TRABAJOS FIN DE GRADO

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Montaje y desarrollo de un clúster funcional de Big Data para el procesamiento de datos en tiempo real utilizando NLP

Autor: **Enrique Sola**

Grado: **Ingeniería Informática**

Director: **Carlos Javier Ogayar Anguita**
Departamento del director: Informática

Fecha: 25/09/2024

(Página intencionalmente en blanco)

Agradecimientos

Quiero agradecer toda mi carrera como ingeniero, este trabajo de fin de grado y todo por lo que pueda agradecer a mi padre, por ser la piedra angular que me ha mantenido en el buen camino y me ha guíado para conseguir todo lo que me he propuesto siempre. A mi abuela, por hacer de madre y ser un apoyo incondicional, por dar años de su vida por la mía, por dar parte de su salud por la mía y por haberme cuidado todos los días y asegurarse que siempre me iba a dormir comido y arropado. A mi abuelo, que siempre mantuvo mi familia unida y que ninguno de nosotros habría podido ser nada si él no se hubiera dejado la vida por nosotros cada día, haciéndonos llegar siempre el orgullo constante que sentía por todos nosotros. A mi tía, por ser dura pero sincera en todo momento, a mi primo, por estar ahí cuando parecía que nos estrellábamos con este proyecto manteniendo la fe en mi cuando otros no lo hacían, a mi hermana, por sacar de mí esa humanidad y cariño que muchas veces se me olvida que tengo y al resto de mi familia por estar siempre presentes de una manera u otra. Y si por cosas del destino, finalmente acabas leyendo esto, también te lo dedico a ti por haber estado en mis peores momentos apoyándome y en mis mejores momentos celebrándolo. También, quiero agradecer a todos mis amigos que entre bromas y risas, no permitieron que tuviese un mal día nunca, que gracias a ellos también he podido descubrir el mundo de otra manera y que sin duda alguna, también han influenciado en mí como profesional y me han convertido un poco más en la persona que soy hoy en día. Además, agradecer a los profesores que me han acompañado a lo largo de mi carrera y de los cuales he aprendido todo lo que he podido, especialmente a mi tutor por todo el trabajo invertido en este proyecto. Finalmente también agradecer a mis compañeros de trabajo que sin duda alguna me convierten cada dia en un mejor profesional y han tenido una gran influencia en mi a nivel personal.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Grado en Ingeniería Informática
Modalidad	Estudio Técnico
Especialidad (solo TFG)	Tecnologías de la Información
Mención (solo TFG)	Sin mención
Idioma	Español
Tipo	Específico
TFT en equipo	No
Autor/a	Enrique Sola
Fecha de asignación	Haga clic aquí o pulse para escribir una fecha.
Descripción corta	Se trata del montaje e instalación de una máquina Big Data que nos proporcione la potencia para lanzar una aplicación en Spark, además de contar con todo el software y entorno necesario para el desarrollo de aplicaciones orientadas al tratamiento masivo de datos. Dicha aplicación, consistirá en una herramienta PLN para la detección de comentarios ofensivos a través un script de scrapping web en twitter.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
*Estas normas se han utilizado **como base o referencia** para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como **proyecto** la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.*	

Contenido

1 ESPECIFICACIÓN DEL TRABAJO

En este capítulo se presenta la especificación del trabajo, con una estructura y contenidos **inspirados** en los criterios y recomendaciones que establece la norma UNE 157801:2007 - "*Criterios Generales para la elaboración de proyectos de Sistemas de Información*".

A lo largo del documento se utilizarán términos y acrónimos cuya descripción aparecen en el apartado 7 (Definiciones y abreviaturas).

1.1 Introducción

El objetivo principal de este trabajo, es el de ofrecer una guía y/o manual completo hacia el montaje e instalación de un clúster Big Data que permita la ejecución de aplicaciones con Spark y un sistema de almacenamiento distribuido. Esto surge con la intención de acercar estos conocimientos a los desarrolladores ya que nos encontramos en un mundo donde el volumen de datos crece constantemente y cada vez se hacen más necesarias las herramientas para afrontar estos nuevos desafíos.

El documento del proyecto inicia con la creación y configuración de la máquina junto a los distintos softwares que conforman el ecosistema Big Data y cómo instalarlos en ésta. Más adelante, se muestra cómo se ha desarrollado un software para obtener comentarios de Twitter utilizando técnicas de scrapping y su posterior análisis utilizando técnicas de NLP, primero con Python en nuestro equipo y luego utilizando Spark en nuestro clúster.

1.2 Objetivos del trabajo

- Montaje de un clúster Big Data que permita la ejecución de aplicaciones Spark y su compenetración con el resto de software del ecosistema.
- Desarrollo de un software Scrapping capaz de extraer y limpiar los distintos comentarios de Twitter para su posterior análisis.
- Desarrollo doble de un software que emplee herramientas NLP para el análisis y detección de ofensividad en los comentarios extraidos anteriormente, tanto en Python como en Spark.

1.3 Antecedentes y estado del arte

Surge la necesidad de crear este tipo de máquinas debido al constante crecimiento exponencial al que se ven sometidos los datos que nos rodean y el objetivo de las empresas de sacar provecho de éstos y convertirlos en información valiosa.

Actualmente contamos herramientas muy avanzadas para el tratamiento de dicho volumen como Spark el cual es un framework que se sigue actualizando a día de hoy y que es compatible tanto con los primeros software Big Data como con los que siguen apareciendo hoy en día.

Además el conocimiento de éstas arquitecturas se va desvaneciendo debido a que los desarrolladores solo aprenden el software o el lenguaje necesario para llevar a cabo un proyecto de éstas magnitudes al basarse en servicios cloud o servicios on-premise que ya estaban montados anteriormente por otros equipos.

1.4 Descripción de la situación de partida

Partimos de un contexto en el que sabemos que hay software especializado para Big Data, que el volumen de datos no es manejable por el hardware o software que empleamos normalmente en nuestros desarrollos y del pensamiento que se hacen necesarias grandes inversiones de dinero por parte de las empresas para que un proyecto de ésta índole pueda seguir adelante.

También a esto se le suma un desconocimiento de todo el ecosistema de tecnologías que rodean un proyecto Big Data, ya que éste ámbito no solo hace referencia a la información en sí sino también a toda una metodología de trabajo especialmente diseñada para ello.

1.4.1 Descripción del entorno actual

La intención del proyecto es desmentir completamente la situación de partida y ofrecer una formación clara sobre la arquitectura de éstos sistemas, las metodologías necesarias para operar con ellos y el conocimiento suficiente para iniciar o emplear software especializado en Big Data.

Partimos de una situación en la que queremos más adelante analizar un gran volumen de datos utilizando técnicas de NLP utilizando el lenguaje Python. Para trabajar con pequeños ficheros en nuestro local puede ser más que suficiente, pero en el momento en el que decidimos escalar el proyecto, nos encontramos descargando miles de comentarios de las redes sociales en tiempo real y éstas técnicas descritas dejan de ser suficientes para realizar dicho análisis en un tiempo prudente.

Es en ese momento, en el que podemos emplear nuestro clúster, diseñado para poder correr aplicaciones implementadas con Spark que permitirán no solo el trabajo distribuido de éste gran volumen sino que además reducirá significativamente los tiempos de ejecución al paralelizar todo el procesamiento necesario.

En las pruebas de benchmarking, notamos las diferencias en tiempo:

MÓDULO	PYTHON	SPARK	UNIDAD
Traducción de comentarios	175	64	s
Algoritmo con Lexicón	4	8	s
Algoritmo de entrenamiento supervisado	16	8	s

Tabla 1.1

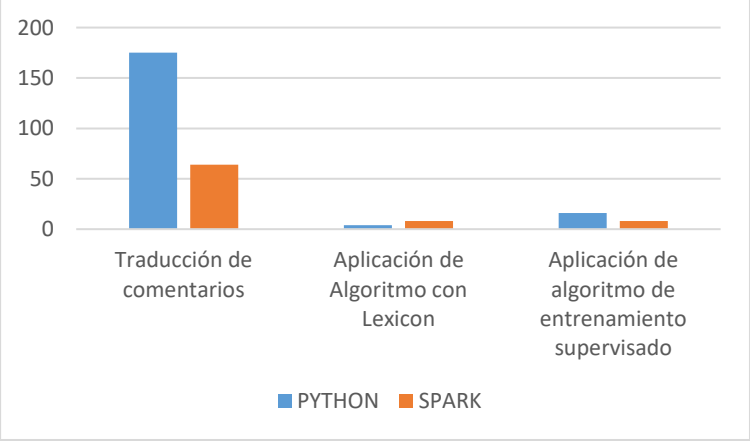


Ilustración 1.1

1.4.2 Resumen de las deficiencias y carencias identificadas

Con la finalización del proyecto, hemos sido capaces de mejorar los tiempos de ejecución en el análisis de los comentarios y además, contamos con una máquina que podrá seguir ejecutando aplicaciones similares y asimilando nuevas piezas de software. También hemos podido notar que la mayor parte del tiempo de ejecución se invierte en el arranque de la máquina y la inicialización de la infraestructura de Spark, pudiendo no ser la mejor solución en el caso de ejecuciones breves, rápidas o de volúmenes de datos reducidos, ya que empleará más tiempo en prepararse e iniciar el programa que la ejecución en sí.

Además, es posible que en el contexto en el que una empresa decide hacer una inversión para un proyecto Big Data, no sea muy satisfactorio para un cliente todo el proceso de montaje e instalación de la máquina, ya que éste no verá cambios notables ni obtendrá resultados hasta que no finalice dicha instalación y se comiencen a ejecutar los procesos Spark.

Sin embargo, si conoce y entiende dicha barrera de entrada, quedará muy satisfecho con la cantidad de valiosa información que se va a generar en poco tiempo y con como ese sistema es capaz de seguir creciendo y funcionando al máximo de su potencial todo el tiempo.

1.5 Requisitos iniciales

El sistema debe estar basado en tecnologías que permitan el procesamiento distribuido de grandes volúmenes de datos. El clúster Big Data, compuesto por Hadoop y Spark, debe ser capaz de manejar aplicaciones que procesen datos en tiempo real. Además, el software desarrollado para scrapping y análisis NLP debe ser escalable y estar optimizado para ejecutarse tanto en un entorno local (Python) como en Spark. Los requisitos incluyen también la capacidad del sistema para gestionar adecuadamente el almacenamiento y la limpieza de datos extraídos, así como para identificar y clasificar comentarios ofensivos de manera precisa.

Además, el clúster debería estar orientado a las necesidades de la empresa, cliente o proyecto, teniendo instalado el posible software específico para el tratamiento de datos.

1.6 Alcance

Entre los entregables de un proyecto de éstas características figuran:

- Acceso a la máquina del clúster o nodo dónde esté instalado para el uso y aprovechamiento del cliente.
- Documentación con todas las especificaciones de la máquina, software y arquitectura.
- Manual de uso de la máquina y tutorial de inicialización de aplicaciones.
- Proyecto o repositorio con los scripts de análisis NLP tanto en su versión en Python como Spark y el software de scrapping junto a su documentación.

1.7 Hipótesis y restricciones

El TFT se define como una asignatura de 12 créditos, lo que supone que la duración total del proyecto será de 300 horas, incluyendo todas las etapas del ciclo de vida, con la excepción del mantenimiento. Por consiguiente, la principal restricción aplicable es la limitación de la duración del trabajo.

1.8 Estudio de alternativas y viabilidad

Es posible ejecutar Spark en modo standalone, lo cual habría permitido ejecutar Spark en nuestro local y realizar la comparativa con Python igualmente. El problema es que no estaríamos explorando todo el potencial de Spark como programa (ya que la máquina sobre la que correría sería una de menor calidad a la que hemos diseñado) y además no tendríamos todo el feedback que podemos obtener de una ejecución gracias al resto de softwares como Yarn que interactúan en nuestro Clúster. Además, realizando el montaje de la máquina también somos capaces de proporcionar un entorno que permite varias ejecuciones por los desarrolladores y al mismo tiempo paralelizar otras funciones como el mantenimiento y extensión del sistema de almacenamiento HDFS.

Queda completamente descarta la idea del standalone cuando además pensamos en la escalabilidad del proyecto, ya que la característica principal de realizar el montaje de la máquina con distintos nodos en arquitectura piramidal es que permite seguir añadiendo nodos y seguir creciendo según las necesidad del proyecto crecen también. En nuestro local tenemos un límite físico que no podremos sobrepasar nunca mientras el hardware que usemos sea el mismo.

1.9 Descripción de la solución propuesta

Se ha propuesto una solución que supera con creces las alternativas y se justifica por la siguientes razones:

- Permitir la escalabilidad. Si el sistema requiere de más hardware o más potencia, solo tenemos que anidar un nodo más a nuestra máquina y ya habremos multiplicado nuestra potencia.
- Permitir la mejor versión de Spark. Al estar Spark completamente orientado a la ejecución en paralelo y distribuida, la arquitectura del clúster permite sacar su mayor potencial.
- Asegurar resultados en tiempos justificables. Gracias a toda la infraestructura y software instalado en la máquina, podemos asegurar a nuestros clientes que las tareas que soliciten o los procesos que soliciten, serán ejecutados y entregados en la menor cantidad de tiempo posible.

1.10 Material y métodos

El desafío principal era el de no solo montar y poner en funcionamiento la máquina sino además permitir una guía y/o manual que permita su reproducción y uso. Esta tarea fue realizada mediante el uso de la documentación oficial de las tecnologías Hadoop y Spark. Más adelante, teníamos que desarrollar el software en Python para el scrapping web el cual fue realizado con metodología agile y por último los softwares de análisis NLP con Python y Spark los cuales siguieron también metodologías agile y además un control de versiones que permitió el paralelismo entre ambos proyectos. Todo esto realizado en el mismo equipo informático, optimizando nuestro aprovechamiento de recursos.

1.11 Tecnologías utilizadas

Según el bloque de nuestro proyecto se han utilizado unas tecnologías u otras, y se pueden enumerar de la siguiente manera:

- Clúster Big Data:
  - **Hadoop**. Es la tecnología principal que ha permitido levantar el sistema de almacenamiento HDFS y además orquestar el resto de tecnologías Big Data bajo la arquitectura de la máquina.
  - **HDFS**. Una máquina Big Data necesita de un sistema de almacenamiento que permita el tratamiento distribuido de ficheros y HDFS encaja a la perfección con nuestra arquitectura.
  - **Spark**. El framework que nos va a permitir ejecutar todos nuestros procesos y/o software en la máquina y aprovechar al máximo su potencial.
  - **Yarn**. El software que nos permitirá orquestar todos los procesos que lancemos y que nos ofrecerá la máxima información posible sobre éstos para su futura optimización.
- Srapping software:
  - **Python**. Hemos elegido este lenguaje al ser muy flexible e integrar las librerías necesarias para nuestro propósito. Además permite su ejecución de manera portable en cualquier sistema que pueda instalar el lenguaje. Dentro de este proyecto destacamos el uso de librerías como:
    - **CSV**. Librería que nos ha permitido el tratamiento de los comentarios tanto en su carga como almacenamiento. Además de ofrecer una manera clara y concisa de presentar los resultados.
    - **Selenium**. Gracias a esta librería hemos podido simular la actividad humana en un navegador y extraer todos los comentarios que serían analizados más adelante.

- Análisis con NLP (Python):
    - **Pandas**. Librería que nos ha permitido el agrupamiento de los datos para su tratamiento en una estructura de datos dirigda a dicho tratamiento. En el caso de Spark, como ya cuenta con su propia estructura para esto como son los Dataframes, esto no se hace necesario, pero Pandas nos ha permitido suplir esto perfectamente.
    - **NLTK**. Librería que contiene los modelos que hemos entrenado en la detección supervisada para el análisis de comentarios como los tokens y dependencias necesarias para dicho análisis.
- Análisis con NLP (Spark):
    - **Pyspark**. Es la librería de Spark que nos permite interactuar con éste utilizando el lenguaje de programación Python, ya que normalmente estaríamos obligados a hacerlo en Scala. Ya uno de los puntos del proyecto era también el benchmarking entre usar o no Spark, era necesario partir del mismo lenguaje de programación.

Los últimos 3 bloques comparten tecnologías como Python y Spark que se explican anteriormente.

1.12 Metodología de desarrollo de software

Para el desarrollo del software, se ha optado por una metodología ágil, debido a su flexibilidad y adaptabilidad a los cambios. Esta metodología permite dividir el trabajo en sprints cortos, cada uno enfocado en un objetivo específico, como la instalación del clúster, el desarrollo del scrapping, y la integración del análisis NLP. Cada sprint incluye fases de desarrollo, pruebas y corrección, lo que garantiza que el proyecto avance de forma controlada y con revisiones constantes. La metodología ágil también facilita la incorporación de mejoras y ajustes a medida que surjan nuevas necesidades o problemas. Lo cual es determinante cuando manejamos una arquitectura tan compleja como la que presentamos aquí.

1.13 Análisis de riesgos

Alguno de los riesgos que conocemos sobre este proyecto son los siguientes:

- Problemas con la instalación y configuración del clúster Big Data, lo cual podría retrasar el desarrollo. El plan de contingencia en este caso es la creación y distribución del manual de montaje y funcionamiento que persigue este proyecto.
- La calidad de los datos no es la esperada. Dicho problema reduciría drásticamente la calidad de nuestro proyecto en general. Como plan de contingencia, podemos seguir utilizando el script de Scrapping para seguir iterando la red hasta que la calidad del dato sea suficiente.

A todos los posibles problemas y riesgos, les sigue una prueba para controlar que la tarea se desarrolla adecuadamente y que coincide con la calidad del entregable que el cliente espera.

1.14 Estimación del tamaño y esfuerzo

Ya que el presente proyecto es un TFT, no existen restricciones de tipo económico, sino de tipo temporal (un número aproximado de horas). Por consiguiente, los cálculos de tamaño del proyecto están supeditados el tiempo disponible. En cuanto al esfuerzo, se dispone de tan un solo efectivo (la persona autora del trabajo).

1.15 Planificación temporal

Se estima que el proyecto requerirá aproximadamente 300 horas de trabajo, que se dividirán en las siguientes fases:

- Instalación y configuración del clúster (150 horas)
- Desarrollo del software de scrapping (50 horas)
- Implementación del análisis NLP en Python (30 horas)
- Implementación del análisis NLP en Spark (30 horas)
- Pruebas y validación (40 horas).

El tamaño del proyecto se medirá en términos de las líneas de código desarrolladas, el número de comentarios analizados y el correcto funcionamiento de la máquina montada junto con todos sus módulos. Estas métricas permitirán evaluar el avance y ajustar los plazos si es necesario.

Diagrama de Gantt

Ilustración 1.25.1

Ilustración 1.35.2

Ilustración 1.45.3

1.16 Presupuesto

- **Costos de hardware**: Para la implementación del clúster de Big Data, se necesita invertir en hardware para el montaje de la máquina. Se estima un costo de **1,000 €** para adquirir servidores con especificaciones suficientes para ejecutar Hadoop y Spark de manera eficiente. Este coste cubre la adquisición de componentes como CPU, RAM y discos duros necesarios para garantizar un rendimiento adecuado.
- **Costos de software**: Por suerte para el cliente, todo el software empleado es software libre sin ningún costo asociado. Se utilizarán herramientas de software de código abierto como Hadoop, Spark, Python y bibliotecas como NLTK para el análisis NLP. Sin embargo, se reserva una partida de **500 €** para cubrir posibles servicios en la nube o licencias que pudieran necesitarse en fases posteriores o para realizar pruebas a gran escala.
- **Costos de desarrollo**: Se estima que el proyecto requerirá un total de 300 horas de trabajo, repartidas en la instalación del clúster, desarrollo del software de scrapping, implementación del análisis NLP y test. Asumiendo un costo de **15 €/hora** (sueldo medio de un desarrollador Big Data senior en España), el total en tiempo de desarrollo asciende a **4,500 €.**

Por posibles imprevistos, se reserva una partida de **200 €** para cubrir cualquier imprevisto, ya sea relacionado con el hardware (fallas técnicas o ampliación de componentes), servicios adicionales (almacenamiento en la nube), o la adquisición de datos para el análisis.

El total de este presupuesto asciende por lo tanto a **6,200 €.**

CONCEPTO	COMENTARIO	UNIDADES	PRECIO
Hardware	Compra de CPUs, RAM, SSDs …	1	1,000 €
Software	Reserva de partida para futura inversión en Cloud	1	500 €
Desarrollo	15 € / hora	300	4,500 €
Imprevistos	Posibles fallos y futuras mejoras	1	200 €
TOTAL			**6,200 €**

Tabla 1.26.1

1.16.1 Mecanismos de control de calidad

El aseguramiento de la calidad en la redacción del trabajo implica la verificación de la completitud (falta de omisiones), la integridad de la documentación, así como una redacción clara, concisa y entendible por todos los participantes e interesados en dicho trabajo.

Al estar el trabajo desarrollado por una sola persona y verificado por un/a tutor/a, no es necesario llevar un documento de control de edición y revisión de la documentación. De esta forma, se han utilizado mecanismos básicos para la verificación de la integridad y completitud, que incluyen un control de versiones a nivel de documento y un control sencillo de la trazabilidad de especificaciones y requisitos.

2 DISEÑO INICIAL

El diseño inicial del sistema se basa en una arquitectura distribuida, utilizando el framework Hadoop para gestionar el almacenamiento y procesamiento de datos a gran escala, y Spark como motor de procesamiento en memoria para mejorar el rendimiento. El sistema se compone de varias capas:

- **Capa de almacenamiento**: Utiliza el sistema de archivos distribuidos de Hadoop (HDFS) para almacenar de manera eficiente los datos. HDFS permitirá la gestión de grandes volúmenes de datos, distribuyéndolos entre múltiples nodos del clúster y garantizando la redundancia y disponibilidad de los datos. Además, se utilizará la capacidad de replicación de HDFS para asegurar la resiliencia frente a fallos de nodos individuales.

- **Capa de procesamiento**: La capa de procesamiento está gestionada por Apache Spark, que se encargará de procesar los datos extraídos utilizando algoritmos de procesamiento de lenguaje natural (NLP). Spark permite la paralelización de tareas y el procesamiento en memoria, lo que lo convierte en la solución ideal para analizar grandes volúmenes de datos textuales en tiempo real. Se integrarán las librerías Spark NLP y NLTK para aplicar técnicas como el análisis de sentimientos y la detección de lenguaje ofensivo.

- **Capa de extracción de datos**: El sistema de extracción de datos utilizará herramientas como Selenium para realizar scrapping de comentarios en Twitter. Esta capa se encargará de recolectar datos relevantes, almacenarlos y garantizar que los datos estén limpios y estructurados adecuadamente para su posterior análisis. Garantizaremos la limpieza de los datos con scripts de limpieza y formato incluidos en el mismo scrapping, ya que el dato que se llega a almacenar es final.

- **Capa de análisis NLP**: Se implementará un sistema de análisis de comentarios ofensivos utilizando técnicas de NLP. Primero, los comentarios serán preprocesados (limpieza, tokenización, eliminación de palabras irrelevantes) y luego se aplicarán modelos de clasificación para detectar comentarios ofensivos. Estos modelos se ejecutarán tanto en Python como en Spark, lo que permitirá comparar la eficiencia de ambos enfoques.

2.1 Especificaciones del sistema

El sistema que planteamos posee la siguiente estructura en un diseño inicial:

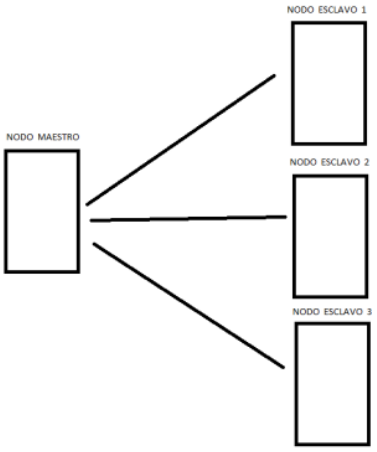


Ilustración 2.1.1

El sistema debe estar compuesto por un clúster Big Data basado en Hadoop y Spark, con capacidad para gestionar grandes volúmenes de datos en tiempo real. Entre los requisitos funcionales se encuentran:

- Capacidad para ejecutar aplicaciones de Spark en múltiples nodos.
- Almacenamiento distribuido de datos utilizando el sistema HDFS de Hadoop.
- Integración con herramientas de análisis de lenguaje natural (NLP) para procesar los datos extraídos de Twitter.

Además, el sistema debe ser escalable, permitiendo agregar más nodos al clúster según sea necesario. Los requisitos no funcionales incluyen un rendimiento óptimo (tiempo de respuesta adecuado para el procesamiento de datos masivos), seguridad en el acceso a los datos y facilidad de uso en la implementación y administración del clúster.

2.2 Análisis y diseño del sistema

El análisis del sistema parte de los requisitos iniciales, donde se identifican los principales problemas que el sistema debe resolver. En este contexto, el sistema debe abordar dos grandes retos: el manejo eficiente de datos masivos en tiempo real y la aplicación de técnicas avanzadas de NLP para analizar comentarios.

- **Problemática**: La creciente cantidad de datos generados en redes sociales, como Twitter, plantea el desafío de extraer, procesar y analizar comentarios de forma eficiente. Las soluciones tradicionales, que procesan los datos de manera local, no pueden escalar adecuadamente cuando el volumen de datos aumenta exponencialmente.
- **Objetivo del sistema**: Desarrollar un sistema capaz de realizar análisis masivos de datos textuales en un entorno distribuido. El sistema debe ser escalable, con capacidad para procesar datos en tiempo real y aplicar modelos de NLP que identifiquen comentarios ofensivos con una alta precisión. Además, debe integrar un proceso de scrapping para la recolección automatizada de los datos desde Twitter.

Para la parte del diseño de la arquitectura, se propone la arquitectura por capas enumeradas en apartados anteriores y que vamos a detallar en la sección de diseño arquitectónico del sistema.

- **Diseño arquitectónico del sistema**. El sistema sigue una arquitectura basada en un **clúster de Big Data** compuesto por Hadoop y Spark, donde Hadoop se utiliza para el almacenamiento distribuido y Spark como motor de procesamiento de datos. Esta arquitectura se divide en las siguientes capas:
    - **Capa de almacenamiento distribuido (HDFS)**: Hadoop se encargará de almacenar los datos de manera redundante en los diferentes nodos del clúster. Los comentarios extraídos de Twitter mediante scrapping se almacenarán en HDFS antes de ser procesados por Spark. El sistema garantiza la escalabilidad mediante la replicación de datos y su distribución en múltiples nodos.

  - **Capa de procesamiento (Spark)**: Spark se encargará de procesar los datos almacenados en HDFS. El procesamiento se llevará a cabo utilizando modelos de NLP para identificar patrones de lenguaje ofensivo. La principal ventaja de Spark es su capacidad para ejecutar tareas en memoria, lo que mejora significativamente el rendimiento en comparación con los sistemas tradicionales basados en disco.
  - **Capa de extracción de datos (Scrapping)**: Esta capa incluye el software encargado de recolectar comentarios desde Twitter. Utilizando Selenium, los datos se extraen y se estructuran para su posterior análisis.

- **Diseño de bases de datos**. Dado que el sistema se basa en un procesamiento distribuido, el almacenamiento se realizará en HDFS, que permite almacenar grandes volúmenes de datos no estructurados. Los datos de comentarios se almacenarán en formato **JSON** y **CSV**, que facilita la posterior transformación y análisis. La estructura de los datos incluirá campos clave como el identificador del comentario, el usuario, la fecha de publicación y el contenido textual.
- **Diagramas de casos de uso**. Los principales casos de uso del sistema incluyen:
  - **Scrapping de comentarios**: El sistema recolecta datos de Twitter utilizando técnicas de scrapping.
  - **Procesamiento de datos en Spark**: El sistema distribuye los datos entre los nodos del clúster para realizar análisis en paralelo. El procesamiento incluye limpieza de datos, tokenización y análisis NLP.
  - **Análisis y clasificación de comentarios ofensivos**: Los comentarios se analizan en base a modelos de NLP, y aquellos identificados como ofensivos se etiquetan en la base de datos.
  - **Visualización de resultados**: Los resultados del análisis se almacenan y se muestran al usuario en un formato csv para su posterior visualización.

- **Diagramas de secuencia**. El diagrama de secuencia muestra el flujo de interacción entre los principales componentes del sistema:
  - **Scraper** obtiene los datos y los almacena en HDFS.
  - **ProcesadorSpark** carga los datos desde HDFS y ejecuta las tareas de procesamiento.
  - **AnalizadorNLP** aplica los modelos de clasificación de comentarios ofensivos y los corre en Python.
  - **ProcesadorSpark** corre el analizador NLP en Spark.

3 DESARROLLO

Software PLN – Python

El software diseñado cataloga según si son ofensivos o no los comentarios que reciben las distintas celebridades. Dicha información la hemos obtenido anteriormente con el script de automatización de scrapping. En esta sección no haremos tanto hincapié sobre su funcionamiento ya que el objetivo del proyecto es presentar la capacidad de Spark para el trato masivo de datos y no el software utilizado para la prueba en sí.

DEPENDENCIAS Y LIBRERÍAS

```
# Librería Pandas para cargar los .tsv en DF directamente y .csv
import pandas as pd
import csv
# Librería string, unicode, re para el tratamiento de cadenas de texto
import string
import unicodedata
import re
# Librería random para adiciones de aletoriedad
import random
```

Ilustración 3.1

Importamos las librerías correspondientes para la ingesta y tratamiento del dato.

```
# Importamos y descargamos las dependencias de NLTK para el procesamiento del texto ( en español ya que
# luego traduciremos los comentarios a español )
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.stem.snowball import SnowballStemmer
from nltk.tokenize import RegexpTokenizer
from nltk.tokenize import sent_tokenize
from nltk.tokenize import TreebankWordTokenizer

stemmer = SnowballStemmer("spanish")
tokenizer = RegexpTokenizer('\w+\'?\w*')
spanish_stops = stopwords.words('spanish')
```

Ilustración 3.2

Las librerías de NLTK se utilizan en el procesamiento del lenguaje natural (NLP) en Python. Facilita tareas como tokenización, lematización, stemming y eliminación de stopwords, que son acciones para el tratamiento y análisis del texto.

```
# Importamos sklearn por lo mismo que NLTK
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import svm

# Función de limpieza para los emojis
from cleantext import clean

# Traducción
from deep_translator import GoogleTranslator
```

Ilustración 3.3

Librerías del algoritmo de clustering que agrupa datos en k clusters para el entrenamiento de modelos de detección de ofensividad. Y por último la librería que permite utilizar el traductor de Google y traducir el texto a castellano desde cualquier idioma.

TRADUCCIÓN DE IDIOMA

```
# Módulo de traducción

# Lo primero será la carga de nuestros comentarios y su traducción en nuestro script
with open('influencers_comments_clean.csv', 'r', encoding='utf-8') as read_obj:
    csv_reader = csv.reader(read_obj)
    list_of_csv = list(csv_reader)

# Traducimos el diccionario
dict_coment = {}
for i in list_of_csv:
    if( len(i) >= 2):
        key_dict = i[0]
        coments = i[1].split('|||')
        value = []
        for j in coments:
            translated = GoogleTranslator(source='auto', target='es').translate(j)
            value.append(translated)
        dict_coment[key_dict] = value
```

Ilustración 3.4

Al tratarse de un script que obtiene los datos de Twitter, es fácil que los comentarios estén en otro idiomas como inglés, italiano, alemán… con este módulo aprovechamos la división del fichero que preparamos y cargamos los datos en un diccionario con los comentarios ya traducidos utilizando la librería del GoogleTranslator. La estructura queda como en el script de scrapping anterior y es algo así:

```
VALUE: Siempre cambio eso.
VALUE: Esto es algo que realmente dirían.
VALUE: Como soy gemela idéntica, me encanta esta forma de responder las preguntas que recibimos TODO EL TIEMPO.
```

Ilustración 3.5

Vemos que los comentarios se traducen correctamente.

Ya tenemos nuestro diccionario con los comentarios por clave, ahora vamos a plantear una estructura de datos nueva que nos va a permitir la clasificación de los comentarios y al mismo tiempo mantener la claridad en ésta.

La estructura que tenemos actualmente es la siguiente:

Dic = { ([CLAVE1] , [VALOR1, VALOR2, ...]) , ([CLAVE2] , [VALOR1, VALOR2, ...]) , ... }

Dic = { ([INFLUENCER1] , [COMENTARIO1, COMENTARIO2, ...]) , ([INFLUENCER2] , [COMENTARIO1, COMENTARIO2, ...]) , ... }

Y la estructura a la que queremos pasar es:

Dic = { ([CLAVE1] , [(VALOR1, ETIQUETA1) , (VALOR2, ETIQUETA2) , ...]), ([CLAVE2] , [(VALOR1, ETIQUETA1) , (VALOR2, ETIQUETA2) , ...]), ... }

Dic = { ([INFLUENCER1] , [(COMENTARIO1, ETIQUETA1) , (COMENTARIO2, ETIQUETA2) , ...]), ([INFLUENCER2] , [(COMENTARIO1, ETIQUETA1) , (COMENTARIO2, ETIQUETA2) , ...]), ... }

Para ello, catalogamos todos inicialmente como 'NON' (no ofensivos)

```
final_dict = {}
for k, v in dict_coment.items():
    l = []
    for i in v:
        value = (i , 'NON')
        l.append(value)
    final_dict[k] = l
```

Ilustración 3.6

CARGA DE FICHEROS

```
# Cargamos los distintos archivos necesarios para la ejecución de la práctica

# - Fichero .tsv de muestra para el entrenamiento del modelo supervisado
train = pd.read_csv('train.tsv', sep='\t', encoding='utf-8')

# - Fichero .txt que contiene el lexicon de insultos
share=open('SHARE.txt', "r", encoding='utf-8')
insultos=share.read().split('\n')
```

Ilustración 3.7

El fichero train es un fichero que ya contiene una serie de comentarios catalogados correctamente como ofensivos y no-ofensivos que va a permitir el entrenamiento de un modelo que usaremos más adelante para catalogar nuestros comentarios traducidos.

El fichero share contiene un conjunto de insultos que nos permitirán aplicar otro algoritmo de clasificación basado en lexicón para la detección de ofensividad. En la etapa final, se cruzarán los resultados de ambos algoritmos, supervisado y no supervisado.

LIMPIEZA DE TEXTO

```
# ALGORITMO DE LIMPIEZA
def CleanText(text):
  # Preparamos traductor
  trans_tab = dict.fromkeys(map(ord, u'\u0301\u0308'), None)
  # Eliminamos acentos
  text = unicodedata.normalize('NFKC', unicodedata.normalize('NFKD', text).translate(trans_tab))
  # Eliminamos enlaces usando expresiones regulares
  text = re.sub('https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
  # Eliminamos etiquetas de otros usuarios o tópicos
  text = re.sub("@[A-Za-z0-9_]+","", text)
  text = re.sub("#[A-Za-z0-9_]+","", text)
  # Quitamos los emojis
  emoji_pattern = re.compile("["
          u"\U0001F600-\U0001F64F"
          u"\U0001F300-\U0001F5FF"
          u"\U0001F680-\U0001F6FF"
          u"\U0001F1E0-\U0001F1FF"
                              "]+", flags=re.UNICODE)
  text = emoji_pattern.sub(r'', text)
  # Lo ponemos en minúscula para tratamiento regular
  text = text.lower()
  # ADD : QUITAR SIGNOS DE INTERROGACION Y EXCLAMACION
  text = text.replace("?","")
  text = text.replace("¿","")
  text = text.replace("!","")
  text = text.replace("¡","")

  return text
```

Ilustración 3.8

Se puede dar que en los comentarios, encontremos caracteres que no aportan al peso del comentario, como pueden ser signos de exclamación y/o interrogación, emojis (aunque en otro tipo de algoritmos, éstos se pueden usar también para la catalogación de comentarios) y espacios o símbolos extraños.

ALGORITMO DE CATALOGACIÓN SUPERVISADO

```
def EntrenamientoSupervisado(trainDF):
  conta = 0
  tag = []
  text = []
  # Iteramos los comentarios del DF de entrenamiento a la vez que el label (tag)
  for i in trainDF['comment']:
    i = CleanText(i)
    aux= ''
    # Clasificamos los tags de los comentarios según si son ofensivos (negativos) o no (positivos)
    if trainDF['label'][conta] == 'OFF':
      tag.append('negative')
    else:
      if ' no ' in i: # Valoramos la negatividad
        tag.append('negative')
      else:
        # Es más probable que si la  que recibe el comentario es mujer, el comentario sea ofensivo
        if trainDF['influencer_gender'][conta] == 'woman':
          if random.random() < 0.10:
            tag.append('negative')
          else:
            tag.append('positive')
        else:
          tag.append('positive')
    # Se puede dar que un comentario esté compuesto por más de una frase, lo separamos
    frases=sent_tokenize(i.lower(), language="spanish")
    # Las recorremos
    for frase in frases:
      token=TreebankWordTokenizer().tokenize(frase)
      tokens=''
      for palabra in token:
        #palabra = lemmatizer.lemmatize(palabra)
        if palabra not in spanish_stops and palabra not in string.punctuation:
          tokens += palabra
      aux+=" " + str(tokens)
    text.append(aux)
    conta+=1
  vectorizer = TfidfVectorizer()
  X = vectorizer.fit_transform(text)
  model = svm.SVC()
  model.fit(X, tag)
  return model,vectorizer
```

Ilustración 3.9

Este algoritmo utiliza los comentarios ya previamente catalogados para entrenar un modelo que permitirá la detección de ofensividad en nuevos comentarios. Utiliza las librerías mencionadas al inicio de tokenización para segmentar el comentario y procesarlo según las pautas del idioma (por eso traducimos al inicio del software) y aporta la información al modelo.

Finalmente, se encarga de procesar los comentarios y entrenar un modelo de clasificación de soporte vectorial (SVM) para identificar si los comentarios son ofensivos o no.

ALGORITMO DE CATALOGACIÓN NO SUPERVISADO (LEXICON)

```
def Lexicon(DictComentarios,ArrayInsultos):
  final_dict_l = {}
  for k,v in DictComentarios.items():
    value = []
    for i in v:
      flag = False
      for j in ArrayInsultos:
        temp = i[0].split()
        if [palabra for palabra in temp if palabra == j]:
            flag = True
            tup = ( i[0] , 'OFF' )
        elif (not flag):
            tup = ( i[0] , 'NON' )
      value.append(tup)
    final_dict_l[k] = value

  return final_dict_l
```

Ilustración 3.20

Este algoritmo es mucho más sencillo en contraste con el anterior, ya que a diferencia de ese, éste parte de un conjunto completamente informado como es el lexicón de insultos. Básicamente, comprobamos si alguno de los comentarios contiene uno de los tantos insultos que hemos recopilado, y si se da la coincidencia lo catalogamos de ofensivo, y en caso contrario, no.

Esto lo hacemos respetando la nueva estructura que hemos creado de 'diccionario etiquetado'.

EJECUCIÓN DE LOS MÉTODOS DE CATALOGACIÓN

```
# LIMPIEZA
for k,v in final_dict.items():
    l = []
    for i in v:
        if(len(i) == 2 and i[0] != None):
            t = CleanText(i[0])
            val = (t,i[1])
            l.append(val)
    final_dict[k] = l
```

Ilustración 3.31

Lo primero que hacemos, es aplicar nuestro algoritmo de limpieza a los distintos comentarios, iterando por nuestra estructura de datos específica.

```
# LEXICON

# --- Cargamos insultos
ArrayInsultos = []
for i in insultos:
  ArrayInsultos.append(i)

# --- Aplicamos insultos
final_dict_lexicon = Lexicon(final_dict,ArrayInsultos)
```

Ilustración 3.42

Cargamos nuestra lista de insultos, y a partir de eso aplicamos nuestro algoritmo de Lexicon definido anteriormente para catalogar y almacenar un nuevo diccionario que usaremos más adelante para la hibridación el supervisado.

```
# SUPERVISADO

# --- Entrenamos modelo utilizando el fichero de testing
m, v = EntrenamientoSupervisado(train)

# --- Aplicamos supervisado
def Supervisado(DictComentarios):
  final_dict_s = {}
  for keys, values in DictComentarios.items():
    value = []
    for i in values:
      vect = v.transform([i[0]])
      prediccion = m.predict(vect)
      if prediccion == "positive":
        tup = ( i[0] , 'NON' )
      else:
        tup = ( i[0] , 'OFF' )
      value.append(tup)

    final_dict_s[keys] = value

  return final_dict_s


final_dict_supervisado = Supervisado(final_dict)
```

Ilustración 3.53

Aplicamos nuestro algoritmo supervisado que tiene que entrenar previamente el modelo como comentábamos con el conjunto de datos ya catalogado. Una vez entrenado, aplicamos el modelo para la predicción en los distintos comentarios a través de la iteración de nuestro diccionario, y aprovechamos para generar un nuevo diccionario de salida como hicimos en el caso del Lexicon.

HIBRIDACIÓN

```
# --- HIBRIDACIÓN ---
# Cuando el de lexicon indique que si es OFF pero el supervisado diga que no,
# recorrer el comentario en búsqueda de insultos, y si contiene 2 o más, se clasifica de ofensivo
final_dict_hybrid = {}
AlgoritmoHibrido = []
posicion = 0
for (k1,v1), (k2,v2), (k3,v3) in zip(final_dict_lexicon.items(), final_dict_supervisado.items(), final_dict.items()):
  value = []
  for i, j, k in zip(v1,v2,v3):
    add = True
    # En el caso de que tanto como el lexicon como el supervisado coincidan, estamos bastante seguros de que es ofensivo //////o no
    if i[1] == 'OFF' and j[1] == 'OFF' and add:
      add = False
      tup = ( i[0] , 'OFF' )
    # En caso de discordia o de que no se haya catalogado directamente
    elif i[1] == 'NON' and j[1] == 'OFF' and add: #Tiene prioridad el supervisado, ya que da mejores resultados
      add = False
      tup = ( i[0] , 'OFF' )
    elif i[1] == 'OFF' and j[1] == 'NON' and add: # Se ha detectado un insulto pero no se ha catalogado como ofensivo por el supervisado
      add = False
      n_insultos = 0
      for t in ArrayInsultos:
        if [palabra for palabra in k[0].split() if palabra == t]:
          n_insultos+=1
      if n_insultos>=2:
        tup = ( i[0] , 'OFF' )
      if n_insultos<2:
        tup = ( i[0] , 'NON' )
    elif i[1] == 'NON' and j[1] == 'NON' and add:
      add = False
      tup = ( i[0] , 'NON' )
    posicion+=1
    value.append(tup)
  final_dict_hybrid[k1] = value
```

Ilustración 3.64

Ahora que tenemos el mismo conjunto de datos catalogado por algoritmos completamente distintos, podemos cruzar ambos conjuntos para quedarnos con la mejor versión de ambos y ofrecer un resultado de calidad. Básicamente en este algoritmo nos dedicamos a iterar los 3 diccionarios en paralelo, el producido por el lexicón, el producido por el supervisado y el que teníamos originalmente, de manera que siguiendo los criterios definidos en los comentarios del código cribamos con cual quedarnos y cual no. Finalmente devolvemos un diccionario que es el resultado de la hibridación de ambos conjuntos.

PUBLICACIÓN

Ahora podemos obtener información relevante sobre los influencers como :

- Calculo de 'Hate' por influencer:

```
######################################################################
INFLUENCER: COLE SPROUSE
Recibe un total de 18 de los cuales son comentarios negativos: 3
Porcentaje de Hate: 17 %
######################################################################
INFLUENCER: CHIARA FERRAGNI
Recibe un total de 19 de los cuales son comentarios negativos: 5
Porcentaje de Hate: 26 %
######################################################################
INFLUENCER: CAMILA COELHO
Recibe un total de 2 de los cuales son comentarios negativos: 0
Porcentaje de Hate: 0 %
######################################################################
```

Ilustración 3.15

- Influencers más "queridos":

```
MOST LOVE INFLUENCERS
1 -> INFLUENCER: CAMILA COELHO ===> 0 %
2 -> INFLUENCER: LAUREN CONRAD ===> 0 %
3 -> INFLUENCER: JULIE SARINANA ===> 0 %
4 -> INFLUENCER: OLIVIA PALERMO ===> 0 %
5 -> INFLUENCER: ALEXA CHUNG ===> 0 %
6 -> INFLUENCER: KAREN WAZEN BAKHAZI ===> 0 %
7 -> INFLUENCER: CAROLINE DAUR ===> 0 %
8 -> INFLUENCER: DANIELLE BERNSTEIN ===> 0 %
9 -> INFLUENCER: LENA PERMINOVA ===> 0 %
10 -> INFLUENCER: ADAM GALLAGHER ===> 0 %
```

Ilustración 3.76

- Influencers más "odiados":

```
MOST HATED INFLUENCERS
1 -> INFLUENCER: JESSICA STEIN ===> 50 %
2 -> INFLUENCER: AIMEE SONG ===> 39 %
3 -> INFLUENCER: HUDA KATTAN ===> 38 %
4 -> INFLUENCER: LEONIE HANNE ===> 33 %
5 -> INFLUENCER: SIMONE GIERTZ ===> 32 %
6 -> INFLUENCER: JACKIE AINA ===> 28 %
7 -> INFLUENCER: MARQUES BROWNLEE ===> 28 %
8 -> INFLUENCER: CHIARA FERRAGNI ===> 26 %
9 -> INFLUENCER: LOUIS COLE ===> 25 %
10 -> INFLUENCER: PAUL'S HARDWARE ===> 25 %
```

Ilustración 3.87

- Influencers más famosos y los menos conocidos: (cantidad comentarios)

```
BIGGESTS INFLUENCERS
1 -> INFLUENCER: JESSICA STEIN ===> 25
2 -> INFLUENCER: AIMEE SONG ===> 25
3 -> INFLUENCER: HUDA KATTAN ===> 25
4 -> INFLUENCER: LEONIE HANNE ===> 23
5 -> INFLUENCER: SIMONE GIERTZ ===> 23
6 -> INFLUENCER: JACKIE AINA ===> 22
7 -> INFLUENCER: MARQUES BROWNLEE ===> 22
8 -> INFLUENCER: CHIARA FERRAGNI ===> 21
9 -> INFLUENCER: LOUIS COLE ===> 21
10 -> INFLUENCER: PAUL'S HARDWARE ===> 21
```

Ilustración 3.98

```
SMALLEST INFLUENCERS
1 -> INFLUENCER: JESSICA STEIN ===> 0
2 -> INFLUENCER: AIMEE SONG ===> 1
3 -> INFLUENCER: HUDA KATTAN ===> 1
4 -> INFLUENCER: LEONIE HANNE ===> 1
5 -> INFLUENCER: SIMONE GIERTZ ===> 1
6 -> INFLUENCER: JACKIE AINA ===> 1
7 -> INFLUENCER: MARQUES BROWNLEE ===> 1
8 -> INFLUENCER: CHIARA FERRAGNI ===> 1
9 -> INFLUENCER: LOUIS COLE ===> 1
10 -> INFLUENCER: PAUL'S HARDWARE ===> 1
```

Ilustración 3.109

Software PLN – Spark

Dado que el código es muy similar al original en Python, señalaremos solo aquellas partes de especial interés y relacionadas con el cambio de paradigma que ofrece Spark. Comenzamos por la importación de funciones y librerías propias de Spark que nos ayudarán en la ejecución del script.

DEPENDENCIAS Y LIBRERIAS

```
from pyspark.sql.functions import udf, col, collect_list, monotonically_increasing_id
from pyspark.sql.types import StringType, ArrayType
from pyspark.sql import Row
from collections import OrderedDict
from pyspark.sql import SparkSession
from pyspark import SparkConf, SparkContext
from pyspark.sql.window import Window
from pyspark.sql import Row, functions as F
from pyspark.sql.functions import col, row_number
from pyspark.sql.functions import split, explode
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField, StringType, IntegerType
```

Ilustración 3.20

INICIALIZACIÓN DE LA SESIÓN DE SPARK

```
conf = SparkConf().setMaster("local")
sc = SparkContext(conf = conf)

spark = SparkSession.builder \
    .appName("PLNSpark") \
    .master("local[*]") \
    .getOrCreate()
```

Ilustración 3.21

El propósito de este código es configurar y crear un entorno de ejecución para una aplicación Spark en un entorno local. Se configuran tanto `SparkConf` y `SparkContext` (para compatibilidad con versiones anteriores de Spark) como `SparkSession` (la API unificada y recomendada a partir de Spark 2.0). Esto permite ejecutar operaciones distribuidas y análisis de datos utilizando todos los núcleos disponibles de la máquina local. Este es un paso fundamental para cualquier aplicación que quiera utilizar Spark para procesamiento de datos en memoria de manera distribuida y paralela.

Además es el objeto sobre el que lanzaremos las operaciones de Spark más importantes, como pueden ser la carga de ficheros, la inicialización de Dataframes o la escritura en tablas.

CARGA DEL DATAFRAME DE COMENTARIOS

```
dict_Dataframe = spark.read.csv("sparkDataframe.csv")
dict_Dataframe.show(truncate=False)

# Ya lo tenemos como un DataFrame, pero su formato aún no es el adecuado para su tratamiento
def rename_columns_with_first_row(df):
    # Seleccionar la primera fila
    first_row = df.head(1)[0]

    # Extraer los valores de la primera fila como nuevos nombres de columna
    new_column_names = list(first_row)

    # Crear un DataFrame con índices para poder filtrar la primera fila
    window_spec = Window.orderBy(col(df.columns[0]))  # Elige una columna para ordenar
    df_with_index = df.withColumn("row_number", row_number().over(window_spec))

    # Filtrar la primera fila
    df_wo_first_row = df_with_index.filter(col("row_number") > 0).drop("row_number")

    # Renombrar las columnas del DataFrame
    for old_name, new_name in zip(df.columns, new_column_names):
        df_wo_first_row = df_wo_first_row.withColumnRenamed(old_name, new_name)

    df_to_return = spark.createDataFrame([df_wo_first_row.collect()[0]])

    return df_to_return

df_renamed = rename_columns_with_first_row(dict_Dataframe)
df_renamed.show(truncate=False)
```

Ilustración 3.22

Gracias a que teníamos los comentarios almacenados en un .csv podemos cargarlos directamente en un Dataframe con la sesión de Spark que hemos creado anteriormente, pero este Dataframe que hemos cargado tiene un formato que no nos facilita el tratamiento de datos por lo que lo vamos a adaptar de manera que cada columna se corresponda con un influencer y el valor de la columna sean los comentarios que éste/a recibe. El dataframe cargado crudo tiene el siguiente aspecto:

```
+--------------------------------------------------------------------------------------------------+--------------------------------+
|_c12                                                                                              |_c13                            |
+--------------------------------------------------------------------------------------------------+--------------------------------+
|DANIELLE BERNSTEIN                                                                                |LENA PERMINOVA                  |
"|[' A magical night with ||| este design de post não é atrativo , foto muito pequena, check ']|['2||| love that Polk-a-dot sweate
+--------------------------------------------------------------------------------------------------+--------------------------------+
```

Ilustración 3.23

Y una vez lo hemos limpiado, luce así :

```
+--------------------------------------------------------------------------------------------------+--------------------------------+
|DANIELLE BERNSTEIN                                                                                |LENA PERMINOVA                  |
+--------------------------------------------------------------------------------------------------+--------------------------------+
"|[' A magical night with ||| este design de post não é atrativo , foto muito pequena, check ']|['2||| love that Polk-a-dot sweate
+--------------------------------------------------------------------------------------------------+--------------------------------+
```

Ilustración 3.24

El cual aún tiene trabajo por delante, pero ya es un formato más agradable y sobre todo a nivel del tratamiento de los datos, es accesible (columna con nombre e identificada) y tratable (el contenido de las columnas es de valor, no son cosas que haya que eliminar).

UDF's Y LIMPIEZA DE TEXTO

```
# --- VERSIÓN CON UDF
# Sencillo verdad ? Esto es gracias a la modularidad de Spark
clean_text_udf = udf(CleanText, StringType())

# Asegurarse de que las columnas son de tipo String
for column in df_renamed.columns:
    df_renamed = df_renamed.withColumn(column, df_renamed[column].cast(StringType()))

# Aplicar la UDF a todas las columnas del DataFrame
for column in df_renamed.columns:
    df_renamed = df_renamed.withColumn(column, clean_text_udf(col(column)))

# Mostrar el resultado
df_renamed.show(truncate=False)
```

Ilustración 3.25

Una de las funciones más potentes de Spark, son las UDFs. (User Defined Functions) ya que nos permiten coger funciones realizadas en Python puro (sin Spark) y adaptarlas de manera que sean compatibles directamente con Dataframes de Pyspark. Como vemos, lo único que necesitamos es el nombre de la función e indicar el tipo del dato de salida. Gracias a eso podemos coger la función que teníamos originalmente de limpieza del texto y pasársela al dataframe con el que estamos trabajando ahora en lugar de un diccionario.

De ésta manera, el dataframe resultado queda algo del tipo:

```
+--------------------+--------------------------------------+--------------------------------------------------------------------
|NICOLE WARNE        |IRENE KIM                             |JENN IM
+--------------------+--------------------------------------+--------------------------------------------------------------------
| i'm not from new york| and i have every reason to speak on it| bozo|||truly insane|||so on brand for america|||cita|||priorities
+--------------------+--------------------------------------+--------------------------------------------------------------------
```

Ilustración 3.26

EXPLOTACIÓN DEL CONTENIDO

```python
# Crear un DataFrame con una sola columna 'value' para almacenar los resultados intermedios
columns = df_renamed.columns

# Iterar sobre cada columna y aplicar las transformaciones
for col_name in columns:
    # Obtener el primer valor de la primera fila de la columna
    row_to_process = df_renamed.head(1)[0]

    # Dividir la cadena larga por el delimitador '|||' y explotar en filas
    split_df = spark.createDataFrame([(value,) for value in row_to_process[col_name].split('|||')], [col_name])

    # Mostrar el DataFrame resultante para esta columna
    print(f"Columna: {col_name}")
    split_df.show(truncate=False)
```

Ilustración 3.27

Para darle una última capa al Dataframe principal con el que estamos trabajando, vamos a aprovechar que marcamos los finales de cadena de los comentarios con el carácter '|||', de ésta manera podemos separar los distintos comentarios y convertirlos en las filas de las distintas columnas, según quien es el influencer que ha recibido dichos comentarios, quedando nuestro Dataframe de la siguiente forma:

```
-+--------------------------------------------------------------------------------------------------------------------------
R|RACH PARCELL
-+--------------------------------------------------------------------------------------------------------------------------
 |traducir post
 |richard hamming said this too
 |when show me
 |yeah but thankfully every sigmoid we see is a small part of a really big sigmoid that\s been happening for 500,000+ years
 |since the first single called organisms information has been growing exponentially, one sigmoid stacked on top of another.
 |what if we find another different sigmoid and then stack it on top of the one thats topping out
 |it looks linear in the middle of the sigmoid
```

Ilustración 3.28

TRADUCCIÓN DE COMENTARIOS

```
translated_df = combined_df

def translate_text( t ):
    return GoogleTranslator(source='auto', target='es').translate(t)

translate_udf = udf(translate_text, StringType())

for col_name in translated_df.columns:
    translated_df = translated_df.withColumn(col_name, translate_udf(translated_df[col_name]))

translated_df.show(1000,truncate=False)
```

Ilustración 3.29

Gracias a las UDFs podemos aplicar directamente la función al Dataframe iterando directamente la estructura y aplicando la función a los valores de éste:

```
+-----------------------------------------------------------------------------------------------
|COLE SPROUSE
+-----------------------------------------------------------------------------------------------
|cole sprouse said what
|gigis at 7:30 tonight boss please don’t forget
|dont forget your reservation at gigis 7:30pm sharpyall better be in the mood for italian
|i miss you so much
|i don’t get it
|i always get that switched.
|i always get that switched.
|this is something theyd actually say
|as an identical twin
```

Ilustración 3.30

```
+-----------------------------------------------------------------------------------------------
|COLE SPROUSE
+-----------------------------------------------------------------------------------------------
|Cole Sprouse dijo qué
|Gigis a las 7:30 esta noche jefe, por favor no lo olvides.
|No olvides tu reserva en gigis a las 7:30 p. m. en punto. Será mejor que todos estén de humor para comida italiana.
|te extraño mucho
|no lo entiendo
|Siempre lo cambio.
|Siempre lo consigo cambiado.
|Esto es algo que realmente dirían.
|como un gemelo idéntico
```

Ilustración 3.31

ADAPTACIÓN DE FORMATO PARA ETIQUETACIÓN

```
def convert_to_tuple(value):
    return (value, "NON")

convert_to_tuple_udf = udf(convert_to_tuple, StructType([
    StructField("comment", StringType(), True),
    StructField("label", StringType(), True)
]))

paired_df = translated_df

for col_name in paired_df.columns:
    paired_df = paired_df.withColumn(col_name, convert_to_tuple_udf(paired_df[col_name]))

paired_df.show(1000,truncate=False)
```

Ilustración 3.32

Ya que más adelante vamos a calificar los comentarios que reciben los influencers según si son ofensivos o no, necesitamos adaptar el formato del Dataframe a esa nueva regla. Para ello, podemos mantener la estructura de columnas, pero convertimos los valores en tuplas que contendrán : { comentario, etiqueta }. Por defecto, los vamos a calificar como 'NON' y el Dataframe queda tal que así:

```
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
|COLE SPROUSE                                                                                                                                                     |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
|{Cole Sprouse dijo qué, NON}                                                                                                                                     |
|{gigis a las 7:30 esta noche jefe por favor no lo olvides, NON}                                                                                                  |
|{No olvides tu reserva en gigis a las 7:30 p. m. en punto. Será mejor que todos estén de humor para comida italiana., NON}                                       |
|{te extraño mucho, NON}                                                                                                                                          |
|{no lo entiendo, NON}                                                                                                                                            |
|{Siempre lo cambio., NON}                                                                                                                                        |
|{Siempre lo cambio., NON}                                                                                                                                        |
|{Esto es algo que realmente dirían., NON}                                                                                                                        |
|{como un gemelo idéntico, NON}                                                                                                                                   |
```

Ilustración 3.33

Y sobre este nuevo conjunto de datos, aplicaremos las funciones de clasificación descritas anteriormente.

LEXICÓN

```
# Registrar la UDF
detect_insult_in_tuple_udf = udf(
    lambda comment_tuple: detect_insult_in_tuple(comment_tuple, ArrayInsultos),
    StructType([
        StructField("comment", StringType(), True),
        StructField("label", StringType(), True)
    ])
)

# Iterar sobre cada columna y aplicar la UDF
for col_name in insultos_df.columns:
    insultos_df = insultos_df.withColumn(col_name, detect_insult_in_tuple_udf(col(col_name)))

# Mostrar el DataFrame resultante
insultos_df.show(1000,truncate=False)
```

Ilustración 3.34

Gracias las UDFs, de nuevo, podemos reutilizar la función tal cual, adaptando únicamente el valor a devolver de la siguiente forma:

```
def detect_insult_in_tuple(comment_tuple, insults):
    comment, label = comment_tuple
    try:
        if len(comment) > 1:
            words = comment.split()
            for word in words:
                if word in insults:
                    return (comment, 'OFF')
            return (comment, 'NON')
    except Exception as e:
        print(f"Error: {e}")
        return (comment, 'NON')
```

Ilustración 3.35

Ya que ahora devolvemos tuplas en lugar de solo la etiqueta, y queda el Dataframe de la siguiente forma:

```
+------------------------------------------------------------------------------------------
|JENN IM
+------------------------------------------------------------------------------------------
|{bozo, NON}
|{verdaderamente loco, OFF}
|{etc. marca para américa, NON}
|{desear, NON}
|{prioridades obviamente. es simplemente una locura, NON}
|{No hay razón para el control de armas... es una cuestión criminal y de salud mental., OFF}
|{una prohibición de tiktok antes de gta vi es aún más loca, OFF}
```

Ilustración 3.36

SUPERVISADO

```
predict_comment_udf = udf(
    lambda predict_tuple: predict_comment(predict_tuple),
    StructType([
        StructField("comment", StringType(), True),
        StructField("label", StringType(), True)
    ])
)

# Supongamos que tienes un DataFrame llamado df con las columnas adecuadas
# Iterar sobre cada columna y aplicar la UDF de predicción
for col_name in supervisado_df.columns:
    supervisado_df = supervisado_df.withColumn(col_name, predict_comment_udf(col(col_name)))

# Mostrar el DataFrame resultante
supervisado_df.show(1000,truncate=False)
```

Ilustración 3.37

Lo mismo que en el Lexicon, solo que el código queda adaptado ahora de la siguiente forma:

```
def predict_comment(comment):
    try:
        vect = v.transform([comment[0]])
        prediction = m.predict(vect)
        if prediction == "positive":
            return (comment[0],'NON')
        else:
            return (comment[0],'OFF')
    except:
        return (comment[0],'NON')
```

Ilustración 3.38

Porque ahora devolvemos las tuplas. Al utilizar la UDF el DF queda tal que así:

```
+------------------------------------------------------------------------------------------
|JENN IM
+------------------------------------------------------------------------------------------
|{bozo, NON}
|{verdaderamente loco, NON}
|{etc. marca para américa, NON}
|{desear, NON}
|{prioridades obviamente. es simplemente una locura, NON}
|{No hay razón para el control de armas... es una cuestión criminal y de salud mental., NON}
|{una prohibición de tiktok antes de gta vi es aún más loca, OFF}
```

Ilustración 3.39

HIBRIDACIÓN

```
final_df_hybrid = insultos_df.union(supervisado_df).select(supervisado_df.columns)

def filter_off_tuples(tuples):
    if tuples and len(tuples) == 2 and tuples[1] == 'OFF':
        return (tuples[0], 'OFF')
    else:
        return None

filter_off_tuples_udf = udf(filter_off_tuples, StringType())

# Registrar la UDF para aplicar la función de predicción
filter_off_tuples_udf = udf(
    lambda predict_tuple: filter_off_tuples(predict_tuple),
    StructType([
        StructField("comment", StringType(), True),
        StructField("label", StringType(), True)
    ])
)

# Iterar sobre cada columna y aplicar el filtro
for col_name in final_df_hybrid.columns:
    final_df_hybrid = final_df_hybrid.withColumn(col_name, filter_off_tuples_udf(F.col(col_name)))

# Mostrar el DataFrame resultante
final_df_hybrid.show(1000,truncate=False)
```

Ilustración 3.40

Finalmente y al igual que en el Lexicon y el supervisado, combinamos ambos conjuntos de datos y lo adaptamos con una UDF, resolviendo el Dataframe completo. Por lo que ahora podemos proceder al análisis de tiempos en comparación con los que teníamos en Python puro ya que el objetivo de todo este proyecto era la mejora y benchmarking de dichas marcas temporales.

```
# SUPERVISADO

# --- Entrenamos modelo utilizando el fichero de testing
m, v = EntrenamientoSupervisado(train)

# --- Aplicamos supervisado
def Supervisado(DictComentarios):

  final_dict_s = {}

  for keys, values in DictComentarios.items():
    value = []
    for i in values:
      vect = v.transform([i[0]])
      prediccion = m.predict(vect)
      if prediccion == "positive":
        tup = ( i[0] , 'NON' )
      else:
        tup = ( i[0] , 'OFF' )
      value.append(tup)

    final_dict_s[keys] = value

  return final_dict_s


final_dict_supervisado = Supervisado(final_dict)
✓ 15.6s
```

Ilustración 3.41

Scrapping

El software diseñado automatiza la extracción de datos de internet sobre los comentarios que reciben distintas celebridades.

DEPENDENCIAS Y LIBRERÍAS

```python
#Libraries and dependencies

# --- READ AND WRITE FILES ---
import csv

# --- REGULAR EXPRESSIONS ---
import re

# --- SELENIUM ---
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.common.by import By
from selenium.webdriver import ChromeOptions, Keys
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.wait import WebDriverWait
import time
```

✓ 0.4s Python

Ilustración 3.42

Importamos librerías para la lectura y escritura de ficheros (csv) y el tratamiento intenso de cadenas de strings (expresiones regulares) junto a toda la funcionalidad de Selenium, la librería que nos permitirá hacer el scrapping web y extraer todos los datos necesarios de la red.

INSTANCIA DE SELENIUM

```python
# --- SELENIUM + CHROME DRIVER INSTANCE ---
options = webdriver.ChromeOptions()
options.add_experimental_option("detach", True)
cService=webdriver.ChromeService(executable_path='chromedriver.exe',options=options)
```

✓ 0.0s Python

Ilustración 3.43

En esta caja, instanciamos nuestro driver de Chrome, el cual es un ejecutable que nos permitirá invocar un navegador de Chrome para realizar todas las operaciones necesarias para el scrapping con Selenium. Hemos elegido Chrome ya que es el navegador con más uso y más usuarios activos, por lo que la mayor parte de servicios de internet son compatibles y no presentan problemas con dicho navegador.

Luego instanciamos el servicio de nuestro driver, que será el objeto que utilicemos para invocar el navegador de Chrome y utilizarlo, indicándole donde se encuentra nuestro ejecutable y las opciones deseadas.

SCRAPPING DE INFLUENCERS

```
# --- INFLUENCERS NAMES RETRIEVING ---
# We initialize the browser
driver = webdriver.Chrome(service = cService)
# We look for our top influencers
driver.get("https://www.amraandelma.com/top-influencers-in-the-world/")
# We extract the info
results = driver.find_elements(By.XPATH, '//*[@id="post-18727"]/div/div[2]/div/section/ \
                                          div[1]/div/div/div/div/div[19]/div')[0].text
# Quit the driver
driver.quit()
```

✓ 6.8s Python

Ilustración 3.44

Invocamos el driver con el servicio que creamos en el bloque anterior, esto no hace otra cosa que instanciar un navegador de Chrome en nuestro equipo, ya que el control de éste viene después. En la siguiente línea, hacemos una solicitud web a través de nuestro navegador que nos permite acceder a la URL deseada. En este caso se trata de una URL que apunta a una página web que contiene los 100 ‘influencers’ más famosos del momento. Una vez nuestra solicitud se resuelve, a través del xPath de la web, extraemos el bloque de HTML que contiene los 100 nombres de los influencers para más adelante darle tratamiento a esa información y poder buscarlos a través de las redes sociales. Por último, cerramos nuestro driver.

LIMPIEZA DE DATOS DE INFLUENCERS

```
# --- CLEANING DATA IN ORDER TO SCRAP IT LATER FROM THE WEB ---
l_results = results.split('\n')
l_results_first_clean = []
re_ex = r":(.*?)@"

for i in l_results:
    if 'Influencers ' in i and ( 'FOLLOWERS' in i or 'SUBSCRIBERS' in i):
        elem = re.findall(re_ex, i)
        if len(elem) < 1:
            re_ex = r":(.*?)(\d)"
            elem = re.findall(re_ex, i)
        l_results_first_clean.append(elem[0])
```

Ilustración 3.45

Ahora tenemos un código HTML que contiene la información de nuestros influencers, pero hay mucha información a parte del nombre que no necesitamos, como son las etiquetas de estructura de la página (, <li> …) u otros. Nos damos cuenta cuando extraemos los datos, que junto al nombre de los influencers se indica su número de followers o suscriptores, por lo que podemos quedarnos solo con aquellas cadenas que contienen esas palabras clave. Después de eso, ya tenemos la línea que contiene la información que queremos, pero nosotros solo queremos el nombre del influencer, es por ello que nos damos cuenta de que en la cadena extraída, el nombre del influencer aparece siempre bajo el mismo patrón:

XXXXXXX **: NOMBRE_DEL_INFLUENCER \d** XXXXXXX

Por lo que aplicamos una expresión regular que se corresponda con el patrón:

re_ex = r":(.*?)(\d)"

La cual extrae únicamente el nombre del influencer de la cadena completa.

```
for i in l_results_first_clean:
    elem = i
    if type(i) is tuple:
        elem = i[0].strip()
        elem = elem.replace('[','')
        elem = elem.replace(']','')
        elem = elem.replace('(','')
        elem = elem.replace(')','')
        influencers_names.append(elem)
```

Ilustración 3.46

Por último, limpiamos la cadena extraída de caracteres no deseados, espacios al inicio y fin del string y también nos deshacemos de esos elementos que se han vuelto dobles por el tratamiento anterior, extrayendo completamente el nombre de los influencers.

Una vez realizada la limpieza, procedemos a la escritura de la información que será utilizada más adelante en otros procesos. Esto nos permite no tener que ejecutar el software completo cada vez que queramos iterar sobre los influencers y además tener un formato limpio que podemos consultar en cualquier momento para su estudio.

```
file_name = "influencers.csv"
with open(file_name, 'w', newline='') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(['INFLUENCER NAME'])
    for i in influencers_names:
        csvwriter.writerow([i])
```

Ilustración 3.47

Y los datos en el fichero quedan de la siguiente manera:

influencers.csv

1	INFLUENCER NAME
2	COLE SPROUSE
3	CHIARA FERRAGNI
4	CAMILA COELHO
5	LAUREN CONRAD
6	AIMEE SONG
7	NEGIN MIRSALEHI

Ilustración 3.48

Una única columna cuyo header es INFLUENCER NAME y cuyos valores son los nombres de los 100 influencers que hemos extraído en la página del inicio.

SCRAPPING DE COMENTARIOS

```
# Module to scrap data from our influencers from Twitter (X) bypassing twitter security
options = ChromeOptions()
options.add_argument("--start-maximized")
options.add_experimental_option("excludeSwitches", ["enable-automation"])

driver = webdriver.Chrome(options=options)
url = "https://twitter.com/i/flow/login"
driver.get(url)
```

Ilustración 3.49

Hemos elegido Twitter como la red social a explotar para la extracción de los comentarios hacia los distintos influencers. Algunos influencers no poseen un perfil en esta red social, por lo que no se extraen comentarios acerca de ellos, pero eso entra dentro de la automatización y es controlado para no provocar errores en nuestro software.

A nuestro driver de Chrome, le añadimos unas opciones que nos permiten "simular" el comportamiento humano ya que al no utilizar la API específica de Twitter detectará nuestro script como un bot y seremos expulsados de inmediato. Una vez configurado, invocamos el driver y accedemos a la URL correspondiente la login de Twitter.

```
username = WebDriverWait(driver, 20).until(EC.visibility_of_element_located((By.CSS_SELECTOR,
                                                    'input[autocomplete="username"]')))
username.send_keys([redacted])
username.send_keys(Keys.ENTER)

password = WebDriverWait(driver, 10).until(EC.visibility_of_element_located((By.CSS_SELECTOR,
                                                    'input[name="password"]')))
password.send_keys([redacted])
password.send_keys(Keys.ENTER)

time.sleep(10)
```

Ilustración 3.50

En la ventana de login, introducimos nuestros credenciales junto a unas pautas de espera de manera que Twitter piense que somos un usuario más introduciendo sus credenciales y esquivamos el bloqueo. Además, en este caso, en lugar de utilizar la localización de elementos por XPATH como hicimos en el scrapping de los influencers, aquí localizamos las cajas de usuario y contraseña a través de CSS. Ésta es una de las razones de porqué utilizamos Selenium y es que permite la simulación completa del comportamiento humano al mismo tiempo que nos permite extraer toda la información que queramos e interactuar con la web a través de herramientas Python muy potentes.

```
file_name = "influencers.csv"
influencers = []
with open(file_name, 'r') as csvfile:
    reader = csv.reader(csvfile)
    for i in reader:
        influencers.append(i)
# We delete the header
influencers.pop(0)

driver.implicitly_wait(10)

full_dict = {}
```

Ilustración 3.51

Previo a la extracción de los comentarios y la navegación por twitter, cargamos en una lista el nombre de los influencers que cargamos en un csv anteriormente, esa lista va a ser la que marque las iteraciones de nuestro bucle principal extrayendo comentarios. Eliminamos el primer elemento que contiene el header ya que este no nos interesa y le damos un valor de espera adicional al driver para que nos podamos ajustar a la tasa de refresco del navegador mientras lo iteramos. Además, preparamos un diccionario que nos permitirá almacenar la información de la siguiente forma:

DICCIONARIO = { (KEY1 , [VALUE1, VALUE2, …]) , (KEY2, […]) , …}
DICCIONARIO = { (INFLUENCER , [COMENTARIO1, COMENTARIO2, …])

```
for i in influencers:
    dict_key = i

    # Search the user
    try:
        search_bar = driver.find_element('xpath','/html/body/div[1]/div/div/div[2]/main/div/div/div/div[2]/ \
                                          div/div[2]/div/div/div/div[1]/div/div/div/form/div[1]/div/div/div/ \
                                          div/div[2]/div/input')
        search_bar.clear()
        search_bar.send_keys(dict_key)
        search_bar.send_keys(Keys.ENTER)
    except:
        search_bar = driver.find_element('xpath','/html/body/div[1]/div/div/div[2]/main/div/div/div/div[1]/ \
                                          div/div[1]/div[1]/div[1]/div/div/div/div/div[2]/div[2]/div/div/div/ \
                                          form/div[1]/div/div/div/div/div[2]/div/input')
        search_bar.clear()
        search_bar.send_keys(dict_key)
        search_bar.send_keys(Keys.ENTER)
```

Ilustración 3.52

En este caso, comenzamos a iterar sobre la lista de influencers, y lo primero que haremos será localizar el elemento de “Barra de búsqueda” para poder buscar por Twitter nuestros influencers. Se ha hecho con la estructura mostrada en la captura, porque la estructura web de la página es dinámica, y puede cambiar a lo largo de la ejecución del programa, es por ello que iteramos sobre las dos posibilidades que se dan para localizar este elemento. Una vez lo hemos localizado, insertamos el nombre del influencer en el campo y simulamos el pulsar la tecla ENTER, realizando así la búsqueda.

```
try:

    # Filter by people
    filter = driver.find_element('xpath','/html/body/div[1]/div/div/div[2]/main/div/div/div/div[1]/div/ \
                                 div[1]/div[1]/div[2]/nav/div/div[2]/div/div[3]/a')
    filter.click()

    # Enter the profile
    person = driver.find_element('xpath','/html/body/div[1]/div/div/div[2]/main/div/div/div/div[1]/div/ \
                                 div[3]/section/div/div/div[1]/div/div/button')
    person.click()

    # NO POSTS EXCEPTION
    post = driver.find_element('xpath','/html/body/div[1]/div/div/div[2]/main/div/div/div/div[1]/div/ \
                               div[3]/div/div/section/div/div/div[1]/div/div/article/div/div/div[2]/ \
                               div[2]/div[2]/div/span[1]')
    post.click()

except:

    time.sleep(10)
```

Ilustración 3.53

A continuación, entramos en el bloque de código que más errores o casuísticas puede enfrentar. Las situaciones a las que nos encontramos son que no encontremos el perfil en twitter o que encontramos el perfil pero no posea ninguna publicación por lo que podemos asumir que no se trata del influencer que estamos buscando sino de otro perfil. Comenzamos aplicando un filtro por usuarios a nuestra búsqueda, a continuación hacemos ‘click’ en el primer perfil resultado de la búsqueda y por último hacemos click en la primera publicación del perfil. Como hemos comentado antes, estas fases son en cascadas y en caso de que una de ellas falle, simplemente abortaremos esta sección y simularemos un “tiempo muerto” para no sobrecargar la página con peticiones y así mantener nuestro acceso y nuestro software funcionando.

```
elem = driver.find_element(By.TAG_NAME, "html")
elem.send_keys(Keys.END)

comments = []
for i in range(1,50):
    try:
        html_code = driver.find_element('xpath','/html/body/div[1]/div/div/div[2]/main/div/div/div/ \
                                        div[1]/div/section/div/div/div[' + str(i) + ']')
        html = html_code.get_attribute('innerHTML')
        print(html)
        comments.append(html)
        if i%2==0:
            elem.send_keys(Keys.END)
    except:
        break

dict_content = comments

#Save dict
full_dict[dict_key] = dict_content
```

Ilustración 3.54

Ya que los comentarios en twitter se muestran bajo petición, volvemos a simular la actividad humana haciendo scroll hacia abajo en la página para que éstos aparezcan. Una vez cargan, simplemente extraemos el cuerpo html de la sección de comentarios, y cada dos iteraciones scrolleamos hacia abajo. Establecemos el rango de 50 de manera arbitraria pero este podría extenderse hasta el infinito sin problema. En nuestro caso se trata de una prueba de concepto en la que nuestro interés principal está en mostrar la capacidad de Spark y no en el scrapping. Una vez extraídos, almacenamos la información en el diccionario que sigue la estructura explicada anteriormente.

LIMPIEZA DE LOS COMENTARIOS

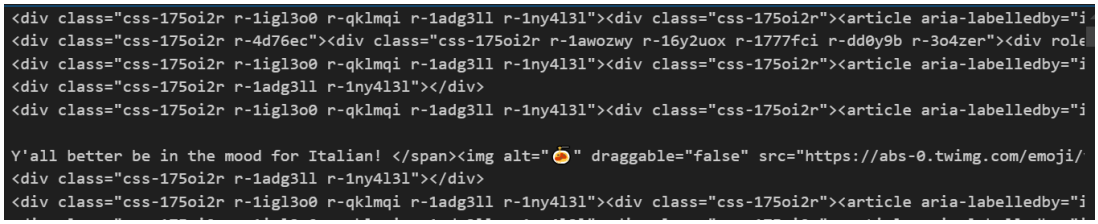


Ilustración 3.55

Esta es la información del HTML que estamos extrayendo y almacenando, ahora se hace necesaria una limpieza como hicimos al inicio con los nombres de los influencers, para ello:

```
# Para cada bloque de len => 4 , el quinto elemento más el cuarto,
# pero cogiendo el primer elemento de splitear hasta </span>

clean_dict = {}

for k,v in full_dict.items():
    value = []
    for j in v:
        temp = j.split('<span class=\"css-1jxf684 r-bcqeeo r-1ttztb7 r-qvutc0 r-poiln3\"  \
                       style=\"text-overflow: unset;\">')
        if len(temp) > 4:
            print(temp[4].split('</span>')[0])
            val = temp[4].split('</span>')[0]
            val = val.replace('\n','')
            value.append(val)
        elif len(temp) == 4:
            print(temp[3].split('</span>')[0])
            val = temp[3].split('</span>')[0]
            val = val.replace('\n','')
            value.append(val)
    clean_dict[k] = value
```

Ilustración 3.56

Nos damos cuenta que los comentarios como tal están encerrados bajo ciertas pautas las cuales podríamos extraer mediante expresiones regulares como hicimos al inicio pero aquí al haber múltiples coincidencias preferimos utilizar otro método como es la separación de bloques. Según la cantidad de bloques en la que se separaba cada cadena, podíamos saber donde se localizaba el comentario y aplicar el tratamiento adecuado, por lo que al final extraemos únicamente el comentario en sí de todo el perfil del influencer a través de la criba del código HTML de la página.

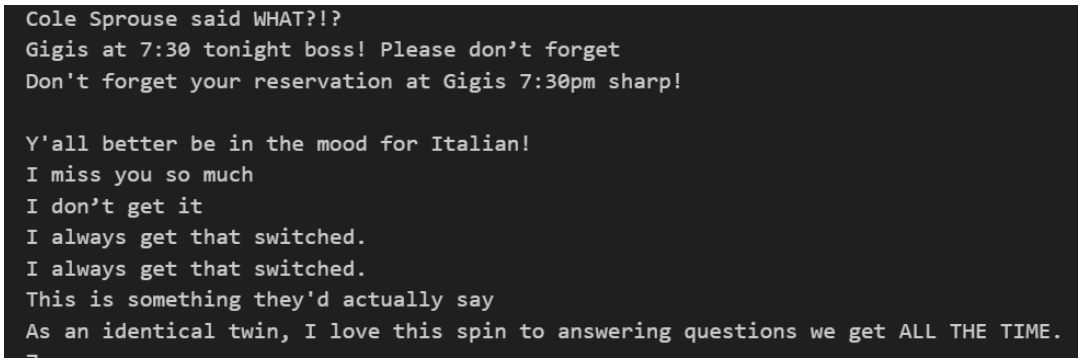


Ilustración 3.57

Ese es el resultado de pasar el HTML mostrado anteriormente por nuestro script de extracción de comentarios, como vemos también se aprecia la separación por influencer y los comentarios tienen un formato que ahora es entendible y tratable.

ALMACENAMIENTO DE COMENTARIOS

```
for k,v in clean_dict.items():
    print('KEY: ' + k)
    for i in v:
        print('VALUE:' + i)
```

Ilustración 3.58

Imprimimos por pantalla nuestra estructura de datos previamente a su escritura para ver que todo está correcto:

```
KEY: COLE SPROUSE
VALUE:Cole Sprouse said WHAT?!?
VALUE:Gigis at 7:30 tonight boss! Please don’t forget
VALUE:Don't forget your reservation at Gigis 7:30pm sharp!Y'all better be in the mood for Italian!
VALUE:I miss you so much
```

Ilustración 3.59

Lo cual tiene buena pinta, por lo que procedemos a escribirlo:

```
with open("influencers_comments_clean.csv", 'w', encoding="utf-8") as csv_file:
    writer = csv.writer(csv_file)
    for key, value in clean_dict.items():
        writer.writerow([key, '|||'.join(value)])
csv_file.close()
```

Ilustración 3.60

Realizar la escritura de esta manera nos permite tener un fichero donde la clave separada mantiene los comentarios unidos, por lo que la información tiene sentido y está diferenciada. Utilizamos un separador especial ‘|||’ para no confundirlo con los separadores de campo o línea típico de los ficheros como podría ser ‘,’ o ‘;’.

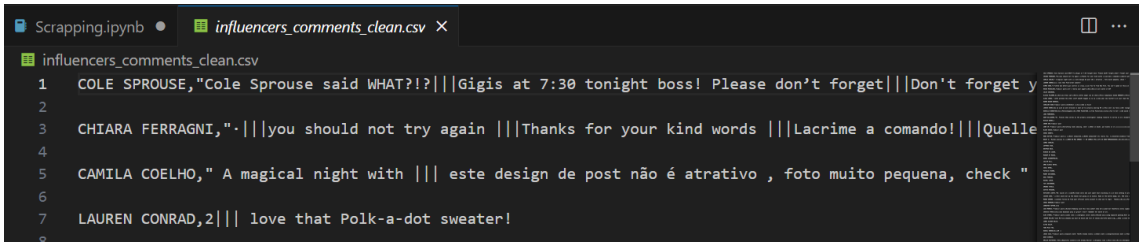


Ilustración 3.61

Y con esto tenemos nuestro .csv con toda la información extraída de Twitter y no tenemos la necesidad de volver a ejecutar el software para las siguientes fases de tratamiento de dicha información o la ejecución de procesos que los ingesten.

3.1 Implementación

MONTAJE Y FUNCIONAMIENTO

Lo primero de todo es preparar nuestra máquina virtual, es decir, una ejecución funcional de Cent OS 7 en nuestro virtual box que funciona correctamente y en la que podremos instalar software en el futuro. Por ahora comenzaremos creando un solo nodo u ordenador a modo de prueba y para comenzar a configurar el resto del ecosistema. Es muy importante pararnos aquí a destacar la arquitectura que hemos pensado para la implementación de nuestro proyecto, ya que ésta determinará los recursos a usar necesarios y la planificación de éstos.

Nuestra arquitectura final vendrá a ser algo tal que así:

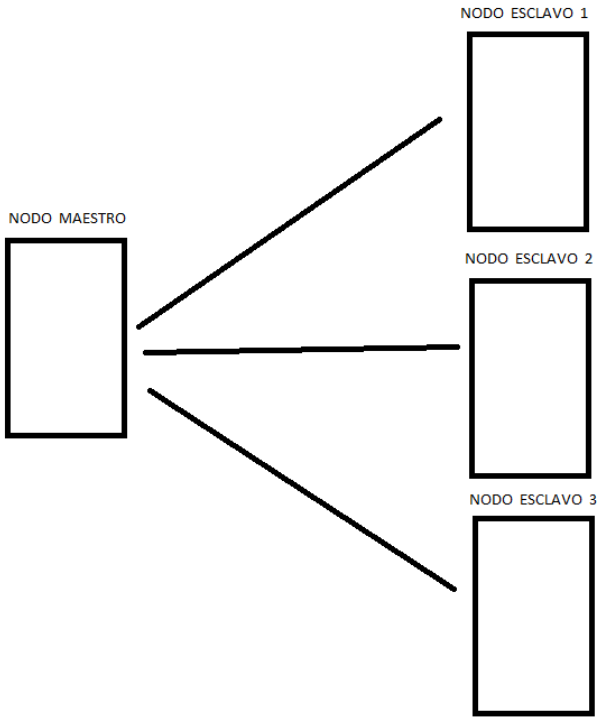


Ilustración 3.11.1

Vamos a comenzar, creando una máquina virtual en la que realizaremos toda la instalación del software necesario paso por paso, luego duplicaremos esta máquina y haremos las modificaciones necesarias para imitar la arquitectura de la imagen anterior.

- Creamos la máquina virtual

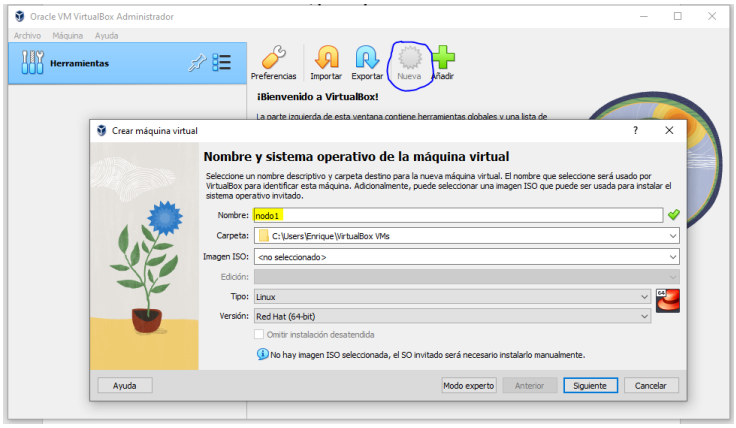


Ilustración 3.12.2

Dado que se trata del futuro nodo maestro, le daremos una mayor cantidad de recursos hardware en comparación con los otros nodos, siendo en este caso, 4 GB y dos cores.

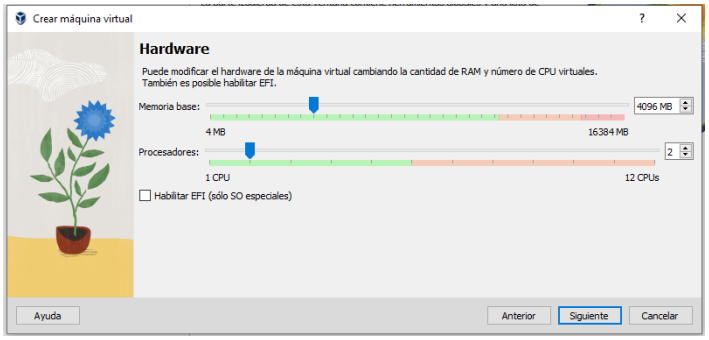


Ilustración 3.13.3

Creamos un disco virtual adecuado a la memoria que vayamos a emplear para almacenar los distintos ficheros de pruebas y similar, en este caso:

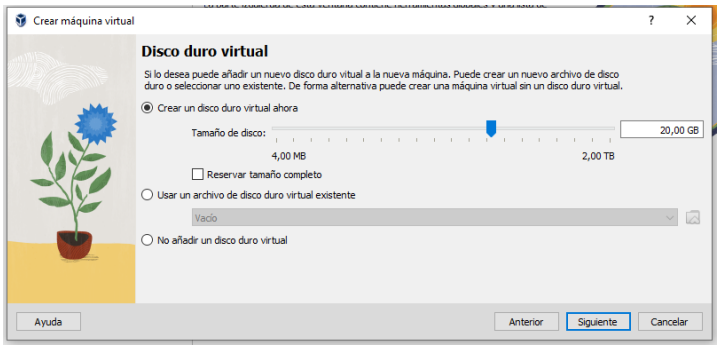


Ilustración 3.14.4

Aquí ofrecemos un breve resumen de la máquina que estamos creando:

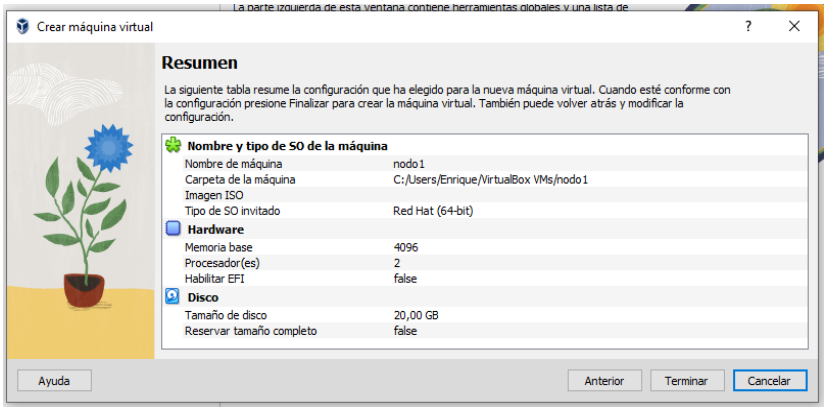


Ilustración 3.15.5

Ahora mismo, contamos con la instancia de una máquina virtual generada, pero ésta carece de sistema operativo alguno con el que arranca, por lo que vamos a asignarle el que nos hemos instalado anteriormente:

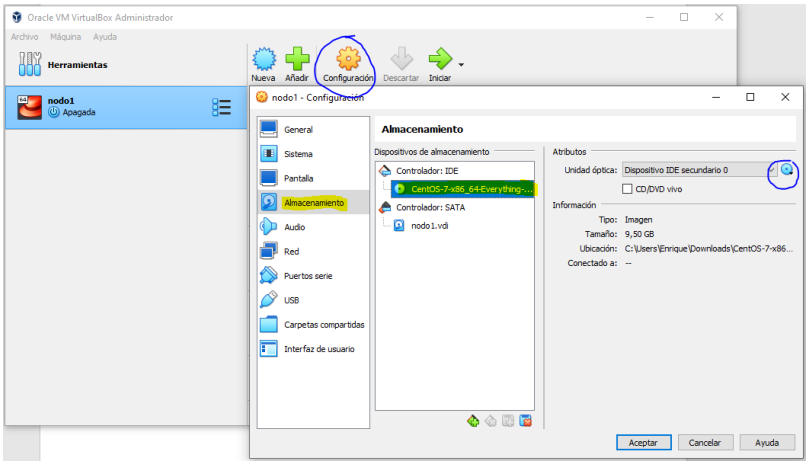


Ilustración 3.16.6

Ya contamos con nuestra máquina virtual preparada, así que vamos a probar a arrancarla y a realizar la instalación y configuración de todo el sistema operativo CentOs en ella:

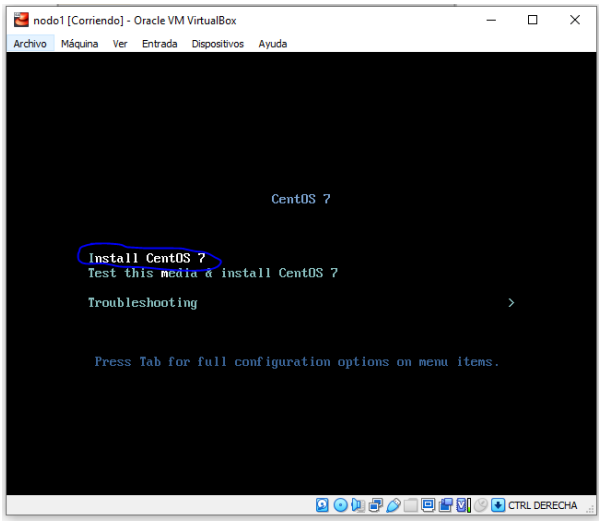


Ilustración 3.17.7

Una vez hemos instalado el sistema operativo, (no comentamos nada de la instalación ya que esto no es un tutorial) procederemos con la configuración específica del sistema para hacerlo compatible con la instalación de software Big Data que realizaremos más adelante.

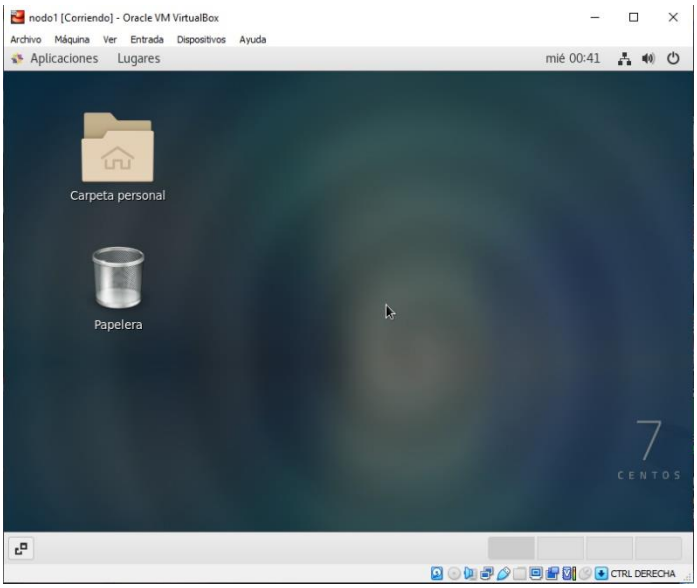


Ilustración 3.18.8

HADOOP

Lo primero, es descargar el software de Hadoop (https://hadoop.apache.org/release/2.7.2.html)

Esta versión es sobre la que más conocimiento tenemos y la que más estandarizada está su uso debido a su gran compatibilidad con Java 1.8 y Java 11

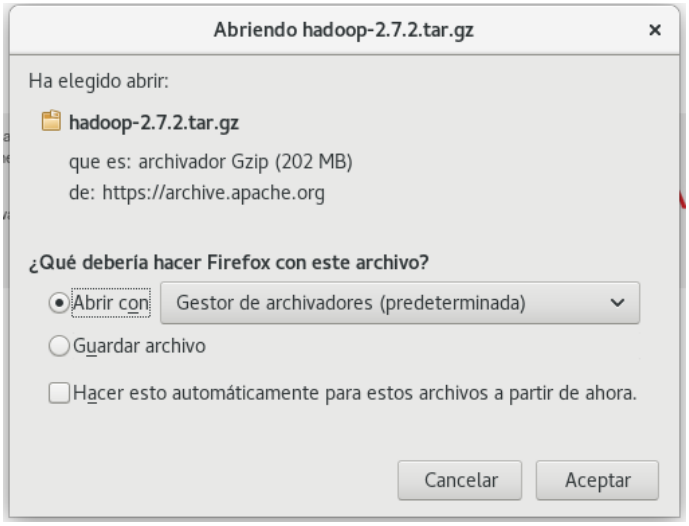


Ilustración 3.19.9

Localizamos nuestro paquete y lo descomprimimos en nuestra carpeta de trabajo:

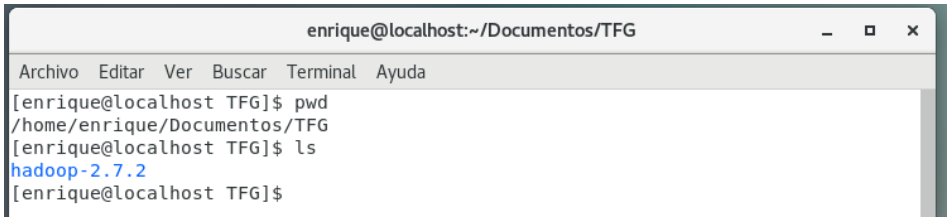


Ilustración 3.20.10

Dado que en este entorno tenemos poderes de super usuario y simulamos que estamos en un entorno en el que somos el administrador del sistemas, vamos a crear una carpeta llamada “hadoop” donde vamos a volcar todo el contenido descargado en la ruta /opt ya que desde ahí podremos administrar el resto de nodos.

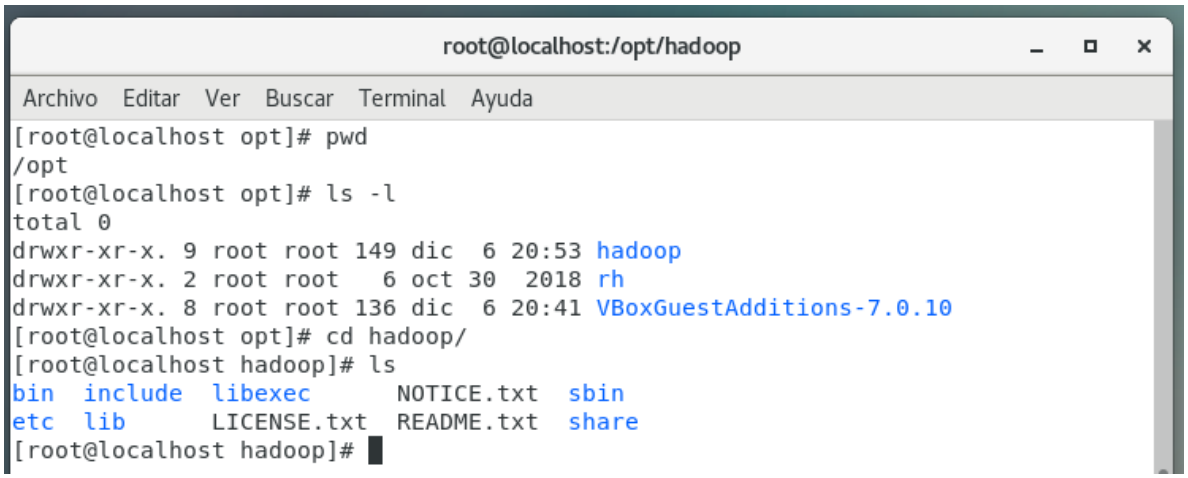


Ilustración 3.21.11

En ese mismo directorio podemos observar las GuestAdditions que son un paquete de software que VBox nos proporciona para la interacción con el usuario lo cual nos va a facilitar el trabajo.

Una vez hecho, procedemos con la instalación de Java y de los JDKs, ya que Spark y Hadoop corren sobre ese lenguaje. Descargaremos el JDK 8 ya que es la versión más compatible con nuestra elección de Hadoop en .rpm al ser más sencillo de descomprimir y procederemos con la instalación.

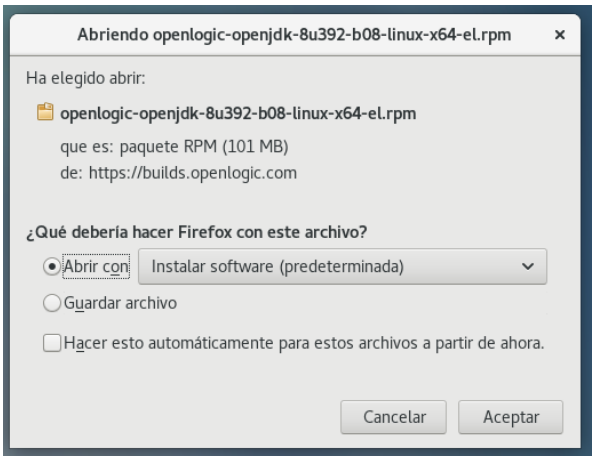


Ilustración 3.22.12

Como ya contábamos con la instalación, simplemente se nos ha actualizado:

```
[root@localhost Descargas]# ls
hadoop-2.7.2.tar.gz  openlogic-openjdk-8u392-b08-linux-x64-el.rpm
[root@localhost Descargas]# rpm -ivh openlogic-openjdk-8u392-b08-linux-x64-el.rp
m
Preparando...                          ################################# [100%]
Actualizando / instalando...
   1:openlogic-openjdk-8-hotspot-8u392################################# [100%]
[root@localhost Descargas]#
```

Ilustración 3.23.13

Nos aseguramos que todo está correcto:

```
[root@localhost Descargas]# alternatives --config java

Hay 3 programas que proporcionan 'java'.

  Selección    Comando
-----------------------------------------------
   1           java-1.7.0-openjdk.x86_64 (/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.261-2.6.22.2.el7_8.x86_64/jre/bin/java)
*+ 2           java-1.8.0-openjdk.x86_64 (/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.262.b10-1.el7.x86_64/jre/bin/java)
   3           /usr/lib/jvm/openlogic-openjdk-8-hotspot/bin/java

Presione Intro para mantener la selección actual[+], o escriba el número de la selección:
[root@localhost Descargas]# java -version
openjdk version "1.8.0_262"
OpenJDK Runtime Environment (build 1.8.0_262-b10)
OpenJDK 64-Bit Server VM (build 25.262-b10, mixed mode)
[root@localhost Descargas]#
```

Ilustración 3.24.14

Ahora procedemos a echar un vistazo a nuestro directorio de Hadoop y a los componentes más relevantes con los que trabajaremos a lo largo de este documento:

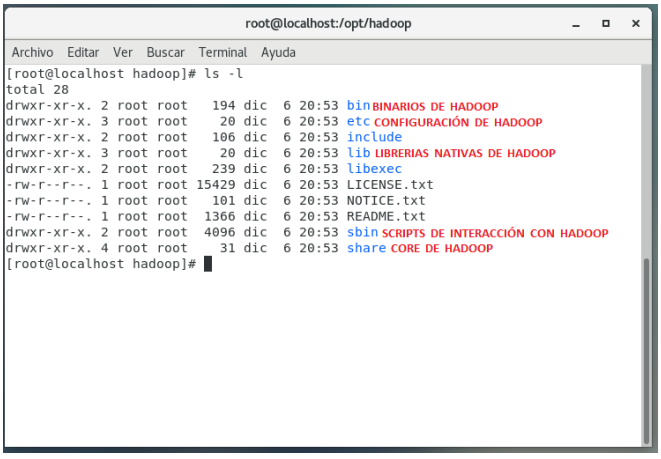


Ilustración 3.25.15

Para proceder con la configuración de Hadoop, primero necesitamos configurar las variables de entorno de nuestro sistema para que nuestra interacción con el software sea directa e intuitiva. Procederemos de la siguiente manera modificando el .bashrc de nuestro usuario, el cual contiene nuestra interacción principal con el sistema:

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions
```

Ilustración 3.26.16

Añadiremos lo siguiente:

```
# User specific aliases and functions
export HADOOP_HOME=/opt/hadoop
export JAVA_HOME=/usr/lib/jvm/openlogic-openjdk-8-hotspot
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

Ilustración 3.27.17

Las cuales son la referencia a la variable global de hadoop y añadiremos la de binarios y scripts a nuestro path principal ya que estsas serán las que mas utilicemos. Evidentemente Hadoop corre sobre Java por lo que también tendremos que añadirlo como referencia. Y ejecutamos el archivo .bashrc para actualizar las referencias nuevas:

```
[enrique@localhost ~]$ . ./.bashrc
```

Ilustración 3.28.18

Hemos utilizado esa versión de Java ya que es una de las compatibles con Hadoop y ademá era la que tenía ya la máquina instalada por lo que no es necesario instalarlo.

PROBANDO HADOOP

Ahora que hemos realizado la instalación completa, vamos a ejecutar Hadoop en el modo stand-alone (es decir, no requiere de clúster y puede correr de manera independiente) para la realización de una serie de pruebas genéricas que incluye el propio Hadoop para aasegurarnos de que su funcionamiento es el correcto. En este caso, nuestra elección es la ejecución de una operación mapreduce:

```
[enrique@localhost hadoop]$ pwd
/opt/hadoop/share/hadoop
[enrique@localhost hadoop]$ ls
common  hdfs  httpfs  kms  mapreduce  tools  yarn
[enrique@localhost hadoop]$
```

Ilustración 3.29.19

Procederemos a la preparación del entorno para el ejempo. Para comenzar, nos copiaremos una serie de ficheros que tiene Hadoop de prueba sobre el que realizaremos la prueba de funcionamiento:

```
[enrique@localhost hadoop]$ mkdir /tmp/entrada
[enrique@localhost hadoop]$ cd /opt/hadoop/etc/hadoop/*.xml /tmp/entrada/
bash: cd: /opt/hadoop/etc/hadoop/capacity-scheduler.xml: No es un directorio
[enrique@localhost hadoop]$ cp /opt/hadoop/etc/hadoop/*.xml /tmp/entrada/
[enrique@localhost hadoop]$ ls /tmp/entrada/
capacity-scheduler.xml  hadoop-policy.xml  httpfs-site.xml  kms-site.xml
core-site.xml           hdfs-site.xml      kms-acls.xml     yarn-site.xml
```

Ilustración 3.30.20

Lanzamos el jar del mapReduce de la siguiente manera:

[grep es un comando interno de hadoop/Java que permite el conteo y acumulación de strings que cumplen cierta condición, en nuestro caso todos los que empiecen por ‘kms’ como indica la expresión regular de la linea de argumentos]

```
[enrique@localhost mapreduce]$ pwd
/opt/hadoop/share/hadoop/mapreduce
[enrique@localhost mapreduce]$ ls
hadoop-mapreduce-client-app-2.7.2.jar
hadoop-mapreduce-client-common-2.7.2.jar
hadoop-mapreduce-client-core-2.7.2.jar
hadoop-mapreduce-client-hs-2.7.2.jar
[enrique@localhost mapreduce]$ hadoop jar hadoop-mapreduce-examples-2.7.2.jar grep /tmp/entrada/ /tmp/salida 'kms[a-z.]+'
```

Ilustración 3.31.21

Lo cual producirá unos ficheros similares a estos:

```
699 mar 21 23:14 part-r-00000
  0 mar 21 23:14 _SUCCESS
```

Ilustración 3.32.22

Y de esta manera nos cercioramos de que nuestro Hadoop está correctamente instalado.

SIGUIENTES PASOS

Procederemos a configurar el ssh en nuestra máquina de manera que podamos conectar los distintos nodos de trabajo más adelante.

Lo primero, será generar nuestro par de claves para la comunicación de SSH:

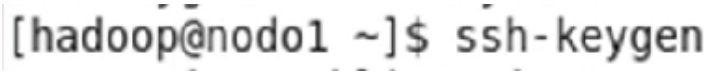


Ilustración 3.33.23

En este caso hemos empleado un passphrase en blanco por no añadir complejidad a este apartado:

```
[hadoop@nodo1 ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoop/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hadoop/.ssh/id_rsa.
Your public key has been saved in /home/hadoop/.ssh/id_rsa.pub.
The key fingerprint is:
5a:c7:c0:66:b8:4d:32:cf:84:7d:52:dc:99:67:5f:81 hadoop@nodo1
The key's randomart image is:
+--[ RSA 2048]----+
|          ... o...|
|       = .. +Eo .|
|      = X .  o ..|
|       % =      .|
|      . S o      |
|       o .       |
|      .          |
|                 |
|                 |
+-----------------+
```

Ilustración 3.34.24

Hacemos una rápida comprobación y vemos el contenido de nuestra clave pública:

```
[hadoop@nodo1 ~]$ cd /home/hadoop/.ssh
[hadoop@nodo1 .ssh]$ ls -l
total 8
-rw-------. 1 hadoop hadoop 1675 mar 24 20:39 id_rsa
-rw-r--r--. 1 hadoop hadoop  394 mar 24 20:39 id_rsa.pub
[hadoop@nodo1 .ssh]$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAvw2F7onioAZfVtFRClre4jaHA6EuHYP36i1ldycMz6T8
BPb0Dwl/LgKwIH6TAO7c2bybjyCuSQedGJo9hchk0XOzKX2leg+Hx7ED4ilTT0wH6wTONp/GSa43xBrp
6i35pLts7dLq7A8HMHDRrUj1rSh6QeR5/JsD0g5RPPWaJVVcj8SXr7eX6w3tgolHAdLLa4UQB+foPsP4
xI0nCjGY4j6WNJkWoTIdCZX1deca6HPcWTH/LobnCTtKgkzXTWuTfO0Rq0hZOiReYa9YQdUcbl3uFcG6
RpED/HFHKCzhqn4YiSKC4QBWW7LvBECB8XkGTXZQA+vYQ2vHZioqky1RWw== hadoop@nodo1
```

Ilustración 3.35.25

Y por ser organizados, lo copiamos a la carpeta que creamos authorized_keys:

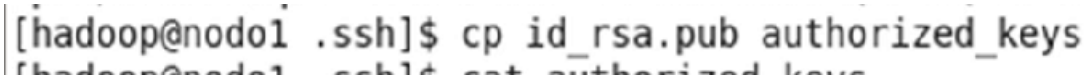


Ilustración 3.36.26

Finalmente y para comprobar que todo es correcto, nos conectaremos por ssh al nodo 1 (es decir, a nosotros mismos o al localhost) y cerraremos la conexión después:

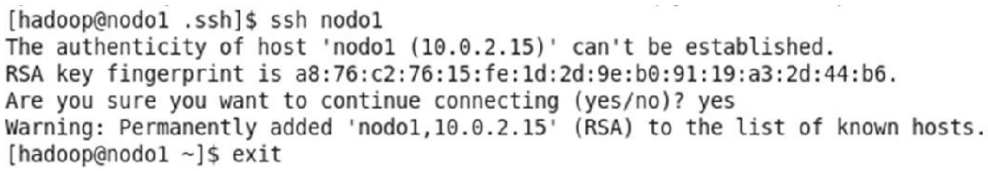


Ilustración 3.37.27

MONTANDO UN CLUSTER HDFS CON UN SOLO NODO (PSEUDODISTRIBUIDO)

Pseudodistribuido es un clúster de un solo nodo, donde éste es maestro y nodo al mismo tiempo. En la realidad esto no ocurre, pero esta máquina será clonada así que la instalación nos sirve también.

Accedemos a los ficheros de configuración de nuestro Hadoop, y aquí modificaremos 2 de ellos principalmente que son:

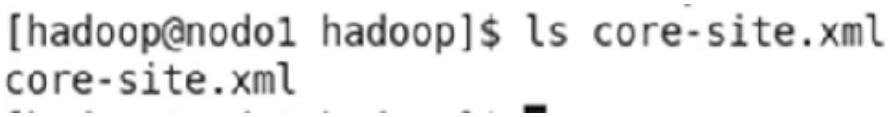


Ilustración 3.38.28

Que es el que contiene las propiedades y configuración general del clúster.

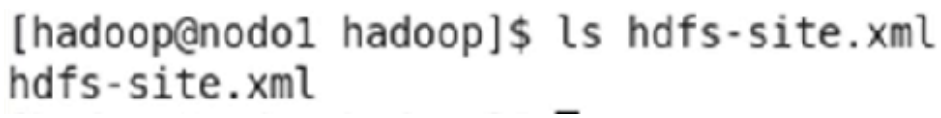


Ilustración 3.39.29

El cual contiene la configuración para los datos, es decir, la del sistema HDFS.

Pero por otro lado, me parece relevante también destacar otros dos que no van a tener tanto protagonismo en esta parte pero que son muy relevantes en un clúster Hadoop como son:

```
[hadoop@nodo1 hadoop]$ ls mapred-site.xml.template
mapred-site.xml.template
```

Ilustración 3.40.30

Es el que configura la operación map reduce sobre la que se basa Spark

```
[hadoop@nodo1 hadoop]$ ls yarn-site.xml
yarn-site.xml
```

Ilustración 3.41.31

Que es el que contiene toda la configuración de la aplicación de Yarn que es la que administra las aplicaciones o ejecuciones en curso de Spark.

Core-site.xml

*core-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>

</configuration>
```

Ilustración 3.42.32

Este es el fichero tal cual nos lo encontramos, así que le vamos a añadir una pequeña propiedad que permitirá indicar la ruta del HDFS que establecerá el sistema principal de almacenamiento:

```
<configuration>
        <property>
                <name>fs.defaultFS</name>
                <value>hdfs://nodo1:9000</value>
        </property>
</configuration>
```

Ilustración 3.43.33

En este caso, como la misma máquina hace de nodo maestro y nodo esclavo, estableceremos el nodo1 en la base de nuestra ruta HDFS, en el puerto por defecto 9000.

Hdfs-site.xml

hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>

</configuration>
```

Ilustración 3.44.34

Como vemos es idéntico al core-site pero recordemos que éste está orientado a la configuración de los datos que vamos a almacenar. Aquí añadiremos diferentes valores como:

```
<property>
        <name>dfs.replication</name>
        <value>1</value>
</property>
```

Ilustración 3.45.35

Ya que solo tenemos un nodo (maestro y esclavo) no podemos acogernos al factor de replicación normal de Hadoop que es de 3, por lo que debemos indicar que los ficheros que carguemos únicamente se replicarán una sola vez, es decir, no se replicarán.

```
<property>
        <name>dfs.namenode.name.dir</name>
        <value>/datos/namenode</value>
</property>
```

Ilustración 3.46.36

Ese parámetro indica en qué ruta se almacenará la información del nodo maestro, es decir, el metadatado.

```
<property>
        <name>dfs.datanode.data.dir</name>
        <value>/datos/datanode</value>
</property>
```

Ilustración 3.47.37

Y éste último indica, dentro de cada nodo esclavo, dónde se van a almacenar realmente los datos. Ya que los datos reales se almacenan en los esclavos y los metadatos y toda la información de la arquitectura se almacena en el nodo maestro, pero en este caso básico, ambos son uno.

En un clúster con varios nodos el namenode estará solo en el maestro y los datanode en los esclavos.

Ahora que hemos definido las rutas del datanode y el namenode, vamos a crearlas para que puedan utilizarse como usuario root:

```
[root@nodo1 ~]# cd /
[root@nodo1 /]# mkdir datos
[root@nodo1 /]# cd datos
[root@nodo1 datos]# mkdir namenode
[root@nodo1 datos]# mkdir datanode
```

Ilustración 3.48.38

Y cambiamos el owner a nuestro usuario hadoop:

```
[root@nodo1 /]# chown -R hadoop:hadoop datos
```

Ilustración 3.49.39

Ahora que ya tenemos el sistema de directorios, vamos a inicializar el sistema de almacenamiento de HDFS (es lo mismo que decir que inicializamos el metadatado):

```
[hadoop@nodo1 hadoop]$ hadoop namenode -format
```

Ilustración 3.50.40

Una vez veamos el mensaje de éxito:

```
16/03/25 00:12:37 INFO util.ExitUtil: Exiting with status 0
16/03/25 00:12:37 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at nodo1/10.0.2.15
************************************************************/
```

Ilustración 3.51.41

El sistema de almacenamiento de HDFS ya estará creado, y podemos cerciorarnos al ver el contenido del namenode:

```
[hadoop@nodo1 hadoop]$ cd /datos
[hadoop@nodo1 datos]$ ls
datanode  namenode
[hadoop@nodo1 datos]$ cd namenode
[hadoop@nodo1 namenode]$ ls -l
total 4
drwxrwxr-x. 2 hadoop hadoop 4096 mar 25 00:12 current
```

Ilustración 3.52.42

Ahora que tenemos todo el sistema de almacenamiento preparado, vamos a arrancar el servicio del DFS para poder interactuar con él:

```
[hadoop@nodo1 datos]$ cd /opt/hadoop/sbin
[hadoop@nodo1 sbin]$ ls
distribute-exclude.sh    start-all.cmd        stop-balancer.sh
hadoop-daemon.sh         start-all.sh         stop-dfs.cmd
hadoop-daemons.sh        start-balancer.sh    stop-dfs.sh
hdfs-config.cmd          start-dfs.cmd        stop-secure-dns.sh
hdfs-config.sh           start-dfs.sh         stop-yarn.cmd
httpfs.sh                start-secure-dns.sh  stop-yarn.sh
kms.sh                   start-yarn.cmd       yarn-daemon.sh
mr-jobhistory-daemon.sh  start-yarn.sh        yarn-daemons.sh
refresh-namenodes.sh     stop-all.cmd
slaves.sh                stop-all.sh
[hadoop@nodo1 sbin]$ start-dfs.sh
```

Ilustración 3.53.43

Y vemos como el comando arranca los servicios del datanode, el namenode y el secondarynamenode (del cual hablaremos más adelante) :

```
[hadoop@nodo1 sbin]$ start-dfs.sh
16/03/25 13:42:01 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Starting namenodes on [nodo1]
nodo1: starting namenode, logging to /opt/hadoop/logs/hadoop-hadoop-namenode-nod
o1.out
localhost: starting datanode, logging to /opt/hadoop/logs/hadoop-hadoop-datanode
-nodo1.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /opt/hadoop/logs/hadoop-hadoop-s
econdarynamenode-nodo1.out
16/03/25 13:42:21 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
[hadoop@nodo1 sbin]$
```

Ilustración 3.54.44

Y vemos que los procesos quedan corriendo en paralelo:

```
[hadoop@nodo1 sbin]$ jps -l
4562 sun.tools.jps.Jps
4259 org.apache.hadoop.hdfs.server.datanode.DataNode
4435 org.apache.hadoop.hdfs.server.namenode.SecondaryNameNode
4125 org.apache.hadoop.hdfs.server.namenode.NameNode
```

Ilustración 3.55.45

Ahora que hemos arrancado el servicio del DFS no solo tenemos el namenode en marcha sino que el datanode también está activo:

```
[hadoop@nodo1 datos]$ cd datanode/
[hadoop@nodo1 datanode]$ ls
current  in use.lock
```

Ilustración 3.56.46

Para comprobar que nuestro hadoop está corriendo actualmente y se encuentra en un estado HEALTHY también podemos utilizar una herramienta gráfica conectándonos al siguiente puerto desde un navegador: (observamos que es la máquina nodo1 en el puerto que definimos)

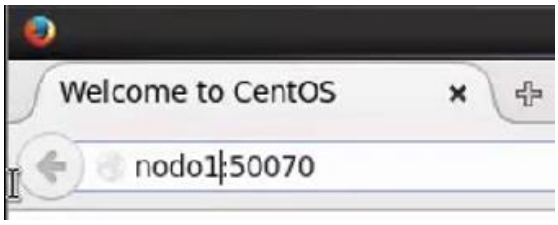


Ilustración 3.57.47

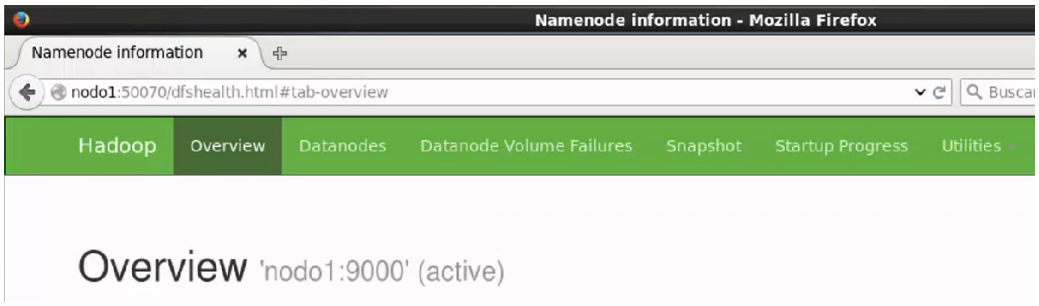


Ilustración 3.58.48

En la página encontramos bastante información acerca de nuestro HDFS y su estado, además de algunas pequeñas interacciones que podemos realizar. Destacar la pestaña de SUMMARY que nos informa de las características del servicio corriendo:

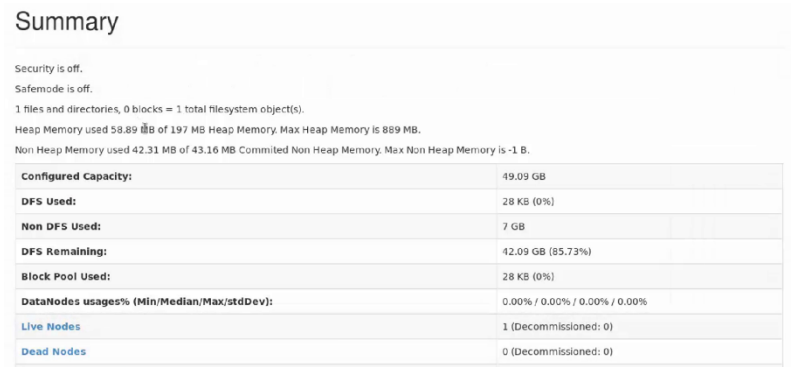


Ilustración 3.59.49

La pestaña DataNodes que nos informa del estado de los distintos nodos conectados a la máquina:

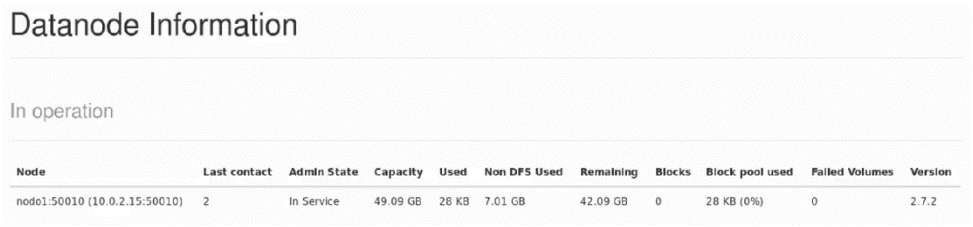


Ilustración 3.60.50

En nuestro caso solo tenemos uno, y podemos ver que está sano:

Ilustración 3.61.51

Y luego las funcionalidades de las que hablaba anteriormente las podemos encontrar en Utilities:

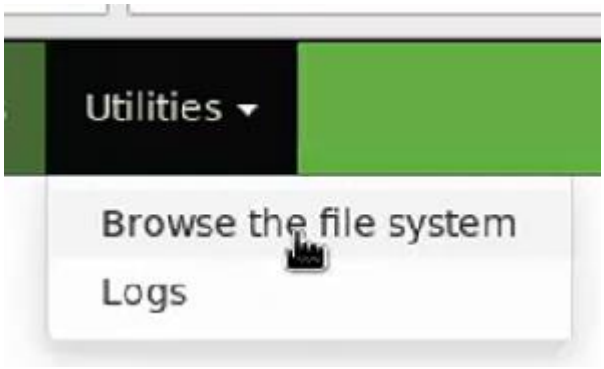


Ilustración 3.62.52

El Browse Directory que nos permite ver los ficheros almacenados actualmente por nuestro HDFS:

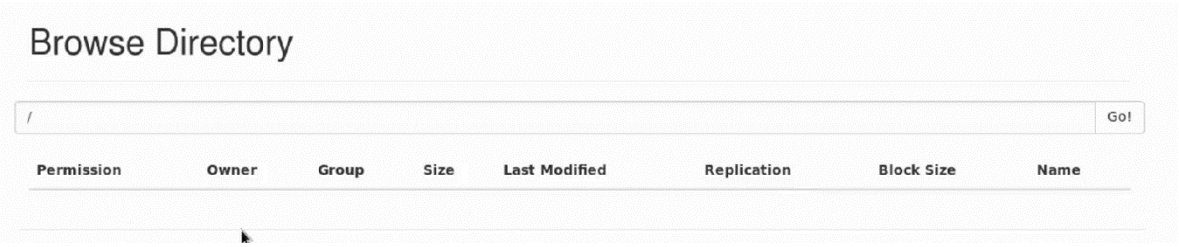


Ilustración 3.63.53

Y la ventana de logs:

Ilustración 3.64.54

UN PEQUEÑO VISTAZO A HDFS

HDFS dispone de unos ficheros que gestionan los cambios que se producen en el clúster de HDFS. Básicamente son:

- Edits_000xxxxxxxx
- Edits_inrpgoress_xxxxxxxx
- Fsimage_00000xxxxxxxxx

FsImage es una foto de un momento concreto del estado de HDFS. Al arrancar HDFS se carga en memoria el último fichero fsimage disponible junto con los edits que no hayan sido procesados, realmente el fsimage no es modificado directamente sino que se construye el edits_inprogress a partir de éste, por temas de rendimiento. Cada cierto tiempo, fsimage y edits_inprogress se sincronizan a través del secondary NameNode.

Nuestros cambios se almacenan en el fichero edits_000xxx y cada hora o cada que 128MB de memoria se llenan (suelen ser un bloque de memoria), el proceso del secondary NameNode genera con los edits y el fsimage anterior, un fsimage nuevo y se sigue trabajando con un fichero de edits nuevo a partir de este punto.

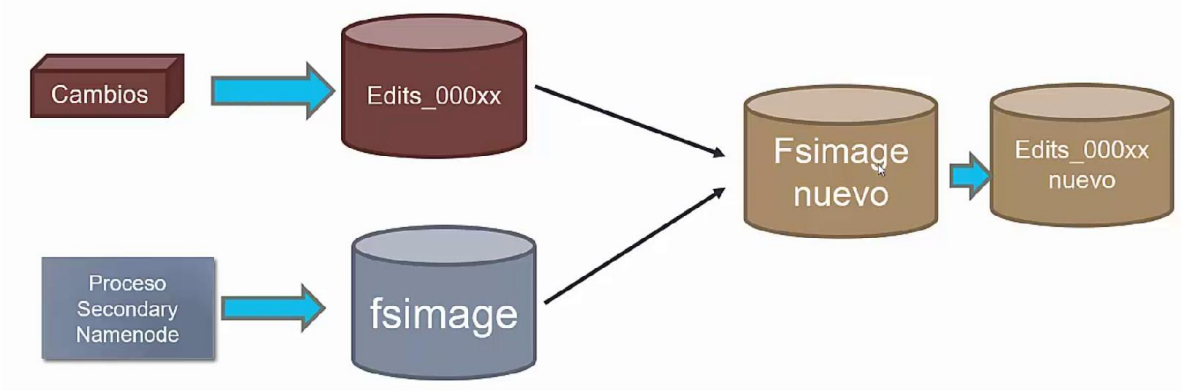


Ilustración 3.65.55

TRABAJAR CON FICHEROS EN HDFS

Observaremos que los comandos que utiliza Hadoop son muy similares a los de Unix, solo que cada vez que ejecutemos una orden a través del HDFS, ésta se estará realizando en el sistema de almacenamiento en lugar de nuestra terminal.

Lo primero será arrancar nuestro servicio de HDFS:

```
[hadoop@nodo1 ~]$ start-dfs.sh
16/03/26 21:20:46 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
Starting namenodes on [nodo1]
nodo1: starting namenode, logging to /opt/hadoop/logs/hadoop-hadoop-namenode-nodo1.out
localhost: starting datanode, logging to /opt/hadoop/logs/hadoop-hadoop-datanode-nodo1.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /opt/hadoop/logs/hadoop-hadoop-secondarynamenode-nodo1.out
16/03/26 21:21:04 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
```

Ilustración 3.66.56

Nos cercioramos que se han arrancado todos los servicios que necesitamos con JPS:

```
[hadoop@nodo1 ~]$ jps
3506 NameNode
3939 Jps
3643 DataNode
3823 SecondaryNameNode
```

Ilustración 3.67.57

Ahora sí, vamos a ver que tenemos en nuestro sistema de almacenamiento HDFS recién creado:

```
[hadoop@nodo1 bin]$ hdfs dfs -ls /
16/03/26 21:24:17 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
```

Ilustración 3.68.58

Como era de esperar no hay nada, por ahora. El mensaje que aparece es un warning que nos indica que las librerías nativas de hadoop (una seríe de librerías utilizadas en proyectos Big Data para el funcionamiento más óptimo posible del software) no están instaladas ahora mismo, por lo que lo podemos ignorar.

Dado que nuestro sistema de almacenamiento se encuentra vacío ahora mismo, vamos a proceder a insertar un fichero de prueba que podamos observar con el comando **hdfs dfs –ls.**

```
[hadoop@nodo1 bin]$ echo Hola >> prueba.txt
[hadoop@nodo1 bin]$ cat prueba.txt
Hola
```

Ilustración 3.69.59

Ahora creamos una carpeta temporal en nuestro sistema de archivos para almacenar el fichero:

```
[hadoop@nodo1 bin]$ hdfs dfs -mkdir /temporal
16/03/26 21:25:46 WARN util.NativeCodeLoader: Unable to load native-hac
ltin-java classes where applicable
[hadoop@nodo1 bin]$ hdfs dfs -ls /
16/03/26 21:25:55 WARN util.NativeCodeLoader: Unable to load native-hac
ltin-java classes where applicable
Found 1 items
drwxr-xr-x   - hadoop supergroup          0 2016-03-26 21:25 /temporal
[hadoop@nodo1 bin]$
```

Ilustración 3.70.60

Subimos el fichero al directorio temporal:

```
[hadoop@nodo1 bin]$ hdfs dfs -put prueba.txt /temporal
```

Ilustración 3.71.61

Y vemos que se haya subido correctamente:

```
[hadoop@nodo1 bin]$ hdfs dfs -ls /temporal
16/03/26 21:27:11 WARN util.NativeCodeLoader: Unable to load native-hadoop library
ltin-java classes where applicable
Found 1 items
-rw-r--r--   1 hadoop supergroup          5 2016-03-26 21:26 /temporal/prueba.txt
```

Ilustración 3.72.62

También podemos utilizar el navegador como hicimos anteriormente para acceder al puerto de Hadoop y ver el sistema de ficheros:

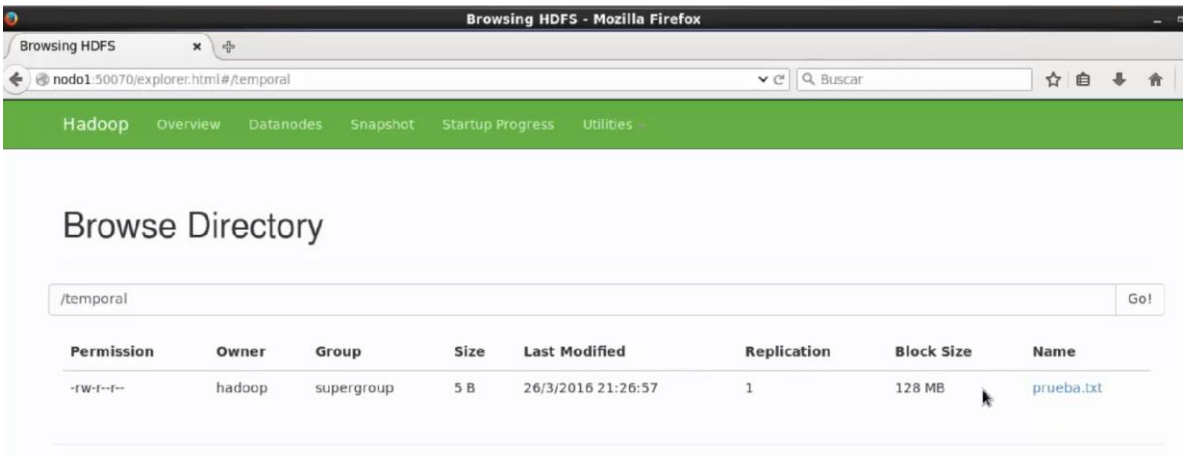


Ilustración 3.73.63

En la imagen también observamos características que hemos mencionado anteriormente como los bloques de memoria (de 128 MB) o la replicación (1 en este caso), ambas características definidas por nosotros anteriormente en los ficheros de configuración de Hadoop.

También desde la propia terminal y utilizando los comandos de Hadoop, podemos ver el contenido de nuestro fichero.

```
[hadoop@nodo1 ~]$ hdfs dfs -cat /temporal/prueba.txt
16/03/27 12:50:09 WARN util.NativeCodeLoader: Unable
ltin-java classes where applicable
Hola
```

Ilustración 3.74.64

Para demostrar un poco la similitud de estos comandos con los de unix, vamos a realizar una pequeña prueba en la que generamos una copia del fichero en el propio HDFS, la movemos a través del clúster, y finalmente la descargamos en nuestro sistema de ficheros local:

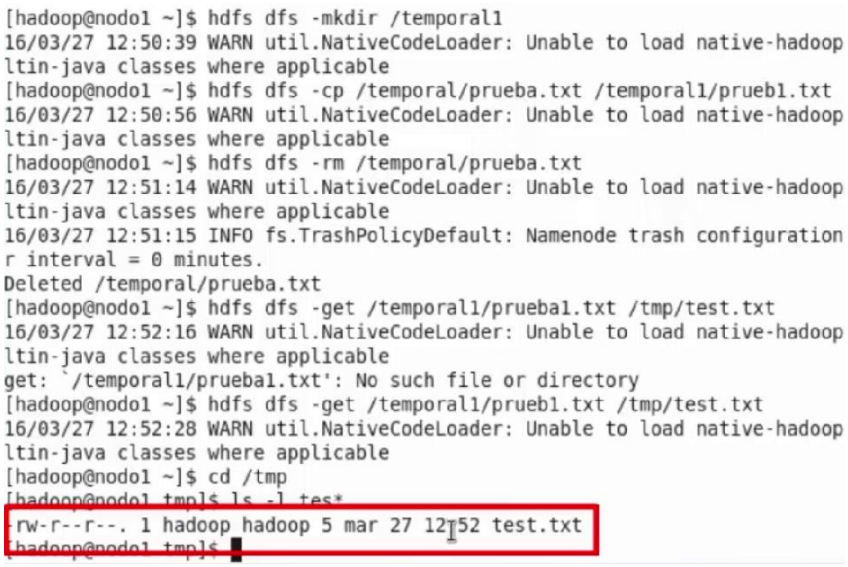


Ilustración 3.75.65

YARN – MapReduce

En el capítulo anterior hemos hablados de los datos de Hadoop, como funciona su estructura, mover ficheros y su sistema de gestión. Ahora, vamos a hablar sobre los distintos procesos con los que funciona este sistema de almacenamiento. Solían existir dos versiones de éstos procesos:

- MapReduce V1
- MapReduce V2 (más conocida como Yarn)

Pero en la actualidad, únicamente se emplea la segunda, e incluso ésta empieza a quedarse un poco por detrás respecto a otros clouds. Las diferencias que podemos encontrar entre ambas son las siguientes:

- MapReduce V1
  - Pensada sobre todo para procesos Batch.
  - Se encarga tanto del proceso de los datos como de la gestión del clúster
- Yarn
  - Admite otro tipo de productos y procesos que no sean Batch (streaming)
  - Tiene procesos distintos para el proceso de datos y para la gestión del clúster.

Es esta dfiferenciación que nos permite mover el MapReduce de manera que deja de ser un eje central para la arquitectura de Hadoop y pasa a ser un componente dedicado únicamente al procesamiento de los datos, y el Yarn se encargará de aquí en adelante de la gestión de recursos de los distintos procesos que se ejecuten.

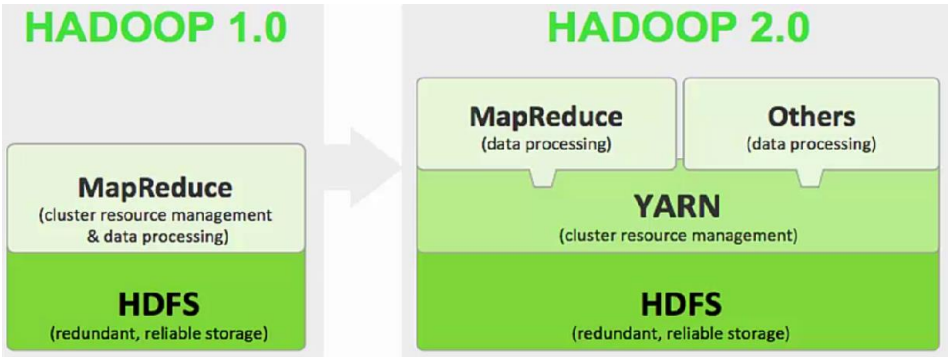


Ilustración 3.76.66

Además, el MR1 presenta problemas de rendimiento, pese a ser en su momento una solucíon muy buena para proyectos Big Data, conforme estos han seguido creciendo, ha empezado a quedarse atrás. Entre sus problemas destacamos principalmente:

- Solo ofrece procesos de tipo Map Reduce
- Tiene problemas de escalabilidad a partir de los 5000 nodos
- Problemas de rendimiento, con aplicaciones que no son diseñadas para trabajar Batch
- No gestiona correctamente los recursos

La arquitectura de MR1 era un poco la siguiente:

- Job Tracker
  - Nodo maestro
  - Gestiona los recursos del clúster y la planificación de los trabajos
- Task Tracker
  - Agente en los esclavos
  - Gestiona las tareas

Y ahora en Yarn, cambia completamente la historia, ya que “desgranamos” las aplicaciones principales en otras con tareas específicas que permiten una distribución de la carga de trabajo y de los procesos mucho más homogénea. Ahora tenemos las 3 siguientes tareas principalmente encargadas de la gestión:

- ResourceManager (RM)
  - En el maestro: Gestiona los recursos del clúster
- NodeManager (NM)
  - En los esclavos: Gestiona los recursos del nodo
- ApplicationMaster (AM)
  - Uno por aplicación en funcionamiento: Gestiona el ciclo de vida y la planificación de la aplicación

MAP REDUCE 1

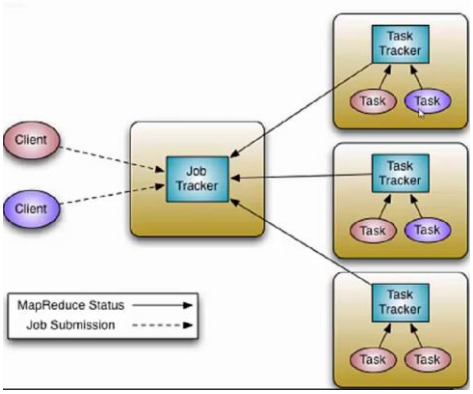


Ilustración 3.77.67

YARN

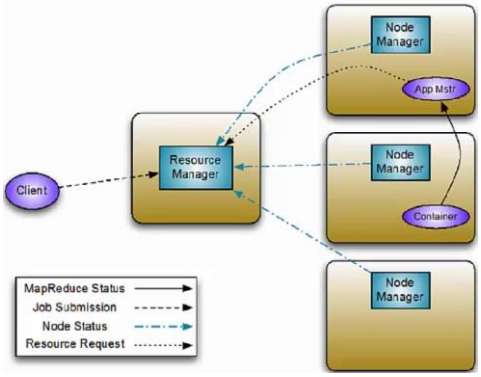


Ilustración 3.78.68

FUNCIONAMIENTO DE YARN

Supongamos una arquitectura con 4 servidores, donde uno de ellos actuará de nodo maestro y los otros 3 de nodos hijo. Dentro de este nodo maestro encontraremos 3 aplicaciones principalmente:

- **Scheduler**: que es el encargado de planificar las tareas a asignar.
- **Application manager**: que como ya hemos visto, se encarga del ciclo de vida de las apps.
- **Security**: implementa algunas medidas de seguridad por parte del nodo.

Cuando se lanza una aplicación, lo primero que ocurre es que se solicita al application manager que inicie un “app master” el cual es un coordinador de la aplicación que hemos lanzado. Dicho app master será lanzado en el nodo hijo que el Scheduler nos comunique. Cada aplicación corriendo en el clúster tiene asociado un app máster, por lo que si tengo un clúster con 101 nodos y estoy lanzando 100 aplicaciones, dentro de cada nodo hijo tendré un app master asociado a su aplicación y que se encargará de coordinarla (éste es uno de los motivos por los que Yarn tiene tanta capacidad de escalabilidad). Ahora el app master, en coordinación con el scheduler, se encargará de levantar un contenedor donde poder correr su aplicación. En nuestro supuesto vamos a pensar que la aplicación lanzada es del tipo MapReduce, por lo que tiene sentido pensar que el contenedor tendrá una operación de Map o de Reduce. Una vez están los contenedores levantados la carga de trabajo queda repartida y el proceso entra en estado RUNNING gracias a Yarn que es la capa que lo coordina completamente.

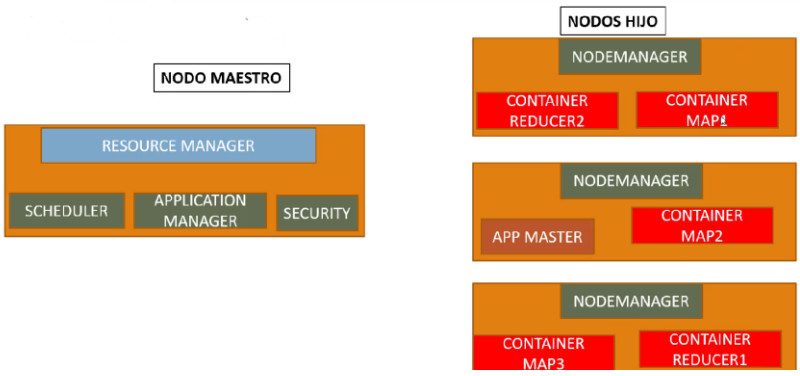


Ilustración 3.79.69

El Yarn no hace ningún trabajo de aplicación como tal, está dedicado únicamente a la gestión de éstas y de los recursos del clúster de manera autónoma gracias a que dentro de cada nodo hijo, encontraremos los NodeManager, que gestionarán los recursos de cada hijo.

CONFIGURANDO YARN

Ahora que sabemos como funciona Yarn, vamos a implementarlo dentro de nuestra máquina Hadoop. De cara a la claridad del montaje, vamos a eliminar el mensaje de WARNING que nos salía anteriormente con los comandos:

```
export HADOOP_HOME_WARN_SUPPRESS=1
export HADOOP_ROOT_LOGGER="WARN,DRFA"
```

Ilustración 3.80.70

Vamos a detener nuestro Hadoop por ahora y vamos a acceder a la carpeta que contiene la configuración del entorno:

```
[hadoop@nodo1 ~]$ stop-dfs.sh
Stopping namenodes on [nodo1]
nodo1: stopping namenode
localhost: stopping datanode
Stopping secondary namenodes [0.0.0.0]
0.0.0.0: stopping secondarynamenode
[hadoop@nodo1 ~]$ cd /opt/hadoop/etc/hadoop/
```

Ilustración 3.81.71

Dado que vamos a añadir una configuración adicional para el funcionamiento de nuestro yarn, vamos a utilizar uno de los ficheros plantilla de configuración que ofrece Hadoop para poder trabajar desde un punto más cómodo:

```
[hadoop@nodo1 hadoop]$ cp mapred-site.xml.template mapred-site.xml
```

Ilustración 3.82.72

Nuestro fichero “virgen” luce así:

mapred-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>

</configuration>
```

Ilustración 3.83.73

Añadimos la siguiente propiedad, que indica que el motor y el framework que va a gestionar el Hadoop va a ser del tipo yarn:

```
<configuration>
        <property>
                <name>mapreduce.framework.name</name>
                <value>yarn</value>
        </property>
</configuration>
```

Ilustración 3.84.74

A continuación, procedemos a modificar el fichero de configuración específica de Yarn:

```
[hadoop@nodo1 hadoop]$ gedit yarn-site.xml
```

Ilustración 3.85.75

Que luce de manera similar a nuestra plantilla anterior, por lo que procedemos a añadir las siguientes propiedades:

- Para indicar que la maquina que va a contener el resourcemanager del clúster y del Yarn va a ser la que estamos usando actualmente (recordemos que por ahora, estamos en una instalación monoclúster):

```
<property>
        <name>yarn.resourcemanager.hostname</name>
        <value>nodo1</value>
</property>
```

Ilustración 3.86.76

- Indicamos que el gestor o servicios auxiliares del nodemanager van a ser del tipo shuffle:

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

Ilustración 3.87.77

- Y ahora especificamos que ese servicio auxiliar anterior al que hacemos referencia, se trata de la operación mapred.ShuffleHandle del propio Hadoop:

```
<property>
        <name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

Ilustración 3.88.78

Y con esto habríamos terminado con la configuración (básica) de nuestro Yarn. Para cerciorarnos de que la instalación ha sido correcta, vamos a arrancar nuestro clúster:

```
[hadoop@nodo1 hadoop]$ start-dfs.sh
```

Ilustración 3.89.79

Con nuestro clúster ya arrancado, vamos a inicializar también el servicio de Yarn:

```
[hadoop@nodo1 hadoop]$ start-yarn.sh
```

Ilustración 3.90.80

Si la configuración ha sido correcta, deberíamos ver como arranca el resourcemanager y el nodemanager. El applicationmanager no lo arrancará hasta que no lancemos una aplicación:

```
starting yarn daemons
starting resourcemanager, logging to /opt/hadoop/logs/yarn-hadoop-resourcemanager-nodo1.out
localhost: starting nodemanager, logging to /opt/hadoop/logs/yarn-hadoop-nodemanager-nodo1.out
```

Ilustración 3.91.81

Y vemos como ahora está corriendo todo en el mismo nodo, por ahora, pero están todos los servicios:

```
[hadoop@nodo1 hadoop]$ jps
15171 Jps
14437 DataNode
14870 NodeManager
14616 SecondaryNameNode
14300 NameNode
14767 ResourceManager
```

Ilustración 3.92.82

Al igual que con HDFS, Yarn también tiene una página web donde visualizar su configuración, la cual podemos encontrarla en el puerto 8088:

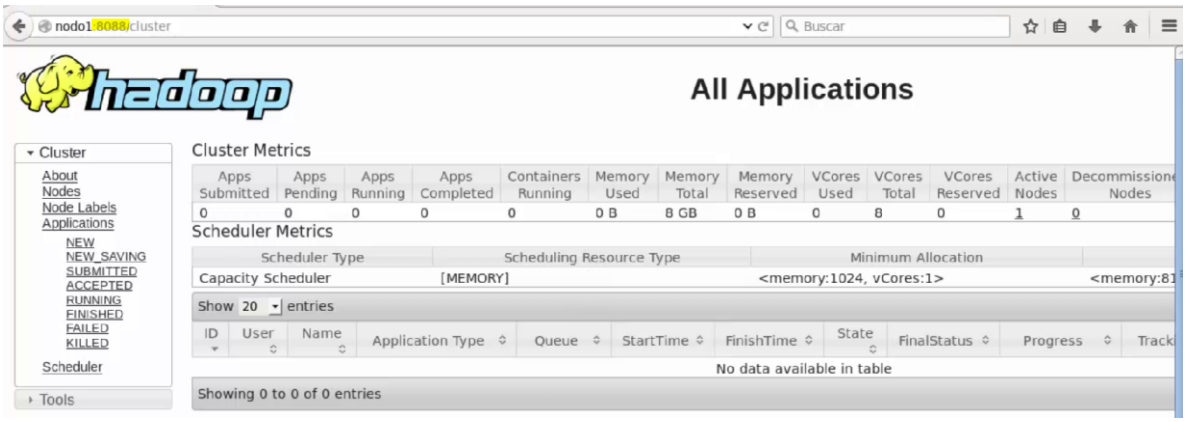


Ilustración 3.93.83

Aquí podemos observar las características del clúster y las métricas del scheduler. Ahora mismo está vació porque no hay ninguna aplicación corriendo. Además el navegador del clúster nos permite ver las características de los nodos, la memoria disponible, los recursos a asignar a cada nodo, etc…

En notas generales, esta página nos permitirá ver de manera horizontal todo lo que ocurre en nuestro clúster y además nos permitirá más adelante interactuar con la interfaz del propio Spark para ver qué recursos y en qué etapa se encuentra cada aplicación con todo lujo de detalles.

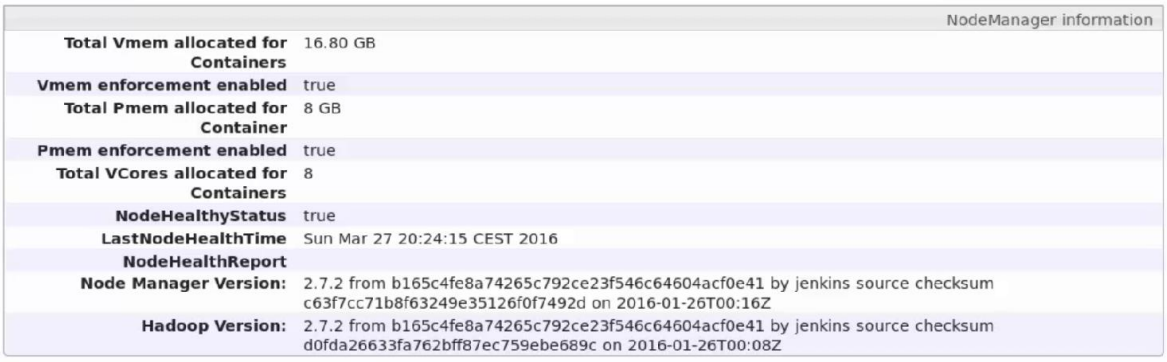


Ilustración 3.94.84

Ésta es una pagina muy potente donde podemos ver mucho detalle de nuestra máquina, pero las más relevantes son las que más hemos destacado junto a la de los contenedores que veremos más adelante cuando lancemos alguna aplicación.

MAP REDUCE

Más adelante realizaremos un ejemplo de ejecución de una aplicación map reduce, pero antes, se hace conveniente repasar en qué consiste ésta operación y qué esperamos de ella cuando la lancemos. Pese a que Yarn ha cogido mucho protagonismo en la máquina ahora, map reduce sigue siendo parte indispensable del clúster, y es el eje entorno al que van a girar las aplicaciones que se ejecuten en ella.

La mayor parte de las herramientas de consulta están diseñadas para realizar consultas simples que deben ejecutarse rápidamente, el dato suele estar indexado y por tanto solo pequeñas porciones de datos se examinan durante la búsqueda. Esta solución, no es útil para datos no indexados de tipo semi estructurado (textos) o sin estructurar (multimedia), ya que para responder a una query en esta situación, se hace necesario examinar todos los datos. Y es aquí dónde entra en juego el Map Reduce que Hadoop utiliza para realizar un análisis exhaustivo de forma rápida.

El Map Reduce es un algoritmo de procesamiento de datos en paralelo, que de forma simple distribuye las tareas a través de los nodos de un cluster y luego las ejecuta. La función map estudia el problema, lo divide en trozos y los manda a diferentes máquinas para que todos los trozos puedan ejecutarse concurrentemente. Los resultados de este proceso paralelo se recogen y se distribuyen a través de distintos servidores que ejecutan una función “reduce”, que toma los resultados de los trozos y los recombina para obtener una respuesta simple.

Vamos a desgranar un poco más las partes internas de un Map Reduce:

- Procesos Mapper
  - Los programas Map Reduce se dividen en Mappers, tareas que se ejecutan en los nodos.
  - Cada tarea MAP ataca a un solo bloque de datos HDFS.
  - Se ejecuta en el nodo donde reside el bloque (salvo excepciones).
- Shuffle y Sort
  - Ordena y consolida los datos intermedios (temporales) que generan los mappers.
  - Se lanzan después de que todos los mappers hyan terminado y antes de que se lancen los procesos Reducer.
- Reducer
  - Operan sobre los datos intermedios Shuffle/sort
  - Generan la salida final.

Esto sería un ejemplo de Map Reduce:

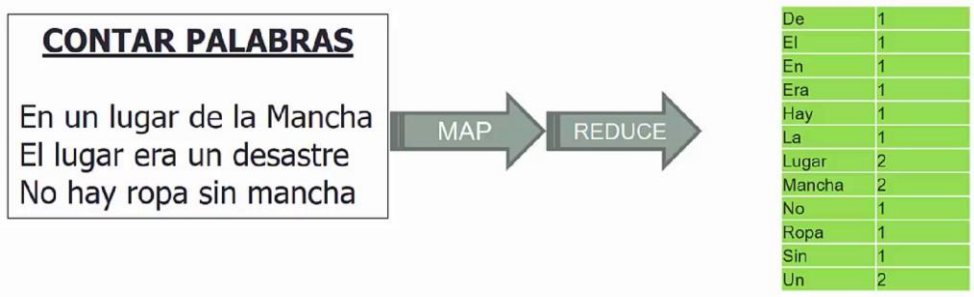


Ilustración 3.95.85

Vamos a ver, siguiendo el procedimiento que explicábamos anteriormente, cómo ha funcionado este proceso de Map Reduce.

- El primer paso como comentábamos antes, son los Mapper. Supongamos que cada una de las frases anteriores representa un bloque de información, y que por lo tanto se distribuye de la siguiente manera:

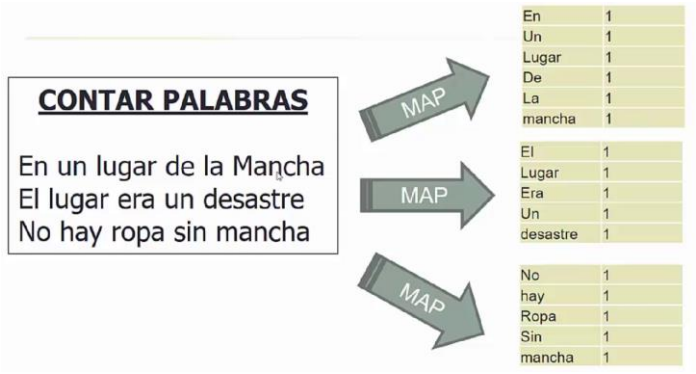


Ilustración 3.96.86

- Justo después de mapear los datos, se produce la fase de Shuffle. Anidamos los datos a modo de diccionario tipo clave-valor donde la clave es la palabra que aparece y el valor, el número de veces que lo hace, pero como aún no realizamos ningún procesamiento ni ningún cálculo, solo estamos haciendo shuffle, queda algo tal que así:

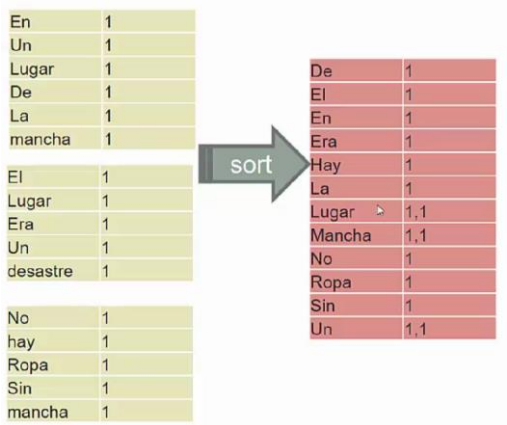


Ilustración 3.97.87

- Y por último, la fase de Reduce, donde cogemos el resultado del shuffle anterior, aplicamos la operación correspondiente (en este caso, contar palabras y por tanto sumar los valores) y nos da como resultado final, el número de veces que aparece cada palabra:

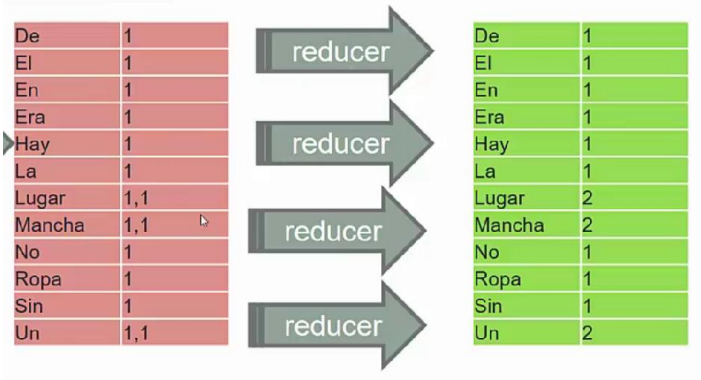


Ilustración 3.98.88

EJECUTANDO UN MAP REDUCE EN NUESTRA MÁQUINA

Para nuestro ejemplo, tanto de ejecución de un map reduce como de lo que es la ejecución de una aplicación en nuestro clúster, vamos a contar las palabras y el número de veces que aparecen en el libro El Quijote. Para ello hemos descargado previamente el libro en formato .txt desde la página El proyecto Gutenberg. Una vez lo tenemos, lo insertamos en nuestro sistema HDFS:

```
[hadoop@nodo1 tmp]$ hdfs dfs -mkdir /libros
[hadoop@nodo1 tmp]$ hdfs dfs -put quijote.txt /libros
[hadoop@nodo1 tmp]$ hdfs dfs -ls /libros
Found 1 items
-rw-r--r--   1 hadoop supergroup        302 2016-03-27 22:12 /libros/quijote.txt
[hadoop@nodo1 tmp]$
```

Ilustración 3.99.89

Cómo ya existe un script propio de Hadoop para la contabilidad de palabras, lo vamos a utilizar para ver que el clúster funciona correctamente y que podemos correr aplicaciones con nuestra nueva configuración del Yarn. Ya que se trata de un script del propio Hadoop, las rutas que empleamos son las mismas del HDFS ya que Hadoop corre en ese sistema de almacenamiento y no en nuestro local, quedando la ejecución algo similar a esto:

```
[hadoop@nodo1 mapreduce]$ hadoop jar hadoop-mapreduce-examples-2.7.2.jar wordcount /libros /salida_libros
```

Ilustración 3.100.90

Donde también le indicamos la salida que puede crear el mismo script. Procedemos a consultar el HDFS para ver si la ejecución ha sido correcta:

```
[hadoop@nodo1 mapreduce]$ hdfs dfs -ls /salida_libros
Found 2 items
-rw-r--r--   1 hadoop supergroup          0 2016-03-27 22:14 /salida_libros/_SUCCESS
-rw-r--r--   1 hadoop supergroup        345 2016-03-27 22:14 /salida_libros/part-r-00000
```

Ilustración 3.101.91

Parece que sí. Vamos a descargar el fichero que contiene los resultados (part-r-00000) y vamos a consultar su contenido para ver si realmente ha generado el resultado que esperaríamos:

```
[hadoop@nodo1 mapreduce]$ hdfs dfs -get /salida_libros/part-r-00000 /tmp/palabras.txt
```

Ilustración 3.102.92

```
[hadoop@nodo1 tmp]$ vi palabras.txt
```

Ilustración 3.103.93

```
¡Mal    1
"Al     1
"Cuando 2
"Cuidados       1
"De     2
"Defects,"      1
"Desnudo        1
"Dijo   1
"Dime   1
"Don    1
"Donde  1
```

Ilustración 3.104.94

Y vemos como se ha realizado la contabilidad completa de palabras del Quijote, habiendo tardado la ejecución de éste menos de 3 segundos.

Una vez finalizada la ejecución, ahora sí hemos lanzado una aplicación en nuestro clúster que en principio ha sido coordina tanto como por el Yarn como por el Application Manager. Así que volvemos a la página de Yarn para ver qué ha ocurrido.

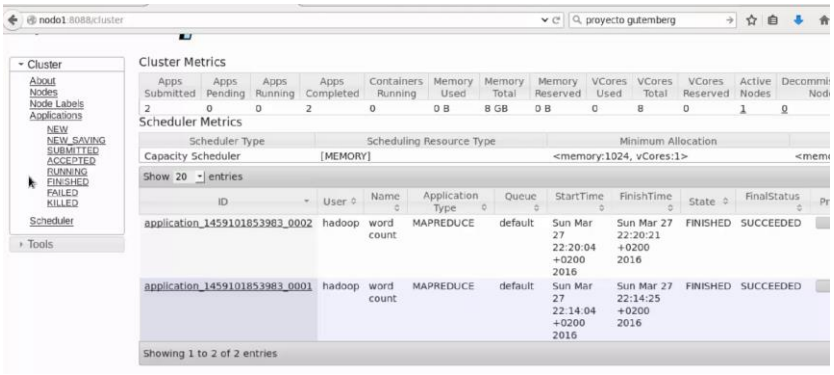


Ilustración 3.105.95

Cómo podemos ver finalizó con éxito y se le asignó un contenedor (ID) en el momento de ejecución. Si hacemos click en una de las aplicaciones, podemos ver todo el detalle que nos ofrece Yarn acerca de su ejecución:

Ilustración 3.106.96

Podemos ver también en qué nodo se ha asignado la tarea y el log:

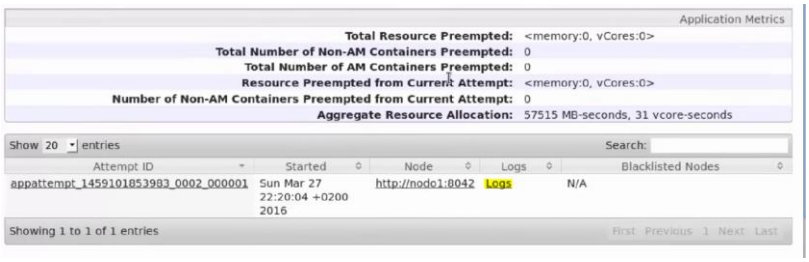


Ilustración 3.107.97

Ilustración 3.108.98

MONTANDO UN CLUSTER REAL

Dado que ya tenemos una máquina de un nodo, el cual funciona correctamente, vamos a aprovechar a la hora de realizar el montaje del clúster ese mismo nodo para clonarlo y que haga las veces de los nodos esclavos. Partimos de la máquina principal que hemos estado utilizando hasta ahora:

```
[hadoop@nodo1 Escritorio]$ uname -a
Linux nodo1 2.6.32-573.el6.x86_64 #1 SMP Thu Jul 23 15:44:03 UTC 2015 x86_64 x86
_64 x86_64 GNU/Linux
```

Ilustración 3.109.99

Con la máquina apagada, vamos a clonar dicha máquina para crear el nodo 2 y el nodo 3. Lo primero que vamos a tener que cambiar, es la configuración de Red:

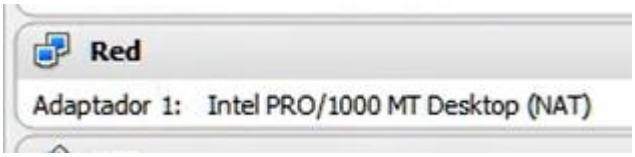


Ilustración 3.110.100

Ya que nuestras máquinas van a convivir en la misma red de equipos informáticos que el sistema de virtualización va a simular por nosotros. Nuestro adaptador 1 lo dejaremos con la siguiente configuración para no perder nuestra salida a internet:

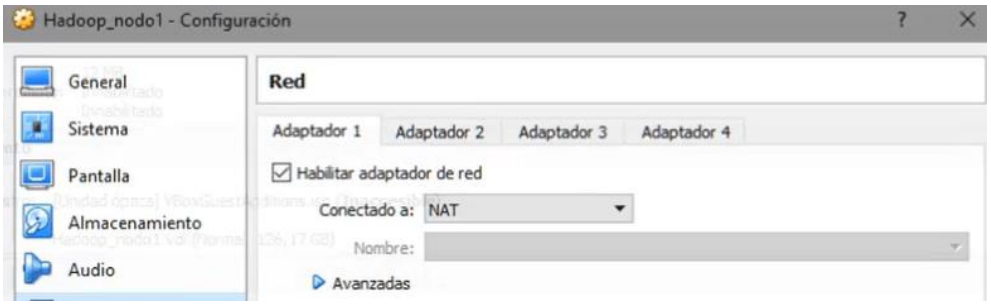


Ilustración 3.111.101

Nuestro adaptador 2 ahora lo configuraremos para que funciona con la red interna (como si estuvieramos simulando otra tarjeta de red) y esta conexión nos permitirá conectarnos con los distintos nodos a nivel interno:

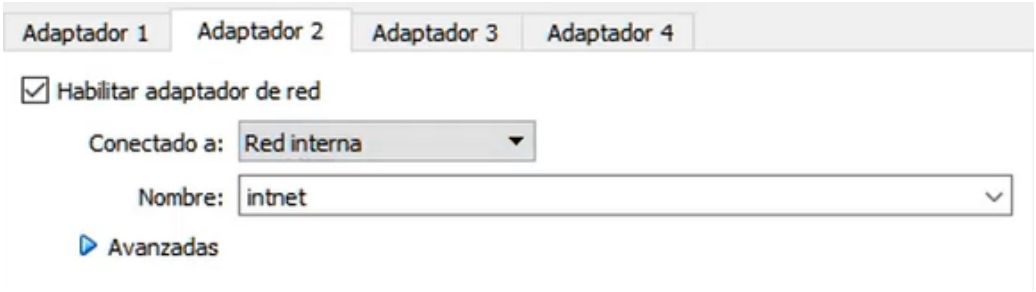


Ilustración 3.112.102

A continuación, y con el nodo ya preparado, vamos a clonar nuestro nodo ya que los 3 compartirán esta configuración de red. Eso lo podemos hacer con botón derecho -> clonar. Es importante marcar la siguiente opción para que los distintos nodos tengan diferente dirección MAC:

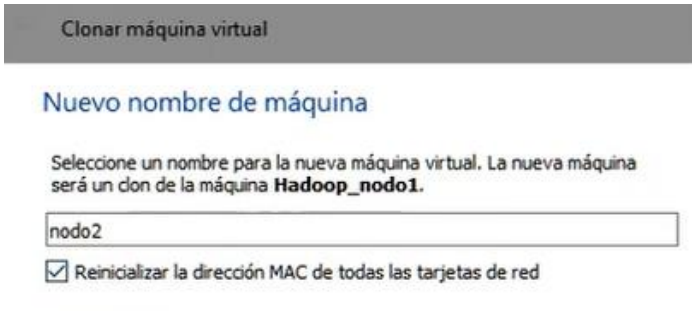


Ilustración 3.113.103

Una vez hecho, deberiamos tener un listado de 3 máquinas, nodo1, nodo2 y nodo3. Ahora mismo, aunque las hayamos renombrado desde el software de virtualización, comparten nombre interno y algunes ajustes de configuración de red. Por lo que iniciamos las distintas máquinas y cambiamos su nombre por el correspondiente.

```
[hadoop@nodo1 Escritorio]$ hostname nodo3
hostname: you must be root to change the host name
[hadoop@nodo1 Escritorio]$ su - root
Contraseña:
[root@nodo1 ~]# hostname nodo3
[root@nodo1 ~]# uname -a
Linux nodo3 2.6.32-573.el6.x86_64 #1 SMP Thu Jul 23 15:44:03 UTC 2015 x86_64 x86_64 x86_64 GNU/
Linux
```

Ilustración 3.114.104

Y configuramos el fichero network para asegurarnos que la máquina es reconocida por ese nombre dentro de la red interna:

```
[root@nodo1 ~]# cd /etc/sysconfig
[root@nodo1 sysconfig]# vi network
```

```
NETWORKING=yes
HOSTNAME=nodo3
```

Ilustración 3.115.105

Ahora, para que las máquinas puedan “verse y reconocerse” entre ellas, vamos a configurar el fichero de Hosts dentro de éstas (recordamos que aquí no tenemos DNS):

```
[root@nodo1 sysconfig]# cd /etc
[root@nodo1 etc]# vi hosts
```

Ilustración 3.116.106

```
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.0.101   nodo1
192.168.0.102   nodo2
192.168.0.103   nodo3
```

Ilustración 3.117.107

Y las asignamos en sucesión para facilitar el trabajo. Una vez hecho esto (en todos los nodos) vamos a aprovechar la interfaz gráfica de CentOs para realizar los últimos ajustes de la red.

En las conexiones de red, deberían aparecernos las dos tarjetas de red que estamos simulando actualmente:

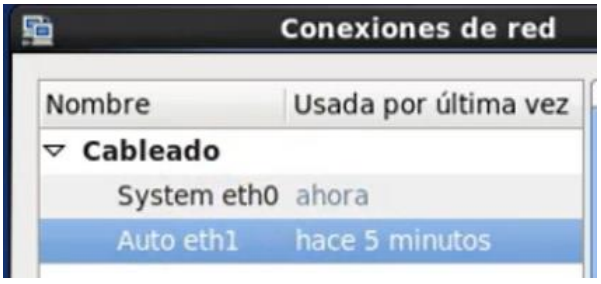


Ilustración 3.118.108

Dado que la IP va a ser fija para nuestras máquinas y la conocemos porque la hemos configurado nosotros mismos, podemos modificar el IPv4 para dichas conexiones de la siguiente manera:

Ilustración 3.119.109

Usamos el comando ifconfig para asegurarnos de que la conexión de nuestra red es la que esperamos:

```
[hadoop@nodo1 Escritorio]$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:3F:F2:7D
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe3f:f27d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:25 errors:0 dropped:0 overruns:0 frame:0
          TX packets:28 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4150 (4.0 KiB)  TX bytes:2553 (2.4 KiB)

eth1      Link encap:Ethernet  HWaddr 08:00:27:A2:D3:B9
          inet addr:192.168.0.101  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fea2:d3b9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:30 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:7548 (7.3 KiB)
```

Ilustración 3.120.110

Ahora que la red está configurada, necesitamos que las máquinas puedan establecer conexiones seguras entre ellas para el intercambio de información. Dado que en pasos anteriores hemos realizado configuraciones del ssh y de la clave pública orientadas al monoclúster, vamos a borrar la configuración de éste de todas las máquinas para poder realizarla en ésta nueva composición:

```
[hadoop@nodo1 ~]$ cd .ssh
[hadoop@nodo1 .ssh]$ ls -l
total 16
-rw-rw-r--. 1 hadoop hadoop  394 mar 28 12:01 authorized_keys
-rw-------. 1 hadoop hadoop 1675 mar 28 12:01 id_rsa
-rw-r--r--. 1 hadoop hadoop  394 mar 28 12:01 id_rsa.pub
-rw-r--r--. 1 hadoop hadoop  802 mar 28 11:59 known_hosts
[hadoop@nodo1 .ssh]$ rm *
```

Ilustración 3.121.111

Ilustración 3.122.112

Una vez la hemos borrado de todos los nodos, las vamos a volver a generar (ya que anteriormente hemos hecho este proceso, no se hace necesario repetirlo aquí). Una vez tenemos los ficheros de clave pública generados, los moveremos a las authorized_keys de los distintos nodos.

```
[hadoop@nodo1 .ssh]$ scp authorized_keys nodo2:/home/hadoop/.ssh
hadoop@nodo2's password:
authorized_keys
```

Ilustración 3.123.113

Observamos que ahora mismo nos pide la contraseña ya que todavía no hemos compartido las claves públicas por los distintos nodos. Comprobamos que el fichero se ha copiado correctamente al nodo2:

```
[hadoop@nodo2 .ssh]$ ls -l
total 4
-rw-rw-r--. 1 hadoop hadoop 394 mar 28 12:06 authorized_keys
```

Ilustración 3.124.114

Repetimos este proceso para el resto de nodos para que compartan las claves y también las agregamos al fichero authorized_keys. De manera que todos los nodos poseen todas las credenciales del resto de nodos.

Ilustración 3.125.115

Ya tenemos los nodos conectados entre sí, con la capacidad de establecer conexiones seguras entre ellos y la red preparada para administrar dichas conexiones, por lo que solo nos queda configurar el clúster para que admita la ejecución trabajando con todos los nodos.

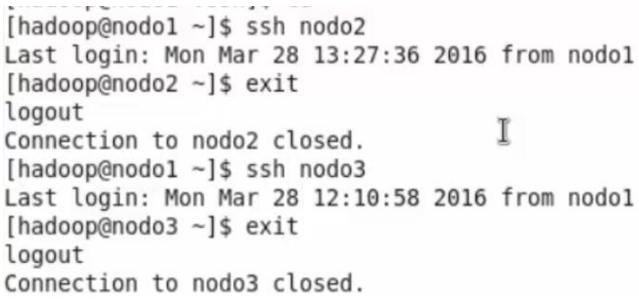


Ilustración 3.126.116

Hemos realizado una serie de pasos un tanto complejos que pueden llevar a confusión, por loque vamos a resumirlos un poco. Es importante destacar que, el hecho de haber clonado las máquinas, nos ahorra mucho de los problemas que puedan surgir y ya nos facilita el camino a la configuración correcta. Los requerimientos para que el clúster funcione al unísono, son los siguientes:

- El mismo usuario en todos los nodos
- Todos los nodos deben ser accesibles a través de SSH sin necesidad de contraseña.
- Debemos tener los mismos directorios de Hadoop en cada nodo
- Haber copiado el software de Hadoop en todos los nodos y en las mismas rutas
- Haber creado el directorio de datos en todos los nodos y haberle dado el permiso correspondiente (en nuestro caso /datos/datanode)

Si recordamos un poco la jerarquía de nuestro clúster, el namenode solo debería aparecer en el nodo maestro, el nodo1 en nuestro caso, por lo que procedemos a borrarlo de los nodos 2 y 3. Además, el datanode contiene los datos del nodo1 al ser un clon, por lo que debemos vaciarlo en los nodos esclavos también.

```
[hadoop@nodo1 etc]$ ssh nodo2
Last login: Mon Mar 28 13:29:47 2016 from nodo1
[hadoop@nodo2 ~]$ cd /datos
[hadoop@nodo2 datos]$ ls -ltr
total 8
drwxr-xr-x. 3 hadoop hadoop 4096 mar 28 00:59 namenode
drwx------. 3 hadoop hadoop 4096 mar 28 00:59 datanode
[hadoop@nodo2 datos]$ rm -rf namenode
[hadoop@nodo2 datos]$ cd datanode
[hadoop@nodo2 datanode]$ ls
current
[hadoop@nodo2 datanode]$ rm -rf current/
```

Ilustración 3.127.117

Esto no va a ser un problema ya que más adelante vamos a regenerar los procesos en los nodos 2 y 3. A continuación, volvemos al nodo1 para configurar el fichero core-site.xml de manera que sea compatible con los nuevos nodos. En principio no tenemos que cambiar su configuración pero sí es importante comprobar que se está reservando el nodo1 como nodo maestro:

```
<property>
        <name>fs.defaultFS</name>
        <value>hdfs://nodo1:9000</value>
</property>
```

Ilustración 3.128.118

Ahora, modificaremos el hdfs-site.xml de la siguiente manera:

- El factor de replicación ahora será 2 ya que tenemos 2 nodos esclavos:

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

Ilustración 3.129.119

Como esta configuración, sí que la tenemos que cambiar en el resto de nodos, vamos a ahorrarnos el modificar uno por uno copiando éste y sobreescribiendo los anteriores que pudieran tener dichos nodos:

```
[hadoop@nodo1 hadoop]$ scp hdfs-site.xml nodo2:/opt/hadoop/etc/hadoop/
hdfs-site.xml                                 100% 1042     1.0KB/s   00:00
[hadoop@nodo1 hadoop]$ scp hdfs-site.xml nodo3:/opt/hadoop/etc/hadoop/
hdfs-site.xml                                 100% 1042     1.0KB/s   00:00
```

Ilustración 3.130.120

El mapred-site.xml no sería necesario configurarlo ya que vamos a seguir utilizando Yarn como nuestra capa de administración. Por último, en el yarn-site.xml tampoco se hace necesaria la configuración, pero es importante destacar que esto se debe a que en nuestro caso el maestro de procesos también va a estar alojado en el nodo maestro, pero en aplicaciones de proyectos reales esto no suele ser así y suele ser un nodo dedicado y se diferencia entre maestro de datos y maestro de procesos, como ya vimos anteriormente.

```
<property>
        <name>yarn.resourcemanager.hostname</name>
        <value>nodo1</value>
</property>
```

Ilustración 3.131.121

Finalmente, le vamos a indicar a nuestro nodo maestro que sus esclavos son el nodo2 y el nodo3 modificanco el fichero slaves:

```
[hadoop@nodo1 hadoop]$ vi slaves
nodo2
nodo3
```

Ilustración 3.132.122

PUESTA A PUNTO Y ARRANQUE DEL CLÚSTER

Antes de arrancar nuestro clúster, tenemos que resolver algunos obstáculos previamente. El primero de todos, será el firewall, que para el propósito de este proyecto lo que haremos será deshabilitarlo completamente, en un proyecto en la vida real lo que haríamos sería habilitar los puertos específicos de nuestra máquina.

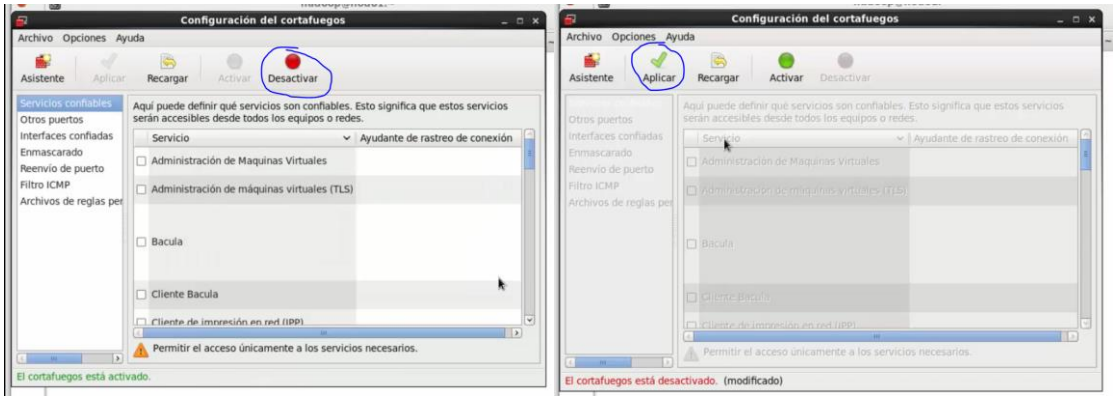


Ilustración 3.133.123

Anteriormente, borramos el namenode de todos los nodos a excepción del maestro, es por ello, que el siguiente paso será formatearlo, para que arranque de manera limpia con la configuración en el nodo maestro.

```
[hadoop@nodo1 ~]$ hdfs namenode -format
```

Ilustración 3.134.124

Y arrancamos el servicio de manera normal con el comando start-dfs.sh como siempre. Es muy importante estar atento al mensaje que devuelva ahora la consola ya que nos va a confirmar si finalmente nuestra configuración del clúster ha sido correcta o no. Esperamos ver que el namenode y el secondary namenode se inician en el nodo maestro y el datanode en los nodos esclavos:

```
[hadoop@nodo1 ~]$ start-dfs.sh
16/03/28 13:58:01 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using
 builtin-java classes where applicable
Starting namenodes on [nodo1]
nodo1: starting namenode, logging to /opt/hadoop/logs/hadoop-hadoop-namenode-nodo1.out
nodo3: starting datanode, logging to /opt/hadoop/logs/hadoop-hadoop-datanode-nodo3.out
nodo2: starting datanode, logging to /opt/hadoop/logs/hadoop-hadoop-datanode-nodo2.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /opt/hadoop/logs/hadoop-hadoop-secondarynamenode-nodo1.out
```

Ilustración 3.135.125

Y si comprobamos los procesos que hay corriendo dentro de cada nodo, observamos:

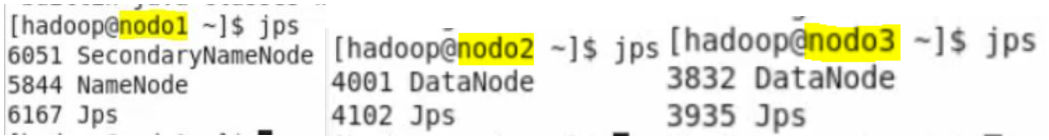


Ilustración 3.136.126

Y también podemos hacerlo a través de la página de administración de Hadoop:

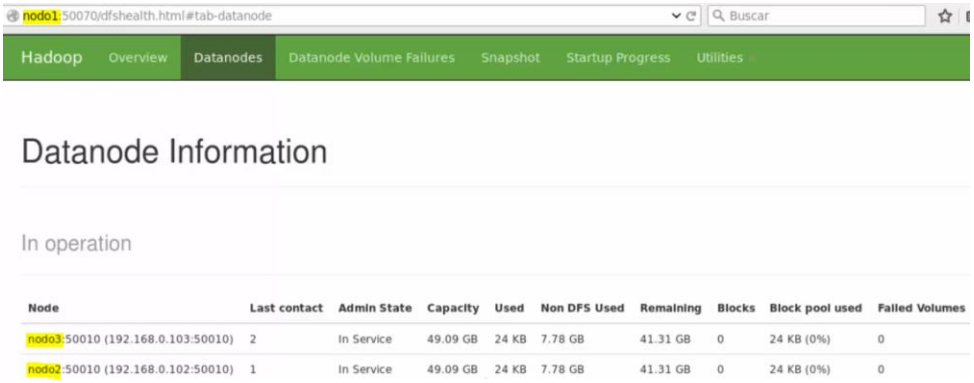


Ilustración 3.137.127

A continuación, arrancamos el Yarn y vemos que la configuración se ha aplicado también correctamente:

Ilustración 3.138.128

Y con JPS vemos que en los nodos 2 y 3 también ha aparecido el NodeManager

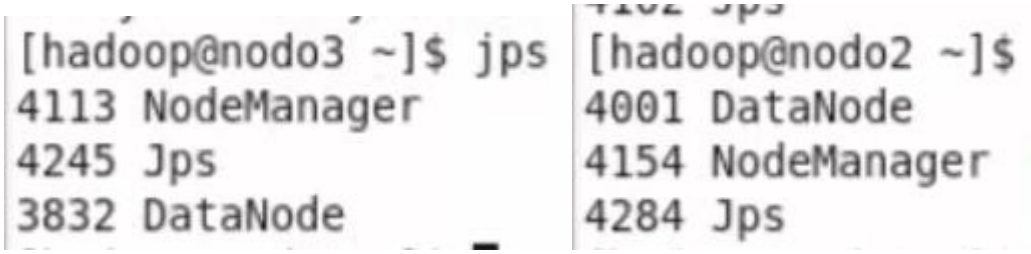


Ilustración 3.139.129

Y finalmente, desde la página de administración del Yarn, observamos que los nodos están listos para trabjar y son reconocidos por el sistema:

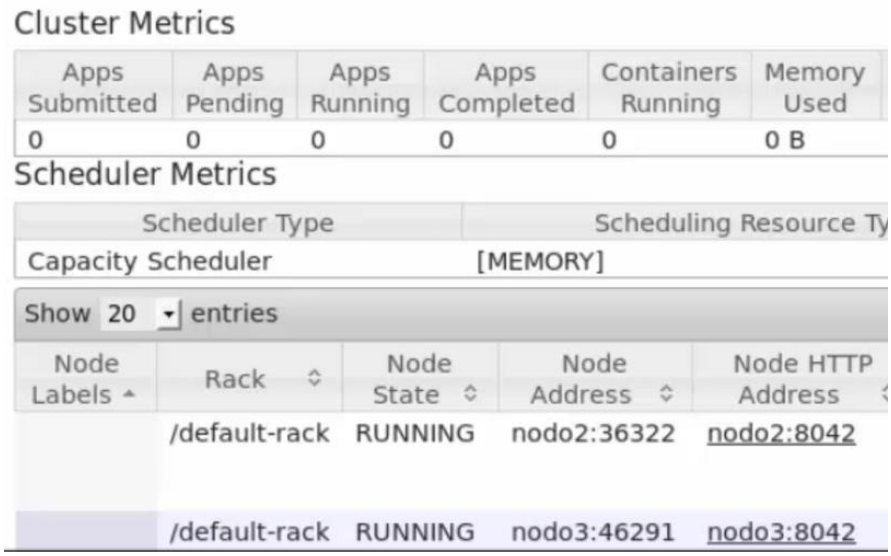


Ilustración 3.140.130

EJECUCIÓN DE UN PROCESO MAP REDUCE CONTRA UN CLÚSTER MULTI-NODO

Para recopilar la mayor información posible en este trabajo de investigación, ahora que vamos a lanzar un proceso contra una máquina Big Data, vamos a lanzar un proceso que nos permitirá usar la funcionalidad del historyserver para almacenar dicha información.

```
[hadoop@nodo1 Escritorio]$ mr-jobhistory-daemon.sh start historyserver
```

Ilustración 3.141.131

En nuestra ejecución, vamos a utilizar un fichero llamado access_log que pesa unos 500 MB y que proporciona la propia plataforma de Hadoop para este tipo de prubas. Dichero fichero lo almacenaremos en nuestro HDFS para la ejecución:

```
[hadoop@nodo1 Escritorio]$ hdfs dfs -mkdir /prueba
16/04/02 16:07:47 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
[hadoop@nodo1 Escritorio]$ hdfs dfs -put /tmp/access_log /prueba
```

Ilustración 3.142.132

Browse Directory

/prueba

Permission	Owner	Group	Size	Last Modified	Replication	Block Size
-rw-r--r--	hadoop	supergroup	481.55 MB	2/4/2016 16:08:28	2	128 MB

Ilustración 3.143.133

File information - access_log

Download

Block information -- Block 2

Block ID: 1073741851

Block Pool ID: BP-785552372-192.168.0.101-1459166264974

Generation Stamp: 1027

Size: 134217728

Availability:

- nodo3
- nodo2

Ilustración 3.144.134

A diferencia de la prueba que hicimos de mapReduce inicialmente, ahora se trata de un fichero más extenso y cuyo factor de replicación es 2, tal y como lo hemos configurado anteriormente, y que además está replicado tanto en el nodo3 como en el nodo2.

Para facilitar un poco la ejecución, hemos trasladado el script propio de Hadoop del conteo de palabras a un fichero en Java llamado ContarPalabras.jar. El código es una copia del mismo de Hadoop pero de ésta manera lo tenemos a mano y es más fácil de invocar.

```
[hadoop@nodo1 practicas]$ hadoop jar ContarPalabras.jar ContarPalabras /prueba/access_log /prue
ba_salida
```

Ilustración 3.145.135

Importante destacar el ResourceManager al que se conecta:

```
16/04/02 16:10:51 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:803
2
```

Ilustración 3.146.136

Y el Application_ID que le asigna el Yarn gracias al ApplicationManager:

```
16/04/02 16:10:53 INFO impl.YarnClientImpl: Submitted application application_1459549022831_000
1
```

Ilustración 3.147.137

Por lo que, y según lo que hemos visto y configurado anteriormente, ahora mismo la aplicación deberíamos poder verla desde el gestor de Yarn en ejecución:

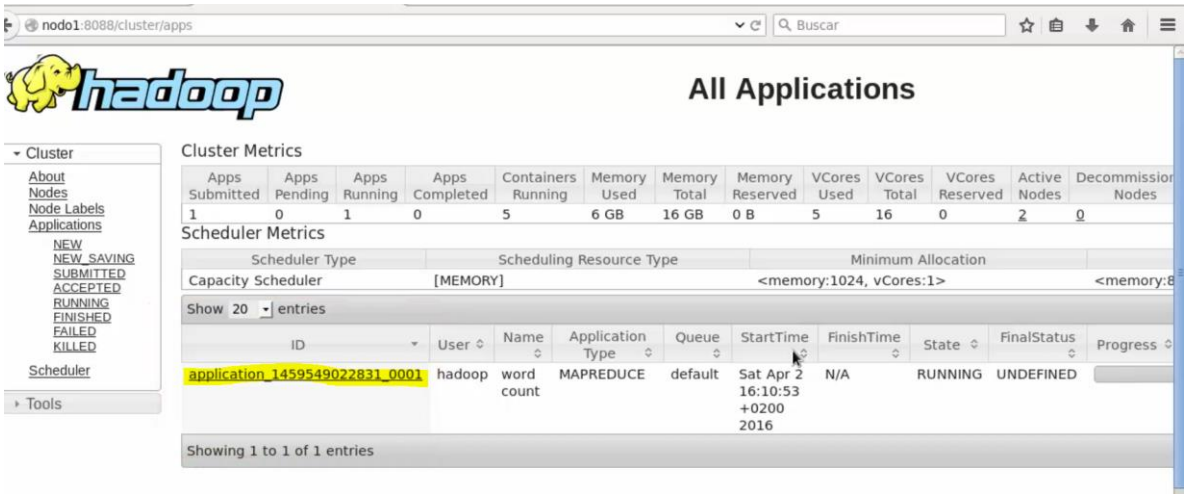


Ilustración 3.148.138

Podemos ver los procesos que se le asignan desde mi local:

```
[hadoop@nodo1 practicas]$ jps
8288 RunJar
5571 ResourceManager
8005 JobHistoryServer
4298 NameNode
4506 SecondaryNameNode
8348 Jps
```

Ilustración 3.149.139

Y dentro de ese portal web, podemos ver que:

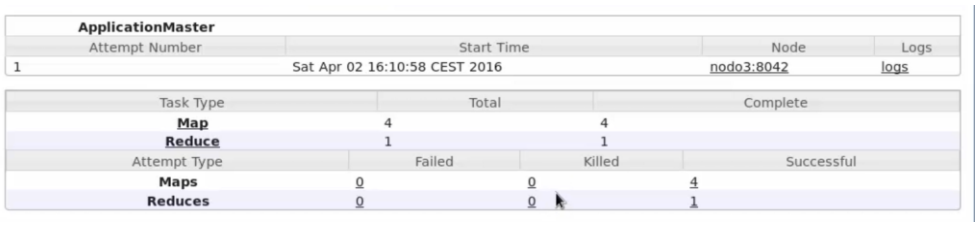
User:	hadoop
Name:	word count
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	RUNNING: AM has registered with RM and started running.
FinalStatus Reported by AM:	Application has not completed yet.
Started:	sáb abr 02 16:10:53 +0200 2016
Elapsed:	30sec
Tracking URL:	ApplicationMaster
Diagnostics:	

Ilustración 3.150.140

Por lo que puede correr con éxito y podemos monitorizar su ejecución desde Yarn. Una vez el proceso finalice veremos la siguiente pantalla gracias a que anteriormente hemos arrancado el jobhistory:

Ilustración 3.151.141

También, gracias a esa funcionalidad, podemos destacar algunos detalles muy relevantes de cara a generar una arquitectura Big Data equlibrada como por ejemplo:



ApplicationMaster

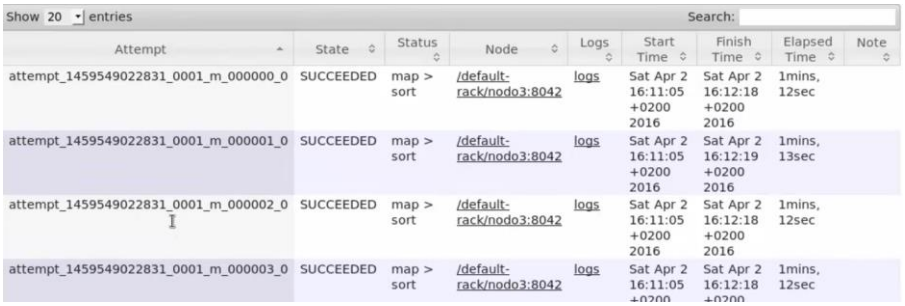
Attempt Number	Start Time	Node	Logs
1	Sat Apr 02 16:10:58 CEST 2016	nodo3:8042	logs

Task Type	Total	Complete
Map	4	4
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	4
Reduces	0	0	1

Ilustración 3.152.142

Vemos que el ApplicationMaster se ha generado en esta ocasión en el Nodo3, que se han realizado 4 operaciones de mapeo las cuales han terminado con éxito y una operación de reducción que también ha terminado correctamente. Además si hacemos click en el número 4 en la pestaña de Successfull podemos ver el detalle de las distintas etapas que ha cruzado la ejecución de la aplicación:



Show 20 entries Search:

Attempt	State	Status	Node	Logs	Start Time	Finish Time	Elapsed Time	Note
attempt_1459549022831_0001_m_000000_0	SUCCEEDED	map > sort	/default-rack/nodo3:8042	logs	Sat Apr 2 16:11:05 +0200 2016	Sat Apr 2 16:12:18 +0200 2016	1mins, 12sec	
attempt_1459549022831_0001_m_000001_0	SUCCEEDED	map > sort	/default-rack/nodo3:8042	logs	Sat Apr 2 16:11:05 +0200 2016	Sat Apr 2 16:12:19 +0200 2016	1mins, 13sec	
attempt_1459549022831_0001_m_000002_0	SUCCEEDED	map > sort	/default-rack/nodo3:8042	logs	Sat Apr 2 16:11:05 +0200 2016	Sat Apr 2 16:12:18 +0200 2016	1mins, 12sec	
attempt_1459549022831_0001_m_000003_0	SUCCEEDED	map > sort	/default-rack/nodo3:8042	logs	Sat Apr 2 16:11:05 +0200	Sat Apr 2 16:12:18 +0200	1mins, 12sec	

Ilustración 3.153.143

Si bien la arquitectura que estamos utilizando no es la más óptima, ya que normalmente esperaríamos tener un nodo maestro junto a 8-10 nodos esclavos, es más que funcional para este proyecto y representa perfectamente la esencia del clústering y la ejecución de procesos Big Data.

SPARK

Apache Spark es un entorno o framework de procesamiento distribuido y paralelo que trabaja en memoria, el cual permite el análisis y tratamiento de grandes conjuntos de datos. Además, integra otros entornos muy potentes con los cuales puede interactuar manteniendo ese paralelismo como son Bases de Datos NoSQL, Real Time (o streaming), Machine Learning, Análisis de Grafos, etc…

Spark aparece como una solución mucho más rápida que el MapReduce y mucho más accesible para usuarios o desarrolladores, a parte de ser totalmente compatible con Hadoop.

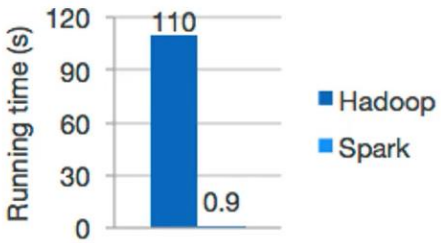


Ilustración 3.154.144

Al contrario que Hadoop Map Reduce que trabaja sobre todo con procesos de tipo Batch y en disco, Spark está orientado al trabajo in-memory y el procesamiento en tiempo real. Mientras MapReduce trabaja secuencialmente, Spark lo hace en paralelo:

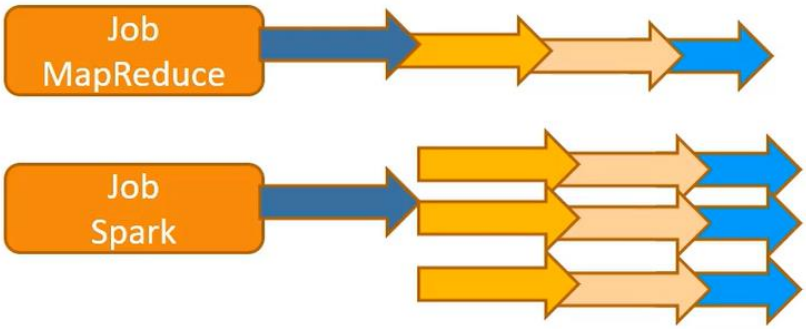


Ilustración 3.155.145

Las compatibilidades con Hadoop que podemos destacar de Spark son:

- Se puede ejecutar sobre HDFS
- MapReduce: Se puede usar en el mismo cluster que MapReduce y aprovechar su configuración para lanzarse de la manera más óptima.
- YARN: Una aplicación Spark se puede lanzar sobre YARN de manera que aprovechamos todo su potencial para la gestión del proceso.
- Se pueden mezclar aplicaciones Spark y MapReduce para Batch y Real Time.

Además soporta múltiples fuentes de datos como pueden ser Hive, Json, Cassandra, CSV, RDBMS…

Es importante destacar, que Spark está escrito en código Scala, pero podemos desarrollar software utilizando otros lenguajes como Java, Python y R, aunque siempre lo más óptimo y lo que mejor aprovechará el potencial de Spark será utilizando Scala.

Spark también poseé un Shell interactivo a diferencia de Hadoop el cual nos permite hacer ejecuciones o pruebas on-premise desde la misma consola, gracias a la arquitectura de Spark de estar compuesto por un Core principal y un gran conjunto de librerías que permiten su gran compatibilidad:

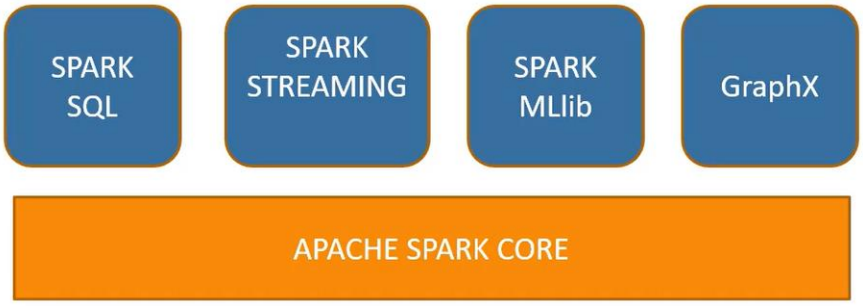


Ilustración 3.156.146

El Spark Core posee las siguientes características:

- Es el motor base para el procesamiento en escala y distribuido.
- Aunque está construido en Scala, hay APIs para Python, Java y R.
- Destacamos entr sus funciones:
  - Gestión de la memoria.
  - Recuperación ante fallos.
  - Planificación, distribución de trabajos en el clúster.
  - Monitorizar el trabajo.
  - Acceso a los sistemas de almacenamiento.

Dicho Core, se apoya en una estructura de datos muy especializada conocia como RDD (Resilient Distributed Datasets) que se caracteriza por:

- Permitir realizar procesos y ser tolerante a fallos
- Colección de registros inmutables y particionadas manejadas en paralelo.
- Pueden contener cualquier clase de objetos Python, Scala, Java o personalizados.
- Al ser inmutables, se crean habitualmente transformándolos de otros RDD o cargando los datos de una fuente externa, como por ejemplo HDFS o Hbase

Hoy en día, en Spark se utilizan los Dataframes, que son estructuras basadas en los RDDs, ya que están mucho más orientados al dato y no a su estructura.

ARQUITECTURA DE PROCESOS DE SPARK

Una aplicación de Spark no deja de ser un conjunto de procesos que se va a lanzar en un clúster mientras que son coordinados por el SparkContext el cuál se encuentra dentro del Driver. Este componente, se conecta al gestor del clúster (Cluster Manager, que bien puede ser un MesOS, Hadoop, …) el cuál le permite reservar un espacio dentro de los nodos trabajadores (Worker Node) conocido como Executor. El Executor es el proceso al cual se va a mandar la aplicación Java, Scala… que finalmente se acabará ejecutando. Dicho componente también levantará las tareas o tasks que se van a ejecutar en el nodo (al final, las tasks no son más que la partición de los jobs o trabajos en fragmentos más pequeños para poder ser ejecutados en paralelo) a través del uso de memoria o Cache. Finalmente, se da el acceso a los datos a través de los Datasets del propio Spark o el sistema de almacenamiento al que nos conectemos.

La siguiente imagen representa un poco este proceso que hemos explicado y es completamente agnóstica del sistema en el que despleguemos nuestro Spark:

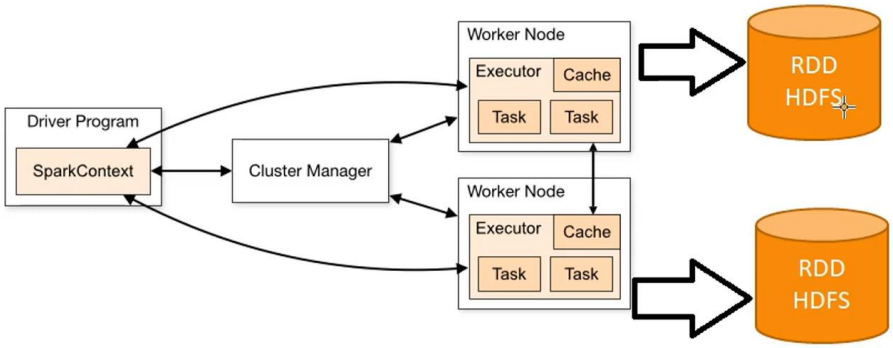


Ilustración 3.157.147

Sobre las distintas librerías que conforman el core de Spark, podemos destacar:

- Spark Streaming
  - Se usa para procesar fuentes de datos en tiempo real
  - Permite procesar con una alta tolerancia a fallos y un gran rendimiento las fuentes “vivas” de información que le suministremos.
  - Su unidad fundamental de trabajo es el Dstream (serie de RDDs)

  Tiene algunas aplicaciones en la actualidad muy destacables como puede ser el software de la red social Twitter o el procesamiento en tiempo real de datos financieros en Global Banking Santander.

- Spark SQL
  - Permite integrar comandos y componentes relacionales junto con la programación funcional de Spark
  - Podemos usar SQL o HQL (Hive Query Language)
  - Permite el acceso a múltiples fuentes de datos
  - Permite el acceso por JDBC o ODBC

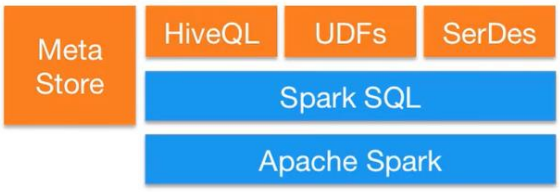


Ilustración 3.158.148

- GraphX (es el API para procesamiento de paralelo en grafos)
  - Spark GraphX implemente Resilient Distributed Graph (RDG – una abstracción de los RDD's)
  - RDG's asocia registros con los vertices y bordes de un grafo. Sin embargo, se pueden seguir viendo como colecciones tradicionales de RDD.
  - Se dispone de una gran cantidad de algoritmos preparados, que permiten agilizar el proceso de construcción de apliaciones y mejora el rendimiento y velocidad.

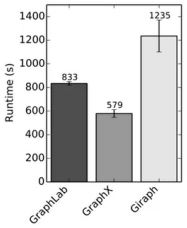


Ilustración 3.159.149

- Mlib (se utiliza para machine learning en Spark)
  - Se dispone de una variedad de algoritmos y otros procesos como "data cleaning"
  - Por ejemplo clasificación, clústering, regresión, extracción, …
  - Permite ejecución sobre HDFS, Hbase, …

DESCARGA E INSTALACIÓN DE SPARK

Es importante destacar que la gran versatilidad de Spark permite instalarlo y ejecutarlo bajo casi cualquier tipo de circunstancia, como pueden ser:

- Podemos descargar Spark desde la página Web
- Podemos hacer una instalación con Hadoop pre-empaquetado.
- Podemos descargar e instalar con Maven o con PyPI.
- Podemos ejecutarlo de forma local en una máquina virtual con el único requisito de tener Java.
- Podemos instalarlo en un entorno de clúster o Standalone.

Para el caso de nuestro clúster, como ya contamos con toda la instalación realizada, lo más sencillo es optar por el primer método y descargarlo directamente desde la página web. Para ello, elegimos la versión compatible con nuestra máquina:

Ilustración 3.160.150

Elegiremos un tipo de paquete diseñado para instalaciones en las que, ya contamos con un producto de Hadoop completo con todos los complementos necesarios como es el caso de nuestro clúster, de ésta manera solo nos tendremos que preoucpar del Spark y de nada más en nuestra instalación:

Ilustración 3.161.151

Y nos debería salir la siguiente opción como enlace de descarga:

Ilustración 3.162.152

Descargamos nuestro paquete y ya estamos listos para instalar Spark en nuestro sistema. Ahora dentro de nuestro clúster, nos situaremos en la carpeta de Hadoop y descomprimiremos el software que acabamos de descargar:

```
[hadoop@nodo1 ~]$ cd /opt/hadoop
[hadoop@nodo1 hadoop]$ tar xvf /home/hadoop/Descargas/spark-2.3.0-bin-without-hadoop.tgz
```

Ilustración 3.163.153

Cambiamos el nombre por simplificar y lo añadimos a nuestras variables de entorno:

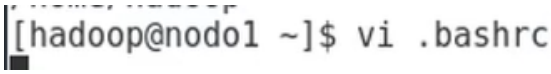


Ilustración 3.164.154

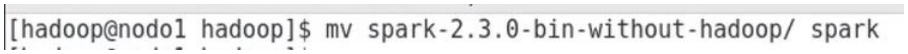


Ilustración 3.165.155

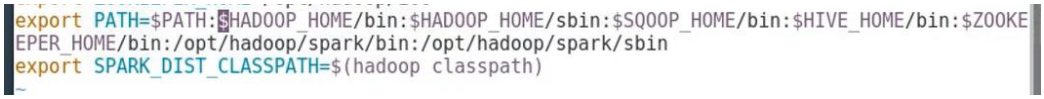


Ilustración 3.166.156

Podemos hacerlo de una manera tan atómica gracias al comando hadoop classpath que es un comando que le pasa a SPARK todas las referencias necesarias para su funcionamiento.

3.2 Pruebas finales

LANZANDO SPARK EN MODO CLIENTE

Vamos a lanzar una prueba con Spark en modo cliente para comprobar que funciona de manera individual, sin adherirlo a nuestro clúster todavía. Así que lo primero será inicializar la shell de Spark, la cual crea una sesión y un contexto en local que nos permite trabajar con ellas, además de que tenemos disponible la Web UI para ver más en detalle dicha ejecución:

```
[hadoop@nodo1 spark]$ spark-shell
2018-04-18 15:27:24 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your p
latform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://nodo1:4040
Spark context available as 'sc' (master = local[*], app id = local-1524058056841).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.0
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_151)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Ilustración 3.2.1

Dado que el entorno que se ha activado es el de Scala, deberemos usar dicho lenguaje para el desarrollo de nuestro ejemplo, pero también podemos echar un vistazo a los comandos que nos ofrece esta terminal:

```
scala> :help
All commands can be abbreviated, e.g., :he instead of :help.
:edit <id>|<line>        edit history
:help [command]          print this summary or command-specific help
:history [num]           show the history (optional num is commands to show)
:h? <string>             search the history
:imports [name name ...] show import history, identifying sources of names
:implicits [-v]          show the implicits in scope
:javap <path|class>      disassemble a file or class name
:line <id>|<line>        place line(s) at the end of history
:load <path>             interpret lines in a file
:paste [-raw] [path]     enter paste mode or paste a file
:power                   enable power user mode
:quit                    exit the interpreter
:replay [options]        reset the repl and replay all previous commands
:require <path>          add a jar to the classpath
:reset [options]         reset the repl to its initial state, forgetting all session entries
:save <path>             save replayable session to a file
:sh <command line>       run a shell command (result is implicitly => List[String])
:settings <options>      update compiler options, if possible; see reset
:silent                  disable/enable automatic printing of results
:type [-v] <expr>        display the type of an expression without evaluating it
:kind [-v] <expr>        display the kind of expression's type
:warnings                show the suppressed warnings from the most recent line which had any
```

Ilustración 3.2.2

Para nuestro ejemplo, lo que haremos será leer un fichero de texto llamado README.md que viene con la propia instalación de Scala y contaremos el número de líneas contenidas en éste para ver que Spark funciona correctamente. Para la carga del fichero, lo haremos especificando la ruta en local ya que por defecto intentará buscarlo en el sistema de almacenamiento de HDFS. El carácter "///" indica que se trata de nuestro local:

```
scala> val fichero=sc.textFile("file:///opt/hadoop/spark/README.md")
```

Ilustración 3.2.3

Y si lo imprimimos por pantalla:

```
scala> fichero
res1: org.apache.spark.rdd.RDD[String] = file:///opt/hadoop/spark/README.md MapPartitionsRDD[1]
 at textFile at <console>:24
```

Ilustración 3.2.4

Vemos que se ha almacenado en forma de RDD[String] que como explicábamos anteriormente son la estructura básica en la que se fundamenta Spark. Finalmente, dichos objetos tienen asociada una función llamada count() que nos permite contar el número de lineas que tiene:

```
scala> fichero.count()
res2: Long = 103
```

Ilustración 3.2.5

Y con esto vemos que no es necesario tener una gran infraestructura para correr Spark, que es muy fácil de instalar y completamente atómico en toda su funcionalidad.

Ahora, vamos a reproducir este ejemplo pero empleando Python en lugar de Scala. Para ello, lanzaremos con el comando pyspark el cual es similar al spark-shell pero nos coloca en la interfaz de Python en lugar de la de Scala.

```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.3.0
      /_/

Using Python version 2.7.5 (default, Aug  4 2017 00:39:18)
SparkSession available as 'spark'.
>>> fichero=spark.read.text("file:///opt/hadoop/spark/README.md")
>>> fichero
DataFrame[value: string]
>>> fichero.count()
103
```

Ilustración 3.2.6

LANZANDO SPARK EN HADOOP Y TRABAJANDO CON HDFS - STANDALONE

Lo primero, será configurar las variables de entorno para adaptar la compatibilidad necesaria.

```
export SPARK_DIST_CLASSPATH=$(hadoop classpath)
export SPARK_HOME=/opt/hadoop/spark
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
```

Ilustración 3.2.7

Para este desarrollo, vamos a crear un directorio llamado spark en nuestro HDFS donde almacenaremos nuestros ficheros de datos para las ejecuciones. Además añadiremos un fichero que nos proporciona la plataforma de Spark denominado puertos.csv que está orientado a tareas de testing de la funcionalidad de Spark.

```
[hadoop@nodo1 ~]$ hdfs dfs -mkdir /spark
[hadoop@nodo1 ~]$ hdfs dfs -put /tmp/puertos.csv /spark
```

Ilustración 3.2.8

Y en otra pestaña, tendremos preparada nuestra shell con Spark mientras en paralelo operamos con el sistema de almacenamiento HDFS. Vamos a intentar cargar el fichero a través de HDFS en una variable en nuestra shell, eso sí, comprobando previamente que se ha cargado correctamente:

```
[hadoop@nodo1 ~]$ hdfs dfs -ls /spark
Found 1 items
-rw-r--r--   3 hadoop supergroup     112636 2018-04-20 00:31 /spark/puertos.csv
[hadoop@nodo1 ~]$
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.0
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_151)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val v1=sc.textFile("/spark/puertos.csv")
v1: org.apache.spark.rdd.RDD[String] = /spark/puertos.csv MapPartitionsRDD[1] at textFi
le at <console>:24
```

Ilustración 3.2.9

```
scala> v1.count()
res0: Long = 1374
```

Ilustración 3.2.10

Parece que está todo correcto y que el fichero ha sido cargado correctamente. Vamos a complicar un poco más el ejemplo, y partiendo del RDD[String] que se ha generado a partir de cargar el fichero en nuestra variable, vamos a crear otro RDD (recordemos que los RDD son inmutables) filtrando el primero, que contendrá aquellos puertos en los que aparece la línea “Barcelona” con la siguiente operación:

```
scala> val v2=v1.filter(line=> line.contains("Barcelona"))
v2: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at <console>:25
```

Ilustración 3.2.11

```
scala> v2.count()
res1: Long = 48
```

Ilustración 3.2.12

Y con esto, hemos ejecutado Spark contra HDFS en modo cliente con una instalación mínima y que no requiere de muchos recursos físicos o de una máquina compleja.

LANZANDO SPARK CONTRA YARN

Ahora que ya hemos comprobado que Spark funciona y que es capaz de comunicarse con Hadoop sin problema, vamos a proceder a hacer una ejecución utilizando todo el potencial del clúster que hemos venido creando y configurando hasta ahora. Para hacer esto, utilizaremos un ejemplo propio de Spark que genera números decimales de PI, a través del spark-submit. El spark-submit se encarga de preparar la petición de un proceso de Spark para el clúster, y se le pasa como argumentos la clase a ejecutar, el master que tiene configurada la máquina, el modo de despliegue (antes era standalone y lo hacía la shell automáticamente por nosotros, ahora es en modo cluster), el nombre que le damos a la aplicación, la ruta donde puede encontrar el .jar a ejecutar y que contiene la clase que indicamos inicialmente, y finalmente, este ejemplo admite como argumento un número que será la cantidad de iteraciones para el cálculo de PI, para nuestro ejemplo hemos usado el 5.

```
[hadoop@nodo1 ~]$ spark-submit --class org.apache.spark.examples.SparkPi --master yarn --deploy-mode cluster --name "apli1" /
opt/hadoop/spark/examples/jars/spark-examples_2.11-2.3.0.jar  5
```

Ilustración 3.2.13

Vemos como se conecta exitosamente al clúster:

```
[hadoop@nodo1 ~]$ spark-submit --class org.apache.spark.examples.SparkPi --master yarn --deploy-mode cluster --name "apli1" /
opt/hadoop/spark/examples/jars/spark-examples_2.11-2.3.0.jar  5
2018-04-20 20:56:15 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java
classes where applicable
2018-04-20 20:56:18 INFO  RMProxy:133 - Connecting to ResourceManager at nodo1/192.168.56.101:8032
2018-04-20 20:56:18 INFO  Client:54 - Requesting a new application from cluster with 5 NodeManagers
```

Ilustración 3.2.14

Y como termina correctamente la ejecución de nuestro spark-submit:

```
2018-04-20 20:56:54 INFO  Client:54 - Application report for application_1523815005783_0017 (state: FINISHED)
2018-04-20 20:56:54 INFO  Client:54 -
         client token: N/A
         diagnostics: N/A
         ApplicationMaster host: 192.168.56.132
         ApplicationMaster RPC port: 0
         queue: default
         start time: 1524250592
         final status: SUCCEEDED
         tracking URL: http://nodo1:8088/proxy/application_1523815005783_0017/
         user: hadoop
2018-04-20 20:56:54 INFO  ShutdownHookManager:54 - Shutdown hook called
2018-04-20 20:56:54 INFO  ShutdownHookManager:54 - Deleting directory /tmp/spark-569c165c-0073-45f3-aee2-ec17c811f08d
2018-04-20 20:56:54 INFO  ShutdownHookManager:54 - Deleting directory /tmp/spark-958ab5d8-63b5-4fb5-befc-2c1d691e2e3b
[hadoop@nodo1 ~]$
```

Ilustración 3.2.15

Vamos a consultar nuestro Yarn para ver la traza completa de la ejecución que acabamos de realizar:

```
application_1523815005783_0017  hadoop  apli1  SPARK  default  0  Fri Apr 20 20:56:32 +0200  Fri Apr 20 20:56:54 +0200  FINISHED  SUCCEEDED  N/A  N/A  N/A
```

Ilustración 3.2.16

Podemos comprobar que, cuando lanzamos la aplicación Spark dentro de nuestro clúster Hadoop, se ejecuta dentro de éste y se aprovecha de las características del entorno. Dando lugar a una infraestructura que posee el sistema de almacenamiento y el software suficientes para otorgar soluciones a proyectos con gran cantidad de datos o de proyectos Big Data.

Ya que la información ofrecida por pantalla (en la shell) ha sido más bien escueta, también podemos aprovechar el Yarn para consultar el log completo de la aplicación y ver qué ha ocurrido realmente con nuestra ejecución.

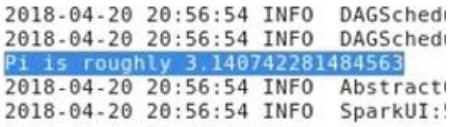


Ilustración 3.2.17

4 EXPERIMENTACIÓN, RESULTADOS Y DISCUSIÓN

Benchmarking – Python / Spark

- SPARK

Módulo de traducción (1 m 4 s)

```
translated_df = combined_df

def translate_text( t ):
    return GoogleTranslator(source='auto', target='es').translate(t)

translate_udf = udf(translate_text, StringType())

for col_name in translated_df.columns:
    translated_df = translated_df.withColumn(col_name, translate_udf(translated_df[col_name]))

translated_df.show(1000,truncate=False)
✓ 1m 4.0s
```

Ilustración 4.1

Aplicación del algoritmo con lexicón (8 s)

```
# Iterar sobre cada columna y aplicar la UDF
for col_name in insultos_df.columns:
    insultos_df = insultos_df.withColumn(col_name, detect_insult_in_tuple_udf(col(col_name)))

# Mostrar el DataFrame resultante
insultos_df.show(1000,truncate=False)
✓ 8.1s
```

Ilustración 4.2

Aplicación del algoritmo de entrenamiento supervisado (8 s)

```
# Iterar sobre cada columna y aplicar la UDF de predicción
for col_name in supervisado_df.columns:
    supervisado_df = supervisado_df.withColumn(col_name, predict_comment_udf(col(col_name)))

# Mostrar el DataFrame resultante
supervisado_df.show(1000,truncate=False)
✓ 8.3s
```

Ilustración 4.3

- Python

Módulo de traducción **(2 m 55 s)**

```
# Traducimos el diccionario
dict_coment = {}
for i in list_of_csv:
    if( len(i) >= 2):
        key_dict = i[0]
        coments = i[1].split('|||')
        value = []
        for j in coments:
            translated = GoogleTranslator(source='auto', target='es').translate(j)
            value.append(translated)
        dict_coment[key_dict] = value

✓ 2m 54.6s
```

Ilustración 4.4

Aplicación del algoritmo con lexicón **(4 s)**

```
# LEXICON

# --- Cargamos insultos
ArrayInsultos = []
for i in insultos:
  ArrayInsultos.append(i)

# --- Aplicamos insultos
final_dict_lexicon = Lexicon(final_dict,ArrayInsultos)

✓ 4.2s
```

Ilustración 4.5

Aplicación del algoritmo de entrenamiento supervisado **(16 s)**

4.1 Resultados y discusión

Vemos claramente como la ejecución en Spark es ampliamente superior a la realizada en Python. Es importante destacar también que la complejidad del código aumenta y no es al que estamos acostumbrados como desarrolladores, ya que Spark al estar basado en Scala produce un código basado en el concepto de programación funcional.

5 CONCLUSIONES Y TRABAJOS FUTUROS

El proyecto ha demostrado la viabilidad de utilizar un clúster de Big Data basado en Hadoop y Spark para el procesamiento de grandes volúmenes de datos textuales, extraídos mediante scrapping de Twitter, y su análisis utilizando técnicas de procesamiento de lenguaje natural (NLP). Se han cumplido los objetivos iniciales de crear un clúster distribuido, desarrollar un sistema de scrapping efectivo, e implementar un análisis de comentarios ofensivos tanto en Python como en Spark.

La vía a futuro y posibles mejoras del proyecto es clara, pasar de un clúster distribuido almacenado en una máquina y montado por nosotros a un servicio Cloud que nos despreocupe completamente del montaje y mantenimiento del sistema y mediante un sistema de facturación o pago emplear solo aquellos servicios necesarios para correr nuestra aplicación y producir nuestros resultados, también utilizando Spark que sigue siendo un software que se actualiza día tras día y que tiene una gran comunidad de desarrolladores detrás.

Las mejoras inmediatas pueden ser:

- Sistema de almacenamiento. En lugar de emplear HDFS, podríamos empezar a utilizar S3 que es el sistema de almacenamiento propuesto por Amazon. No es un software libre y tampoco es gratuito pero es cierto que ofrece mejores resultados en cuanto a optimización del espacio y velocidad en las búsquedas e inserciones.

- Servicio en cloud para ejecución de aplicaciones. Podríamos alojar todas nuestras aplicaciones y servicios en un clúster que ya cuenta con todo el software que necesitamos y que se puede seguir actualizando conforme a nuestras necesidades periódicamente. Es cierto que esto tiene un incremento del coste del proyecto asociado, pero podemos tener la configuración de hardware y software que deseemos en cualquier motivo.

- Servicio en cloud para cuadernos y control del código. No solo nos permitiría una gran integración con el servicio cloud en el que estemos corriendo nuestro software, sino que además nos proporcionaría un sistema de cuadernos donde tener organizado y almacenado los distintos procesos a ejecutar. Además permitiría pruebas y test on-premise en el mismo clúster.

6 APÉNDICES

6.1 Guía original del Trabajo Fin de Título

Ésta es la guía original del trabajo de fin de grado, y como se puede observar, no nos hemos desviado apenas del objetivo principal:

- Descripción
    - Este proyecto consiste en el montaje de una máquina tipo clúster de Big Data que permita el procesamiento y tratamiento de volúmenes de datos en tiempo real utilizando un software de NLP para la detección de “ofensividad” en texto. El proyecto se divide en dos núcleos principales, primero, esa máquina montada con máquinas virtuales con todo el software y configuración necesario para el tratamiento de esos datos en tiempo real y que permita el acceso, tratamiento y almacenamiento de esa información. Por otro lado, el segundo, ese software que permitirá tratar esos datos utilizando NLP de manera que nos permitirá detectar si existe o no ofensividad en esos grupos de textos que estaremos extrayendo y tratando en tiempo real a través de nuestra máquina. Dicho proyecto puede ser utilizado como enseñanza del funcionamiento interno de un clúster de Big Data, ya que se recogerá con todo detalle paso por paso la configuración completa del clúster, e incluso del framework Spark como herramienta Big Data a través de los lenguajes de programación Python y Scala. También es destacable el algoritmo NLP utilizado ya que en el proyecto entraremos en detalle sobre su desarrollo, planificación e implementación.

- Conocimientos y requisitos previos
    - Es necesario tener conocimiento y experiencias con máquinas y entornos virtuales, el framework Spark, el lenguaje de programación Python, APIs en general y el software Kafka.

- Objetivos del trabajo
    - Diseñar y montar un clúster y un software con capacidad real para el tratamiento de datos en streaming gracias a la capacidad de Spark.

      Ofrecer una guía completa de montaje, indicaciones y funcionamiento tanto del software como de la máquina en sí para que sea fácilmente reproducible y además se pueda adaptar a cualquier otro posible uso.
    - Explicar los distintos componentes que forman el trabajo de manera que se vuelvan más accesibles y "fáciles" de trabajar por personas sin tanto conocimiento técnico en el área.

- Metodología de trabajo
    - Diseñar una máquina con la configuración adecuada para poder lanzar scripts de Python en ella, junto al software de Spark, Kafka, Hadoop ... y todo lo necesario para actuar como clúster Big Data.
    - Diseñar y desarrollar un software basado en NLP utilizando el lenguaje Python para el tratamiento de la información que será ingestada en streaming.
    - Detallar la estimación temporal y costes de desarrollo.
    - Realizar pruebas para la máquina y documentar los resultados.
    - Redacción de la documentación final requerida.

- Documentos y formato de entrega
    - Documentación generada durante el proceso, de acuerdo a las buenas prácticas de la ingeniería del software. Esto se aplicará tanto en los proyectos de Ingeniería como en los trabajos teóricos o experimentales, es decir, siempre que se genere algún tipo de desarrollo, tanto software como webs, scripts, etc. Para el formato de documentación de los Proyectos de Ingeniería se utilizará la norma UNE 157801. Tal y como especifican las normas de estilo de la EPSJ: 'Los Proyectos de Ingeniería deberán presentar una estructura y contenido adaptados a la norma UNE 157001 – Criterios generales para la elaboración de proyectos - u otra derivada de ésta, en función de la naturaleza del proyecto'. '...para el caso de los sistemas de información informatizados (sistemas informáticos, sistemas de información geográfica, etc.) está la norma derivada de la anterior UNE 157801 - Criterios generales para la elaboración de proyectos de sistemas de información'. Para ello, se puede utilizar también la Norma CCII-N2016-02 (Norma Técnica para la realización de la Documentación de Proyectos en Ingeniería Informática) del Consejo General de Colegios Profesionales de Ingeniería en Informática, que incluye e implementa la norma UNE 157801. Memoria del trabajo, incluyendo manuales y anexos, en formato PDF.
    - Código fuente completo y ejecutables del prototipo o software desarrollado.
    - Documentos y archivos adicionales a los que se haga mención expresa en la memoria.
    - Vídeo de demostración de la aplicación o de la experimentación realizada.
    - Anexos para la presentación del trabajo:
        - Informe favorable del tutor.
        - Autorización de publicación, si procede.

- Estudio técnico
- Para el estudio técnico, diseñaremos y montaremos una serie de máquinas virtuales utilizando el sistema operativo CentOS (en la versión más estable o compatible en el momento) utilizando un software de virtualización, o bien VirtualBox o VMWare. Dichas máquinas, estarán conectadas entre sí a nivel local, de manera que puedan comunicarse entre ellas de manera fácil y directa. Dicha conexión, nos facilitará que, tras el montaje del software adecuado, las máquinas puedan actuar de manera conjunta como un clúster de Big Data. Entre el software necesario para trabajar de este modo contamos con: Spark, Hadoop, HDFS y Kafka.

Una vez estén las máquinas creadas, levantadas y funcionando correctamente, podremos lanzar nuestro software que recogerá los datos en streaming y a través de Spark se repartirá la carga de trabajo entre los distintos nodos (o máquinas virtuales) que luego será tratada por nuestro software de NLP en Python.
De esta manera, tendremos un clúster Big Data completamente funcional utilizando pocos recursos y de manera clara y sencilla.

7 DEFINICIONES Y ABREVIATURAS

- **PLN** = Procesamiento del Lenguaje Natural.
- **NLP** = Natural Language Processing.
- **Scrapping** = El scraping o rascado de datos, de un modo general, se refiere a una técnica en la cual un programa informático extrae datos del resultado generado por otro programa.
- **Clúster** = Conjunto de computadores que trabajan juntos de manera coordinada.
- **Framework** = Entorno o marco de trabajo que actúa a modo de herramienta o estándar para trabajar con determinados programas.
- **Hardware** = Equipo, aparato, pieza de un ordenador.
- **Software** = Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.
- **Cloud** = La computación en la nube es una tecnología que permite acceso remoto a softwares, almacenamiento de archivos y procesamiento de datos por medio de Internet.
- **On Premise** =Una aplicación o software es on-premise es el que se instala en tus servidores u ordenador, es decir, en tus equipos informáticos, sean los que sean.
- **Feedback** = Retroalimentación.
- **Standalone** = Autónomo. Un programa que puede trabajar de manera autónoma, es decir, que se puede instalar y ejecutar, o simplemente ejecutar, en un sistema sin necesidad de nada más.
- **Dataframe** = Estructura de datos con dos dimensiones en la cual se puede guardar datos de distintos tipos (como caractéres, enteros, valores de punto flotante, factores y más) en columnas. Es la estructura característica de Spark.
- **Sprint** = Un sprint es un período breve de tiempo fijo en el que un equipo de scrum trabaja para completar una cantidad de trabajo establecida.

- **HDFS** = Hadoop Distributed File System = Sistema de archivos distribuidos Hadoop.
- **NLTK** = Natural Language Toolkit
- **Tokenización** = La tokenización es una forma que tienen las empresas de proteger los datos sensibles
- **Nodo** = En informática y en telecomunicación, de forma muy general, un nodo es un punto de intersección, conexión o unión de varios elementos que confluyen en el mismo lugar. En nuestro contexto es una máquina que hace de punto de intersección para los datos y cruces de las aplicaciones.
- **JSON** = JavaScript Object Notation = Es una extensión para almacenar información con cierto formato.
- **CSV** = Comma Separated Values = Valores separados por comas, es un formato utilizado por programas como Excel y ampliamente conocido en el mundo del dato para el almacenamiento organizado de la información.
- **Clústering** = Técnica mediante la cual se integran dos o más computadoras para que trabajen simultáneamente en el procesamiento de una determinada tarea.
- **Translator** = Traductor.
- **SVM** = Máquina de soporte de vector, es un algoritmo supervisado usado en el ámbito del Machine Learning.
- **Influencer** = Se trata de una persona con capacidad para influir sobre otras, principalmente a través de las redes sociales.
- **UDF** = User Defined Function. Función definida por el usuario, que no pertenece al core de Spark pero que el usuario puede definir para que funcione con éste.
- **Lexicon** = Conocimiento léxico sobre una lengua.
- **xPath** = Es un lenguaje que permite construir expresiones que recorren y procesan un documento XML, en este caso para recorrer una página web y encontrar los elementos que nos interesan.

- **HTML** = Lenguaje de Marcas de Hipertexto, es el lenguaje en el que se fundamentan las páginas web.
- **String** = Cadena formada por caracteres básicos del lenguaje.
- **Header** = Encabezado. Parte frontal y primera de un documento con datos ordenados.
- **Bot** = Es una aplicación de software automatizada que realiza tareas repetitivas en una red.
- **Login** = Ingresar o Entrar. Es el proceso que controla el acceso individual a un sistema.
- **GB** = GigaBytes
- **RM** = ResourceManager
- **NM** = NodeManager
- **AM** = ApplicationMaster
- **IP** = Es una dirección única que identifica a un dispositivo en Internet o en una red local.
- **DNS** = Sistema de nombres de dominio. Es el directorio telefónico de Internet. Las personas acceden a la información en línea a través de nombres de dominio que son traducidos por este servicio.
- **Network** = Red.
- **SSH** = Secure Shell. Es un protocolo de red destinado principalmente a la conexión entre máquinas de manera segura que se “conocen” entre ellas.
- **Batch** = Lote. Normalmente se habla del procesamiento por batches, que es cuando procesamos una cantidad de datos finita, que tiene comienzo y fin.
- **RDD** = Resilient Distributed Dataset. Conjunto de elementos tolerantes a fallos que se pueden distribuir entre varios nodos en un clúster y trabajar en paralelo. Los Dataframes de Spark se basan en dichas estructuras.

- **SQL** = Structured Query Language. El lenguaje de consulta estructurada es un lenguaje de programación para almacenar y procesar información en una base de datos relacional
- **HQL** = Hive Query Language. Igual que la definición anterior pero orientado a Hive y Hadoop.
- **RDG** = Resilient Distributed Graph
- **API** = Application Program Interface. Las interfaces de programación de aplicaciones son mecanismos que permiten a dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos.
- **Streaming** = En vivo. Contrario al procesamiento de datos en batch donde los datos son finitos, en el caso del streaming, partimos de la suposición de que los datos son infinitos y no van a dejar de llegarnos.

8 BIBLIOGRAFÍA

Pressman, R. (2010). *Ingeniería del Software.* McGraw-Hill.

Jules S. Damji, Brooke Wenig, Tathagata Das, Denny Lee (2020). Learning Spark, 2nd Edition. O'Reilly Media, Inc.

Bill Chambers & Matei Zaharia (2018). Spark: The definitive Guide. O'Reilly Media, Inc.

Danil, Zburivsky (2013). Hadoop Cluster Deployment. PACKT

Mayank Bhushan (2023). Big Data and Hadoop: Fundamentals, tools, and techniques for data-driven success - 2nd Edition. BPB Publications.

Viktor Mayer-Schnberger and Kenneth Cukier (2013). Big Data: A Revolution That Will Transform How We Live, Work and Think. HACHETTE

Lewis Tunstall, Leandro von Werra & Thomas Wolf (2022). Natural Language Processing with Transformers: Building Language Applications With Hugging Face. O'Reilly Media, Inc.

Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta & Harshit Surana. Practical Natural Language Processing: A Comprehensive Guide to Building Real-World Nlp Systems

Apache Spark - Spark (18:00 – 10/09/2024)
Apache Hadoop - Hadoop (18:00 – 10/09/2024)
Cloudera - Cloudera (18:00 – 10/09/2024)
The CentOS Project - CentOS (18:00 – 10/09/2024)
DataCamp - NLP (18:00 – 10/09/2024)