    .map(word => (word, 1)).reduceByKey(_ + _)  
    .saveAsTextFile("hdfs://...")
```

Otro aspecto importante al aprender a usar Apache Spark es el shell interactivo (REPL) que proporciona listo para usar. Con REPL (**R**eader-E**E**valuate-P**R**int Loop), se puede probar el resultado de cada línea de código sin necesidad de codificar y ejecutar primero todo el trabajo. Por lo tanto, el camino hacia el código de trabajo es mucho más corto y se hace posible el análisis de datos ad-hoc.

Las características clave adicionales de Spark incluyen:

- Actualmente proporciona API en Scala, Java y Python, con soporte para otros lenguajes (como R)
- Se integra bien con el ecosistema Hadoop y las fuentes de datos (HDFS, Amazon S3, Hive, HBase, Cassandra, etc.)
- Puede ejecutarse en clústeres administrados por Hadoop YARN o Apache Mesos, y también puede ejecutarse de forma independiente

El núcleo de Spark se complementa con un conjunto de potentes bibliotecas de alto nivel que se pueden usar sin problemas en la misma aplicación. Estas bibliotecas actualmente incluyen SparkSQL, Spark Streaming, MLlib (para aprendizaje automático) y GraphX, cada una de las cuales se detalla más en esta sesión. Actualmente también se están desarrollando bibliotecas y extensiones adicionales de Spark.



### *Spark Core*

Spark Core es el motor base para el procesamiento de datos paralelos y distribuidos a gran escala. Es responsable de:

- Gestión de memoria y recuperación de fallos
- Programar, distribuir y monitorear trabajos en un clúster
- Interactuar con los sistemas de almacenamiento

Spark presenta el concepto de un RDD (Conjunto de datos distribuido resistente), una colección de objetos distribuidos, tolerantes a fallas e *inmutables* que se pueden operar en paralelo. Un RDD puede contener cualquier tipo de objeto y se crea cargando un conjunto de datos externo o distribuyendo una colección desde el programa controlador.

Los RDD admiten dos tipos de operaciones:

- **Transformations** son operaciones (como mapa, filtro, combinación, unión, etc.) que se realizan en un RDD y que generan un nuevo RDD que contiene el resultado.
- **Actions** son operaciones (como reducir, contar, primero, etc.) que devuelven un valor después de ejecutar un cálculo en un RDD.

Las transformaciones en Spark son "perezosas", lo que significa que no calculan sus resultados de inmediato. En cambio, simplemente "recuerdan" la operación que se realizará y el conjunto de datos (por ejemplo, un archivo) en el que se realizará la operación. Las transformaciones solo se calculan realmente cuando se llama a una acción y el resultado se devuelve al programa controlador. Este diseño permite que Spark funcione de manera más eficiente. Por ejemplo, si un archivo grande se transformó de varias maneras y pasó a la primera acción, Spark solo procesaría y devolvería el resultado de la primera línea, en lugar de hacer el trabajo para todo el archivo.

De forma predeterminada, cada RDD transformado se puede volver a calcular cada vez que ejecuta una acción en él. Sin embargo, también puede conservar un RDD en la memoria utilizando el método persistente o de caché, en cuyo caso Spark mantendrá los elementos en el clúster para un acceso mucho más rápido la próxima vez que lo consulte.

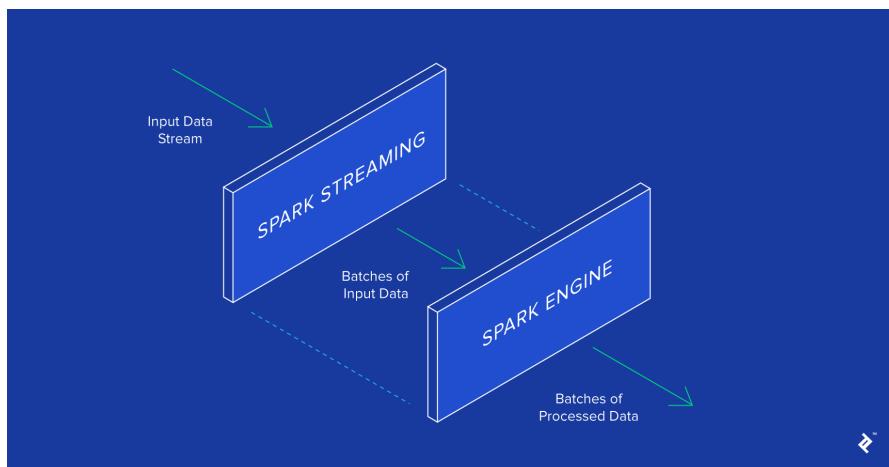
### *SparkSQL*

SparkSQL es un componente de Spark que admite la consulta de datos a través de SQL o del lenguaje de consulta de Hive. Se originó como una migración de Apache Hive para ejecutarse sobre Spark (en lugar de MapReduce) y ahora está integrado con la pila Spark. Además de brindar soporte para varias fuentes de datos, permite entrelazar consultas SQL con transformaciones de código, lo que resulta en una herramienta muy poderosa. A continuación se muestra un ejemplo de una consulta compatible con Hive:

```
// sc is an existing SparkContext.  
  
val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)  
  
sqlContext.sql("CREATE TABLE IF NOT EXISTS src (key INT, value  
STRING)")  
sqlContext.sql("LOAD DATA LOCAL INPATH  
'examples/src/main/resources/kv1.txt' INTO TABLE src")  
  
// Queries are expressed in HiveQL  
sqlContext.sql("FROM src SELECT key,  
value").collect().foreach(println)
```

## *Spark Streaming*

Spark Streaming admite el procesamiento en tiempo real de streaming de datos, como archivos de log del servidor web de producción (por ejemplo, Apache Flume y HDFS/S3), redes sociales como Twitter y varias colas de mensajería como Kafka. Bajo el capó, Spark Streaming recibe los flujos de datos de entrada y los divide en lotes. Luego, el motor Spark los procesa y genera el flujo final de resultados en lotes, como se muestra a continuación.

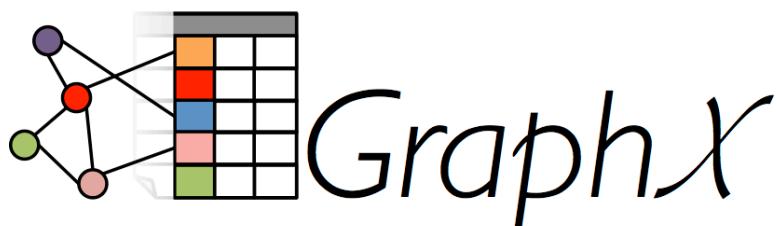


La API de Spark Streaming se parece mucho a la de Spark Core, lo que facilita a los programadores trabajar en el mundo de los datos por lotes y de transmisión.

## *Mllib*

Mlib es una biblioteca de aprendizaje automático que proporciona varios algoritmos diseñados para escalar en un clúster para clasificación, regresión, agrupamiento, filtrado colaborativo, etc. Algunos de estos algoritmos también funcionan con transmisión de datos, como la regresión lineal que utiliza mínimos cuadrados ordinarios o el agrupamiento de k-medias (y más en el camino). Apache Mahout (una biblioteca de aprendizaje automático para Hadoop) ya se alejó de MapReduce y unió fuerzas en Spark Mlib.

## *GraphX*



GraphX es una biblioteca para manipular gráficos y realizar operaciones paralelas de gráficos. Proporciona una herramienta uniforme para ETL(Extract, Transform, Load), análisis exploratorio y cálculos de gráficos iterativos. Además de las operaciones integradas para la manipulación de gráficos, proporciona una biblioteca de algoritmos gráficos comunes, como PageRank.

## Cómo usar Apache Spark: caso de uso de detección de eventos

Ahora que hemos respondido a la pregunta "¿Qué es Apache Spark?", pensemos en qué tipo de problemas o desafíos podría usarse de manera más efectiva.

Hay un artículo por ejemplo sobre un experimento para detectar un terremoto mediante el análisis de una transmisión de Twitter. Curiosamente, se demostró que esta técnica probablemente informaría de un terremoto en Japón más rápido que la Agencia Meteorológica de Japón. A pesar de que usaron una tecnología diferente en el artículo, parece que es un gran ejemplo para ver cómo podemos usar Spark.

Primero, tendríamos que filtrar los tweets que parecen relevantes como "terremoto" o "sacudida". Fácilmente podríamos usar Spark Streaming para ese propósito de la siguiente manera:

```
TwitterUtils.createStream(...)  

    .filter(_.getText.contains("earthquake") ||  

    _.getText.contains("shaking"))
```

Luego, tendríamos que realizar un análisis semántico de los tweets para determinar si parecen estar haciendo referencia a un terremoto actual. Tweets como "¡Terremoto!" o "Ahora está temblando", por ejemplo, se considerarían coincidencias positivas, mientras que tuits como "Asistir a una conferencia sobre terremotos" o "El terremoto de ayer fue aterrador" no lo serían.

Los autores del artículo utilizaron una máquina de vectores de soporte (SVM) para este propósito. Haremos lo mismo aquí, pero también podemos probar una versión de transmisión. Un ejemplo de código resultante de MLlib sería similar al siguiente:

```
// We would prepare some earthquake tweet data and load it in LIBSVM  

format.  

val data = MLUtils.loadLibSVMFile(sc, "sample_earthquake_tweets.txt")  
  

// Split data into training (60%) and test (40%).  

val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)  

val training = splits(0).cache()  

val test = splits(1)  
  

// Run training algorithm to build the model  

val numIterations = 100  

val model = SVMWithSGD.train(training, numIterations)  
  

// Clear the default threshold.
```

```

model.clearThreshold()

// Compute raw scores on the test set.
val scoreAndLabels = test.map { point =>
    val score = model.predict(point.features)
    (score, point.label)
}

// Get evaluation metrics.
val metrics = new BinaryClassificationMetrics(scoreAndLabels)
val auROC = metrics.areaUnderROC()

println("Area under ROC = " + auROC)

```

Si estamos satisfechos con la tasa de predicción del modelo, podríamos pasar a la siguiente etapa y reaccionar cada vez que descubramos un terremoto. Para detectar uno, necesitamos un cierto número (es decir, densidad) de tweets positivos en una ventana de tiempo definida (como se describe en el artículo). Tenga en cuenta que, para los tweets con los servicios de ubicación de Twitter habilitados, también extraeríamos la ubicación del terremoto. Armados con este conocimiento, podríamos usar SparkSQL y consultar una tabla de Hive existente (almacenando usuarios interesados en recibir notificaciones de terremotos) para recuperar sus direcciones de correo electrónico y enviarles un correo electrónico de advertencia personalizado, de la siguiente manera:

```

// sc is an existing SparkContext.
val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
// sendEmail is a custom function
sqlContext.sql("FROM earthquake_warning_users SELECT firstName,
lastName, city, email")
.collect().foreach(sendEmail)

```

## Otros casos de uso de Apache Spark

Los posibles casos de uso de Spark se extienden mucho más allá de la detección de terremotos, por supuesto.

Aquí hay una muestra rápida (¡pero ciertamente no exhaustiva!) de otros casos de uso que requieren lidiar con la velocidad, la variedad y el volumen de Big Data, para los cuales Spark es tan adecuado:

En la industria de los juegos, el procesamiento y el descubrimiento de patrones de los eventos del juego en tiempo real y la capacidad de responder a ellos de inmediato es una capacidad que

podría generar un negocio lucrativo, para fines tales como la retención de jugadores, publicidad dirigida, auto -ajuste del nivel de complejidad, etc.

En la industria del comercio electrónico, la información de transacciones en tiempo real podría pasarse a un algoritmo de streaming clustering como k-means o filtrado colaborativo como ALS. Los resultados podrían incluso combinarse con otras fuentes de datos no estructurados, como comentarios de clientes o reseñas de productos, y usarse para mejorar y adaptar constantemente las recomendaciones a lo largo del tiempo con las nuevas tendencias.

En la industria de las finanzas o la seguridad, Spark Stack podría aplicarse a un sistema de detección de fraude o intrusión o autenticación basada en riesgos. Podría lograr resultados de primer nivel al recopilar grandes cantidades de registros archivados, combinándolos con fuentes de datos externas como información sobre filtraciones de datos y cuentas comprometidas (ver, por ejemplo, <https://haveibeenpwned.com/>) e información de la conexión/ solicitud como la geolocalización de IP o la hora.

## Features



¿Cuáles son las características principales de Apache Spark?

Las empresas de comercio electrónico como Alibaba, las empresas de redes sociales como Tencent y el motor de búsqueda chino Baidu ejecutan operaciones Apache Spark a escala. Aquí hay algunas características que son responsables de su popularidad.

### 1) Velocidad de procesamiento rápida

La velocidad ha sido la USP de Apache Spark desde sus inicios. La primera y principal ventaja de usar Apache Spark para su big data es que ofrece 100 veces más rápido en la memoria y 10 veces más rápido en el disco en los clústeres de Hadoop. Habiendo establecido el récord mundial de clasificación de datos en disco, Apache Spark ha demostrado una velocidad ultrarrápida cuando se almacena una gran escala de datos en el disco.

### 2) Almacenamiento en caché: aceleración de las aplicaciones de Big Data

La eficacia de una aplicación de big data depende en gran medida de su rendimiento. El almacenamiento en caché en Apache Spark proporciona una mejora de 10x a 100x en el rendimiento de las aplicaciones de big data a través de capacidades de persistencia de disco.

### 3) Despliegue y Compatibilidad

Spark puede ejecutarse en Hadoop, Apache Mesos, Kubernetes, de forma independiente o en la nube. Puede operar diversas fuentes de datos.

### 4) Procesamiento en tiempo real

Spark tiene MapReduce que puede procesar datos almacenados en Hadoop y también tiene Spark Streaming que puede manejar datos en tiempo real.

### 5) Polyglot: admite una variedad de lenguajes de programación

Las aplicaciones Spark se pueden implementar en una variedad de lenguajes como Scala, R, Python, Java y Clojure. Con API de alto nivel para cada uno de estos lenguajes de programación

y un shell en Python y Scala, los desarrolladores pueden trabajar fácilmente según sus preferencias.

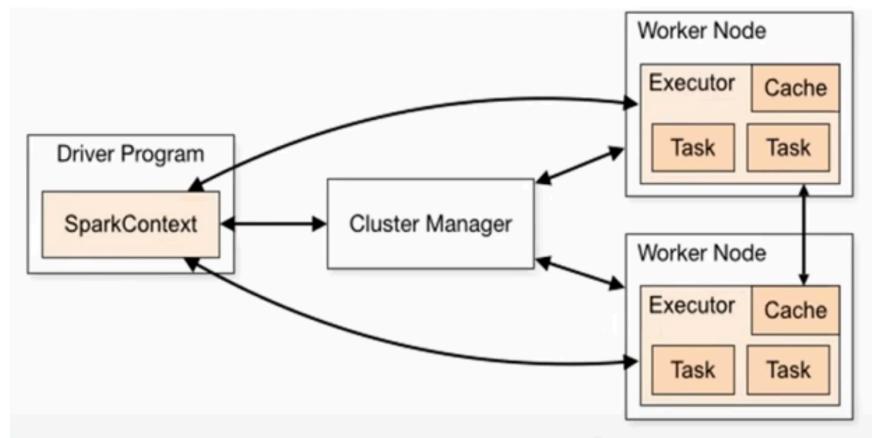
#### 6) Bibliotecas poderosas

Contiene más que solo mapear y reducir funciones. Contiene bibliotecas SQL y marcos de datos, MLlib (para aprendizaje automático), GraphX y transmisión en Spark que ofrecen herramientas poderosas para el análisis de datos.

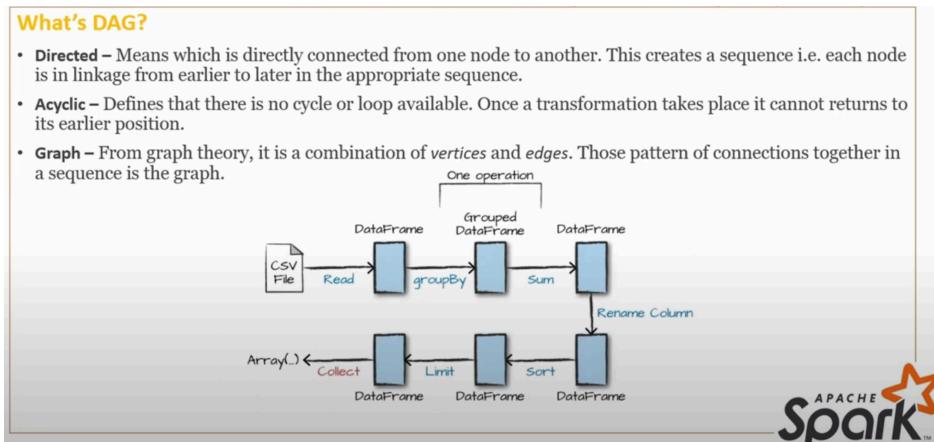
Ahora que conoce sus interesantes características, exploremos Spark Architecture para darnos cuenta de lo que la hace tan especial. Esta parte es un recurso integral que brinda una descripción general de la arquitectura Spark con la ayuda de un diagrama de arquitectura Spark y es un buen recurso para principiantes que buscan aprender Spark.

## Arquitectura

El diagrama de arquitectura base de Apache Spark se proporciona en la siguiente figura:



Cuando se ejecuta el programa Driver en la arquitectura Apache Spark, llama al programa real de una aplicación y crea un SparkContext. SparkContext contiene todas las funciones básicas. Spark Driver incluye varios otros componentes, incluidos un programador de DAG, un programador de tareas, un programador de backend y un administrador de bloques, todos los cuales son responsables de traducir el código escrito por el usuario en trabajos que realmente se ejecutan en el clúster.



El Administrador de clústeres administra la ejecución de varios trabajos en el clúster. Spark Driver funciona junto con Cluster Manager para controlar la ejecución de varios otros trabajos. El Administrador de clústeres realiza la tarea de asignar recursos para el trabajo. Una vez que el trabajo se ha dividido en trabajos más pequeños, que luego se distribuyen a los worker nodes, SparkDriver controlará la ejecución.

Se pueden usar muchos nodos trabajadores para procesar un RDD creado en SparkContext, y los resultados también se pueden almacenar en caché.

Spark Context recibe información de la tarea del Administrador de clústeres y la pone en cola en los worker nodes.

El executor está a cargo de llevar a cabo estos deberes. La vida útil de los ejecutores es la misma que la de la aplicación Spark. Podemos aumentar el número de workers si queremos mejorar el rendimiento del sistema. De esta manera, podemos dividir los jobs en partes más coherentes.

### **Aplicaciones de arquitectura Spark**

Una vista de alto nivel de la arquitectura de la aplicación Apache Spark es la siguiente:

#### **Spark Driver**

El nodo maestro (proceso) en un proceso controlador que coordina a los trabajadores y supervisa las tareas. Spark se divide en trabajos y se programa para ejecutarse en ejecutores en clústeres. El controlador crea contextos de Spark (puertas de enlace) para monitorear el trabajo que se ejecuta en un clúster específico y para conectarse a un clúster de Spark. En el diagrama, los programas del controlador llaman a la aplicación principal y crean un contexto Spark (actúa como una puerta de enlace) que monitorea conjuntamente el trabajo que se ejecuta en el clúster y se conecta a un clúster Spark. Todo se ejecuta utilizando el contexto Spark.

Cada sesión de Spark tiene una entrada en el contexto de Spark. Los controladores Spark incluyen más componentes para ejecutar trabajos en clústeres, así como administradores de clústeres. El contexto adquiere nodos trabajadores para ejecutar y almacenar datos a medida que los clústeres de Spark están conectados a diferentes tipos de administradores de clústeres. Cuando se ejecuta un proceso en el clúster, el trabajo se divide en etapas con etapas de ganancia en tareas programadas.

#### *Spark Executors*

Un ejecutor es responsable de ejecutar un trabajo y almacenar datos en un caché desde el principio. Los ejecutores primero se registran con el programa controlador al principio. Estos ejecutores tienen una serie de intervalos de tiempo para ejecutar la aplicación al mismo tiempo.

El ejecutor ejecuta la tarea cuando ha cargado datos y se eliminan en modo inactivo. El ejecutor se ejecuta en el proceso de Java cuando se cargan y eliminan datos durante la ejecución de las tareas. Los ejecutores se asignan dinámicamente y se agregan y eliminan constantemente durante la ejecución de las tareas. Un programa controlador monitorea a los ejecutores durante su desempeño. Las tareas de los usuarios se ejecutan en el proceso de Java.

#### *Cluster Manager*

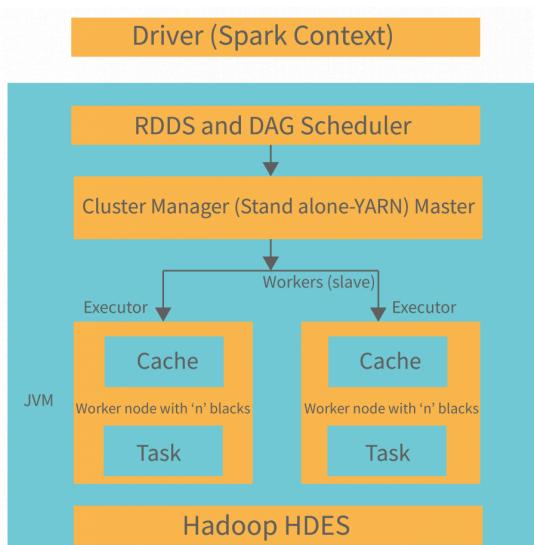
Un programa controlador controla la ejecución de trabajos y almacena datos en un caché. Al principio, los ejecutores se registran con los drivers. Este ejecutor tiene una serie de intervalos de tiempo para ejecutar la aplicación al mismo tiempo.

Los ejecutores leen y escriben datos externos además de atender las solicitudes de los clientes. Un trabajo se ejecuta cuando el ejecutor ha cargado datos y se han eliminado en el estado inactivo. El ejecutor se asigna dinámicamente y se agrega y elimina constantemente según la duración de su uso.

Un programa controlador monitorea a los ejecutores mientras realizan las tareas de los usuarios. El código se ejecuta en el proceso de Java cuando un ejecutor ejecuta la tarea de un usuario.

## Worker Nodes

Los nodos esclavos funcionan como ejecutores, procesan tareas y devuelven los resultados al contexto de Spark. El nodo maestro envía tareas al contexto de Spark y los worker nodes las ejecutan. Simplifican el proceso al impulsar los worker nodes (1 a n) para manejar tantos trabajos como sea posible en paralelo al dividir el job en sub-jobs en varias máquinas. Un Spark worker supervisa los worker nodes para garantizar que el cálculo se realice de forma sencilla. Cada worker node maneja una tarea de Spark. En Spark, una *partición* es una unidad de trabajo y se asigna a un executor para cada uno.



Vale la pena recordar los siguientes puntos sobre este diseño:

Existen múltiples procesos ejecutores para cada aplicación, que ejecutan tareas en múltiples subprocessos a lo largo de toda la aplicación. Esto permite aislar las aplicaciones tanto en el lado de la programación (los controladores pueden programar tareas individualmente) como en el lado del ejecutor (las tareas de diferentes aplicaciones pueden ejecutarse en diferentes JVM). Por lo tanto, los datos deben escribirse en un sistema de almacenamiento externo antes de que puedan compartirse entre diferentes aplicaciones de Spark.

Incluso en un administrador de clústeres que también admite otras aplicaciones, Spark se puede ejecutar si puede adquirir procesos ejecutores y estos se comunican entre sí. Es relativamente fácil para Spark operar incluso en un administrador de clústeres si esto se puede hacer incluso con otras aplicaciones (por ejemplo, Mesos/YARN).

El programa driver debe escuchar y aceptar las conexiones entrantes de sus ejecutores a lo largo de su vida útil (por ejemplo, consulte spark.driver.port en la sección de configuración de la red). Los trabajadores deben poder conectarse al programa del controlador a través de la red.

El driver es responsable de programar tareas en el clúster. Debe ejecutarse en la misma red local que los nodos trabajadores, preferiblemente en la misma máquina. Si desea enviar solicitudes al clúster, es preferible abrir una RPC y hacer que el controlador envíe operaciones desde cerca en lugar de ejecutar el controlador lejos de los nodos trabajadores.

### **Modos de ejecución**

Puede elegir entre tres modos de ejecución diferentes: local, compartido y dedicado. Estos determinan dónde se encuentran físicamente los recursos de su aplicación cuando ejecuta su aplicación. Puede decidir dónde almacenar los recursos localmente, en una ubicación compartida o en una ubicación dedicada.

- Modo de clúster
- Modo cliente
- Modo local

*Modo de clúster:* el modo de clúster es la forma más frecuente de ejecutar aplicaciones Spark. En el modo de clúster, un usuario entrega un JAR precompilado, una secuencia de comandos de Python o una secuencia de comandos R a un administrador de clústeres. Una vez que el administrador del clúster recibe el JAR precompilado, el script Python o el script R, el proceso del controlador se inicia en un nodo trabajador dentro del clúster, además de los procesos ejecutores. Esto significa que el administrador del clúster está a cargo de todos los procesos relacionados con la aplicación de Spark.

*Modo cliente:* en contraste con el modo de clúster, donde el controlador Spark permanece en la máquina cliente que envió la aplicación, el controlador Spark se elimina en el modo cliente y, por lo tanto, es responsable de mantener el proceso del controlador Spark en la máquina cliente. Estas máquinas, generalmente denominadas máquinas de puerta de enlace o nodos perimetrales, se mantienen en la máquina cliente.

*Modo local:* el modo local ejecuta toda la aplicación Spark en una sola máquina, a diferencia de los dos modos anteriores, que pusieron en paralelo la aplicación Spark a través de subprocessos en esa máquina. Como resultado, el modo local usa subprocessos en lugar de subprocessos paralelizados. Esta es una forma común de experimentar con Spark, probar sus aplicaciones o experimentar iterativamente sin tener que hacer ningún cambio por parte de Spark.

En la práctica, no recomendamos usar el modo local para ejecutar aplicaciones de producción.

### **Tipos de administradores de clústeres:**

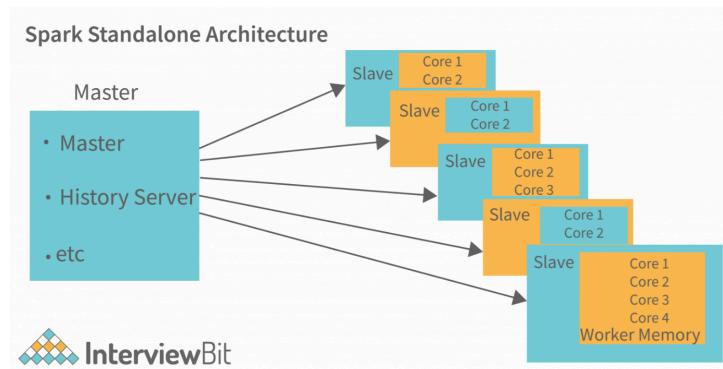
Hay varios administradores de clústeres compatibles con el sistema:

#### *Standalone*

Se incluye un administrador de clústeres Spark con el paquete de software para facilitar la configuración de un clúster. Resource Manager y Worker son los componentes únicos de Spark

Standalone Cluster que son independientes. Solo hay un ejecutor que ejecuta tareas en cada nodo trabajador en el modo de clúster independiente. Cuando un cliente establece una conexión con el maestro autónomo, solicita recursos y comienza el proceso de ejecución, un maestro autónomo en clúster inicia el proceso de ejecución.

El cliente aquí es el maestro de la aplicación y quiere los recursos del administrador de recursos. Tenemos una interfaz de usuario web para ver todos los clústeres y estadísticas de trabajo en el Administrador de clústeres.

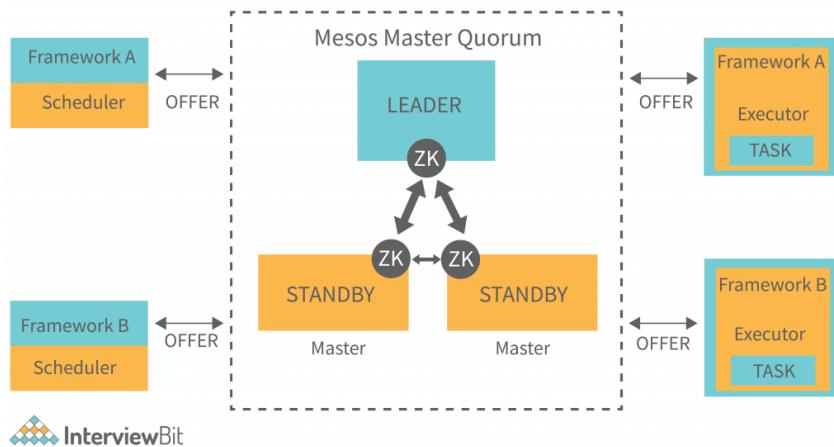


## Apache Mesos

Puede ejecutar Hadoop MapReduce y aplicaciones de servicio, además de ser un administrador general de clústeres. Apache Mesos contribuye al desarrollo y la gestión de clústeres de aplicaciones mediante el uso compartido y el aislamiento de recursos dinámicos. Permite la implementación y administración de aplicaciones en entornos de clústeres a gran escala.

El framework Mesos incluye tres componentes:

- Mesos Master: un clúster Mesos Master proporciona tolerancia a fallas (la capacidad de operar y recuperarse de pérdidas cuando ocurre una falla). Debido al diseño de Mesos Master, un grupo contiene muchos Mesos Master.
- Mesos Slave: un Mesos Slave es una instancia que entrega recursos al clúster. Cuando un Mesos Master asigna una tarea, Mesos Slave no asigna recursos.
- Mesos Frameworks: las aplicaciones pueden solicitar recursos del clúster para que la aplicación pueda realizar las tareas. Mesos Frameworks permite esto.



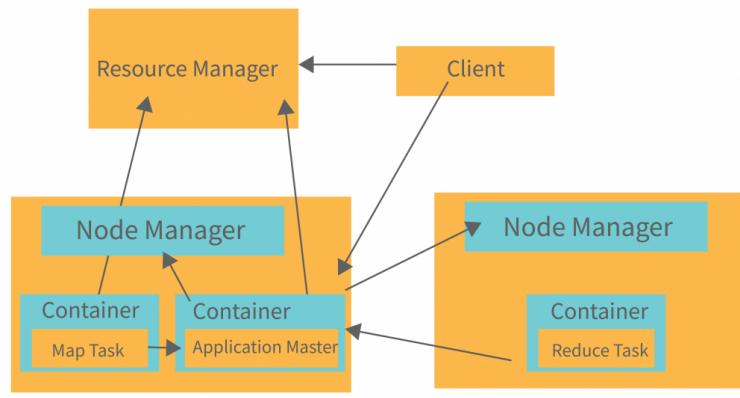
 InterviewBit

## Hadoop Yarn

Una característica clave de Hadoop 2.0 es el administrador de recursos mejorado. El ecosistema de Hadoop se basa en YARN para manejar los recursos. Consta de los siguientes dos componentes:

- Administrador de recursos: controla la asignación de recursos del sistema en todas las aplicaciones. Se incluyen un programador y un administrador de aplicaciones. Las aplicaciones reciben recursos del Programador.
- Administrador de nodos: cada trabajo o aplicación necesita uno o más contenedores, y el administrador de nodos supervisa estos contenedores y su uso. El administrador de nodos consta de un administrador de aplicaciones y un administrador de contenedores. Cada tarea en el marco MapReduce se ejecuta en un contenedor. El administrador de nodos supervisa los contenedores y el uso de recursos, y esto se informa al administrador de recursos.

YARN Architecture



 InterviewBit