Cloud computing y datos masivos (*big data*)

Remo Suppi Boldrito

PID_00247724

Tiempo mínimo previsto de lectura y comprensión: **6 horas**

UOC
Universitat Oberta de Catalunya

Índice

Introducción

Los datos masivos (*big data*) es un término que ha adquirido especial relevancia durante la última década y su crecimiento y utilización no paran de crecer.

Este concepto (también referido como *datos a gran escala*) se utiliza para un conjunto de datos que, dada su cantidad/tamaño, no es posible analizar/procesar con los métodos tradicionales, y es necesario optar por herramientas y algoritmos alternativos que puedan extraer información en tiempos aceptables.

Muchas son las noticias en donde la utilización de *big data* es una realidad.

Por ejemplo: la utilización de las bases de datos de búsqueda como corrector de ortografía de Google que sugiere (incluso busca con anticipación antes que terminemos de escribir) la palabra adecuada porque otros miles de personas lo han buscado antes, los datos utilizados por la liga NFL para proveer de datos para la toma de decisiones, o la NBA, o cómo el *big data* ayudó aganar a Obama las elecciones en Estados Unidos, o aplicaciones como PriceStats, que de la información de precios de tiendas en línea «obtiene» información diaria de la evolución de la inflación en veintidós países del mundo, o la aplicación Illustreets, que sobre la base del *open data* disponible en Reino Unido permite escoger la zona donde comprar/alquilar una vivienda a partir de criterios «diferentes» sobre el barrio (violencia, edad, alquiler social, tipos de viviendas, distancia a los comercios/hospital, etc.) y estableciendo un listado de *standard of living* para la ayuda a la toma de decisiones.

No obstante, es necesario tener en cuenta que los datos son datos, y a veces su interpretación puede dar lugar a dudas o errores.

Por ejemplo, fue el caso de la predicción de epidemias de gripe que realizaba Google (Google Flu Trends). En 2008, los investigadores de la compañía exploraron la potencialidad de los datos masivos de las búsquedas en temas vinculados con la gripe partiendo de que la persona enferma busca información relacionada, lo cual les permitía obtener predicciones instantáneas de las epidemias a escala mundial (*Nature*, 2008). Pero en 2013 las predicciones fracasaron estrepitosamente (con un 140 % de error en el pico) y algunos científicos, en *Science*, indicaron que el problema no estaba en el valor del *big data*, sino en las problemáticas derivadas de su uso e interpretación. En este caso en particular fueron un cúmulo de situaciones, más allá de las buenas intenciones de los investigadores de Google (aunque algunos científicos le critican la opacidad en el método y los datos), ya que el algoritmo era vulnerable a búsquedas no correlacionadas, a no contemplar los cambios del comportamiento de las búsquedas en el tiempo, a la introducción de nuevos complementos en el algoritmo de búsquedas sobre salud que no se tuvieron en cuenta para las predicciones y que les afectaban, y una larga lista más. Con ello se demostró un aspecto crítico del *big data* donde el problema no está en los datos sino en el modo en que son tratados, filtrados, procesados, etc. (ved el artículo de *Wired*).

El *big data* ha generado la creación/adecuación de técnicas y metodologías y la creación de nuevas herramientas con un crecimiento constante, ya que las necesidades derivadas de la captura, almacenamiento, búsqueda, compartición, análisis y visualización son totalmente diferentes a las conocidas hasta ahora y requieren nuevos métodos, procedimientos y estrategias para llevarlas a cabo.

Muchos campos de la ciencia, pero también orientados a negocio, están obteniendo beneficios de estos datos y sus necesidades/requerimientos son cada vez más específicos/diversos.

Entre los diferentes segmentos se pueden mencionar la ciencia (genética, biología, medicina, ingeniería, física/astrofísica...), los análisis de negocio (banca, publicidad, bolsa, compras, moda, fraude, tendencias de negocio...), la salud (epidemias, incidencias y contagios de enfermedades infecciosas...), u otros tan diversos como el espionaje y seguimiento a la población o la lucha contra el crimen organizado.

Es muy interesante la disertación «*Each of us, All of us*», de Eric Schmidt (director ejecutivo de Google hasta 2011 y hoy presidente de Alphabet Inc.) en el año 2011, donde explica que había, en aquel entonces, 295 exabytes de información en el mundo sobre la base del estudio de Hilbert y López «*The world's technological capacity to store, communicate and compute information*» (*Science*). El estudio del período 1986-2007 establecía como punto de inicio de la era digital el año 2002 como aquel en el que la información digital almacenada superó a la capacidad de almacenamiento analógica, y consideraba que en al año 2007 el 94 % de toda la información era digital.

En 2016, el estudio *How Much Data is Produced Every Day?* especifica que las sociedades actuales generan cada día miles de millones de fotos, vídeos, texto, y se opina sobre preferencias, compras y tendencias, que los gobiernos/instituciones juntan cantidades enormes de datos de sus ciudadanos con diferentes objetivos –lo mismo pasa con las empresas y sus negocios y actividades–, y que todo ello se puede cuantificar en 2,5 exabytes/día (1 exabyte = 10^{18} bytes = 1 millón de terabytes). Una representación visual del tamaño de estos datos se puede hacer considerando que un disco de un terabyte tiene una altura de 20,17 mm, por lo cual la altura de los discos equivalente sería de 50,4 kilómetros. ¿Y el futuro? Si la cantidad de datos en el mundo en 2013 fue de 4,4 zettabytes, en 2020 se espera llegar a 44 zettabytes (10^{21} bytes = 1.000 millones de terabytes), y con las tendencias de IoT (internet de las cosas) todas estas tendencias no hacen más que incrementarse (informe IDG, infografía de AIS).

Pero los datos solo son datos, y lo que se necesita es información, por lo cual se debe filtrar, procesar, cruzar y analizar para extraer la información subyacente; para ello se necesita el ***cloud computing***, que es el único que puede proveer de los recursos necesarios para realizar estas tareas y sin grandes inversiones. ***Big data*** y ***cloud computing*** son dos términos de esta nueva década que serán fundamentales para la generación de nuevos negocios y oportunidades empresariales (informe de EMC).

Es interesante la proyección que hace el informe del McKinsey Global Institute en el que se analizan las posibilidades de negocio del *big data* y su utilización, y cómo este genera nuevas oportunidades empresariales que se pueden cuantificar en miles de millones de dólares [Bdn].

Los expertos de *Gartner* en 2011 definieron como **características** del *big data* las llamadas **3V** (volumen, variedad, velocidad) e IBM, otro de los grandes actores en este tema, extendió estas características a «veracidad» y generó así las 4V (infografía); con todo, han sido ampliadas a medida que se han ido consolidando las tecnologías [Bdd] y actualmente podemos enumerarlas de este manera (orden alfabético):

1) Complejidad: aspecto que prevalecerá sobre todo cuando se disponga de grandes volúmenes de datos ensamblados de diversas fuentes.

2) Variabilidad: no todos los datos serán homogéneos y habrá inconsistencias que el sistema de almacenamiento y procesamiento deberá contemplar para la gestión de los mismos. Las herramientas utilizadas deberán ser flexibles para adaptarse a estas situaciones y no generar errores cuando el tipo/formato/estructura de los datos no sea el esperado.

3) Variedad: formas y tipos de fuentes de datos que tendrán (o no) diferentes estructuras en comparación de los ya conocidos hasta ahora y donde un sistema de procesamiento deberá ser capaz de almacenar y procesar sin tener que realizar un preproceso para estructurar o indexar la información.

4) Velocidad: a la que se producen, y a las que será necesario capturar, almacenar, analizar y extraer valor para aprovechar las ventanas de oportunidades basadas en estos datos. Esto constituye un reto para la tecnología y el *cloud computing* es el principal actor que puede satisfacer estas necesidades en este escenario.

5) Veracidad: calidad de los datos capturados (predictibilidad y disponibilidad).

6) Volumen: cantidad de datos utilizada (que indicará si se trata de datos masivos o no).

La potencialidad de nueva información, negocios y oportunidades generada por el *big data* ha evolucionado en paralelo a la tecnología necesaria para tratarla. Nombres como Hadoop, MapReduce, NoSQL, Cassandra, *business intelligence* o *machine learning* han adquirido una popularidad derivada de las herramientas/métodos utilizados para tratar el *big data*, las cuales pueden trabajar sobre tres tipos de datos masivos de acuerdo a su **estructura**:

1) Datos estructurados (*structured data*): datos como los utilizados por la tecnología actual, definidos su longitud y su formato y que se pueden (generalmente) almacenar en tablas (bases de datos relacionales/hojas de cálculo).

2) Datos no estructurados (*unstructured data*): mantienen una estructura similar a como han sido recolectados, sin formato específico; generalmente son una mezcla de tipos/formatos de datos (datos en PDF, documentos multimedia, correos electrónicos o documentos de texto).

3) Datos semiestructurados (*semistructured data*): datos que se encuentran entre las dos definiciones anteriores; no tienen campos determinados, pero incluyen marcas para separar los diferentes elementos (por ejemplo, archivos HTML, XML, JSON...).

Toda esta «revolución» ha permitido reforzar y evolucionar técnicas y metodologías existentes, pero una parte importante de este movimiento es la nueva «forma de pensar y hacer» que se ha tenido que crear para tratar con estos datos y de la que han surgido nuevas metodologías, técnicas y herramientas no conocidas hasta ahora, así como otras ya conocidas y en desuso se han reformulado y cobrado nuevo impulso para adecuarse a los nuevos datos/tiempos.

Entre las técnicas de búsqueda y análisis de datos que se han tenido que adaptar y reformular para tratar con el *big data*, se pueden mencionar:

- **asociación**: encontrar relaciones entre diferentes variables causales,
- **minería de datos**: *data mining*, encontrar comportamientos predictivos combinando métodos estadísticos y de *machine learning*,
- **agrupación**: *clustering*, división de grandes grupos de datos en grupos más pequeños para encontrar similitudes,
- **análisis de texto**: *text analytics*, dado que gran parte del conjunto de datos está en modo texto se han tenido que definir nuevos métodos para extraer información de grandes volúmenes de datos en texto.

Para ello, en muchos casos el tratamiento está orientado hacia tipos de datos no estructurados, por lo cual se han tenido que buscar nuevas formas de almacenarlos/buscarlos. En este sentido, las técnicas NoSQL (*not only SQL*) se refieren a datos que no cumplen esquemas tradicionales de entidad-relación de las bases de datos relacionales (tradicionales); permiten además formas más flexibles y concurrentes de manipular grandes cantidades de información y de manera mucho más rápida. Este tratamiento no estándar es utilizado para almacenar/recuperar/ buscar la información en diferentes formas, como por ejemplo por medio de una clave-valor, de un grafo, o métodos orientados a columnas entre las más importantes, y esto a su vez se ha traducido en nuevas herramientas que se han tenido que desarrollar (o evolucionar las existentes),

por ejemplo, en gestores de bases de datos (Casandra, MongoDB, MariaDB, Hive, HBase, GraphDB...) y en nuevos métodos para procesarlos (Hadoop, Apache Stark...).

Es interesante analizar la línea de tiempo del *big data* propuesta por Winshuttle, donde se pueden encontrar los acontecimientos e hitos más importantes, así como la predicción del Horizonte 2020; el informe de BBVA muestra, de forma muy gráfica y clara, que los modelos de negocio basadosen datos masivos serán empresas fructíferas a corto y medio plazo (publicidad mezclada con tendencias sociales, segmentación de los clientes, captura de oportunidades en venta y marketing, toma de decisiones, detección de pérdidas/comportamiento del cliente/fraude, cuantificación de riesgo, sentimientos del mercado, planificación y predicción, análisis de costes, rendimientos...).

En la opinión de algunos expertos, la experiencia del *big data* aplicada a los negocios tendrá como premisa «conocer al cliente» como evolución de los «antiguos» CRM (*customer relationship management*), ya que esto permitirá obtener no solo la información (interna) de este, sino cruzar esta información con la externa (redes sociales, geolocalización, opinión, sentimientos...), lo que a su vez generará valor añadido, pues se estará más cerca de lo que necesita y piensa el cliente para poder ofrecerle productos/servicios con un alto grado de especificación y detalle, aunque ajustado para lo que necesita o considera necesario.

1. Arquitectura y paradigmas del *big data*

Una arquitectura para el procesamiento de *big data*, como en cualquier entorno orientado a datos, está formada por cuatro niveles que interactúan (recolección de datos, almacenamiento, procesamiento y visualización), más uno global de administración.

Esta arquitectura es la habitual que se puede encontrar en cualquier entorno orientado a datos (por ejemplo, *datamining*, *business intelligence* o *deep learning*), pero, cuando se asocia a estos entornos el concepto de *big data*, cada uno de estos niveles ha evolucionado o se han generado nuevas herramientas/algoritmos/entornos para adecuarse a los datos masivos [Bad][Bdb].

En la **recolección** de los datos orientados al *big data* han surgido gran cantidad de herramientas orientadas a este fin (*data ingestion*), que permiten el procesamiento secuencial habitual de datos almacenados (*batch* o lotes) para obtener el siguiente tramo o secuencia (*chunk*) de datos desde el último leído; en todo caso, en su mayoría, y dado el volumen de los mismos, las herramientas han evolucionado para recolectar eficientemente flujos de datos (*streams*) en tiempo real.

En el **almacenamiento** también han surgido grandes cambios para adaptarse a las características de los datos; ello se ha reflejado en una gran cantidad de desarrollos de motores de bases de datos (NoSQL, documentos, columnas, llave/valor, grafos) y sistemas de archivos distribuidos para adecuarse a la realidad de procesamiento y, así, disponer de escalabilidad, fiabilidad y cercanía de los datos para su tratamiento (por ejemplo, Apache HDFS, BeeGFS, Disco DDFS, Google GFS, BaiduFS, GlusterFS, QuantcastFS, CephFS, GridGainin-memoryFS, LustreFS).

Para el **procesamiento** han surgido nuevos paradigmas que han caracterizado todo el entorno (y en cada una de las capas para adecuarlas a sus necesidades); se conocen como MapReduce (MR) o Massive Paralell Processing (MPP).

MapReduce es un paradigma de programación cuyo nombre viene dado por las dos funciones que lo componen (*map* y *reduce*); por medio de un entorno *open-source*, llamado Apache Hadoop, y es hoy en día uno de los paradigmas más utilizados para el procesamiento del *big data*. Este paradigma, si bien no es la solución para todos los problemas, trabaja con grandes conjuntos de datos (petabytes) y se ejecuta sobre sistemas de archivos distribuidos (HDFS) y vinculados a herramientas como HBase, Hive, Impala o Cassandra (bases de datos). Este paradigma es apto para procesar aquellos datos que pueden tra-

bajar sobre tuplas del tipo (clave, valor) y donde la función *map()* procesa en paralelo las tuplas de un dominio y genera una lista de pares en un dominio diferente, las cuales se agruparán bajo la misma clave utilizando un esquema de procesamiento *master-worker* en forma de árbol. Posteriormente, la función *reduce()* se aplica en forma paralela para cada grupo y genera una colección de valores para cada dominio. Un ejemplo típico es el procedimiento para contar el número de veces que aparecen las palabras en un texto. Las funciones serían:

```
map(string name, string document):
  foreach word w in document:
    generate-tupla(w, 1);
reduce(string word, iterator partialList):
  int total = 0;
  foreach value in partialList:
    total += Int(value);
  generate(total);
```

Donde la función *map()* divide el documento por palabras y genera las tuplas (palabra, valor), por ejemplo, la famosa frase de A. Turing «La ciencia es una ecuación diferencial. La religión es una condición de frontera» quedará (La,1), (ciencia,1), (es,1), (una,1), (ecuación,1), (diferencial,1), (La,1), (religión,1), (es,1), (una,1), (condición,1), (de,1), (frontera,1). El entorno agrupará todas las claves iguales ('La:1,1', 'es,1,1', 'una,1,1' y el resto 'clave,1') como entrada a *reduce()*, que generará la agrupación y la suma de los valores de las claves en las cuales, para el ejemplo, será (La,2), (es,2), (una,2) y el resto (clave,1).

El paradigma MPP ofrece una solución tradicional adaptada al *big data* basada en dividir los grandes conjuntos de datos en secciones (*slices*) que serán más fáciles de gestionar; cada uno de ellos serán asignados a un elemento de cómputo para su procesamiento. El dispositivo de cómputo los procesará y, cuando termine el sistema, combinará los resultados parciales para dar un resultado final (equivalente a una secuencia *fork-join* en un modelo *master-workers*). Dado que los elementos de cómputo (procesadores) trabajarán sobre su segmento de datos asignado de la BD y se comunicarán a través de mensajes cuando generen los resultados, no hay interacción entre ellos; esto se conoce como «débilmente acoplados» (otros autores lo denominan *shared nothing*).

Entre las características principales de estos sistemas se pueden enumerar sus posibilidades de sintonización y escalado ilimitado (en principio al número de nodos disponible), con el consiguiente incremento de su rendimiento (prácticamente en forma lineal) y sin cuellos de botella. Existe gran cantidad de bibliografía sobre la comparación MR y MPP (por ejemplo, *Hadoop vs. MPP*, *Introduction to MPP*); no obstante, muchas veces la solución no proviene de uno u otro paradigma, y dado que se complementan bien en cuanto a las prestaciones y los tipos de datos que obtienen mejores rendimientos, es habitual

que se utilicen arquitecturas mixtas donde generalmente se aplica primero MR sobre datos no estructurados y, posteriormente, MPP para procesar y visualizar los datos estructurados generados por el primero.

El proyecto Apache™ Hadoop® es una plataforma *open-source* para el cómputo distribuido, confiable y escalable y utilizado por una gran cantidad de empresas/instituciones de renombre que implementa MapReduce. Esta plataforma se inspira en el trabajo de GoogleFS/MapReduce y su desarrollo se inició dentro del proyecto Apache Nutch y luego bajo el proyecto Hadoop (2006); Yahoo! es uno de sus grandes contribuidores (y D. Cutting uno de los creadores de esta compañía, quien lo bautizó así por el elefante de juguete de su hijo). El código inicial fue de 5k líneas para HDFS y 6k líneas para MapReduce, y la primera versión Hadoop 0.1.0 fue publicada en abril de 2006; la última disponible (estable) es de agosto de 2016, la versión 2.7.3 [Ahd].

Apache Hadoop es un entorno para el procesamiento distribuido de grandes conjuntos de datos a través de *clusters* de cómputo; utiliza modelos de programación sencillos y tiene la posibilidad de escalar desde servidores individuales a miles de procesadores. Básicamente Hadoop (V2.x) está formado por cuatro módulos:

1) Hadoop YARN: marco para la programación de tareas y gestión de recursos del *cluster*.

2) Hadoop MapReduce: sistema basado en YARN para el procesamiento paralelo de grandes conjuntos de datos.

3) Hadoop Distributed File System (HDFS): sistema de archivos distribuido que proporciona acceso de alto rendimiento a los datos de la aplicación.

4) Hadoop Common: las utilidades comunes que soportan los otros módulos de Hadoop.

No obstante, Hadoop se puede relacionar/integrar/ampliar de forma fácil y rápida con los siguientes proyectos (solo algunos de los más habituales/referenciados):

- **Ambari**: herramienta web para el aprovisionamiento, gestión y monitorización de *clusters* Hadoop con HDFS, MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig y Sqoop.

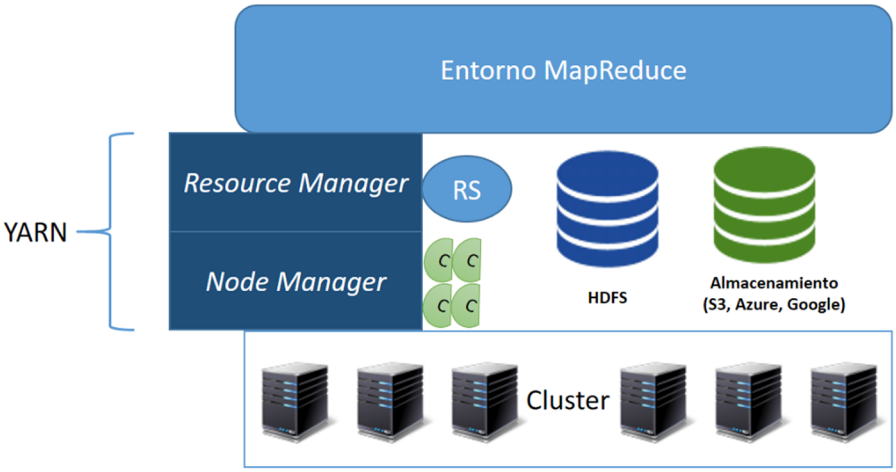
- **Avro**: serialización de datos.

- **Cassandra**: base de datos escalable sin puntos de fallo únicos.

- **Chukwa**: recopilación de datos en sistemas distribuidos.

- **HBase**: base de datos distribuida y escalable.
- **Hive**: almacén de datos que proporciona resumen de datos y consultas *ad hoc*.
- **Mahout**: entorno de *machine learning* y *datamining*.
- **Pig**: lenguaje de flujo de datos de alto nivel.
- **Spark**: motor de cálculo en memoria y general para los datos de Hadoop.
- **Tez**: entorno de programación de flujo de datos generalizado para grafos dirigidos.
- **ZooKeeper**: servicio de coordinación de alto rendimiento para aplicaciones distribuidas.

En relación con la arquitectura de Hadoop y de MapReduce, se puede consultar la documentación de E.Coppa [Hao], pero en resumen las capas y módulos de Hadoop quedan representados en la figura siguiente:



Donde:

a) Entorno MapReduce: capa de software que permite la ejecución de aplicaciones bajo este paradigma.

b) YARN (*Yet Another Resource Negotiator*): es la parte responsable de gestionar los recursos (CPU, memoria, etc.) que utilizarán las aplicaciones; se compone de dos grandes módulos:

- *Resource Manager*, RM (uno por *cluster*), que actúa como supervisor y conoce dónde están ubicados los *workers* y de cuántos recursos dispone. Incluye

una serie de servicios, pero el más importante es el *Resource Scheduler* (RS), que decide cómo y a quién asignarlos.

- *Node Manager* (>1 por *cluster*) es el *worker* de la infraestructura y mantiene informado al *Resource Manager* sobre su estado; gestiona la memoria y *vcores*, que pueden ser asignados bajo la gestión de RS, que decidirá cómo se utiliza esta capacidad a través de fracciones de la capacidad del NM llamadas *containers*.

c) HDFS Federation: es la parte del entorno responsable de proveer almacenamiento permanente, fiable y distribuido; es utilizado para almacenar las entradas y salidas de la aplicación (no las intermedias). Se puede complementar con otras opciones de almacenamiento *cloud* (por ejemplo, S3, Google *Storage Objects*, etc.).

d) *Cluster*: es el conjunto de nodos de la infraestructura.

Es importante destacar que la infraestructura YARN y el HDFS están totalmente desacoplados; sobre ellos se podrían ejecutar otros entornos, aunque de momento es el único que está implementado.

En el ciclo de vida de la aplicación, gestionado por YARN, intervienen tres módulos: *job submitter* (cliente), *resource manager* (supervisor o máster) y *node manager* (*worker*), por lo cual el *workflow* de una aplicación sería:

1) el cliente envía la aplicación al RM,

2) este sobre la base de la información de RS,

3) asigna el contenedor,

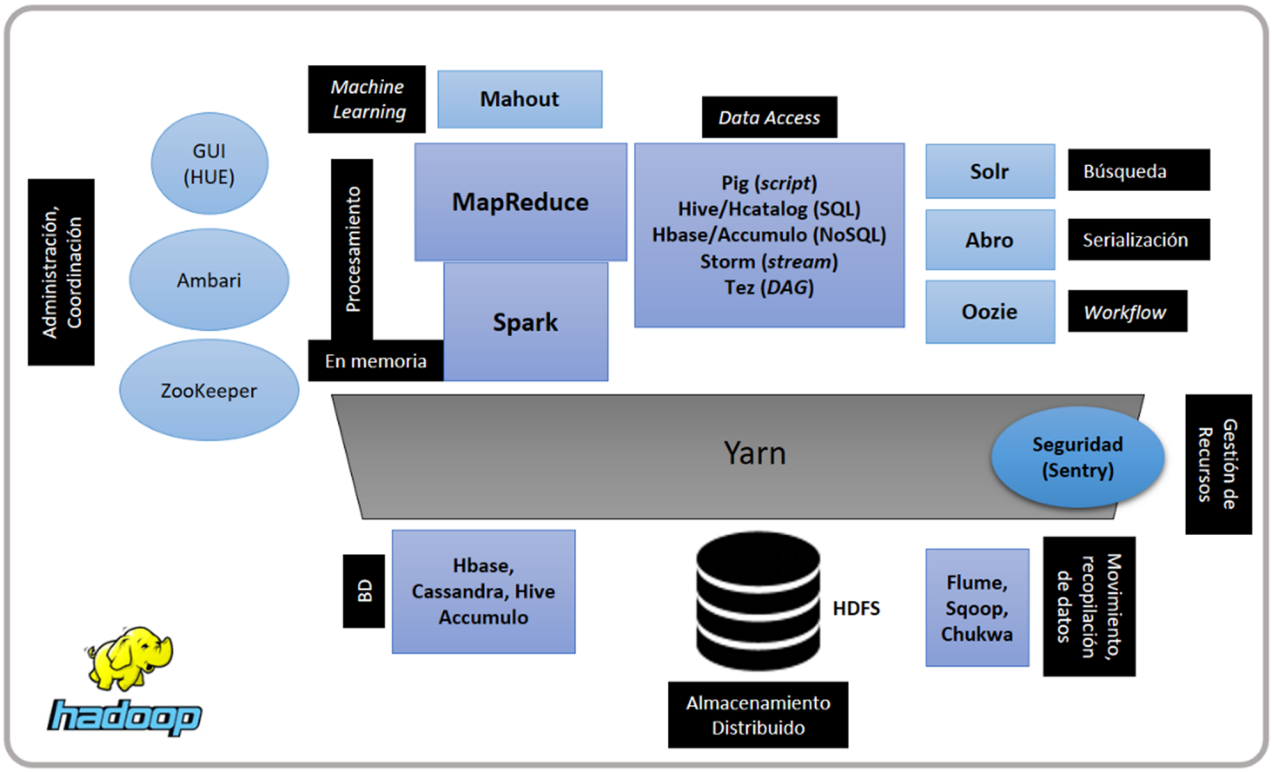
4) que además contacta con el *node manager*,

5) que a su vez lanza el contenedor

6) y este ejecuta la aplicación.

En la documentación [Hao] antes mencionada se recomienda analizar la anatomía de una aplicación MR y cómo interactúan cada una de sus fases con cada uno de los módulos, así como los detalles de cada uno de ellos, las tareas que se generan y la granularidad del modelo de computación derivado de submódulos del RM y NM [Ydo].

La figura siguiente muestra un entorno Hadoop con sus actores más importantes y su rol (uno de los posibles) dentro de la plataforma.



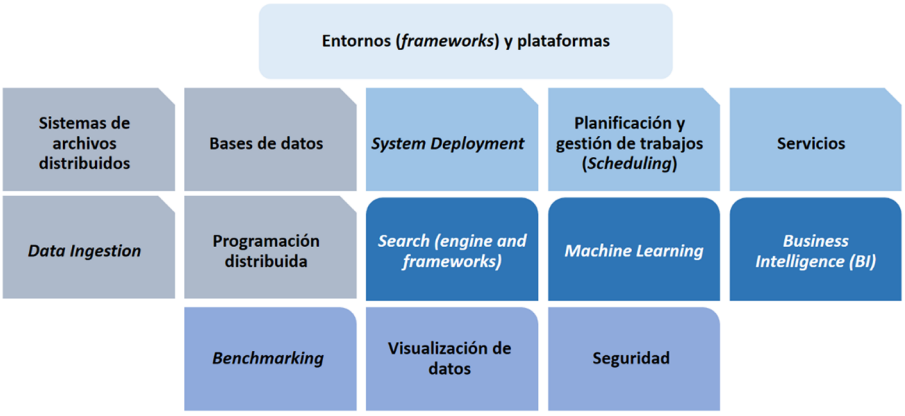
Los grandes proveedores de IaaS/PaaS/SaaS disponen de entornos basados en MR como Amazon EMP (Elastic MapReduce), Google Cloud DataProc, Azure HDInsigh, IBM Hadoop, o distribuciones específicas, entre las más importantes (informe de Forrester 2016 [Fdb]), Cloudera, HortonWorks, MapR o IBM-BigInsights. El informe Forrester menciona Pivotal HD, pero esta ha pasado (febrero de 2017) comoEnd of Availability (EoA), si bien existe alguna MV disponible en GitHub.

También es importante considerar que no todo el tratamiento del *big data* se centraliza sobre Hadoop, sino que existen en *cloud* otras herramientas como por ejemplo Azure Intelligence + Analytics, AWS Big Data, Gloogle Cloud Big Data Products, entre otras.

2. Herramientas

La «revolución» del *big data* ha generado una gran cantidad de métodos y herramientas (o evoluciones de las ya existentes) que han surgido en diversas categorías para gestionar volúmenes masivos de datos. Es muy interesante el mapa de las herramientas/servicios/entornos generado por FirstMark [Bds] y la extensa lista de AweSome-Bigdata [Awd], donde se puede encontrar **todo** (o una mayor parte) de lo que conforma el ecosistema del *big data* y *cloud computing*.

Sin ánimo de repetir el mapa o la lista, a continuación se muestra una gran división por categorías y se mencionan algunas de las más importantes/habituales (lista no exhaustiva y en su mayoría *open-source*, pero se incluyen algunas con licencias propietarias/*freemium*/uso no comercial).



2.1. Entornos (*frameworks*) y plataformas

- Apache Hadoop, entorno para el procesamiento distribuido que incluye MapReduce (*parallel processing*), YARN (*job scheduling*) y HDFS (*distributed file system*).
- Cloudera, plataforma Hadoop *open-source*.
- Hortonworks, HDP plataforma basada en Hadoop.
- MapR, plataforma para *big data* basada en Hadoop.
- Tigon, entorno para el procesamiento de altas prestaciones de flujos de datos (*stream*) en tiempo real que utiliza Hadoop y Hbase.
- Pachyderm, plataforma de almacenamiento construida sobre Docker y Kubernetes para proveer procesamiento de datos y análisis.
- Soluciones empresariales para *big data* (solo algunas de ellas): Amazon, Google, IBM Hadoop, Microsoft, Oracle, SAS Hadoop, TeraData, Vertica HP.

2.2. Sistemas de archivos distribuidos

- Apache HDFS, sistema para almacenar archivos de gran tamaño en múltiples máquinas.
- BeeGFS, Disco DDFS, Google GFS, Baidu File System, GlusterFS, Quantcast File System QFS, *distributed file systems*.
- Ceph Filesystem, sistema de archivos compatible Posix de tamaño ilimitado.
- GridGain, sistema de archivo *in-memory* compatible con Hadoop.
- Lustre file system, *distributed filesystem* de altas prestaciones.

2.3. Bases de datos

RDBMS

- MySQL/MariaDB, popular base de datos en sus versiones de Oracle (bajo licencia para algunos casos) y la versión *open-source* compatible y diseñada e implementada por los mismos desarrolladores de MyQSL.
- PostgreSQL BD, muy popular y con excelentes prestaciones *open-source*.

Document Data Model

- Crate Data, almacenamiento de datos masivos en tiempo real *open-source*.
- MongoDB, gestor de bases de datos NoSQL orientado a documentos.
- RavenDB, BD transaccional orientada a documentos *open-source*.
- RethinkDB, BD orientada a documentos para *realtime-web*.

Key Map/Value Data Model

- Aerospike, BD NoSQL *in-memory* y *open-source*.
- Apache Accumulo, almacenamiento de datos distribuido basados en *key/value* construidos sobre Hadoop.
- Apache Cassandra, almacenamiento de datos distribuido orientado a columna.
- Apache HBase, almacenamiento de datos distribuido orientado a columna.
- Baidu Tera, BD a *internet-scale*.
- Google BigTable, BD NoSQL orientado a columna de Google.
- Hypertable, almacenamiento orientado a columna *open-source*.
- InfiniDB utiliza una interface MySQL y *massive parallel processing* para paralelizar *queries*.
- Redis, almacenamiento *in memory* orientado a *key/value*.
- Tephra, transacciones para HBase.

Graph Data Model

- Apache Giraph, implementación de Pregel sobre Hadoop.
- Apache Spark Bagel, implementación de Pregel sobre Spark.

- DGraph, BD escalable y distribuida orientada a grafos.
- EliasDB, BD «liviana» orientada a grafos sin librerías de terceros.
- Google Cayley, BD orientada a grafos *open-source*.
- Google Pregel, entorno de procesamiento de grafos.

Columnar Databases

- MonetDB, DB orientada a columnas.
- Parquet, almacenamiento orientado a columnas para Hadoop.
- Google BigQuery, *datawarehouse* en *cloud* para *big data*.
- Amazon Redshift, almacenamiento basado en columnas para *cloud* de Amazon.
- IndexR, almacenamiento *open-source* basado en columnas para análisis en tiempo real de *big data*.

Procesamiento SQL-*like*

- Apache HCatalog, gestión del almacenamiento y tablas para Hadoop.
- Apache Hive, *datawarehouse SQL-like* para Hadoop.
- Apache Phoenix, capa SQL para HBase.
- Cloudera Impala, entorno para análisis interactivo.
- Spark Catalyst, entorno de optimización de *queries* para Spark y Shark.
- SparkSQL, entorno para utilizar datos estructurados sobre Spark.

2.4. *Data ingestion*

- Apache Chukwa, sistema de recolección de datos.
- Apache Flume, servicio para manejar gran cantidad de datos de *log*.
- Apache Kafka, sistema de mensajes de datos bajo método de *publish-subscribe*.
- Apache Sqoop, herramienta de transferencia de datos entre Hadoop y un repositorio de datos estructurado.
- Cloudera Morphlines, entorno para ayuda ETL a Solr, HBase yHDFS.
- Embulk, cargador de datos *open-source* que ayuda a la transferencia en BD, repositorios, formatos y servicios *cloud*.
- Fluentd, herramienta para recolectar eventos y *logs*.
- Heka, sistema de procesamiento *open-source* de *streams*.
- Logstash, herramienta para manejar eventos y *logs*.

2.5. Programación distribuida

- Apache Crunch, API Java para tareas como agregación de datos que son complejas de implementar en MapReduce.
- Apache Flink, entorno de procesamiento de *streams* distribuido y de altas prestaciones (Stratosphere).

- Apache MapReduce, modelo de programación para el procesamiento de datos masivos utilizando un algoritmo paralelo/distribuido sobre un *cluster/cloud*.
- Apache Pig, lenguaje de alto nivel para describir análisis de datos para Hadoop.
- Apache Spark, cómputo distribuido *in-memory*.
- Apache Storm, procesamiento de *streams* (de Twitter).
- Apache Samza, procesamiento de *streams* basados en Kafka & YARN.
- Concurrent Cascading, procesamiento/analítica de datos sobre Hadoop.
- Datasalt Pangool, entorno alternativo a MapReduce.
- Onyx, cómputo distribuido en el *cloud*.
- OpenMPI, entorno de programación por paso de mensajes.
- Pydoop, MapReduce y HDFS API para Hadoop en Python.
- Skale, procesamiento distribuido en NodeJS.

2.6. *System deployment*

- Apache Ambari, entorno para gestión de Hadoop.
- Apache Bigtop, entorno de desarrollo para Hadoop.
- Apache Helix, entorno de gestión de *clusters*.
- Apache Mesos, gestor de *clusters*.
- Apache Slider, aplicación en YARN para el despliegue de aplicaciones en YARN.
- Apache YARN, gestión de *clusters*.
- Apache Tez, entorno para la ejecución de grafos dirigidos complejos y construidos sobre YARN.
- Brooklyn, entorno que simplifica el despliegue y gestión de aplicaciones en *clusters*.
- Buildoop, similar a Apache BigTop.
- Cloudera HUE, entorno web para interactuar con Hadoop.
- Hortonworks HOYA, aplicación que permite desplegar un *cluster* HBase sobre YARN.
- Marathon, entorno Mesos para ejecuciones/servicios de larga duración.

2.7. Planificación y gestión de trabajos (*scheduling*)

- Apache Aurora, servicio de planificación que se ejecuta sobre Apache Mesos.
- Apache Falcon, entorno de gestión de datos.
- Apache Oozie, planificador de trabajos de *workflow*.

2.8. Servicios

- Apache Avro, sistema de serialización de datos.
- Apache Karaf, *runtime* OSGi (*Open Services Gateway Initiative*) que se ejecuta sobre cualquier entorno OSGi.
- Apache Thrift, entorno para construir protocolos binarios.

- Apache Zookeeper, servicio centralizado para la gestión de procesos.

2.9. *Machine learning*

- Cloudera Oryx, *machine learning* en tiempo real.
- convnetjs, *deep learning* en JavaScript.
- Deeplearning4j, *deep learning* para Java, Scala, Clojure.
- Mahout, librería de *machine learning* para Hadoop.
- MOA, *machine learning* sobre *big data stream* en tiempo real.
- PredictionIO, servidor de *machine learning server* sobre Hadoop, Mahout y Cascading.
- Spark MLlib, *machine learning* para Spark.
- WEKA, entorno de *machine learning*.

2.10. *Search* (*engine and frameworks*)

- Apache Lucene, librería de búsqueda.
- Apache Solr, plataforma de búsqueda basada en Apache Lucene.
- Elassandra, proyecto derivado de Elasticsearch sobre Apache Cassandra.
- ElasticSearch, plataforma de búsqueda y analítica basada en Apache Lucene.
- Sphinx Search Server, entorno de búsqueda de texto.

2.11. *Business intelligence* (BI)

- Qlik, plataforma de *business intelligence and analytics* (versión básica gratuita).
- Redash, SpagoBI, plataformas *open-source* para BI.
- Saiku, plataforma de análisis *open-source*.
- Jethrodata, plataforma interactiva de análisis de *big data*.

2.12. *Benchmarking*

- Apache Hadoop Benchmarking, test para analizar prestaciones de Hadoop.
- Berkeley SWIM Benchmark, ejemplos reales de *big data*.
- PUMA Benchmarking, conjunto de test para aplicaciones MapReduce.

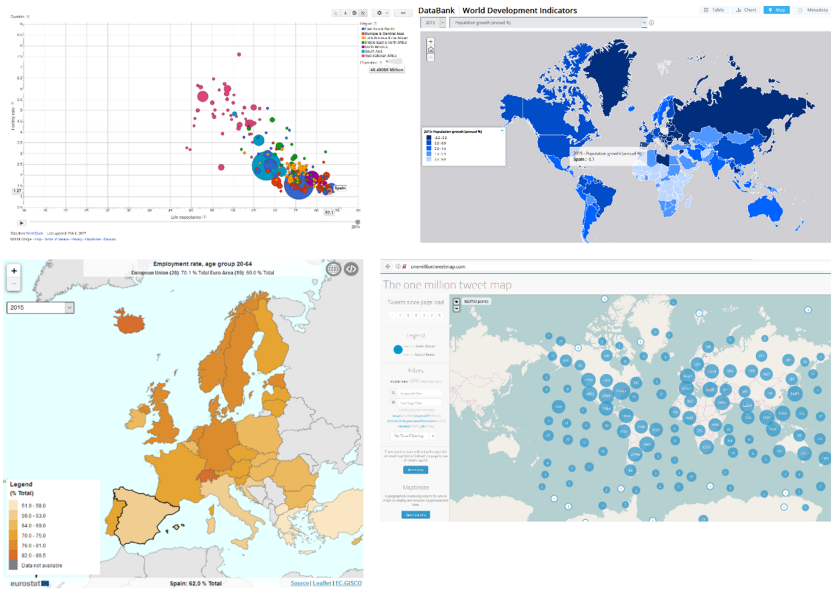
2.13. Seguridad

- Apache Eagle, monitorización en tiempo real.
- Apache Knox Gateway, acceso único seguro a *clusters* Hadoop.
- Apache Sentry, módulo de seguridad para datos almacenados en Hadoop.

2.14. Visualización de datos

- Librerías (solo algunos ejemplos): Bokeh, Chart.js, Chartist.js, Crossfilter, Cubism, DC.js, D3, D3Plus, D3.compose, DyGraphs, Envisionjs, Leaflet, NVD3, Plotly.js, Plot.ly, Sigma.js
- Plataformas: Freeboard, Fusion Tables (en línea), Gephi, Grafana, Graphite, Lumify, Mondrian, Redash

Ejemplos de cómo la visualización muestra el *big data* se pueden ver en las gráficas que presenta Google en un servicio lanzado en 2010 llamado Public Data Explorer para representar diferentes fuentes de datos abiertos (*open data*) como, por ejemplo, del World Bank, Eurostat y otras grandes fuentes de datos. A continuación, se muestran cuatro ejemplos de visualización de *big data* donde los tres primeros son la esperanza de vida procesada por Google, el crecimiento de la población mundial del WB y la tasa de empleo de Eurostat sobre datos preprocesados y sus respectivas interfaces, mientras que el último es un ejemplo de *big data* pero con procesamiento en tiempo real sobre los *tweets* emitidos en el mundo (con posibilidad de filtrarlos y saber de qué se está hablando a escala mundial), en el mapa de *The one million tweet map* de Maptimize.



Como ejemplos de *big data* se puede consultar:

- Amazon Public DataSets grandes (y famosos) *datasets* (por ejemplo, el genoma humano).
- AwesomePublicDatasets, referencia a los grandes repositorios de datos.
- Commoncrawl, información sobre 3.140 millones de páginas web y alrededor de 250 TiB (WebDataCommons con Hyperlink Graphs).

3. Distribuciones Hadoop y casos de uso

En este apartado, se mostrarán ejemplos de las distribuciones y procesamiento de datos masivos basado en Hadoop en diferentes tipos de instalaciones, ya sea desde el propio repositorio y/o desde alguna de las distribuciones de Hadoop de mayor renombre (según el informe *The Forrester Wave: Big Data Hadoop Distributions* de 2016 [Fdb]): Cloudera, HortonWorks, MapR. También se mostrará un caso de uso de *big data* sobre el *cloud* (Google) como representativo de la utilización de herramientas y tecnología como PaaS (o SaaS en algunos apartados), y un caso de *machine learning* (sobre Azure ML) como ejemplos de la evolución de los entornos para acercar el procesamiento de los datos a los usuarios sin tener que preocuparse por nada del resto.

3.1. Instalación de Hadoop sobre una MV KVM Ubuntu 16.04

En esta primera prueba se instalará Hadoop con HDFS en modo **pseudodistribuido** sobre una MV KVM (Ubuntu 16.04, 3 GB Ram, 15 GB de disco, pero se puede con máquinas más pequeñas y con `ssh-server`, `Lxde`, `Firefox`). Hadoop puede funcionar en tres modos: *local (standalone)*, *pseudo-distributed*, *fully-distributed*, donde en modo local se incluye en un proceso Java y es útil para depuración; en el modo pseudodistribuido, cada *daemon* es un proceso Java y se utiliza para tener la funcionalidad de Hadoop, pero sin disponer de la infraestructura correspondiente, ya que todo se ejecutará en un nodo, y finalmente la totalmente distribuida para máquinas en producción [Hui][Hsn]. Para ello:

1) Instalar y verificar la versión de Java.

```
sudo apt-get update
sudo apt-get install default-jdk
java -version
             openjdk version "1.8.0_121"
             OpenJDK Runtime Environment (build 1.8.0_121-8u121-b13-0ubuntu1.16.04.2-b13)
             OpenJDK 64-Bit Server VM (build 25.121-b13, mixed mode)
```

2) Crear un usuario, asociarlo a un grupo, insertarlo dentro de grupo *sudo* y verificar.

```
sudo addgroup hadoop
sudo adduser --ingroup hadoop hduser
groups hduser
                 hduser : Hadoop
sudo adduser hduser sudo
```

3) Cambiarse como *hduser* y crear las llaves (pública y privada) y verificar.

```
su - hduser
ssh-keygen
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
ssh localhost
```

Pedirá la confirmación del *fingerprint* y se podrá acceder sin *passwd*.

4) Descargar e instalar Hadoop (se puede utilizar cualquier repositorio).

```
wget http://apache.uvigo.es/hadoop/common/hadoop-2.7.3/hadoop-2.7.3.tar.gz
tar xvzf hadoop-2.7.3.tar.gz
mv hadoop-2.7.3 /usr/local/Hadoop
sudo chown -R hduser:hadoop /usr/local/hadoop
```

5) Modificar los archivos de configuración de Hadoop, que son:

```
~/.bashrc
/usr/local/hadoop/etc/hadoop/hadoop-env.sh
/usr/local/hadoop/etc/hadoop/core-site.xml
/usr/local/hadoop/etc/hadoop/mapred-site.xml.template
/usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

Antes de modificar el primero, obtener el directorio de instalación de Java y después editar el archivo.

```
update-alternatives --config java
... (providing /usr/bin/java): /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
```

6) Editar el archivo `~/.bashrc` y agregar al final:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-D java.library.path=$HADOOP_INSTALL/lib"
```

Y ejecutar a continuación `source ~/.bashrc`

7) Verificar.

```
javac -version
       javac 1.8.0_111
which javac
       /usr/bin/javac
readlink -f /usr/bin/javac
       /usr/lib/jvm/java-8-openjdk-amd64/bin/javac
```

8) Cambiar en */usr/local/hadoop/etc/hadoop/hadoop-env.sh* la variable JAVA_HOME con `export` *JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64*

9) Crear un directorio que Hadoop utilizará para guardar configuraciones y objetos durante su arranque y modificar el archivo *core-site*.

```
sudo mkdir -p /app/hadoop/tmp
sudo chown hduser:hadoop /app/hadoop/tmp
vi /usr/local/hadoop/etc/hadoop/core-site.xml
    <configuration>
       <property>
           <name>hadoop.tmp.dir</name>
           <value>/app/hadoop/tmp</value>
           <description>A base for other temporary directories.</description>
       </property>
       <property>
           <name>fs.default.name</name>
           <value>hdfs://localhost:54310</value>
           <description>The name of the default file system. A URI whose scheme and authority
            determine the FileSystem implementation. The uri's scheme determines the config
            property (fs.SCHEME.impl) naming the FileSystem implementation class. The uri's
            authority is used to determine the host, port, etc. for a filesystem.</description>
       </property>
    </configuration>
```

10) Configurar el entorno MapReduce.

```
cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template \
 /usr/local/hadoop/etc/hadoop/mapred-site.xml
vi /usr/local/hadoop/etc/hadoop/mapred-site.xml
    <configuration>
      <property>
          <name>mapred.job.tracker</name>
          <value>localhost:54311</value>
          <description>The host and port that the MapReduce job tracker runs at. If "local", then
           jobs are run in-process as a single map and reduce task.</description>
      </property>
    </configuration>
```

11) Configurar el HDFS (en cada *host* del clúster), que es especificar los directorios que serán utilizados por el *namenode* (pieza central de HDFS que guarda el árbol de todos los archivos sobre el sistema) y el *datanode* (almacena los datos el HDFS y está replicado en cada *host*) sobre aquel *host*.

```
sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode
sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode
sudo chown -R hduser:hadoop /usr/local/hadoop_store
vi /usr/local/hadoop/etc/hadoop/hdfs-site.xml
    <configuration>
      <property>
          <name>dfs.replication</name>
          <value>1</value>
          <description>Default block replication. The actual number of replications can be
           specified when the file is created. The default is used if replication is not specified
           in create time.</description>
      </property>
      <property>
         <name>dfs.namenode.name.dir</name>
         <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
      </property>
      <property>
          <name>dfs.datanode.data.dir</name>
          <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
      </property>
    </configuration>
```

12) Formatear el sistema de archivos HDFS que creará un directorio en */usr/local/hadoop_store/hdfs/namenode*:

```
hadoop namenode -format
```

Ir con cuidado con este comando, ya que este destruye todos los datos sobre el entorno Hadoop.

13) Para iniciar el entorno se utiliza el *script* `start-all.sh` (o también `start-dfs.sh` y `start-yarn.sh`) en */usr/local/hadoop/sbin*. Para iniciar el *DataNode daemon*:

```
/usr/local/hadoop/sbin/start-dfs.sh
```

Y se podrá consultar utilizando un navegador con la URL *http://localhost:50070/*

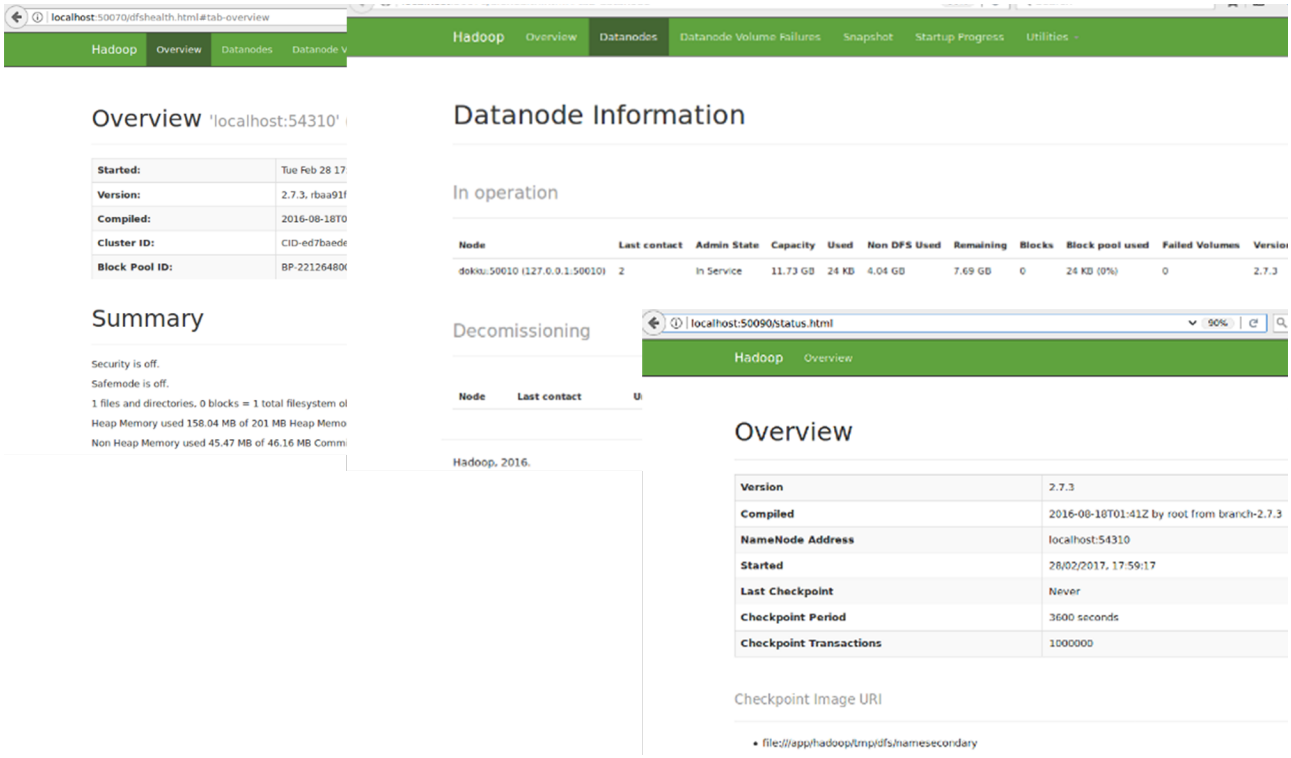
14) Para iniciar *ResourceManager daemon* y el *NodeManager daemon*:

```
/usr/local/hadoop/sbin/start-yarn.sh
```

Durante la ejecución se podrá ver una advertencia del estilo *Unable to load native-hadoop library...*, pero no nos debemos preocupar por ello; si se desea quitar, se deberá compilar el código fuente para tener las librerías adaptadas a 64bits para el entorno de trabajo. Para verificar que todo está ejecutándose, utilizar el comando `jps`.

15) Para detener la ejecución, `stop-all.sh` o (`stop-dfs.sh` y `stop-yarn.sh`).

16) Después de haber iniciado YARN se podrá ver en URL *http://localhost:50070/* las características del entorno, estado, el estado del *SecondaryNode* y los *logs*. Como se muestra a continuación.



17) Para probar el entorno ejecutaremos una aplicación de pruebas, que es el cálculo de Pi utilizando un método de Monte Carlo con MapReduce, y se visualizará este en el entorno del Resource Maganer. Para ello primero modificar */usr/local/hadoop/etc/hadoop/yarn-site.xml* agregando:

```
<configuration>
   <property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
   </property>
</configuration>
```

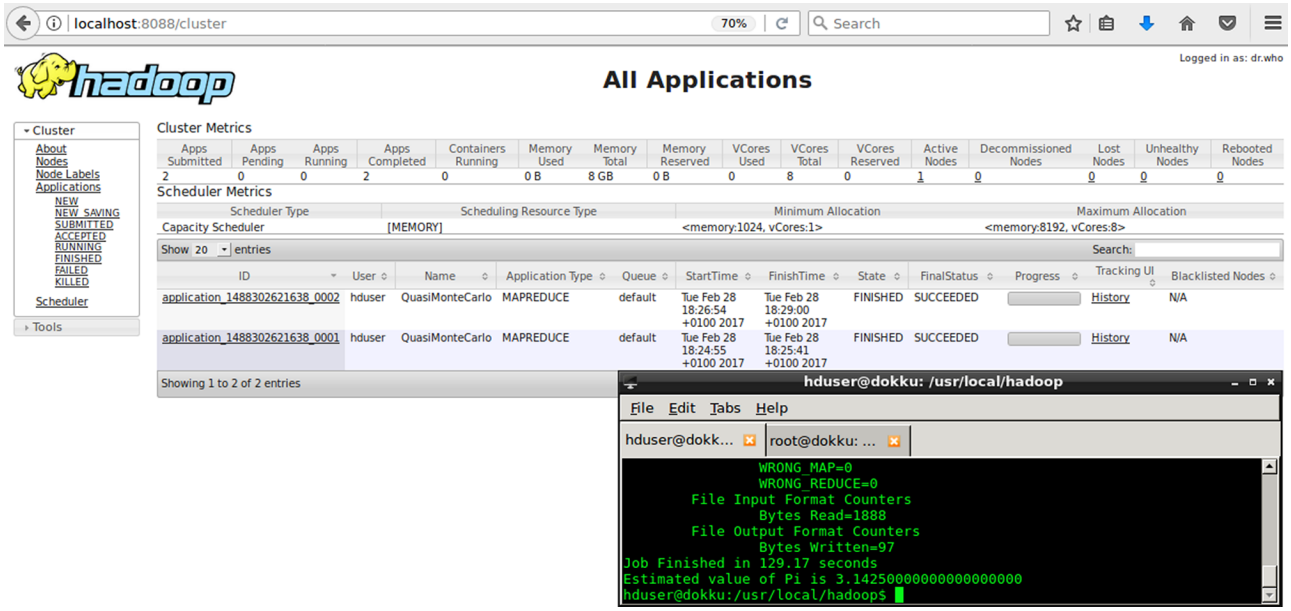
Y */usr/local/hadoop/etc/hadoop/mapred-site.xml* agregando:

```
<configuration>
   <property>
      <name>mapred.job.tracker</name>
      <value>localhost:54311</value>
   </property>
   <property>
      <name>mapreduce.framework.name</name>
      <value>yarn</value>
   </property>
</configuration>
```

Y parar y reiniciar los *daemons* a través de los *scripts*. Después ejecutar:

```
hadoop jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.5.jar pi 2 8
```

Donde 2 es el número de *maps* y 10 el número de muestra por *map* = 2, pero se puede utilizar otra combinación para tener mayor precisión (por ejemplo, 16 y 1000). A continuación, conectarse a la URL *http://localhost:8088/* y se podrá ver el estado de las tareas durante su ejecución, como se muestra en la figura siguiente.



Para completar esta prueba de concepto, se utilizará una aplicación que cuenta palabras de un texto, se generará la entrada en HDFS y se compilará y ejecutará la aplicación [Mrt].

Primero, se creará el archivo sobre el que se aplicará MR; para ello, se ha generado un texto del LoreIpsum y se ha guardado en el archivo *li.txt*, y luego se ha creado el directorio sobre el HDFS y se ha copiado el archivo (aunque se podría haber utilizado el comando *put* también) y verificado.

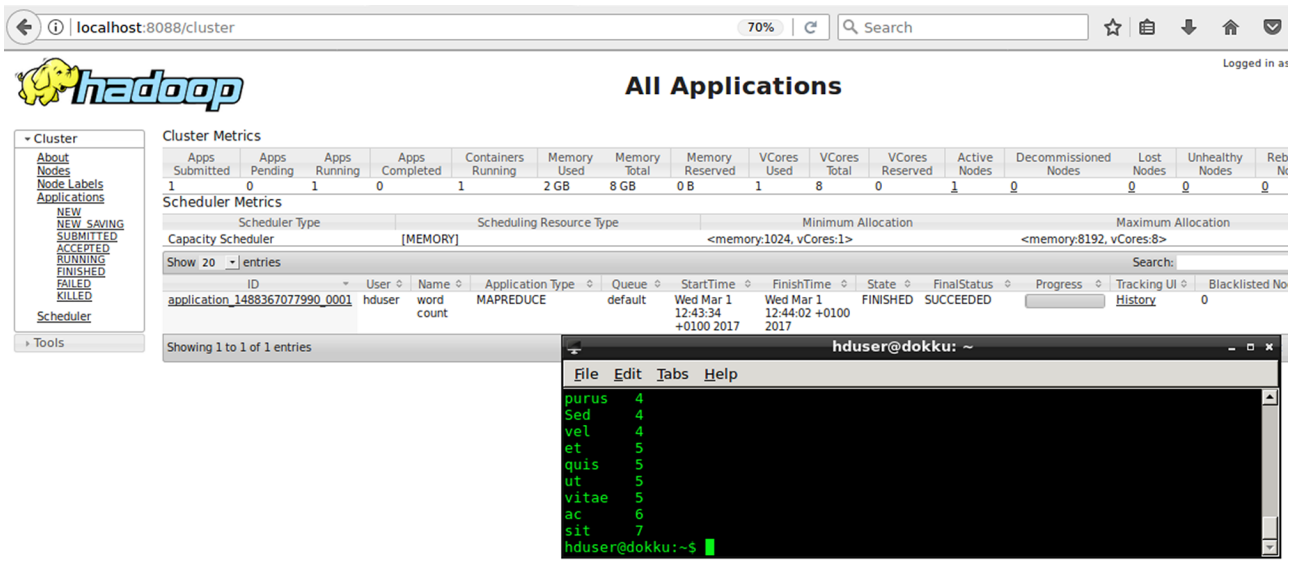
```
hdfs dfs -mkdir -p /user/hduser/in
hdfs dfs -copyFromLocal ./li.txt /user/hduser/li.txt
hdfs dfs -ls
hdfs dfs -cat /user/hduser/in/li.txt
```

A continuación, se ha creado el archivo WourdCount.java con el ejemplo de [Mrt], se han inicializado las variables, compilado, verificado que los archivos de entrada existen (utilizando *hadoop* en lugar de *hdfs*) y ejecutado.

```
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
hadoop com.sun.tools.javac.Main WordCount.java
jar cf wc.jar WordCount*.class
hadoop fs -ls /user/hduser/in/
hadoop fs -cat /user/hduser/in/li.txt
hadoop jar wc.jar WordCount /user/hduser/in /user/hduser/output
```

La imagen a continuación muestra la ejecución y la salida ordenada de los contadores de palabras después de la ejecución en el archivo del HDFS y visualizado con:

```
haddop fs -cat /user/hduser/output/part-r-0000 | sort -k2
```



Si se desea experimentar, se podría complementar el experimento agregando otra MV y crear un clúster Hadoop sobre ellas. Para crear/instalar los paquetes de los repositorios es conveniente utilizar Apache BigTop, que permitirá instalar o crear los paquetes sin preocuparse por dependencias o versiones.

3.2. Cloudera (CDH)

CDH es una distribución *open-source* muy completa, probada y popular de Apache Hadoop que, junto con otros proyectos relacionados, ofrece la versatilidad de Hadoop en el almacenamiento escalable y cómputo distribuido junto con una interfaz de usuario basada en web, además de procesamiento por lotes unificado, búsqueda/SQL interactivo y controles de acceso basados en roles [Cdo].

La plataforma CDH ofrece:

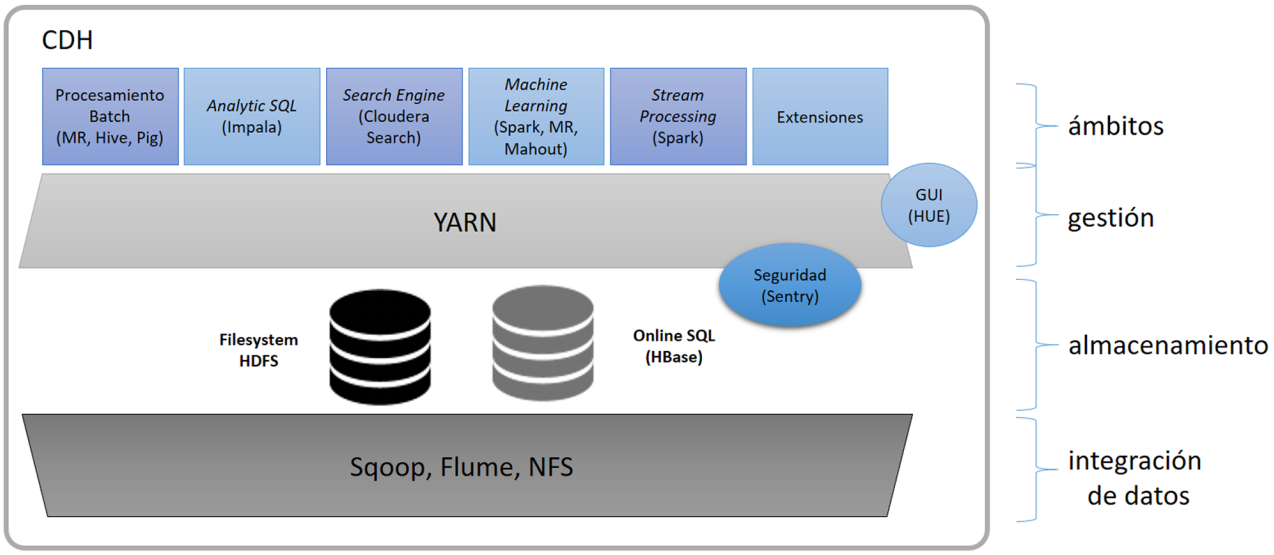
- flexibilidad (permite almacenar cualquier tipo de datos dentro de diversos entornos de computación como procesamiento por lotes, SQL interactivo, búsqueda de texto libre, aprendizaje automático y cálculo estadístico),
- integración (despliegue rápido de la plataforma Hadoop sobre una base de «lista para usar»),
- seguridad (se han cuidado todos los aspectos de integración y configuración para mantener la confidencialidad de los datos procesados),
- escalabilidad (permite integrar nuevos módulos/proyectos/aplicaciones sobre la misma base),
- alta disponibilidad (integrada y fácil de configurar) y
- compatibilidad (con los sistemas actuales de la infraestructura de TI existente).

Los componentes de CDH (v.5.8.x) son:

- Apache Hadoop: entorno de procesamiento/almacenamiento distribuido de *big data*.
- Apache HBase: BD (registro/tablas) que permite el almacenamiento escalable/distribuido para *big data*.
- Apache Impala: entorno para el procesamiento de consultas concurrentes SQL con baja latencia sobre HDFS, S3, o HBase.
- Apache Spark: procesamiento en memoria para acelerar la ejecución de los trabajos en *big data*.
- Apache Avro: serialización de datos (estructuras complejas, formato binario, RPC).
- Apache Flume: recolección/agregación de eventos/*streams* en tiempo real sobre HDFS.
- Apache Hive: entorno para lectura y escritura de *big data* en almacenamiento distribuido utilizando SQL.
- Apache Kafka: servicio de mensajería distribuido y basado en el paradigma *publish-subscribe*.
- HUE: GUI basada en web para la infraestructura Hadoop.
- Apache Oozie: planificación del *workflow* para la gestión eficiente de tareas Hadoop.
- Cloudera Search: búsqueda de texto libre (estilo Google Search) sobre Hadoop.

- Apache Sentry: control de acceso (granular) basado en roles para Hadoop.
- Apache Sqoop: transferencia de datos (en ráfagas) eficiente y escalable sobre HDFS.
- Apache ZooKeeper: servicio de coordinación fiable y distribuido utilizado en HBase.
- Apache Crunch: librería Java para escribir/depurar/ejecutar *pipelines* MR.
- Apache DataFu: librería para análisis estadístico de *big data*.
- Kite SDK: API, ejemplos y documentación para construir aplicaciones sobre Hadoop.
- Apache Parquet: representación de datos por columnas en forma comprimida/eficiente para Hadoop.
- Apache Mahout: librerías para ML (clasificación, agrupación y filtrado colaborativo).
- Apache Pig: entorno de lenguaje de alto nivel para análisis de *big data*.

La figura siguiente muestra un esquema de los principales módulos de CDH.



Además, Cloudera desarrolla el Cloudera Manager (no es *open-source*, pero se puede utilizar sin licencia), que es un entorno sofisticado para el despliegue, administración y monitorización de entornos CDH que, a través de un *dashboard* (web), permite que el *sysadmin* pueda tener todo el control sobre la infraestructura distribuida de Hadoop.

Esta plataforma también dispone de una API que permite obtener información del entorno o configurar el propio CM. Otro paquete (orientado a Cloudera Enterprise, versión con prestaciones adicionales bajo licencia del entorno Cloudera Hadoop) es Cloudera Director, que permite desplegar y gestionar la plataforma en el *cloud*.

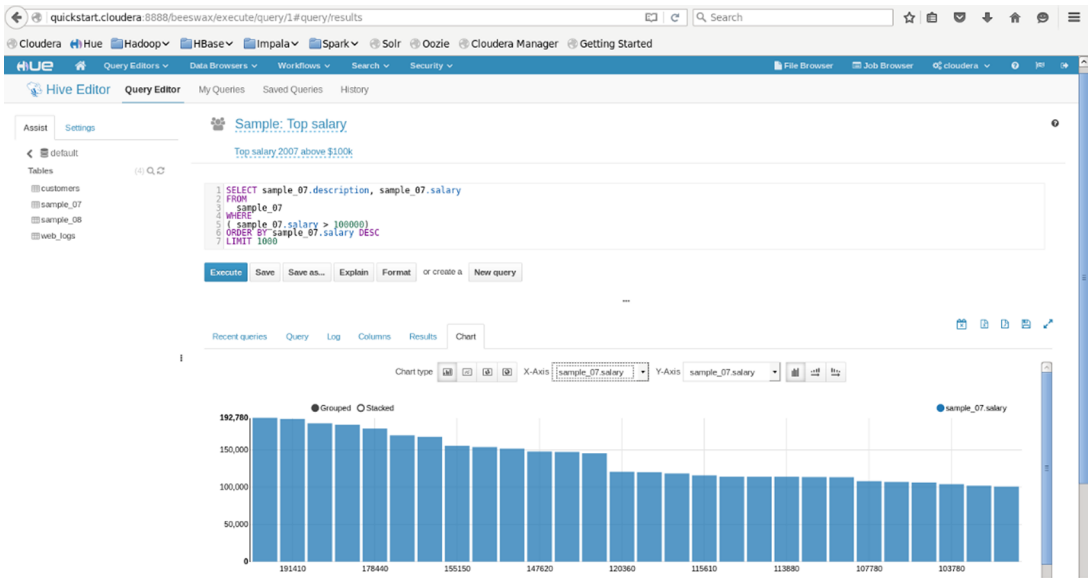
3.2.1. CDH sobre Virtualbox

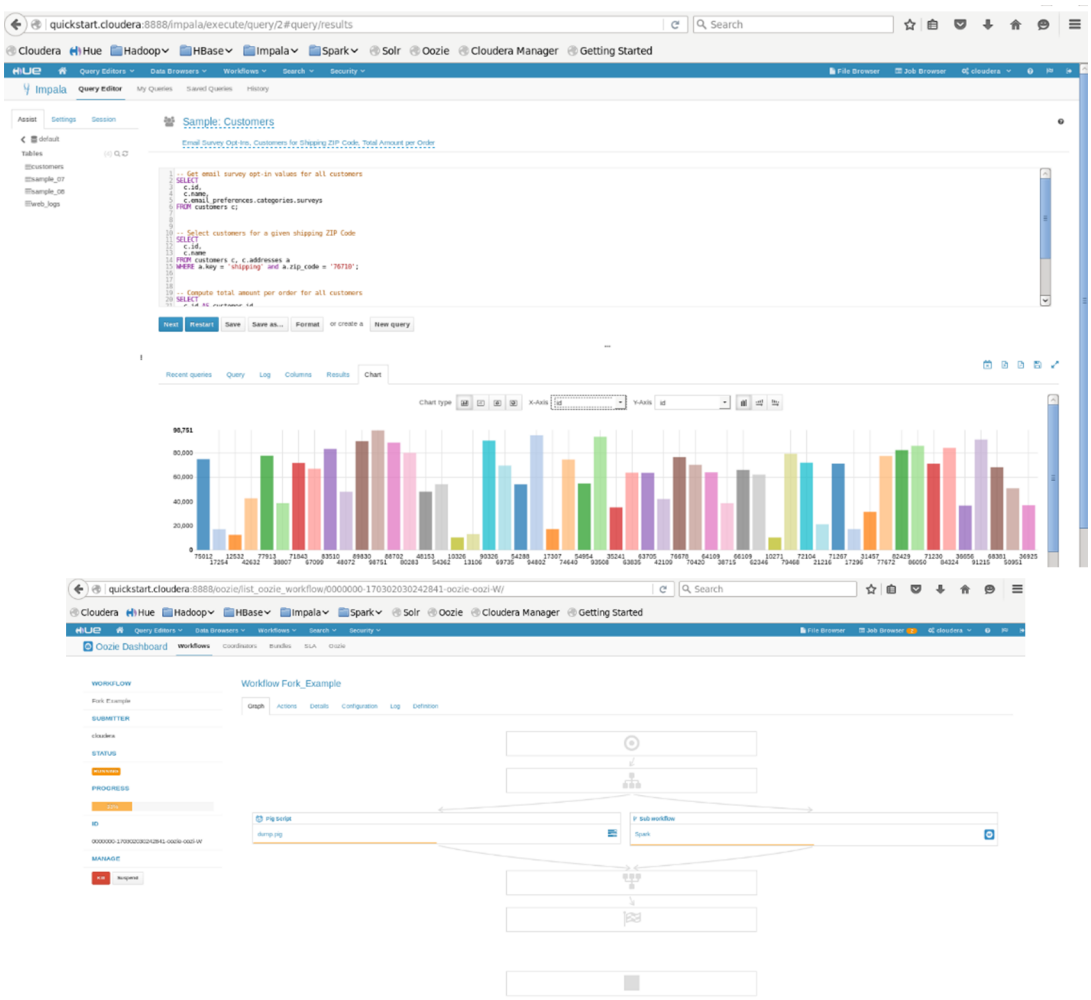
Para hacer una prueba de concepto con la distribución Cloudera, se descargará la MV (para este caso se ha utilizado la de VirtualBox, pero puede ser Docker, KVM, VMware), se descomprimirá el archivo e importará la MV (archivo OVF) y se arrancará la MV. Por defecto, se arrancará CDH con Hadoop (con servicios Linux funciona sobre 4G de RAM o incluso menos) en *pseudo-distributed* con un RManager y un nodo y se podrán ver sobre el navegador al inicio de la sesión.

Sobre el navegador se podrá acceder a Hue (por el *bookmark* o *http://quickstart.cloudera:8888/about/* con usuario y *passwd* **cloudera/cloudera**), donde primero verificará y mostrará la configuración y luego se podrán instalar los ejemplos que presenta el entorno *Step 2* (Hue → *About*) y analizar diferentes casos de usos (para ello, en el *home* de Hue (*http://quickstart.cloudera:8888/home2*) se verán los diferentes archivos en los que haciendo un doble clic se cargará el ejemplo y se podrá ejecutar).

En las siguientes figuras se muestra el procesamiento a través de Hive de unos datos sobre profesiones/salarios, otros sobre direcciones/mails de clientes a través de Impala, o se ejecuta un *workflow* con Ozzie y Spark como procesamiento.

En la documentación se pueden ver los detalles de cada aplicación y, a través de la interfaz de Hue, ver los trabajos y *logs* o acceder al RM de Hadoop y ver el estado de las tareas, como se ha visto anteriormente.





Una segunda forma de trabajar es migrar a Cloudera Manager, que se puede poner en marcha desde el escritorio ejecutando el enlace a Cloudera Express; sin embargo, dará un error si no se tiene la cantidad de memoria adecuada, ya que este necesita 8 GB y 2 VCPU. No obstante, para una prueba se puede ejecutar desde un terminal el comando forzando a que se ejecute y la versión *express* (podría ser la empresarial, pero solo por sesenta días de validez):

```
sudo /home/cloudera/cloudera-manager --force --express
```

Luego, accediendo a la dirección indicada en el terminal o la pestaña del navegador donde indica Cloudera Manager, se accederá a la pantalla de gestión donde se podrán tomar decisiones sobre cada servicio, ver el estado, arrancar/parar CDH/CM, agregar nodos, y todo un conjunto de opciones que permitirán rápidamente gestionar toda la infraestructura Hadoop. No obstante, es necesario tener en cuenta que depende mucho de la RAM disponible y por debajo de 6 GB muchos servicios generan advertencias por poca disponibilidad de memoria; además, el sistema se degrada de forma notable.

Para una prueba de funcionalidad se ha seguido el ejemplo dado en la plataforma (*tutorial 1*), donde primero se ha hecho la ingestión de los datos y luego se han procesado a través de una consulta.

En primer lugar, se debe traspasar los datos estructurados de RDBMS a HDFS utilizando Apache Sqoop (que permite cargar datos de MySQL en HDFS preservando su estructura), por lo cual desde un terminal del entorno ejecutar:

```
sqoop import-all-tables \
    -m {{cluster_data.worker_node_hostname.length}} --connect \
    jdbc:mysql://{{cluster_data.manager_node_hostname}}:3306/retail_db \
    --username=retail_dba \
    --password=cloudera \
    --compression-codec=snappy \
    --as-parquetfile \
    --warehouse-dir=/user/hive/warehouse \
    --hive-import
```

Esta propia carga tardará, ya que lanza tareas MapReduce para obtener los datos de MySQL y guardarlos en HDFS en formato Apache Parquet; con eso se crean las tablas en Hive con el mismo esquema de MySQL.

Parquet es un formato adecuado para el análisis en Hadoop que, en lugar de agrupar los datos en filas, los agrupa en columnas, lo cual es más adecuado para obtener relaciones entre los registros.

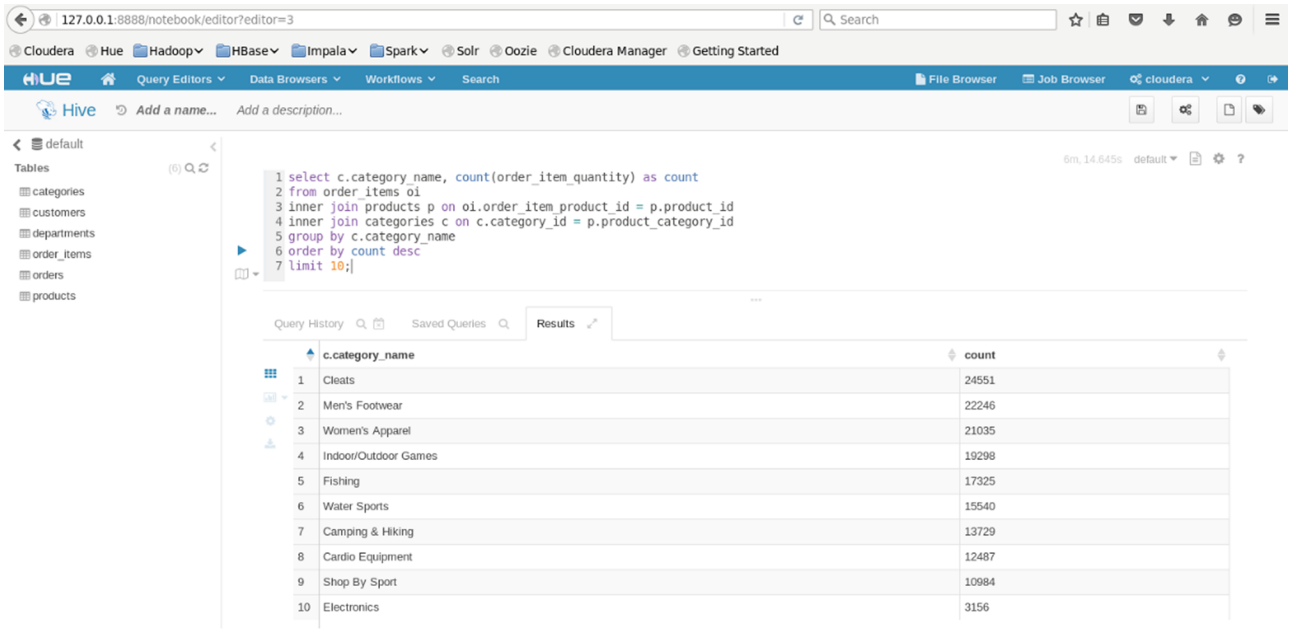
La comprobación de que se han creado los archivos se podrá hacer con:

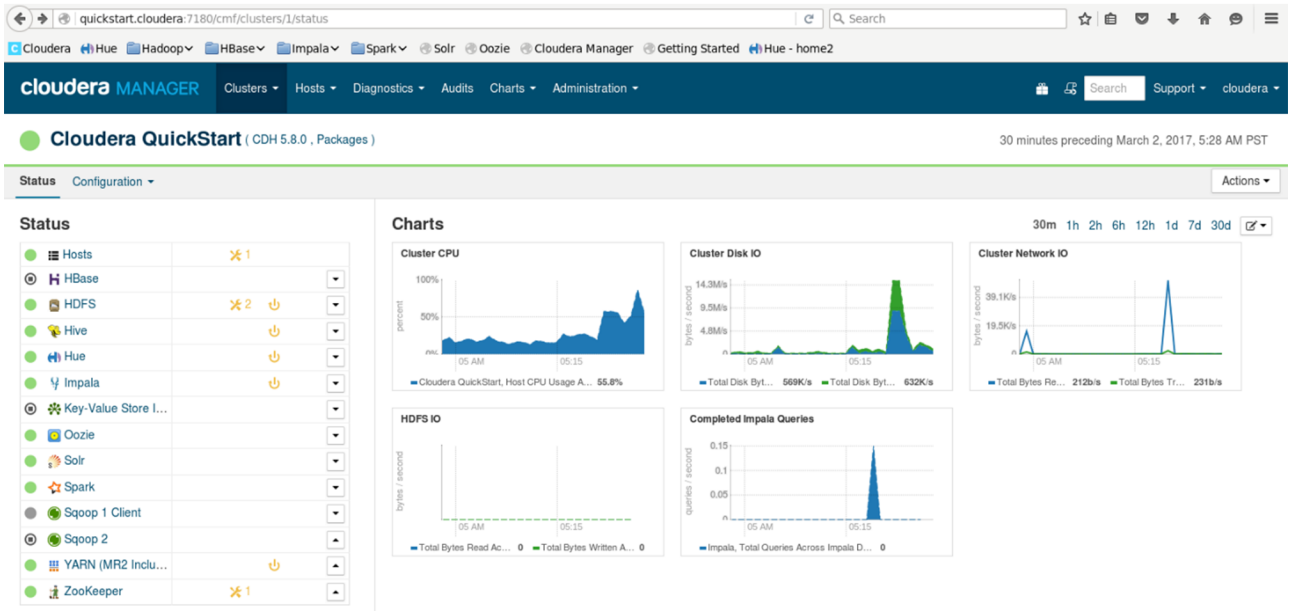
```
hadoop fs -ls /user/hive/warehouse/
hadoop fs -ls /user/hive/warehouse/categories/
```

Dentro de Hue se puede ir a *Query Editors* → *Hive* (en la documentación se hace a través de Impala, pero en este caso, y para reducir la memoria utilizada, se ha hecho a través de Hive), refrescar las bases de datos/tablas disponibles y escribir y ejecutar la siguiente consulta para determinar las diez categorías más populares de los productos:

```
select c.category_name, count(order_item_quantity) as count
from order_items oi
inner join products p on oi.order_item_product_id = p.product_id
inner join categories c on c.category_id = p.product_category_id
group by c.category_name
order by count desc
limit 10;
```

La imagen, a continuación, muestra la ejecución de la consulta a través de hacer clic en ▶ (cabe tener en cuenta que la interfaz en esta última versión ha cambiado y es diferente a la interfaz dada en la documentación) y la interfaz de CM donde se muestra el estado de la infraestructura.





En la documentación hay diferentes consultas u otros ejemplos para analizar la potencialidad de CDH + CM [Cma].

3.2.2. CDH sobre Docker

Otra forma de probar CDH es a través de la máquina Docker. Es importante tener en cuenta que, para esta prueba, se debe disponer de un espacio de memoria RAM libre superior a los 8 GB, ya que, sino, el clúster CDH no arrancará (ver recomendaciones al final de párrafo).

Para ello, se debe en primer lugar tener instalado Docker (en una versión superior a la 1.11) y correctamente configurado. En este caso, se ha instalado Docker CE (*community edition*) sobre Ubuntu 16.04 siguiendo las indicaciones del desarrollador.

Luego, se ha accedido al proyecto Clusterdock sobre el Hub de Docker, que permite desplegar un *cluster* multinodo sobre la misma Docker *host* (como requiere CDH para pruebas de concepto).

A diferencia de otras herramientas como Docker Compose (diseñado para manejar arquitectura con microservicios), Clusterdock despliega múltiples contenedores Docker que actuarán como máquinas tradicionales; por ejemplo, en una arquitectura Apache Hadoop de cuatro nodos, este utilizará cuatro contenedores. Como se mencionó anteriormente, esta prueba requiere gran cantidad de memoria, y los desarrolladores recomiendan como mínimo 16 GB de RAM libre para dos nodos o 24 GB para cuatro nodos.

Clusterdock (en sí mismo) está empaquetado en los contenedores junto con CDH para simplificar su ejecución y despliegue; se accederá a ellos a través de un *script* `clusterdock.sh` ejecutando alguna de las funciones internas como *clusterdock_run* o *clusterdock_ssh*. Para ello, se utiliza el comando `source` para cargar el *script* y luego se podrán ejecutar los comandos mencionados; en este ejemplo hay un clúster CDH de dos nodos:

```
source /dev/stdin <<< "$(curl -sL http://tiny.cloudera.com/clusterdock.sh)"
clusterdock_run ./bin/start_cluster cdh
```

Este comando descargará dos contenedores del Docker Hub que tienen la distribución completa de Cloudera Manager/CDH; se pondrá en marcha el *cluster* CDH a través de una red *bridged*, y con ello se modificará también el */etc/host* para adecuarlo a los contenedores.

Una vez validados los servicios CDH, se podrá acceder al *cluster* a través de Cloudera Manager (la URL y el puerto serán dados al final del despliegue, y el usuario y *passwd* para acceder en esta versión es **admin/admin**), o también se puede acceder por SSH directamente a los nodos utilizando la función *clusterdock_ssh* con el nombre FDQ del nodo (por ejemplo, *node-1.cluster*).

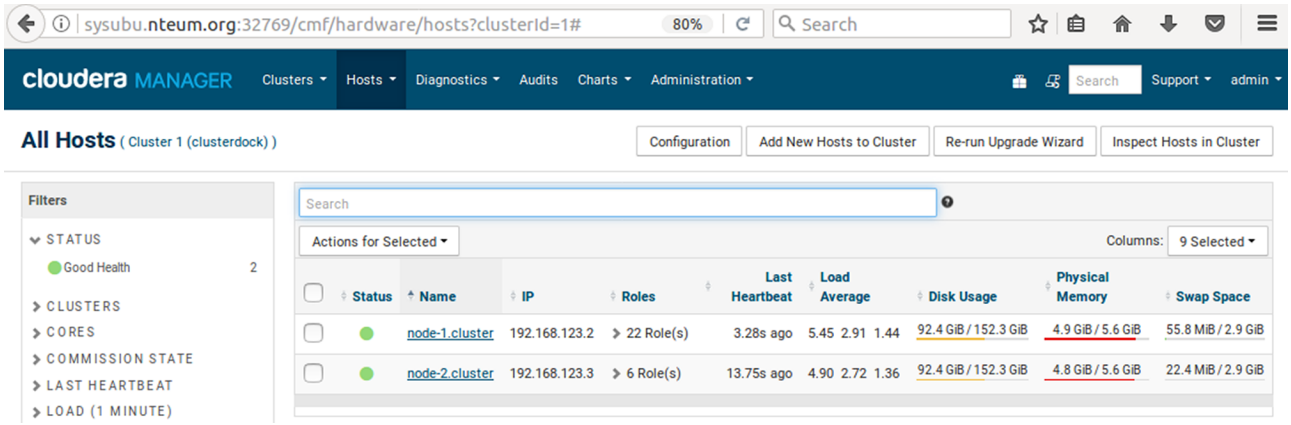
Es muy importante tener en cuenta las restricciones de memoria para probar este apartado ya que, si bien se instalará todo, habrá dificultades cuando inicie el clúster CDH y no será posible arrancarlo por este motivo (mostrará un error como la figura siguiente).

```
INFO:clusterdock.topologies.cdh.actions:Detected Cloudera Manager server after 123.17 seconds.
INFO:clusterdock.topologies.cdh.actions:CM server is now accessible at http://sysubu.nteum.org:32769
INFO:clusterdock.topologies.cdh.cm:Detected CM API v13.
INFO:clusterdock.topologies.cdh.cm_utils:Updating database configurations...
INFO:clusterdock.topologies.cdh.cm:Updating NameNode references in Hive metastore...
INFO:clusterdock.topologies.cdh.actions:Once its service starts, Hue server will be accessible at http://sysubu.nteum.org:3
2768
INFO:clusterdock.topologies.cdh.actions:Deploying client configuration...
INFO:clusterdock.topologies.cdh.actions:Starting cluster...
Traceback (most recent call last):
  File "./bin/start_cluster", line 70, in <module>
    main()
  File "./bin/start_cluster", line 63, in main
    actions.start(args)
  File "/root/clusterdock/clusterdock/topologies/cdh/actions.py", line 154, in start
    raise Exception('Failed to start cluster.')
Exception: Failed to start cluster.
```

Si igualmente se desean hacer las pruebas con una cantidad de memoria inferior a lo sugerido, se puede poner en marcha la infraestructura sin arrancar el *cluster* y después hacerlo de forma controlada dentro del CM.

```
clusterdock_run ./bin/start_cluster cdh -dont-start-cluster
```

La imagen siguiente muestra un despliegue realizado con Docker (sin iniciar los servicios de Hadoop) y dos nodos donde se pueden ver los requerimientos de memoria a través de CM.



3.3. HortonWorks

La plataforma *open-source* de datos Hortonworks (HDP) está basada en Apache Hadoop, es escalable y permite almacenar, procesar y analizar grandes volúmenes de datos de diferentes fuentes y formatos de una forma rápida, fácil y rentable. Esta integra diferentes proyectos de Apache Software Foundation (ASF) que se enfocan en el almacenamiento y procesamiento de *big data* como Hadoop, HDFS, YARN, Ambari, Falcon, Flume, HBase, Hive, Kafka, Knox, Oozie, Ranger, Slider, Spark, Sqoop, Storm, Tez y ZooKeeper.

La empresa contribuye a su desarrollo de forma activa en alguno de estos proyectos y, a diferencia de otras plataformas que tienen como base Apache Hadoop, Hortonworks aporta todo el código producido a la ASF, por lo cual HDP tiene licencia Apache y es completamente abierta. El modelo de negocio se basa en servicios, despliegue por expertos y capacitación donde la tecnología es libre y de código abierto [Hdp].

Para probar esta distribución el desarrollador provee un Sandbox, que es un entorno de aprendizaje sencillo y preconfigurado que contiene los últimos desarrollos de las herramientas relacionadas con Apache Big Data, que se puede ejecutar en la nube (Azure) o en una máquina personal (VirtualBox, VMware, Docker), lo cual permite experimentar y explorar el entorno HDP.

Se debe tener en cuenta que esta MV necesita 8 GB de RAM libre; con menos la máquina se degrada de forma notable.

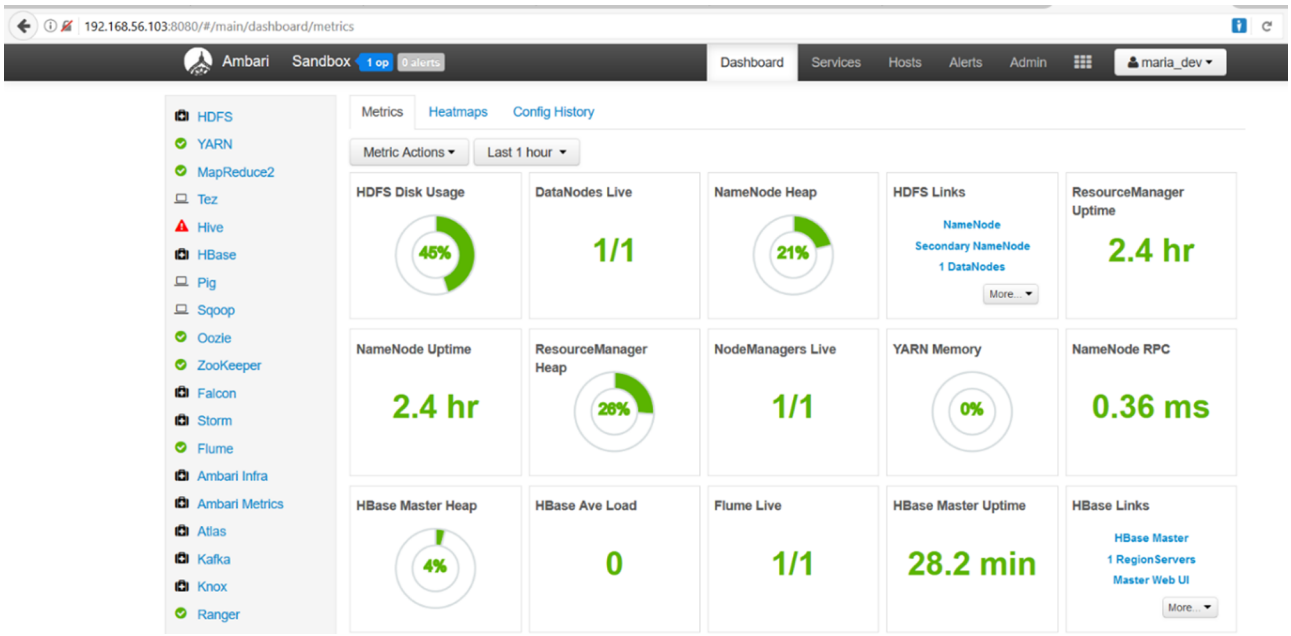
Para hacer una prueba funcional se ha utilizado una máquina con 8 GB de RAM y se ha dejado para la MV en Virtualbox 6 GB, y después de una serie de advertencias se ha podido arrancar.

El acceso a los servicios se puede hacer a través de la *IP 127.0.0.1:puerto* sobre un navegador en el *host*, ya que la MV tiene habilitado el *port-forwarding* para la interfaz NAT de un gran conjunto de puertos (ver en *MV* → *Network* → *Adaptador1* → *Advanced* → *Port Forwarding*). También, si se prefiere, se puede agregar un segundo adaptador configurado como *HostOnly* sobre Virtualbox (IP 192.168.56.100), ya que la MV tiene un dispositivo de red configurado en 192.168.56.103 (se debe hacer un `ifup` cuando se arranque la máquina, pues este no se levanta durante el *boot*). Cualquiera de los dos métodos puede ser utilizado para acceder a la interfaz gráfica a través de un navegador.

Después de arrancada la MV, se puede acceder a través de la consola donde el usuario/*passwd* es **root/hadoop**, o se puede acceder de muchas formas tal como indica la guía de Hortonworks.

Dentro del sistema se deberá ejecutar `ip address` para ver los dispositivos de red (uno de ellos estará en NAT; por lo tanto, tendrá IP=10.0.2.15) y levantar el otro disponible con `ifup <dispositivo>`. Luego, desde el *host* se podrá acceder a la URL *http://192.168.56.103:8888* y, a través de los *quick links* o la URL *http://192.168.56.103:8080*, al *dashboard* (usuario/*passwd* **maria_dev/maria_dev** o consultar otros en la guía antes mencionada). Si se accede por el *port-forwarding*, reemplazar la IP 192.168.56.103 por 127.0.0.1 en la URL.

La figura siguiente muestra el *dashboard* del Sandbox (con muchos de los servicios funcionando –y otros que necesitan atención– y con alguna alerta –por ejemplo, Hive– por la –reducida– cantidad de memoria).



Para hacer una prueba de concepto sobre la infraestructura Hortonworks, se ha utilizado el Sandbox sobre una máquina de 16 GB de RAM, cargando un conjunto de datos de los desarrolladores y analizando la potencialidad de la distribución. Sobre la MV se han seguido los mismos pasos (con un interfaz *HostOnly*) antes descritos para acceder a la interfaz de HDP con un navegador sobre el *host*, pero también se puede acceder utilizando la IP *127.0.0.1:puerto* sobre el *host* utilizando el *port-forwarding* que tiene habilitado la MV sobre la interfaz NAT.

Para el análisis se han descargado los datos *Geolocalization.zip* de la guía *Getting Started with HDP*, donde se utilizan datos de una empresa de transportes que incluyen vehículos, dispositivos y personas que se mueven a través de la geografía de un país y donde la empresa deseavincular la información de ubicación con sus datos analíticos y analizar el riesgo que puede representar para los trabajadores diversas rutas/horas de conducción y vincularlo a otros parámetros.

En este ejemplo solo se han cargado los datos en HDFS y, posteriormente, se ha realizado un análisis con Hive, pero en el tutorial mencionado se explica cómo calcular el factor de riesgo de los conductores utilizando Pig, un análisis con Spark y visualización con Zeppelin.

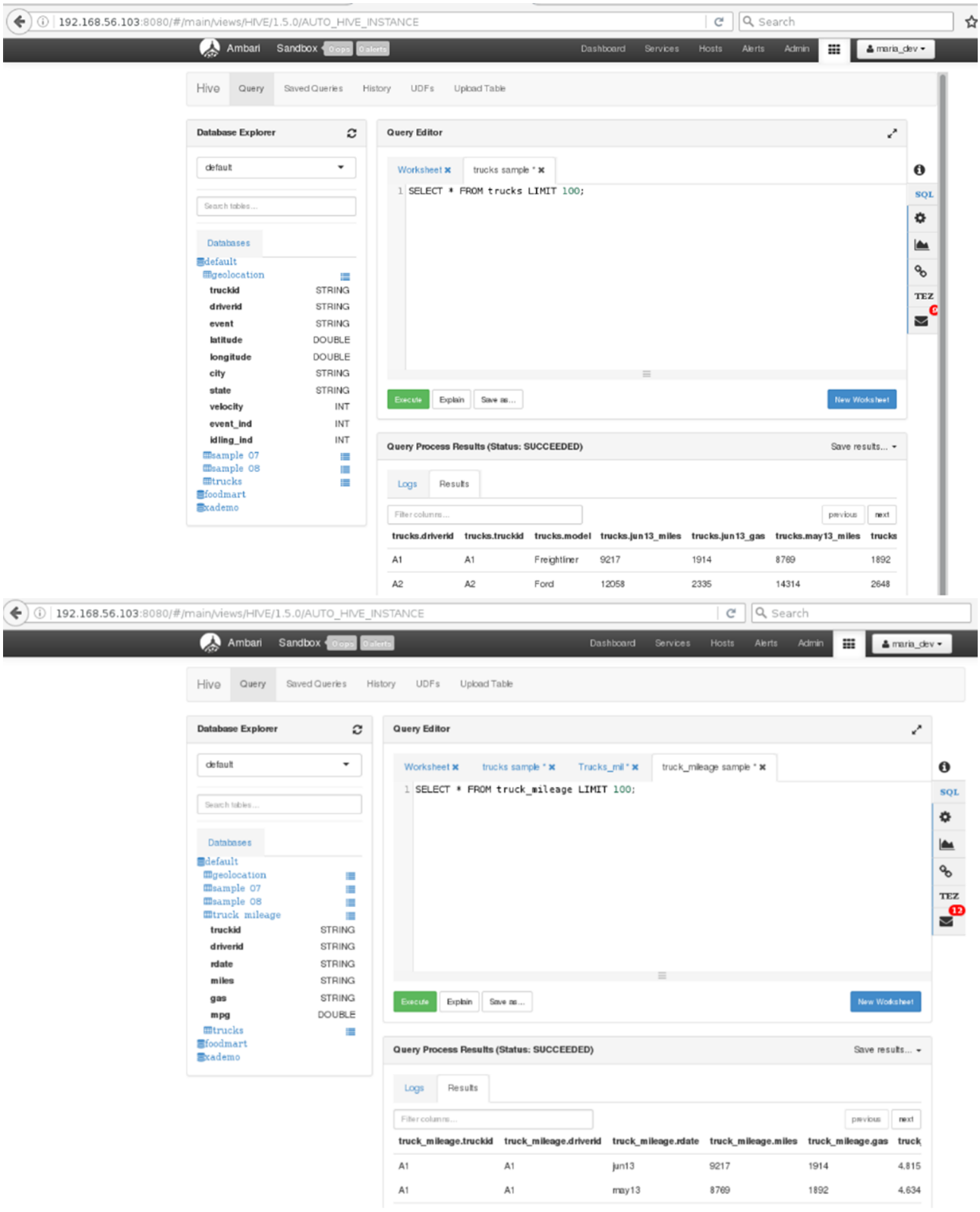
Para este mínimo ejemplo, se han seguidos los pasos indicados en *Loading Sensor Data into HDFS* y posteriormente Data manipulation with Hive.

Como muestra la gráfica, a continuación se ha ejecutado una consulta sobre la tabla *trucks*; con ello se ha mostrado el resultado de los cien primeros registros y luego se ha creado una nueva tabla:

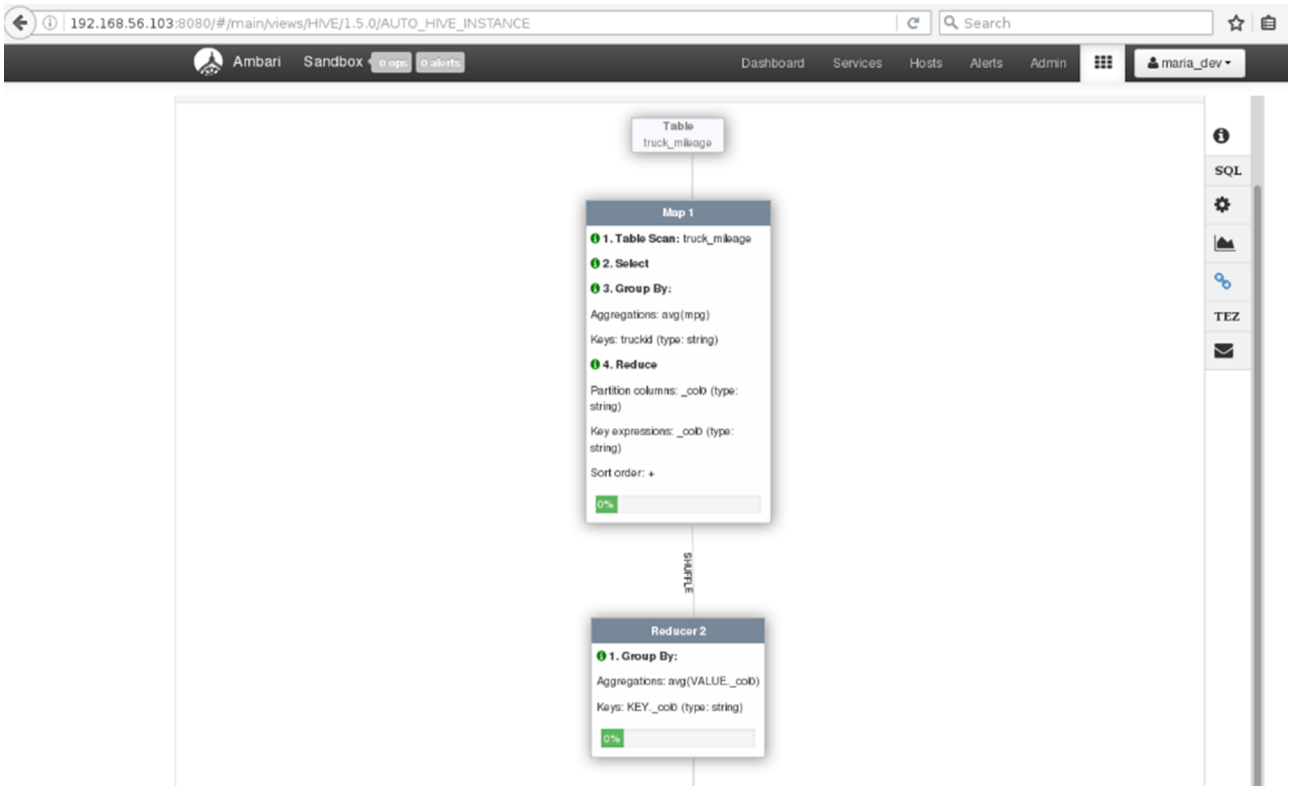
```
CREATE TABLE truck_mileage STORED AS ORC AS SELECT truckid, driverid, rdate, miles, gas, miles /
```

```
    gas mpg FROM trucks LATERAL VIEW stack(....) dummyalias AS rdate, miles, gas;
```

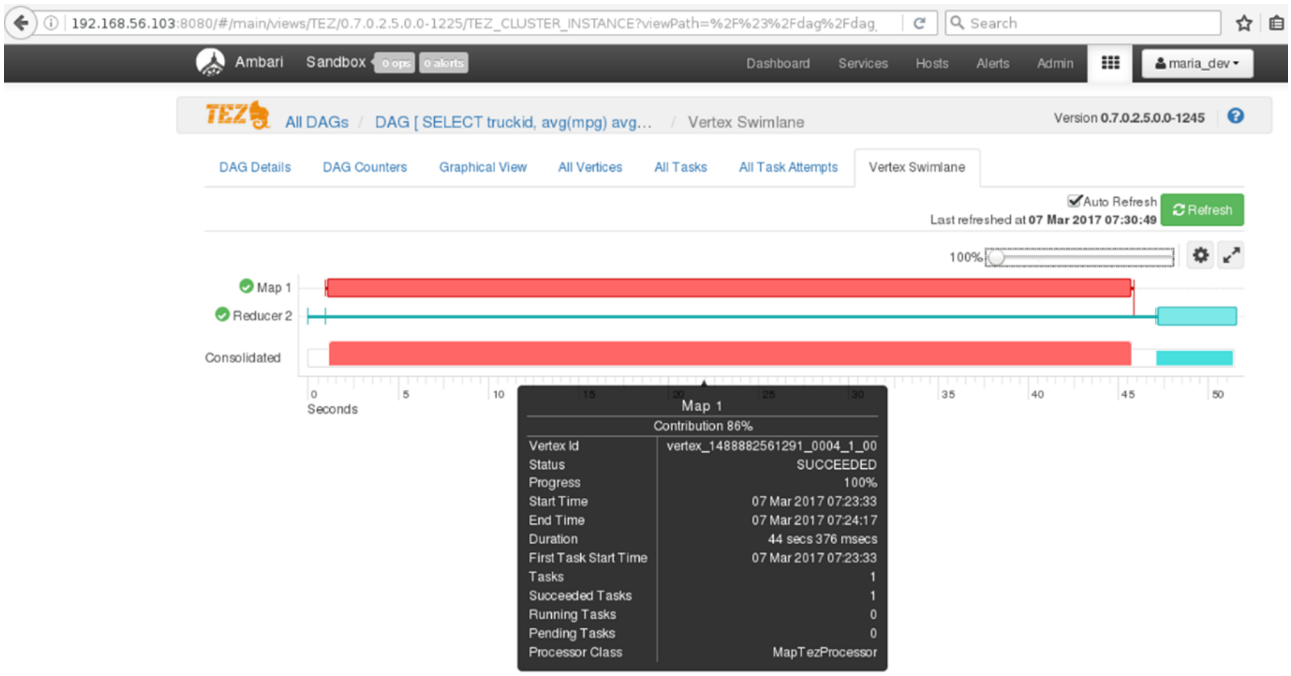
para después generar consulta para mostrar las millas/galón de combustible de cada camión, como se muestra en la segunda imagen.



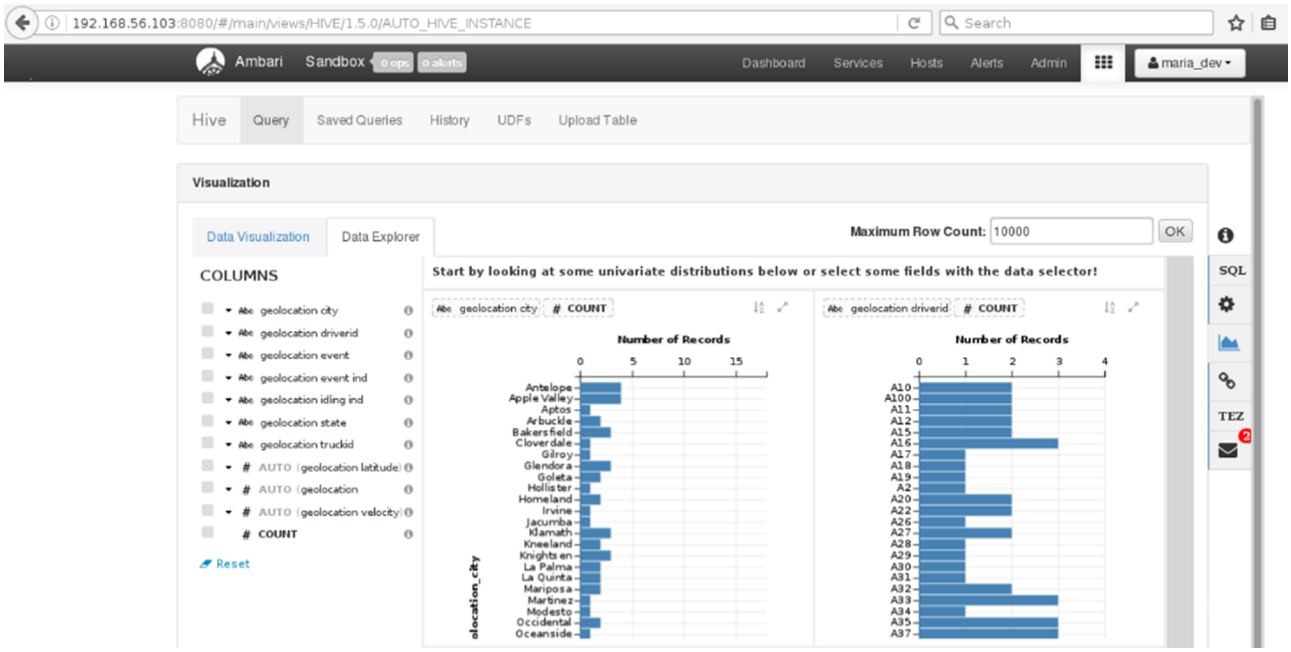
Como parte de la experiencia se ha accedido al «*Explain*», que permite conocer cómo se realiza la consulta y qué fases intervienen visualizando esta de modo gráfico, como muestra la siguiente figura.



A continuación, a través de Tez, se han visualizado los detalles de la ejecución de la consulta anterior, obteniendo información sobre cada fase de MapReduce con información detallada de tiempos y detalles sobre la ejecución.



Finalmente, se ha accedido a la visualización de los datos mediante Hive donde, a través del Data Explorer, se ha podido observar la distribución de los datos y acceder fácilmente a los datos procesados por la *Query*, como se puede apreciar en la imagen siguiente.



Como se puede observar (y solo se ha visto un mínimo ejemplo), la potencialidad de HDP es muy grande, al igual que las posibilidades que representa para el tratamiento de grandes conjuntos de datos. Es por ello que se recomienda hacer pruebas y experimentar con ellas (sobre una máquina con la memoria adecuada), ya que es una plataforma muy completa y con una funcionalidad avanzada, además de tener una alta integración con todo el entorno Hadoop.

3.4. MapR

MapR Converged Data Platform, es una plataforma de ámbito empresarial que permite aplicar los conocimientos analíticos a procesos operativos en tiempo real para crear conocimiento sobre grandes conjuntos de datos.

Según la visión del desarrollador, es una plataforma que permite analizar la convergencia de segmentos/productos históricamente separados para generar nuevos valores de forma simple y fácil con una infraestructura de datos segura y abierta, que reduce drásticamente el TCO, y analizar aplicaciones basadas en datos en tiempo real.

En su diseño, MapR ha incorporado los criterios más importantes que se deben aplicar en un centro de datos de gran tamaño que utilizan Hadoop/Spark, que deben tener un funcionamiento 24/7 y bajo diferentes escenarios, como son el análisis por lotes o en tiempo real y la consulta interactiva. Esto ha permitido una evolución de la arquitectura de la plataforma, que mantiene la compatibilidad binaria con el sistema de archivos distribuido Apache Hadoop

(HDFS) para garantizar la compatibilidad, pero que además incluye MapR-FS, que es un sistema de archivos distribuido, con notables prestaciones dada su capacidad de acceder directamente al hardware de almacenamiento.

Desde el punto de vista estructural, MapR se diferencia de otras distribuciones de Hadoop, que requieren clústeres separados para múltiples aplicaciones, ya que permite el procesamiento de archivos distribuidos, tablas de base de datos y flujos de eventos en una única capa unificada. Esto facilita tener aplicaciones operativas (por ejemplo, HBase) y analíticas (por ejemplo, Apache Drill, Hive o Impala) en el mismo clúster, lo cual se traduce en una reducción de costos significativa.

Desde el punto de vista de la integración con el entorno Hadoop y los proyectos de la AFS, MapR contribuye al ecosistema (sobre todo en el desarrollo de Apache Mahout y Apache Drill) y todas las aplicaciones pasan rigurosas pruebas de integración/ejecución antes de su adopción; las aportaciones propias y librerías también son publicadas en abierto a través de repositorios públicos (GitHub y Maven) [Wmr].

MapR Converged Data Platform está disponible en dos ediciones *Converged Community Edition*, para utilización libre (con restricciones especificadas en la *End User License Agreement* -EULA-), y la versión *Converged Enterprise Edition*, para desarrollos de servicios críticos que requieren continuidad de negocio (ver diferencias).

Su instalación no presenta dificultades y está automatizada a través de un *script* y posterior instalación a través de una página web (si bien es necesario disponer de por lo menos dos nodos con unos requerimientos muy específicos) [Mrd].

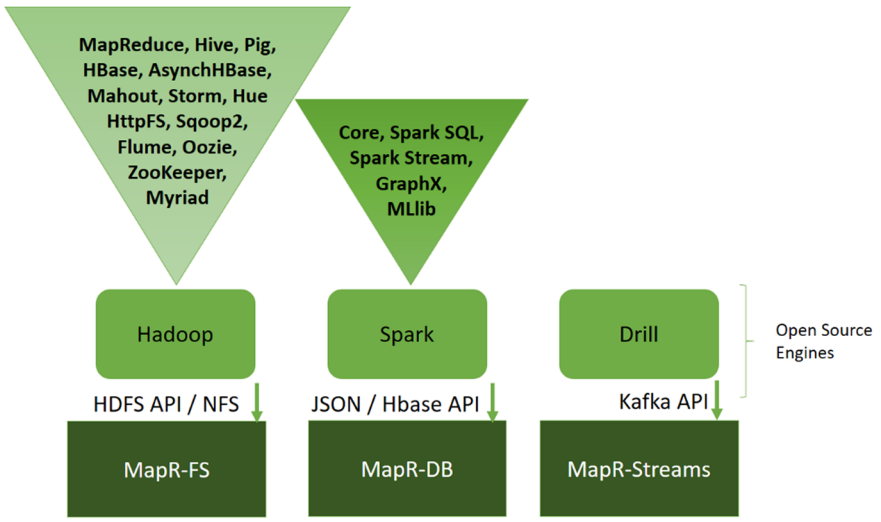
Para experimentar con MapR, los desarrolladores proveen un Sandbox para VMware o Virtualbox, si bien es necesario disponer de 20 GB de disco disponibles, 4 *cores* de 64bits –mínimo a 1.3+ GHz con soporte VT-x– y 8 GB de RAM libres.

Los analistas en BI o desarrolladores interesados en MR y Apache Drill pueden utilizar otro Sandbox específicamente diseñado para tal fin. MapR 5.2 Sandbox provee tutoriales y aplicaciones de demostración con interfaces basadas en web que permiten, rápidamente, conocer y experimentar con la plataforma, ya que ofrece una instancia de MapR totalmente funcional que se ejecuta sobre una MV. Existe una gran cantidad de cursos gratuitos del desarrollador en MapR Academy [Mra], que permiten conocer en detalle, además de cuestiones generales sobre el *big data* y Hadoop, la plataforma, su administración, cada uno de los módulos y cómo experimentar con ellos.

La plataforma incorpora tres servicios –MapR-FS, MapR-DB y MapR Streams– que forman un núcleo unificado en la arquitectura MapR-CDP y son los que proveen de alta disponibilidad, acceso en tiempo real, seguridad unificada, *multi-tenancy*, espacio de nombres global, gestión y monitorización.

Sobre ellos se pueden instalar *open-source engines* (como Hadoop, Spark, Drill) u otras privadas (como Vertica, Sap, MySQL).

MapR 5.2 Sandbox incluye Apache Hadoop (MapReduce, Hive, Pig, HBase, AsynchHBase, Mahout, Storm, HttpFS, Sqoop/Sqoop2, Flume, Oozie, ZooKeeper, Hue, y Myriad), Apache Spark (Core, Spark SQL, GraphX, Spark Streaming, MLlib), MapR-DB (con soporte para tablas JSON y HBase), MapR Streams (con soporte para la API Apache Kafka) y MapR-FS. La figura siguiente muestra la imagen integrada de la MV [Mrd].



MapR Sandbox se distribuye como *appliance* OVA, por lo cual se debe importar desde Virtualbox; en este caso, el desarrollador aconseja utilizar el *port-forwarding* que se ha hecho sobre la interfaz NAT hacia el *host*, por lo cual, para acceder a los servicios/interfaz sobre el *host*, hacer *http://127.0.0.1:8443* (o utilizar *localhost* en lugar de 127.0.0.1).

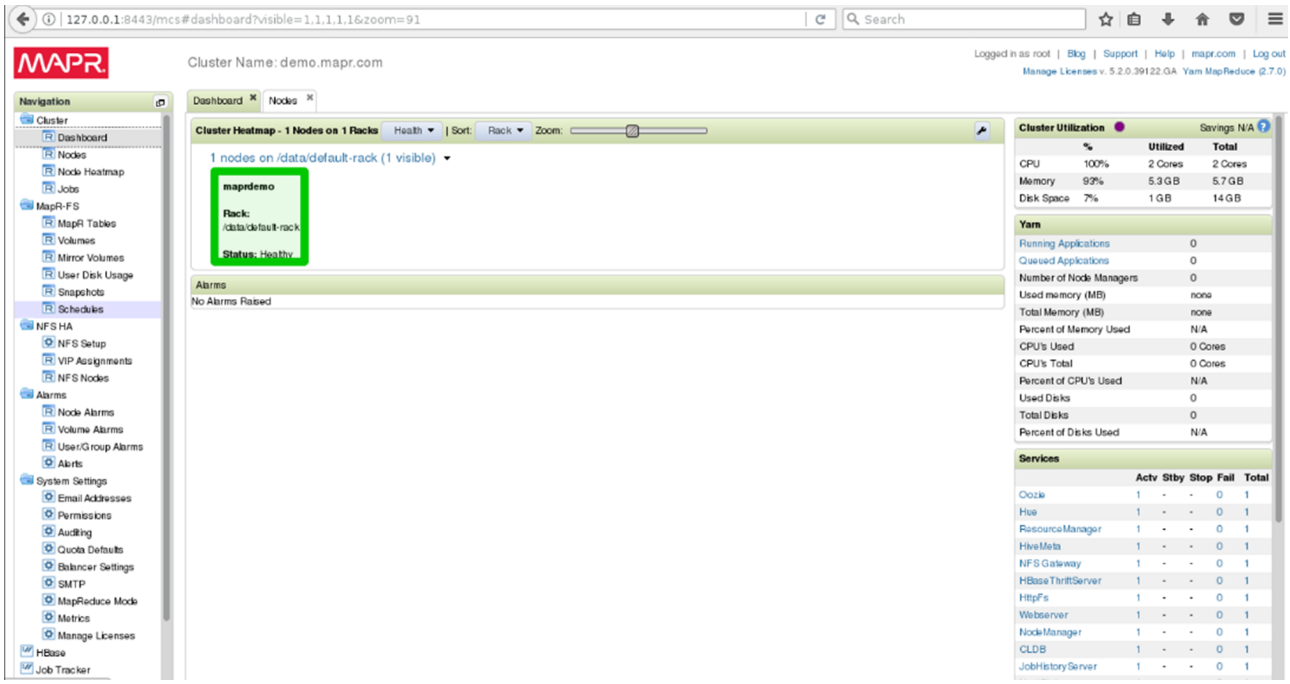
También se puede acceder desde un cliente de SSH ejecutando (el servicio SHH se ha mapeado sobre el puerto 2222 del *host* para no interferir con el propio servicio SSH de este) con usuario/*passwd* **mapr/mapr** (el *passwd* para el usuario *root* también es **mapr**).

```
ssh mapr@localhost -p 2222
```

Cuando se accede a la URL *http://127.0.0.1:8443* del *host* (cabe recordar que la interfaz NAT de la MV tiene un *port-forwarding* hacia el *localhost* del *host*), se podrá acceder a los dos roles definidos:

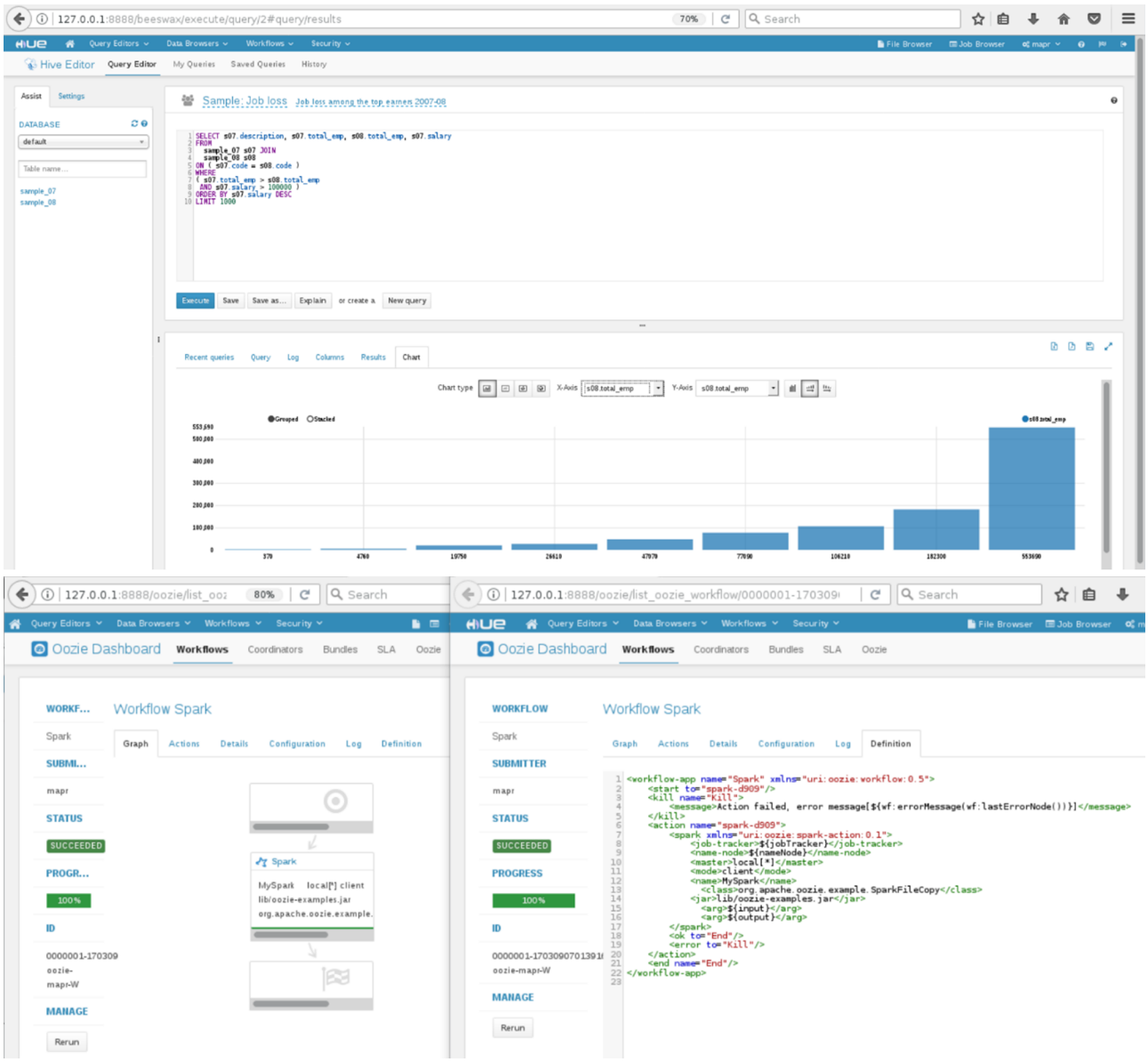
1) Desarrolladores y analistas: orientado a los desarrolladores de Hadoop o analistas de datos que desean experimentar o trabajar con Hadoop y MapR utilizando los ejemplos incluidos o cargando nuevos datos y procesándolos. El acceso será a Hue con usuario/*passwd* **mapr**.

2) Administradores: perfil de administrador de infraestructura que permite acceder a MapR (MCS), lo que permitirá configurar, monitorizar y gestionar el *cluster* a través de una interfaz web (usuario/*passwd* **root/mapr**), como la mostrada en la figura siguiente.



Es interesante, para ganar conocimiento en la plataforma y las herramientas, seguir los tutoriales que ha desarrollado MapR para conocer el entorno y las herramientas.

Para hacer una prueba funcional, y con los datos ya integrados en MapR, se ha ejecutado una consulta SQL sobre Hive para obtener la cantidad de trabajos perdidos entre los que más ganan y su representación gráfica y la ejecución de un *workflow* de copia de un archivo mediante Spark, como se puede apreciar en las figuras siguientes.



En esta última se puede ver la representación del *workflow* y su definición Oozie.

Como prueba funcional se compilará y ejecutará el programa WordCount, pero utilizando Spark y siguiendo las líneas de la guía *Getting Started with Spark on MapR Sandbox* y el código de Github. Para ello, primero en la MV MapR se ha instalado, necesario para compilar el proyecto, Apache Maven, que es una herramienta de gestión de proyectos de software y de comprensión que gestiona la compilación, elaboración de informes y la documentación de un proyecto a partir de un archivo de especificaciones llamado Project Object Model (POM) y que existe en el directorio de la aplicación como *pom.xml*.

Este archivo es el núcleo de la configuración del proyecto en Maven, que contiene la mayoría de la información requerida para construir el proyecto, como en el presente caso. Para instalar Maven sobre CentOS (que es el SO de la MV MapR), ejecutar como *root*:

```
cd /opt
wget http://www-eu.apache.org/dist/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.tar.gz
tar xzf apache-maven-3.3.9-bin.tar.gz
ln -s apache-maven-3.3.9 maven
vi /etc/profile.d/maven.sh
   export M2_HOME=/opt/maven
   export PATH=${M2_HOME}/bin:${PATH}
source /etc/profile.d/maven.sh
mvn -version
   Apache Maven 3.3.9
   Maven home: /opt/maven
   Java version: 1.7.0_79, vendor: Oracle Corporation
   ...
```

Ahora se deberá descargar la aplicación desde GitHub y compilar (esto tardará unos instantes ya que deben bajarse todos los paquetes necesarios para compilar):

```
cd /user/user01
yum install git
git clone https://github.com/caroljmcdonald/sparkwordcountapp wordcount
cd wordcount
mvn package
cp /user/user01/wordcount/target/sparkwordcount-1.0.jar /user/user01
```

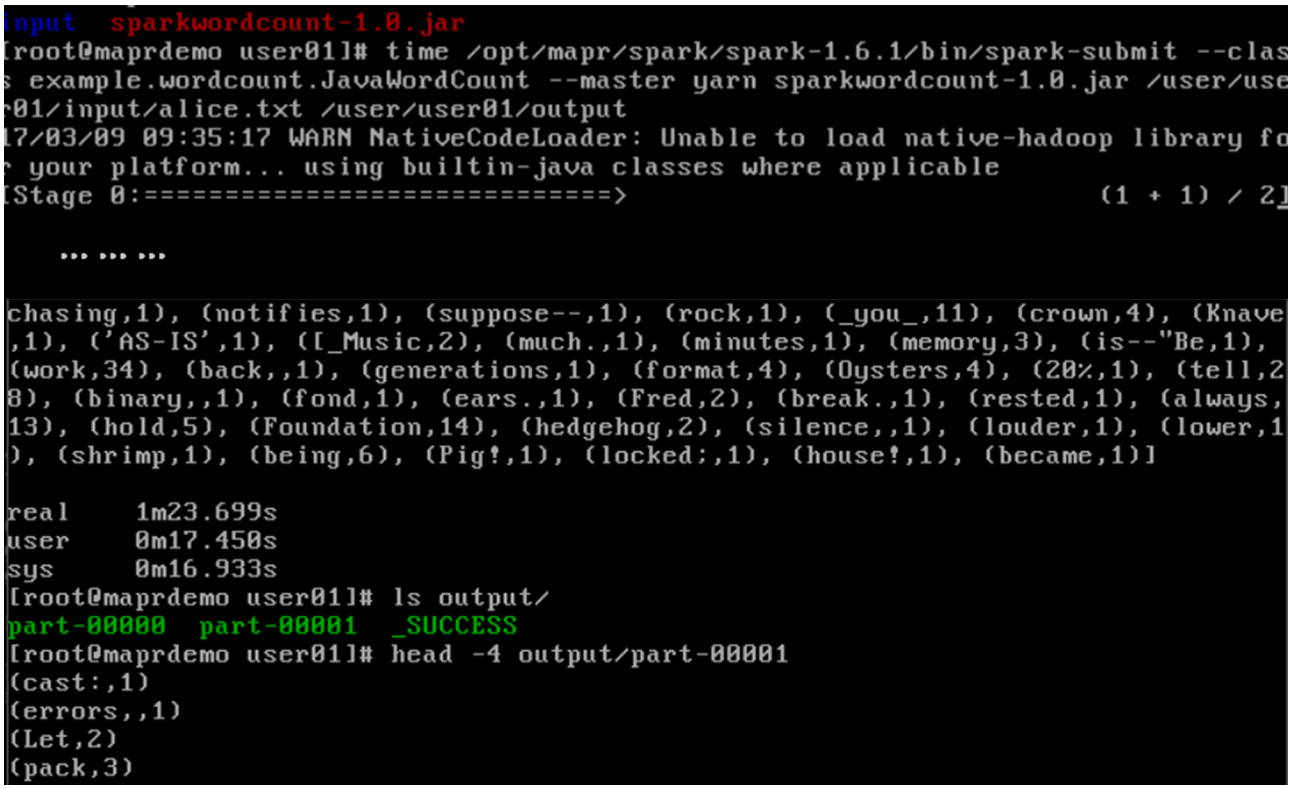
Para ejecutarlo, primero debemos tener un texto; se utilizará *Alice in Wonderland*, del proyecto Gutenberg:

```
mkdir /user/user01/input
cd /user/user01/input
wget -O alice.txt http://www.gutenberg.org/cache/epub/35688/pg35688.txt
```

Finalmente, se podrá ejecutar sobre Spark (verificar que la llamada a `spark-submit` esté toda en una línea):

```
cd /user/user01/
/opt/mapr/spark/spark-1.6.1/bin/spark-submit --class example.wordcount.JavaWordCount \
--master yarn sparkwordcount-1.0.jar /user/user01/input/alice.txt /user/user01/output
```

Las imágenes a continuación muestran la ejecución del comando y su finalización y visualización de los resultados (se ha antepuesto el comando `time` para generar el tiempo gastado en su ejecución y tener constancia de la reducción significativa que existe cuando se procesa por Spark).



Tal y como se mencionó para las plataformas anteriores, Mapr es una plataforma con una gran funcionalidad y con una visión un poco diferente a las otras plataformas, ya que incorpora MapR-FS como reemplazo a HDFS y, por lo tanto, puede incluir Pig, Hive y Sqoop sin dependencias de Java; provee acceso a NFS desde cualquier nodo, por lo cual se puede montar MapR *file system* sobre NFS permitiendo con ello a las aplicaciones acceder a datos de Hadoop de la forma tradicional; es, además, considerada por usuarios de renombre una de las plataforma con mejores prestaciones para el procesamiento de *big data*.

Es interesante el artículo [Gmr] de *Forbes* a raíz de la inversión de capital en MapR (2014) por parte de Google; en él se analizan los objetivos y mercados/negocios de cada una de tres grandes compañías tratadas en este apartado que tienen como base Hadoop.

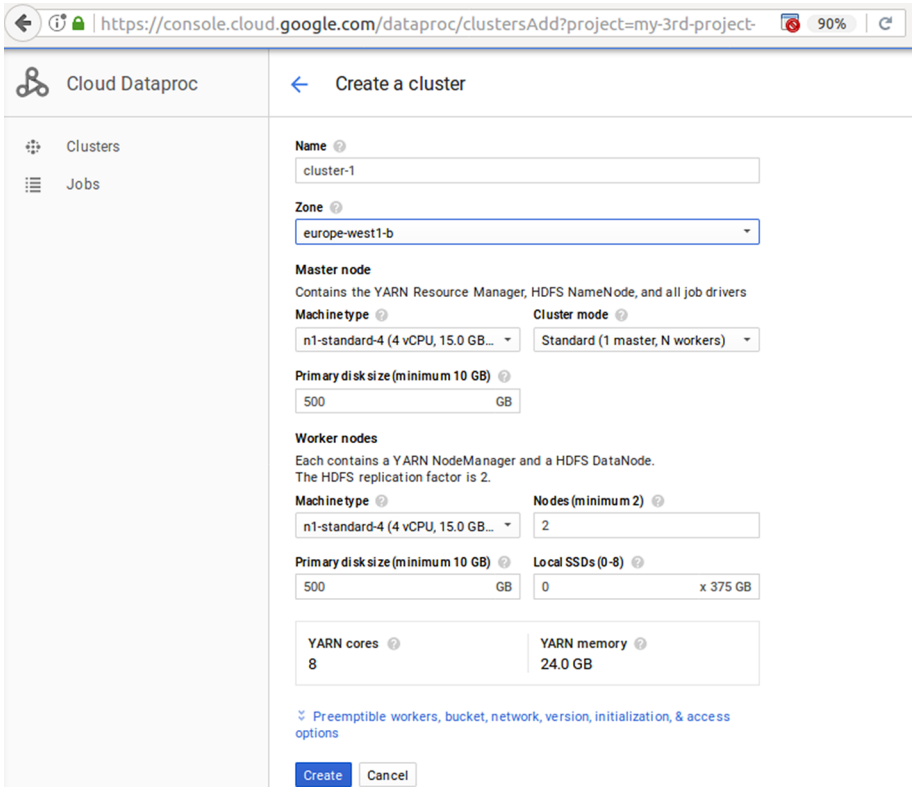
3.5. Caso de uso de *cloud* público y Hadoop/Spark: Google Cloud DataProc

El auge de *big data* también ha llegado a los *clouds* públicos (por ejemplo, Amazon ERM (*Elastic MapReduce*), Azure HDInsight, Google Cloud Dataproc entre otros) y es una opción como las tratadas en capítulos anteriores, ya que proporciona un entorno pseudo-SaaS orientado a los datos masivos donde el usuario solo debe desplegar sus trabajos, ejecutarlos y obtener los resultados.

En este apartado se realizará una prueba de concepto dentro del entorno de Google en dos apartados, pero es similar (más allá de las interfaces y pasos a seguir) en los otros proveedores (consultar la información correspondiente al inicio del párrafo).

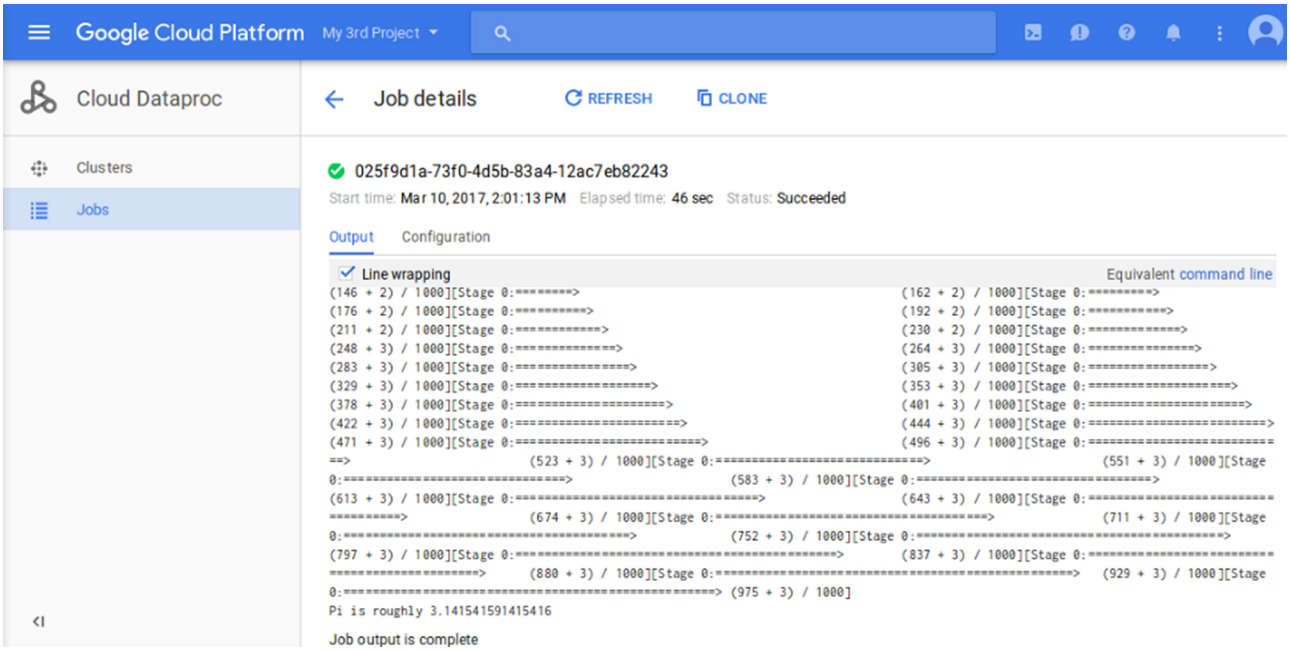
Cloud Dataproc es un servicio de Apache Spark y Apache Hadoop que permite aprovechar las herramientas de datos de código abierto para el procesamiento por lotes/consulta interactiva/*streaming*/*machine learning* a través de configuraciones automáticas donde se puede rápidamente crear un *cluster* y enviar los trabajos a ejecutar solo pagando por su uso.

La creación de un *cluster* se puede hacer desde la consola en el menú lateral → *Big Data* → *DataProc* → *Cluster* → *Create a Cluster* de acuerdo a las indicaciones dadas, o también a través del SDK sobre la máquina local y utilizando la API, o la CLI (`gcloud`). Se deben seleccionar las máquinas –en tipo y disco– tanto para el *master* como para los *workers*, región y otros parámetros (como muestra la figura siguiente) y a continuación *Create*.

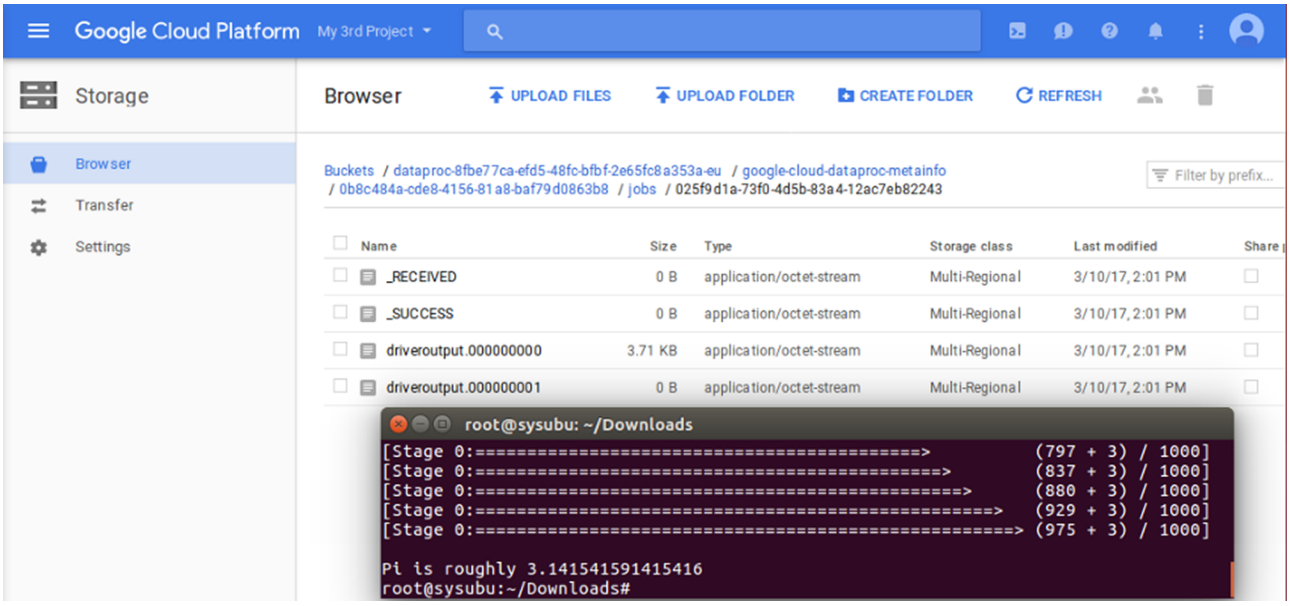


Para ejecutar un trabajo, por ejemplo Spark (en este caso será un programa que calcula el valor de PI por el método de Monte Carlo), se puede hacer en *Jobs* → *Submitjob* → indicar el nombre del *cluster* (*cluster-1*) → seleccionar el tipo de trabajo (*Spark*) → indicar el archivo a ejecutar en el campo de Jar (*file:///usr/lib/spark/examples/jars/spark-examples.jar*) → la clase en el campo *MainClass* (*org.apache.spark.examples.SparkPi*) → y el argumento (1000).

Luego, se puede lanzar la ejecución del trabajo y, haciendo clic sobre él, se verá el resultado de la ejecución (seleccionar *Line Wrapping* para evitar el *scrolling*), como muestra la imagen a continuación.



No obstante, se pueden consultar los archivos de salida de la ejecución del trabajo en *Storage* → seleccionando el trabajo y el directorio de salida, como muestra la figura siguiente.



Considerando las diferentes soluciones que propone Google como casos de estudio, se ha seleccionado uno de *Financial Services* → Monte Carlo *Methods* using Google Cloud Dataproc and Apache Spark. Google Cloud Dataproc y Apache Spark son soluciones óptimas para simulaciones de Monte Carlo que se utilizan para responder a cuestiones en finanzas, ingeniería, ciencia y matemáticas que, de otra forma, no tendrían respuestas. Uno de los problemas

que tiene este método es que es necesario trabajar con gran cantidad de simulaciones, por lo que si el cómputo se realiza sobre una arquitectura convencional, consumirá mucho tiempo y recursos, mientras que en Cloud Dataproc permite escalar y ejecutarlo de acuerdo a las necesidades pagando por uso (por minuto), con las consiguientes ventajas de reducción de tiempo, costo de la infraestructura y de la administración.

Siguiendo la guía antes indicada, verificar que el *cluster* está creado y activo; luego, se deberá acceder almenú lateral → *VM Instances* → *Nodo Máster* del *cluster* → botón SSH a la derecha (para abrir un terminal y conectarnos por SSH a este nodo, que es desde donde se lanzará el trabajo).

Sobre el terminal modificar el archivo */etc/spark/conf/log4j.properties* y cambiar *log4j.rootCategory* a *log4j.rootCategory=ERROR*, *console* para visualizar los posibles errores en el terminal. Para este ejemplo, se utilizará Python (a través de un intérprete interactivo llamado `pyspark`), que intenta predecir cómo se podría hacer una inversión. Es decir, a través de muestras aleatorias de resultados en un rango de condiciones probables del mercado, la simulación de Monte Carlo puede responder a preguntas sobre cómo una cartera podría evolucionar en promedio. Para ello, ejecutar (respetar la sangría de la función *grow*):

```
pyspark
>>> import random
>>> import time
>>> from operator import add

>>> def grow(seed):
        random.seed(seed)
        portfolio_value = INVESTMENT_INIT
        for i in range(TERM):
            growth = random.normalvariate(MK_AVG_RETURN, MK_STD_DEV)
            portfolio_value += portfolio_value * growth + INVESTMENT_ANN
        return portfolio_value
```

Después de terminar de introducir la función, presionar <*enter*> hasta que salga nuevamente el *prompt* de `pyspark` (>>>).

Este código define una función que modela lo que podría suceder cuando un inversionista tiene un plan de pensiones que invierte en el mercado de valores y que además aporta capital adicional cada año. La función genera un rendimiento aleatorio de la inversión, como un porcentaje durante cada año durante y durante un período especificado.

Es importante tener en cuenta que la función toma un valor de semilla como parámetro, que utiliza para reiniciar el generador de números aleatorios, lo cual garantiza que la función no recibe la misma lista de números aleatorios cada vez que se ejecuta. Además, se genera un crecimiento distribuido aleato-

riamente a través de una distribución normal (para una media y la desviación estándar especificadas) y se aumenta el valor de la cartera por el crecimiento (puede ser positivo o negativo) y se le suma un valor que representa una inversión adicional.

El paso siguiente es crear diferentes semillas para introducir en la función (10.000 en este caso):

```
>>> seeds = sc.parallelize([time.time() + i for i in xrange(10000)])
```

Esto genera un conjunto de datos optimizados (RDD) para el procesamiento paralelo, basado en la hora del sistema. Cuando se crea este RDD, Spark divide los datos en función del número de *workers/cores* disponibles (en este caso ocho conjuntos) para los 10.000 datos utilizados.

El paso siguiente es poner estos a disposición de la función *grow*:

```
>>> results = seeds.map(grow)
```

El método *map* pasa cada semilla en el RDD a la función *grow* y agrega cada resultado a un nuevo RDD, que se almacena en los resultados realizando una transformación y no genera resultados hasta que sean necesarios.

A continuación, se especifican los parámetros de la función:

```
>>> INVESTMENT_INIT = 100000  # valor inicial de la inversión
>>> INVESTMENT_ANN = 10000  # inversión adicional c/año
>>> TERM = 30  # período en años
>>> MK_AVG_RETURN = 0.11 # porcentaje medio
>>> MK_STD_DEV = 0.18  # desviación estándar
```

Ahora se debe ejecutar el *reduce* para agregar los valores:

```
>>> sum = results.reduce(add)
```

Y finalmente, imprimir el resultado (no descuidar el carácter '.' al final ya que significa *float*):

```
>>> print sum / 10000.
```

Se podría cambiar por ejemplo el `MKT_AVG_RETURN` y volver a ejecutar:

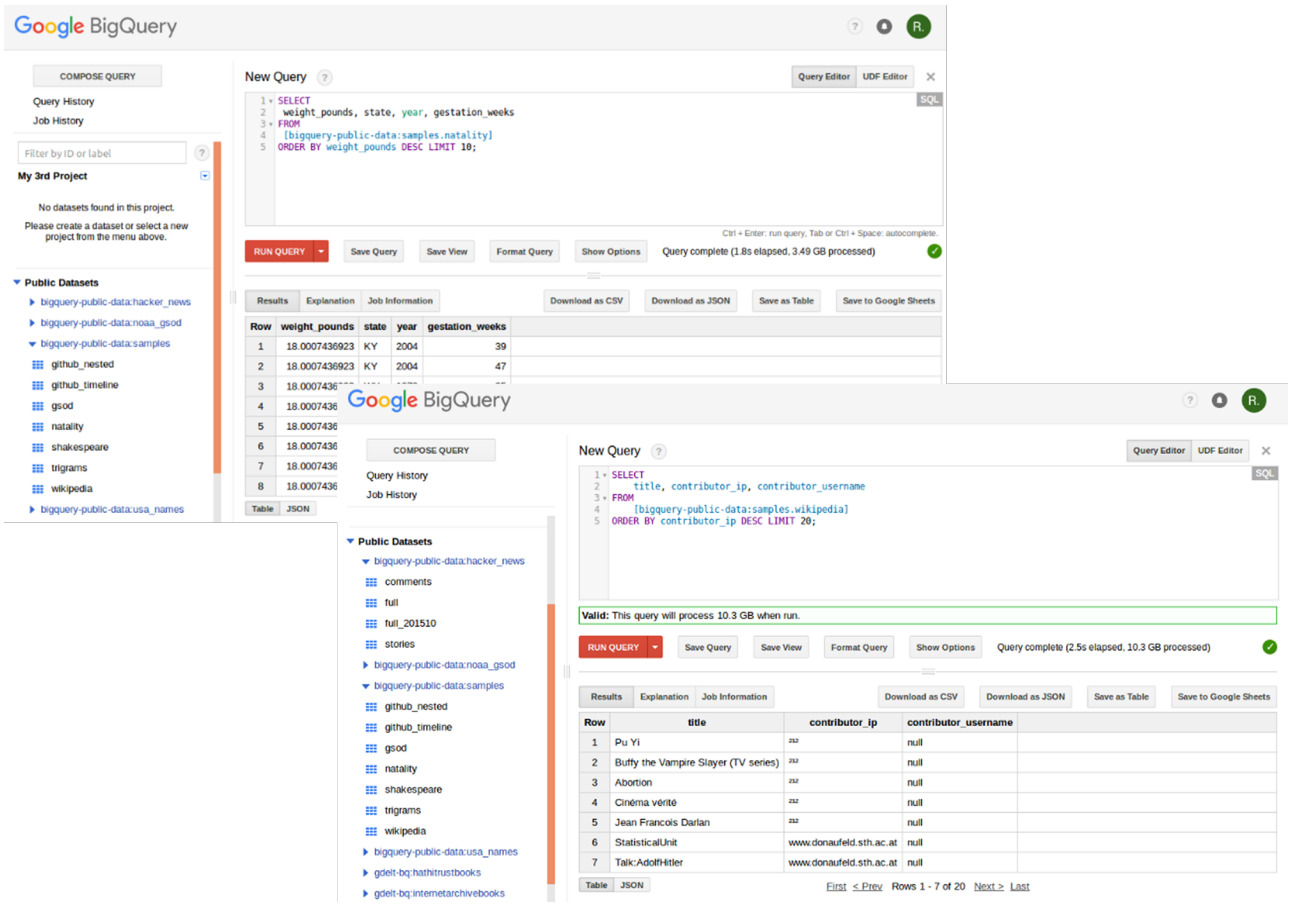
```
>>> MKT_AVG_RETURN = 0.07
>>> print sc.parallelize([time.time() + i for i in xrange(10000)]) \
            .map(grow).reduce(add)/10000.
```

Hacer *Ctrl* + *D* para salir del intérprete y recordar que hay que eliminar el clúster y los ficheros del *Storage* (a través de la consola o de la línea de comandos) para no incurrir en gastos adicionales. La figura siguiente muestra la ejecución en el terminal del nodo máster y los resultados obtenidos para los valores MK_AVG indicados.

Otro ejemplo interesante es utilizar datos públicos o propios para hacer consultas a través de BigQuery. Esta aplicación permite almacenar datos y hacer consultas (a nivel de petabytes) totalmente de forma administrada y de bajo costo, a partir del paradigma *NoOps* (sin infraestructura para administrar y sin administrador de base de datos), lo cual permite al usuario centrarse en analizar datos para encontrar valor agregado utilizando SQL y bajo un modelo de pago por uso.

Para hacer una prueba se ha accedido a través de la consola y el menú lateral → *BigQuery* → *Compose Query* y se ha introducido la consulta SQL sobre dos bases de datos públicas (una de datos sobre la natalidad y otra de *Hacker News*).

Las consultas y los resultados se muestran a continuación (es importante verificar la tilde verde a la derecha antes de ejecutar la *query*, ya que si está en rojo indica algún error en la sintaxis).



Como se puede observar en la primera prueba (natalidad), se realizó la consulta sobre 3,5 GB en 1,8 segundos y en la segunda (*Hacker News*) sobre 10,3 GB, y tardó 2,5 segundos, todo ello sin preocuparse de la infraestructura/base de datos/administración. En interesante seguir la documentación indicada importar/exportar los datos y cuestiones respecto a la seguridad y privacidad de los mismos.

Las herramientas de *big data* de Google (Cloud DataProc, BigQuery) muestran una plataforma con una gran funcionalidad y conectores para vincular otras herramientas, como por ejemplo BigTable –DB NoSQL–, y se sugiere consultar la documentación y los casos de uso para ganar conocimiento y experimentar con ellas.

Además de trabajar con mayor detalle en las herramientas utilizadas, se aconseja extender el análisis a otras herramientas no menos potentes de que dispone Google Cloud para el tratamiento del *big data*, como por ejemplo Pub/Sub, Dataflow y ML Engine entre otras [Ghs].

3.6. Caso de uso de *cloud* público y *machine learning*: Azure ML

El aprendizaje automático (ML) es una técnica que se aplica a la ciencia de datos y que está muy ligado a la inteligencia artificial (IA), que permite extraer información y generalizar comportamientos a través de aplicaciones/dispositivos capaces de aprender de unos datos existentes y, a través de un proceso de inducción, obtener unos resultados que permitan conocer tendencias/comportamientos o, simplemente, agrupar la información por sus características.

Estas técnicas, a veces con mucha parte de estadística y muy ligadas a la complejidad computacional de un problema, permiten que aplicaciones o dispositivos puedan «aprender»de las indicaciones dadas o de los datos introducidos y hacer sugerencias/clasificar y/o ayudar a tomar decisiones, dado que si el problema es de complejidad elevada, este puede compararlo con los datos para los cuales ha sido entrenado y así ofrecer un camino posible que es probable que sea mejor, o al menos sea una decisión más acertada que la que se podría tomar si no se dispone de información. Para muchos autores es la evolución del método científico donde algunas partes han sido automatizadas mediante procedimientos computacionales, estadísticos y matemáticos y con gran aportación de la IA.

Por ejemplo, un servicio de *mail* que utiliza ML puede decidir si un correo es *spam* o no en función de una gran cantidad de conocimiento que tiene almacenado sobre lo que ha aprendido de casos pasados (entendiendo que quien envía el *spam* ha cambiado notablemente las características de cada correo para que no pueda ser detectado por las reglas habituales, como origen, contenido, tamaño, etc.); un robot de limpieza utiliza un algoritmo de ML para tomar la decisión cuando ha terminado o no la limpieza de una habitación, o también se utiliza ML en la detección de secuencias y clasificación de ADN/RNA, en la interpretación de palabras/traducción de idiomas o el reconocimiento de la escritura y su traducción automática, entre otros muchos ejemplos.

Con dos grandes divisiones de los algoritmos de ML es como se produce el entrenamiento/aprendizaje, en que se puede distinguir el supervisado o el no supervisado (si bien hay otras clasificaciones como semisupervisado, por esfuerzo, multitarea, etc.).

En el primero se entrena al sistema para que pueda establecer una relación entre las entradas y las salidas, es decir, qué conjunto de datos forman parte de uno u otro caso, proporcionándole información sobre, por ejemplo, las categorías de una imagen (paisaje, edificio, persona) y el sistema podrá luego clasificar una imagen sobre la base de lo aprendido. En cambio, el aprendizaje no supervisado no cuenta con datos que definan qué información satisface los objetivos del aprendizaje o no, y debe encontrar patrones entre sus características que permitan establecer categorías y poder hacer conjuntos de datos que tengan los mismos atributos.

Siguiendo el ejemplo anterior de clasificación de imágenes, no podría decir si es un edificio o no, pero podría agrupar todos los edificios, las personas y los paisajes en grupos. Esta información luego deberá ser «interpretada» para indicar que un grupo de imágenes de paisajes son paisajes.

Azure Machine Learning es un servicio de análisis predictivo en el *cloud* que permite crear e implementar, de forma simple y rápida, modelos predictivos y analizar (a partir de una librería de algoritmos ya integrados) interactivamente su ejecución y resultados a través de un servicio totalmente administrado que se puede usar para implementar los modelos predictivos como servicios web listos para utilizar.

El *workflow* que adopta Azure para esta propuesta es:

- recolección de datos que pueden ser de Azure Storage (*blobs* y tablas), HDinsight (Hadoop), Azure SQL (BD), Azure Data Lake (almacén de datos masivos),
- aplicación ML con aprendizaje automático utilizando modelos ya probados, adaptaciones de estos o propios del diseñador,
- servicio web que puede recibir datos externos (por ejemplo, de un BD),
- utilización de la información generada para agregar «inteligencia» o proporcionar información.

Hay que consultar la documentación para tener un mayor detalle de los términos y conceptos implicados en Azure ML, así como del ciclo de vida y cuestiones vinculadas a la ciencia de los datos [Aml].

Azure dispone de un entorno llamado Machine Learning Studio donde se pueden crear de forma gráfica los modelos predictivos por medio de la interconexión de módulos, ya sean de su propia galería (Cortana Intelligence) o con modificaciones sobre esta base o propios.

Su utilización es muy sencilla, ya que presenta un entorno de trabajo (*canvas*) donde se arrastrarán los módulos, los cuales disponen de puertos de interconexión de entrada y salida que permitirán vincular las acciones e indicar cómo los datos van fluyendo y son procesados.

En la parte izquierda se encontrarán una serie de íconos que indican proyectos (colecciones de experimentos, conjuntos de datos, *notebooks* y otros recursos que representan un proyecto individual), experimentos (que se han creado/ejecutado y guardado), servicios web (implementado a partir de experimentos), *notebooks*, conjunto de datos, modelos entrenados y configuración.

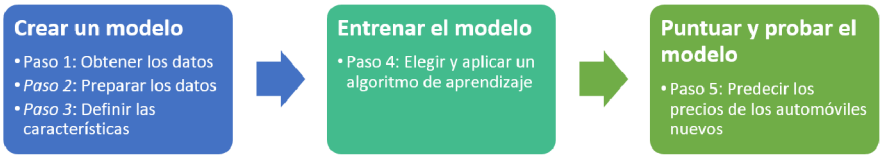
A continuación, se encontrará un menú de búsqueda de recursos para trabajar, sobre la parte derecha un menú de propiedades sobre cada recurso seleccionado en el *canvas*, y en la parte inferior los controles para crear y ejecutar experimentos/modelos.

La dinámica es sencilla, se crea un experimento en *New* ✚ en la parte inferior (se puede seleccionar uno de la galería o uno en blanco), se busca el recurso en la caja de búsqueda, se arrastra al *canvas*, se modifican las propiedades y se interconecta con sus fuentes y destinos (para ello, simplemente seleccionar el puerto de origen y con un clic del ratón estirar la línea hasta el destino).

Cuando se tiene terminado se puede ejecutar (*Run* ▶) para verificar su funcionalidad o visualizar los datos a través del botón derecho en cada módulo (u otras tareas que permitana una fuente de datos, por ejemplo, descargarla en local).

Como prueba de concepto se utilizará un caso de uso de ML basado en la guía Estudio de aprendizaje automático de Azure, que permitirá aplicar los conceptos más importantes y hacer una prueba funcional sobre la plataforma Azure.

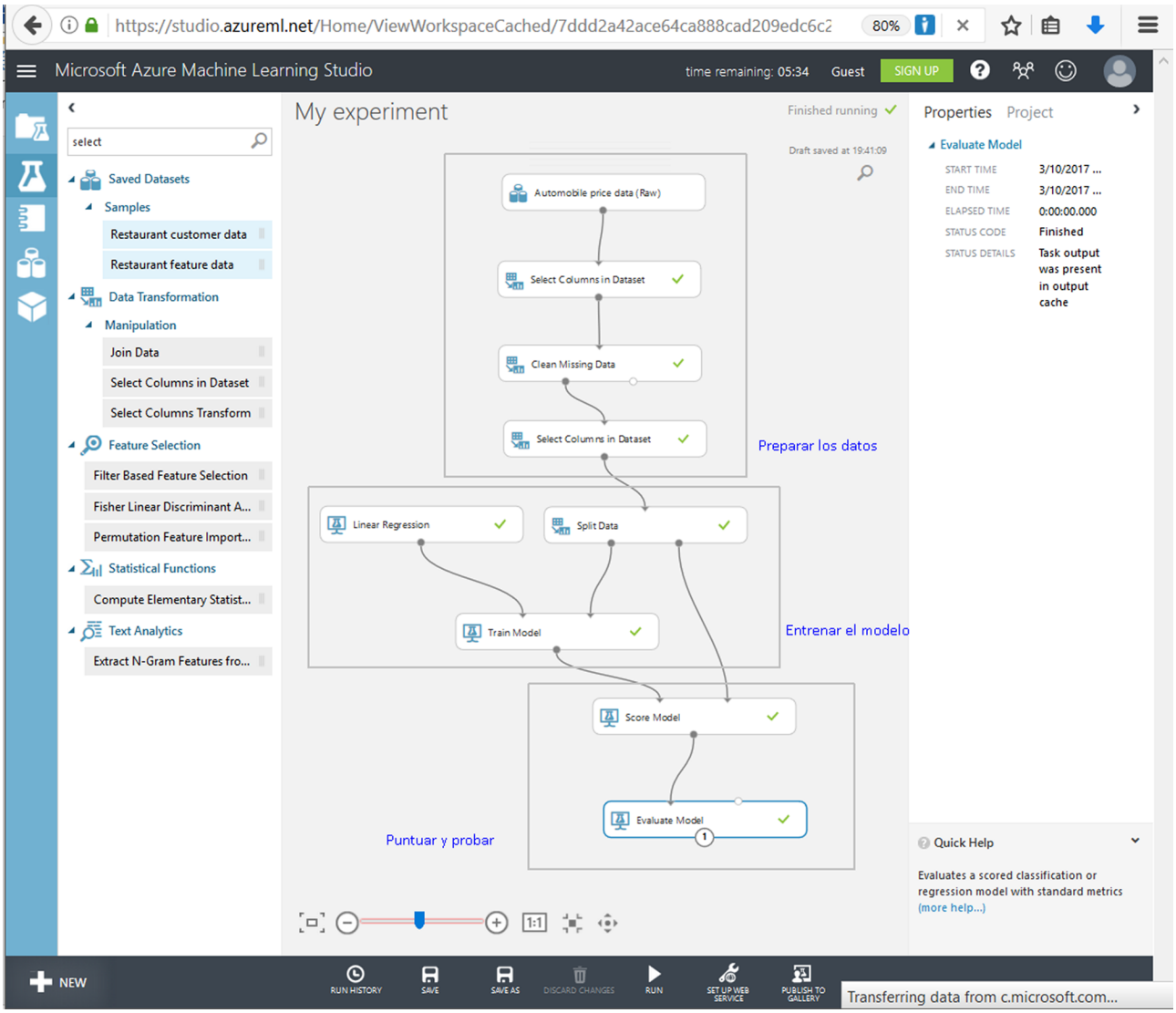
El *workflow* para una aplicación de este tipo es muy simple y queda representado por la figura siguiente:



Para esta prueba se accederá a la plataforma Azure ML Studio y se seleccionará el tipo de acceso (invitado de 8 horas y sin registro o gratuito si se dispone de una cuenta en Microsoft).

Sobre esta plataforma se trabajará con un conjunto de datos que ya dispone que incluye datos de automóviles de diferentes fabricantes, modelos, especificaciones técnicas y precio.

El *workflow* terminado se muestra en la figura siguiente y se irán refiriendo a cada una de las cajas indicadas anteriormente.



Para iniciar el trabajo:

1) En la parte inferior hacer *New* (✚) y seleccionar la galería *Blank experiment*, que mostrará un esqueleto para insertar los módulos; cambiar, en la parte superior, el nombre por otro más significativo (seleccionar y cambiar).

2) En la caja de búsqueda (puede estar plegada) se introducirá *automobile* para seleccionar los datos y de la lista arrastrar el conjunto de datos **Automobile price data (Raw)** al *canvas*. Con el botón derecho sobre el punto de salida podremos visualizar este conjunto de datos, salvar una copia en local u otras operaciones. Como se puede ver, desde la visualización hay datos incompletos, por lo cual se deberán «limpiar» estas filas/columnas.

3) En la caja de búsqueda se introduce *select* y de la lista *Manipulation* → *Select columndata set* se arrastra debajo del módulo anterior y se interconectan como en la figura anterior. Para seleccionar la «limpieza» se hace clic en el módulo

agregado y en *Properties → Launch column selector → With rules → All Columns → Exclude → Column Names → normalized-losses* → clic en la tilde inferior para aceptar.

4) Repetir el procedimiento agregando el módulo *Clean Missing Data → Properties → Clean Mode → Remove entire row*. Desde el botón *Run* inferior (▶) ejecutar el *workflow* definido, que mostrará una tilde verde al costado de cada módulo cuando se haya ejecutado correctamente.

En ML se denomina «características» a las propiedades individuales y que se pueden medir por su interés; en este conjunto de datos cada fila representa un automóvil y cada columna una característica de ese automóvil. Se necesita experiencia y conocimiento para decidir qué características analizar o quitar (por ejemplo, si dos de ellas están fuertemente correlacionadas, se puede quitar una sin que afecte a la predicción); no obstante en este caso, los desarrolladores sugieren *make*, *body-style*, *wheel-base*, *engine-size*, *horsepower*, *peak-rpm*, *highway-mpg*, *Price*, pero podrían ser otras a voluntad (y conocimiento) del usuario que hace el análisis.

5) Para ello, agregar otro módulo *Select Columns in Dataset → Properties → Launch column selector → With rules → No Columns → Include → Column Names* → y agregar la indicada anteriormente y aceptar. Luego conectar la salida izquierda de *Clean* con la entrada de este módulo (ver figura anterior) y con ello finalizará la preparación de los datos (primer rectángulo).

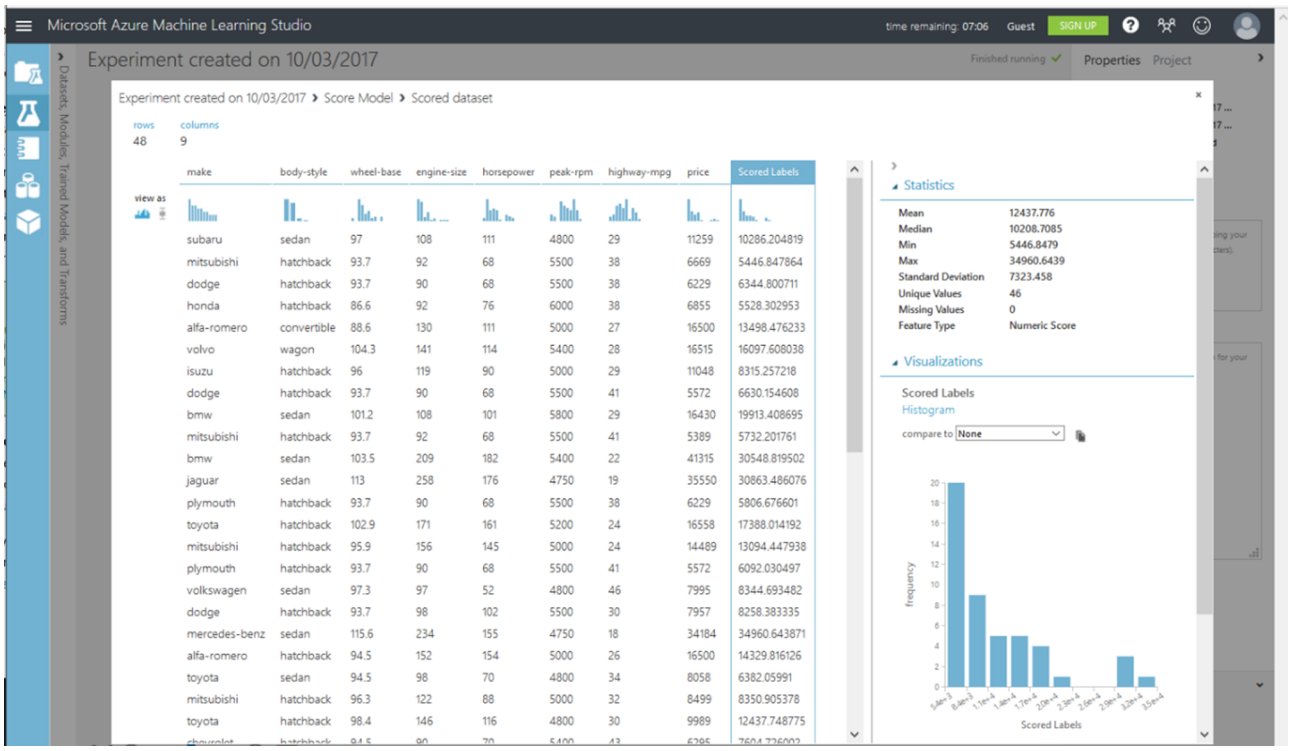
6) Ahora se deben seleccionar los datos para entrenar el modelo y los datos para predecir; para ello se agregará un módulo de *Split Data* y se conectará al anterior indicando en *Properties → Fraction of rows in the first output dataset* un valor de 0,75, que indica que el 75 % de los datos serán utilizados para el entrenamiento y el 25 % restante para la predicción. Ejecutar nuevamente el *workflow* para verificar que todo funciona correctamente (tilde verde en cada módulo).

En este ejemplo, utilizaremos aprendizaje supervisado para predecir el precio a partir de unas características; como algoritmos básicos de ML se pueden utilizar los de clasificación o de regresión, donde el primero permite predecir una respuesta a partir de un conjunto definido de categorías, como el color (rojo, azul o verde), mientras que la regresión se usa para predecir un número. Como se desea predecir el precio, se utilizará un modelo de regresión lineal, que analizará los datos y buscará, en la fase de entrenamiento, correlaciones entre las características de un automóvil y su precio para luego darle, en la fase de predicción, un conjunto de características de automóviles y predecir el precio.

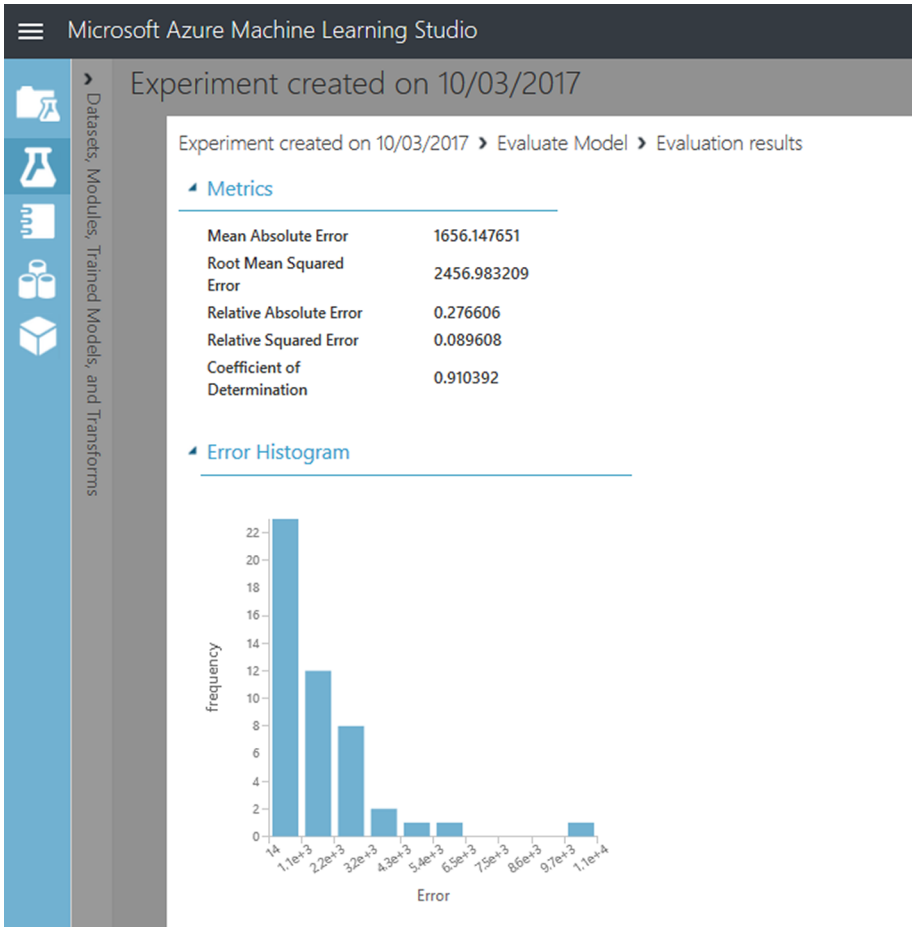
7) Buscar/agregar el módulo de *Linear Regresion*, y buscar/agregar el módulo de *Train Model* e interconectarlos y también a *Split Data*, como se muestra en la figura anterior. En *Train Model → Properties → Launch column selector →* seleccionar la columna *Price* y moverla hacia la derecha. Finalmente, ejecutar

el experimento y verificar que todo es correcto. A partir de este momento se dispone de un modelo de regresión entrenado que puede usarse para realizar predicciones de precios en función de sus características.

8) Ahora se debe puntuar y verificar cómo funciona el modelo con el 25 % de los datos restantes; para ello, buscar/agregar el módulo *Score Model* e interconectarlo con *Train Model* y *Split Data*, como se indica en la figura anterior, y ejecutar el modelo y observar la salida (visualizar), que mostrará los valores dados por la predicción que se pueden comparar con los reales ya conocidos para analizar cómo ha ido esta. Ver las últimas dos columnas de la figura siguiente (se pueden seleccionar los gráficos correspondientes para ver si el histograma tiene una forma similar).



Finalmente, para probar la calidad de los resultados, buscar/agregar/interconectar el módulo *Evaluate Model* y ejecutar/visualizar los resultados, donde se mostrará una serie de estadísticas que darán información sobre la adecuación del modelo: desviación media del error o MAE (la media de errores absolutos, diferencia entre el valor de predicción y el valor real), raíz cuadrada de errores o RMSE, el error absoluto relativo o RAE, el error al cuadrado relativo o RSE, y el coeficiente de determinación o CD (valor R cuadrado es una métrica estadística que indica cómo de bien se ajusta un modelo a los datos; cuanto más cerca está el valor de uno, mejores son las predicciones).

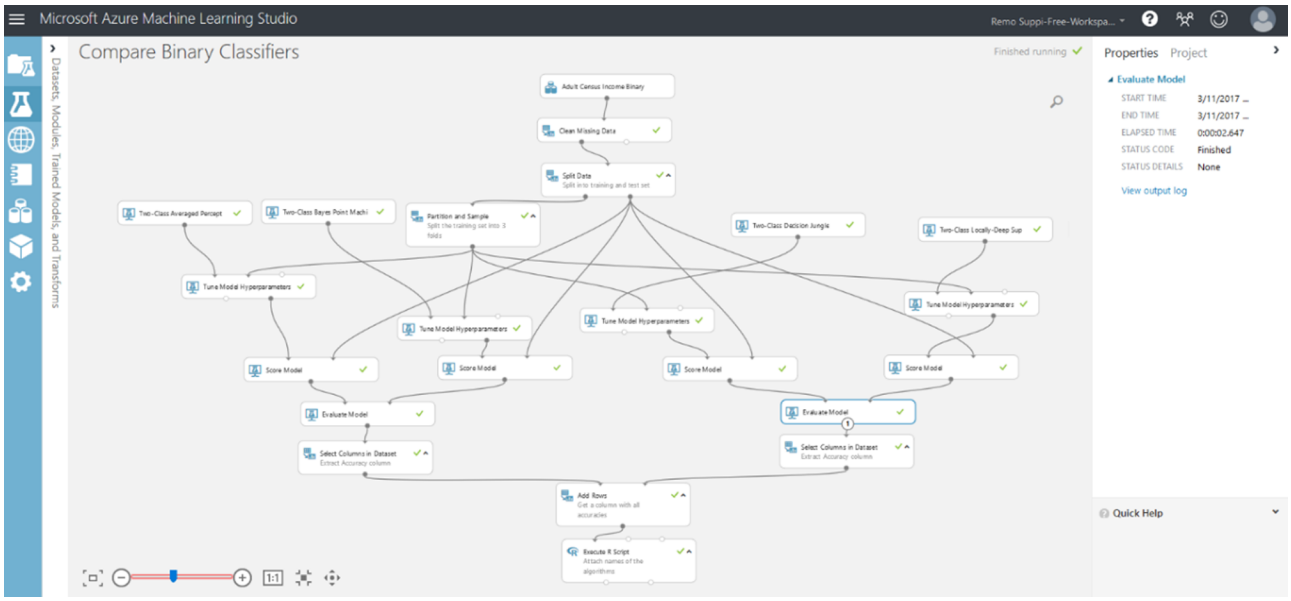


9) Como pasos siguientes se podrían modificar los parámetros del modelo de regresión linear, cambiar los parámetros de *Split Data* o agregar otro modelo adicional, y como el módulo *Evaluate Model* dispone de dos entradas, comparar los resultados de la predicción de dos modelos diferentes o finalmente, cuando se tenga ajustado, implementarlo como un servicio web predictivo.

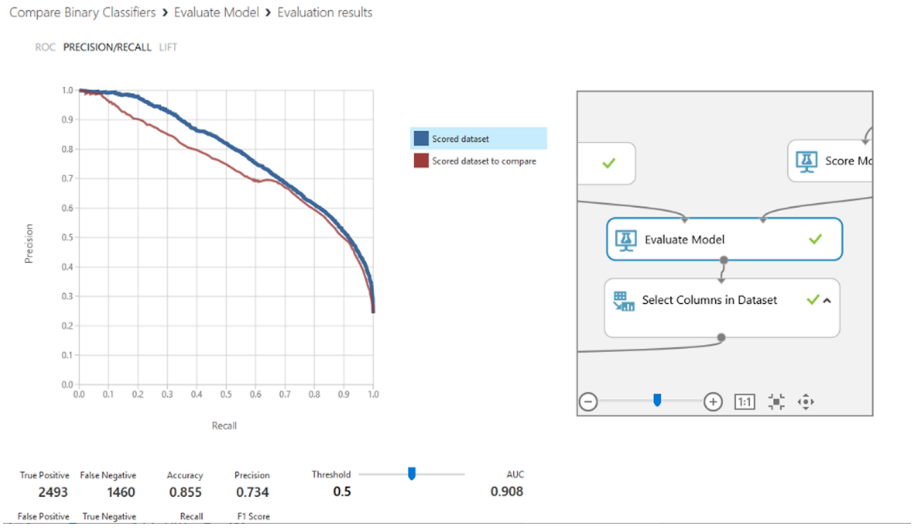
10) Para ver un ejemplo de cómo comparar varios modelos en un único experimento, consultar *Compare Regressors* en la galería de Cortana Intelligence.

No hay que olvidar salvar el experimento y consultar la documentación indicada para mayor detalle y/o aspectos adicionales indicados por el desarrollador.

11) Más complejo para comparar diferentes algoritmos es el propuesto en la galería de Cortana Compare Binary Classifiers, que permite, a través de los datos de una población (32 k + filas de 15 características) del repositorio UCI, y centrándose si el salario/año es > que $50 K, comparar diferentes clasificadores binarios (*Two-Class Averaged Perceptron*, *Two-Class Bayes Point Machine*, *Two-Class Decision Jungle and Two-Class Locally-Deep Support Vector Machine*). La figura siguiente muestra el modelo y su ejecución.



Y la comparación a la salida de la comparación de los dos modelos de la derecha.



Como se ha podido observar, las posibilidades y funcionalidades que presenta la plataforma son muy extensas y es necesario tiempo y dedicación para experimentar con ella y utilizar el ML en el *cloud* aplicado a los datos de interés.

Es evidente que las prestaciones y posibilidades sobre el *big data* que presentan este tipo de plataformas es realmente muy interesante; el usuario puede experimentar y analizar sus modelos/datos sin tener que disponer/gestionar la infraestructura o la plataforma subyacente. Se recomienda consultar la información y los casos de uso propuestos en la documentación para mayor detalle y ejemplos [Aml].

4. Conclusiones

Tal y como se ha podido apreciar, existe una **gran** cantidad de metodologías, herramientas y plataformas para tratar el *big data* disponibles para que los usuarios, simplemente, tengan que dedicarse a su trabajo de análisis o a desarrollar los modelos de predicción sobre sus datos. Los modelos de trabajo pueden ser locales o en el *cloud* o mixtos, por ejemplo, MapR en local para los desarrollos preliminares y luego pasar a AWS con MapR con la escalabilidad y conjunto de datos adecuado.

Se debe tener en cuenta que toda esta tecnología es emergente y que se puede comprobar que hay una gran «ebullición» sobre estos temas ya que, dado el interés despertado y las facilidades de procesamiento a través de herramientas y el *cloud*, existen gran cantidad de empresas e instituciones que desean analizar sus datos para mejorar sus procesos/predecir comportamientos o simplemente tener información de cómo evolucionan sus productos/servicios.

La ciencia de los datos es una realidad, y en buena parte gracias al *cloud*, ya que al transformar todo en PaaS o SaaS ha permitido acercar la tecnología a los datos y estos a los usuarios que no entienden de infraestructura/administración, pero sí de datos y negocio. Como muestra de esta ciencia de los datos, en el último tiempo ha surgido formación específica en las universidades para generar profesionales que sepan de datos y su procesamiento; se les ha denominado *data scientist*. Desde que en 2013 se creó el *IEEE Task Force on Data Science and Advanced Analytics*, todo ha sido una sucesión de acontecimientos que muestran este interés sobre el tema; por ejemplo, el siguiente año, la primera conferencia internacional sobre Data Science and Advanced Analytics (en este momento está abierta la presentación de trabajos para la edición del 2017) o publicaciones como *Journal of Big Data* (Springer), *Big Data Research* (Elservier) o *Transactions on Big Data* (IEEE), entre otros, son una muestra de la importancia e investigación realizada sobre estos temas.

Como punto final es interesante (a la luz de lo analizado y experimentado en los apartados anteriores) releer el informe *Big data: The next frontier for innovation, competition, and productivity* [Bdn], mencionado al inicio del capítulo, para comparar la evolución y las predicciones en el año 2011 y la actualidad.

Actividades

1. Instalar y desplegar una aplicación de pruebas sobre Apache Hadoop. Analizar y valorar las posibilidades de la plataforma y de las diferentes herramientas y su potencialidad sobre diferentes conjuntos de *big data* obtenidos de repositorios públicos.

2. Considerando las diferentes plataformas Hadoop, desplegar dos de ellas para ejecutar una de las aplicaciones desarrolladas en el ejercicio anterior. Valorar y analizar las posibilidades de cada plataforma creando una tabla comparativa de las diferentes opciones y equivalencias, ventajas y desventajas.

3. Repetir la experiencia sobre Google DataProc.

4. Realizar un análisis comparativo de datos utilizando diferentes conjuntos de datos/años de repositorios públicos con la herramienta Google BigQuery.

5. Hacer un análisis y evaluación detallada de un caso de predicción utilizando Azure ML basándose en datos públicos y sobre la base de alguno de los ejemplos publicados en la galería de Cortana. Ampliar este estudio a otros datos propios o de fuentes no incluidas en Azure y extender este como servicio web.

Glosario

Ambari, Avro, Cassandra, Chukwa, HBase, Hive, Mahout, Pig, Spark, Tez, ZooKeeper Principales herramientas del entorno *open-source* Apache Hadoop.

Azure Data Lake Herramienta de Azure que implementa un almacén de datos masivos.

Azure Machine Learning Entorno de Azure para el aprendizaje automático.

Azure SQL Entorno de Azure para el trabajo con BD.

Azure Storage (*blobs* y tablas) Fuente de datos aceptadas por Azure ML.

big data Datos masivos.

BigQuery Herramienta de Google (SaaS) para el procesamiento de grandes conjuntos de datos (petabytes) sin infraestructura.

business intelligence Técnicas para transformar los datos en información y esta en conocimiento para mejorar el proceso de toma de decisiones en el ámbito empresarial.

canvas Espacio donde se puede «dibujar» un diagrama en el entorno de Azure ML.

CDH, Cloudera Manager, Cloudera Director Distribución de Hadoop, entorno de gestión y entorno de traspaso de datos al *cloud* respectivamente de la compañía Cloudera.

chunk Secuencia/segmento de datos.

cloud computing Propuesta tecnológica que permite acceder a aplicaciones o servicios remotos en forma ubicua a través de un navegador; estos pueden ser aprovisionados/liberados bajo demanda y en un tiempo reducido.

Cloud Dataproc Servicio de Apache Spark y Apache Hadoop en Google Cloud.

Cloudera, HortonWorks, MapR Tres de las distribuciones más importantes que utilizan Apache Hadoop como base.

clustering Algoritmos de agrupación de datos en aprendizaje automático.

Cortana Intelligence Entorno de Azure ML con algoritmos y casos de uso para aprendizaje automático.

CRM (*customer relationship management*) Herramienta/modelo de gestión de una organización que se basa en la satisfacción del cliente.

daemon Proceso que se ejecuta independiente de resto en un SO y que generalmente atiende las peticiones de un servicio.

data mining Técnicas que permiten encontrar comportamientos predictivos combinando métodos estadísticos sobre los datos.

deep learning Técnicas de ML que modelan abstracciones de alto nivel sobre los datos utilizando transformaciones no-lineales múltiples.

desviación media del error (MAE), raíz cuadrada de errores (RMSE), error absoluto relativo (RAE), error al cuadrado relativo (RSE), coeficiente de determinación (CD) Medidas estadísticas habituales para la comparación de resultados (utilizadas en las predicciones obtenidas por ML).

exabytes Medida en bytes equivalente a 1 millón de terabytes (1 exabyte=10^{18}bytes) utilizada para dar medida al *big data*.

fork-join Sentencia de programación (también utilizada en los procesos) que implica una bifurcación con procesamiento concurrente y una unificación/sincronización posterior.

Gartner Consultora en TIC.

Hadoop YARN, MapReduce, HDFS, Hadoop Common Las cuatro partes esenciales del entorno Apache Hadoop.

Hadoop Plataforma *open-source* para el cómputo distribuido, confiable y escalable.

HDinsight (Hadoop) Entorno Hadoop de Azure.

Hortonworks (HDP) Plataforma Hadoop de la compañía de igual nombre.

IBM Uno de los grandes actores en el tema del *big data*.

IoT (internet de las cosas) Interconexión a través de internet de dispositivos de consumo.

linear regresion Algoritmo de predicción basada en la relación de una variable escalar y otra (u otras) a través de una relación lineal.

local (standalone), pseudo-distributed, fully-distributed Tres de los modos en los cuales se puede instalar y funcionar la plataforma Hadoop.

machine learning Técnicas de aprendizaje automático.

Machine Learning Studio Entorno de ML interactivo de Azure.

MapR Converged Data Platform Plataforma Hadoop de la compañía MapR.

MapReduce (MR) Modelo de programación orientado al cómputo paralelo.

***massive paralell processing* (MPP)** Ejecución coordinada/sincronizada de un programa sobre múltiples procesadores que trabajan en diferentes partes del mismo programa.

master-workers Paradigma de computación distribuida donde el *master* reparte trabajo que realizan los *workers* concurrentemente y retornan los resultados al *master*.

NoSQL (Not Only SQL) Base de datos para datos no estructurados.

paradigma NoOps Técnica que permite la ejecución de una aplicación en modo SaaS sin tener que preocuparse por la infraestructura, ni por la base de datos si se necesita.

Pub/Sub, Dataflow, ML Engine Herramientas *big data* de Google complementarias a DataProc y BigQuery.

semistructured data Datos parcialmente estructurados a través de un archivo XML o Json.

shared nothing Arquitectura de cómputo distribuida donde cada nodo es independiente/autosuficiente del resto.

streams Flujo de datos (dinámico).

structured data Datos clasificados por una estructura/tipo utilizados en una BD relacional.

text analytics Técnica de análisis de datos contenido en texto en lenguaje natural.

unstructured data Datos no estructurados, como por ejemplo los utilizados en BD NoSQL.

workflow Secuencia de procesos/flujos de información que representa una secuencia ordenada de trabajo/procesamiento.

zettabytes Medida en bytes equivalente a 1.000 millones de terabytes (10^{21} bytes) utilizada para cuantificar el *big data* de un futuro cercano.

Bibliografía

Todos los enlaces han sido visitados en febrero de 2017.

[Abd] Awesome-bigdata. Onur Akpolat. <https://github.com/onurakpolat/awesome-bigdata>

[Ahd] Apache Hadoop Project. 2017. <http://hadoop.apache.org/>

[Aml] Azure Machine Learning en el *cloud*. 2017. <https://docs.microsoft.com/es-es/azure/machine-learning/machine-learning-what-is-machine-learning>

[Bad] Big Data Storage Architecture Design in Cloud Computing. X. Chen; S. Wang; Y.Don;X. Wang. Big Data Technology and Applications. 2016. BDTA 2015, CCIS 590 (págs. 7-14), 2016. DOI: 10.1007/978-981-10-0457-5_2.

[Bda] Big Data: A Revolution That Will Transform How We Live, Work, and Think. V. 2013. Mayer-Schönberger, K.Cukier. Houghton Mifflin Harcourt.

[Bdd] Big Data for Development. A Review of Promises and Challenges. 2015. Hilbert M. <http://www.martinhilbert.net/big-data-for-development/>

[Bdn] Big data: The next frontier for innovation, competition, and productivity. 2011. J. Manyika; M. Chui; B. Brown; J. Bughin; R. Dobbs; C. Roxburgh; A. Hung Byers. McKinsey Global Institute. <http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/big-data-the-next-frontier-for-innovation>

[Bds] Is Big Data Still a Thing? (The 2016 Big Data Landscape). 2016. Matt Turck. FirstMark. <http://mattturck.com/2016/02/01/big-data-landscape/>

[Ccb] Del cloud computing al big data. 2012. Jordi Torres i Viñals. FUOC. <http://www.jorditorres.org/wp-content/uploads/2012/03/Del.Cloud_.Computing.al_.Big_.Data_.JordiTorres.ES_.pdf>

[Cdo] Cloudera Documentation (CDH). 2017. <https://www.cloudera.com/documentation/enterprise/5-6-x/topics/introduction.html>

[Cma] Cloudera Manager. 2017. <https://www.cloudera.com/documentation/manager/5-1-x.html>

[Dbd] Design of big data processing system architecture based on Hadoop under the cloud computing. 2014. C. Duan. Applied Mechanics and Materials (vols. 556-562, págs. 6.302-6.306). DOI:10.4028/www.scientific.net/AMM.556-562.6302.

[Fbd] The Forrester Wave™: Big Data Hadoop Distributions. 2016. <https://www.cloudera.com/content/dam/www/static/documents/analyst-reports/forrester-wave-big-data-hadoop-distributions.pdf>

[Gmr] Why Google Capital Placed Its Hadoop Bet On MapR? 2014. *Forbes*. <https://www.forbes.com/sites/danwoods/2014/06/30/why-google-capital-placed-its-hadoop-bet-on-mapr/#3cff542678e7>

[Ghs] Spark and Hadoop on Google Cloud Platform Documentation. 2017. <https://cloud.google.com/hadoop/>

[Hao] Hadoop Architecture Overview. 2017. Emilio Coppa. <http://ercoppa.github.io/HadoopInternals/HadoopArchitectureOverview.html>

[Hdp] Hortonworks Data Platform. 2017. <http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.5.3/bk_release-notes/content/index.html>

[Hui] Hadoop 2.6.5 Installing on Ubuntu 16.04 (Single-Node Cluster). 2017. K. Hong. <http://www.bogotobogo.com/Hadoop/BigData_hadoop_Install_on_ubuntu_16_04_single_node_cluster.php> <http://www.bogotobogo.com/Hadoop/BigData_hadoop_Running_MapReduce_Job.php>

[Hsn] Hadoop: Setting up a Single Node Cluster. 2017. <https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html>

[Hfd] Hadoop for Dummies. 2014. John Wiley & Sons. ISBN: 978-1-118-60755-8

[Mra] MapR Academy. Free Courses. 2017. <http://learn.mapr.com/>

[Mrd] MapR Documentation. 2017. <http://maprdocs.mapr.com/home/MapROverview/c_overview_intro.html>

[Mrt] MapReduce Tutorial. 2017. <http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html>

[Wbd] A. de Mauro; M. Greco; M. Grimaldi. 2015. What is big data? A consensual definition and a review of key research topics. AIP Conference Proceedings 1644 (págs. 97-104). <http://aip.scitation.org/doi/abs/10.1063/1.4907823>

[Wmr] Why MapR? 2017. <https://www.mapr.com/why-hadoop/why-mapr>

[Ydo] YARN Documentation. 2017. <http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>

[Iar] Iconos con licencia de uso libre. <http://www.customicondesign.com> <http://icons8.com>