

Universidad Politécnica de Madrid  
Escuela Técnica Superior de Ingenieros de Telecomunicación



**CREACIÓN DE UN ENTORNO PARA EL ANÁLISIS  
DE DATOS GEOGRÁFICOS UTILIZANDO TÉCNICAS  
DE BIG DATA, GEOMESA Y APACHE SPARK**

**TRABAJO FIN DE MÁSTER**

**Alejandro Rodríguez Calzado**

2017



Universidad Politécnica de Madrid  
Escuela Técnica Superior de Ingenieros de Telecomunicación

**Máster Universitario en  
Ingeniería de Redes y Servicios Telemáticos**

**TRABAJO FIN DE MÁSTER**

**CREACIÓN DE UN ENTORNO PARA EL ANÁLISIS  
DE DATOS GEOGRÁFICOS UTILIZANDO TÉCNICAS  
DE BIG DATA, GEOMESA Y APACHE SPARK**

Autor

**Alejandro Rodríguez Calzado**

Director

**Borja Bordel Sánchez**

Ponente

**Diego Martín de Andrés**

Departamento de Ingeniería de Sistemas Telemáticos

2017



## Resumen

Las nuevas tecnologías como el IoT y los smartphones han impulsado las cantidades de datos que generamos y almacenamos. Esto, junto al desarrollo de las técnicas y tecnologías de geolocalización, ha supuesto un aumento significativo de los datos asociados a posiciones geográficas.

La utilidad de los datos geográficos es enorme y muy diversa, pero como todos los datos, precisan de un análisis para ser convertidos en información útil. Este análisis se ha llevado a cabo durante mucho tiempo con Sistemas de Información Geográfica (GIS) tradicionales que, debido al gran aumento de la cantidad de datos geográficos, han tenido que evolucionar para adaptarse a las técnicas y tecnologías de Big Data.

El objetivo de este Trabajo Fin de Máster es el diseño y la implementación de un entorno para el procesamiento de datos, donde sea posible analizar flujos de información geográfica. Para ello se hará uso de técnicas de Big Data y se emplearán las soluciones tecnológicas Apache Spark y GeoMesa. Se desplegará una plataforma de procesamiento donde se combinen algunas de las tecnologías más altamente demandadas en la actualidad como GeoMesa, Apache Spark o Apache Accumulo.



## **Abstract**

New technologies such as IoT and smartphones have boosted the amounts of data we generate and store. This and the development of geolocation techniques and technologies, has meant a significant increase in the data associated with geographical positions.

The geographic data is very useful, but like all the data, they need an analysis to be turned into useful information. This analysis has been carried out for a long time by traditional Geographic Information Systems (GIS) which, due to the large increase in the amount of geographic data, have had to evolve to adapt to the techniques and technologies of Big Data.

The objective of this Final Master Project is the design and implementation of an environment for data processing, where it is possible to analyze flows of geographic information. This will be done using Big Data techniques and technology solutions like Apache Spark and GeoMesa. A processing platform will be deployed to combine some of today's most highly demanded technologies such as GeoMesa, Apache Spark or Apache Accumulo.





## Índice general

Resumen .....	v
Abstract.....	vii
Índice general.....	ix
Índice de figuras.....	xiii
Siglas .....	xvii
1 Introducción.....	1
1.1 Motivación.....	1
1.2 Objetivos .....	2
1.3 Estructura del documento.....	2
2 Servicios y tecnologías utilizadas.....	5
2.1 Amazon Web Services .....	5
2.1.1 Amazon EC2 .....	6
2.2 Apache Maven.....	7
2.3 Apache Hadoop.....	9
2.3.1 HDFS.....	10
2.4 Apache Zookeeper .....	12
2.5 Apache Accumulo .....	13
2.5.1 Modelo de datos .....	14
2.5.2 Arquitectura.....	15
2.5.3 Gestión de datos .....	17
2.6 Apache Spark.....	17
2.6.1 Stack unificado.....	18
2.7 GeoMesa .....	22
2.7.1 Arquitectura.....	23
2.8 Jupyter Notebook .....	27

2.8.1	Notebook Web Application .....	28
2.8.2	Kernel .....	28
2.8.3	Notebook .....	28
2.9	Apache Toree .....	29
2.10	Docker .....	30
2.11	Ansible .....	32
2.11.1	Ansible Playbooks .....	32
3	Arquitectura del entorno.....	35
4	Implementación del entorno.....	39
4.1	Adquisición de recursos de computación en AWS .....	39
4.1.1	Creación y configuración de una instancia EC2 .....	40
4.1.2	Conexión con la instancia.....	46
4.2	Instalación del servicio No-IP.....	52
4.2.1	Registrar un hostname.....	53
4.2.2	Instalación del Dynamic Update Client (DUC).....	53
4.2.3	Ejecución automática del DUC.....	55
4.3	Instalaciones previas .....	56
4.3.1	JDK 8 .....	56
4.3.2	Git .....	56
4.3.3	Apache Maven.....	56
4.4	Instalación Apache Hadoop.....	57
4.4.1	Configurar conexión SSH sin contraseña.....	58
4.4.2	Descarga e instalación.....	59
4.4.3	Configuración .....	59
4.4.4	Ejecución de HDFS.....	62
4.5	Instalación Apache Zookeeper .....	63
4.5.1	Descarga e instalación.....	63
4.5.2	Configuración .....	64
4.5.3	Ejecución.....	65
4.6	Instalación Apache Accumulo.....	66
4.6.1	Descarga e instalación.....	66

4.6.2	Configuración .....	67
4.6.3	Ejecución.....	70
4.7	Instalación Apache Spark.....	72
4.7.1	Descarga e instalación.....	73
4.7.2	Configuración .....	74
4.7.3	Ejecución.....	75
4.8	Instalación de GeoMesa.....	77
4.8.1	Instalación con binarios .....	77
4.8.2	Instalación mediante compilación del código fuente.....	78
4.8.3	Instalación de la biblioteca “Accumulo Distributed Runtime” .....	78
4.8.4	Instalación de “Accumulo Command Line Tools” .....	79
4.8.5	Actualización versión de Geomesa.....	80
4.8.6	Ejecución.....	80
4.9	Instalación Docker CE.....	81
4.10	Instalación Jupyter Notebook y Apache Toree .....	82
4.10.1	Requisitos previos .....	82
4.10.2	Instalación Jupyter Notebook.....	83
4.10.3	Configuración Jupyter Notebook.....	84
4.10.4	Instalación Toree.....	86
4.10.5	Enlazar Jupyter y Apache Toree .....	86
4.10.6	Ejecución.....	88
4.11	Script para el arranque de servicios.....	91
5	Ingestión (ingest) de datos con GeoMesa-Accumulo.....	93
5.1	Ingestión de datos usando técnicas MapReduce .....	93
5.1.1	Estructura del código MapReduce.....	95
5.2	Ingestión de datos usando GeoMesa Command Line Tools.....	98
5.2.1	Ingestión de Shapefile.....	99
6	Ejecución de aplicaciones GeoMesa-Spark en el entorno.....	101
6.1	Ejecución mediante spark-submit.....	101
6.2	Ejecución en Jupyter-Toree .....	102
6.3	Ejemplos.....	103

6.3.1	CountByDay.....	103
6.3.2	ShallowJoin.....	106
7	Automatización de workers de Spark.....	115
7.1	Docker.....	115
7.1.1	Dockerfile.....	115
7.1.2	Docker Image.....	117
7.1.3	Docker Hub: subir y bajar imágenes.....	117
7.1.4	Docker Hub: Automate Build.....	117
7.1.5	Ejecución de un contenedor.....	118
7.2	Ansible.....	119
7.2.1	Instalación de Ansible.....	119
7.2.2	Conexión con los nodos.....	120
7.2.3	Playbook.....	121
8	Conclusiones.....	125
8.1	Líneas de desarrollo.....	126
	Bibliografía.....	127

## Índice de figuras

Figura 1. Logo Amazon Web Services .....	5
Figura 2. Logo Apache Maven .....	7
Figura 3. Logo Apache Hadoop .....	9
Figura 4. Arquitectura de Hadoop .....	10
Figura 5. Logo Hadoop HDFS.....	10
Figura 6. Arquitectura HDFS.....	11
Figura 7. Logo Apache Zookeeper.....	12
Figura 8. Arquitectura servicio Zookeeper.....	13
Figura 9. Logo Apache Accumulo .....	13
Figura 10. Modelo de datos de Accumulo.....	14
Figura 11. Arquitectura Apache Accumulo.....	15
Figura 12. Distribución de datos en Accumulo.....	17
Figura 13. Logo Apache Spark .....	18
Figura 14. Stack de Apache Spark.....	20
Figura 15. Componentes de una aplicación Spark distribuida.....	22
Figura 16. Logo GeoMesa.....	22
Figura 17. Arquitectura Geomesa .....	24
Figura 18. Posible arquitectura de ingestión de datos con GeoMesa .....	24
Figura 19. Posible arquitectura para peticiones de datos con GeoMesa .....	25
Figura 20. Representación de una Z-curve de GeoMesa .....	26
Figura 21. Estructura de un registro almacenado por GeoMesa en Accumulo.....	27
Figura 22. Logo Jupyter .....	27
Figura 23. Logo Apache Toree.....	29
Figura 24. Interacción aplicación cliente - Apache Toree -Apache Spark .....	30
Figura 25. Logo Docker .....	30
Figura 26. Arquitectura de virtualización basada en hypervisor y basada en contenedores .....	31
Figura 27. Logo Ansible .....	32
Figura 28. Esquema general de arquitectura del entorno.....	35
Figura 29. Diagrama automatización despliegue de workers de Spark.....	37
Figura 30. Precios y características instancias Amazon EC2.....	39
Figura 31. Listado de servicios de AWS.....	40
Figura 32. Consola de Amazon EC2.....	41
Figura 33. Listado de instancias EC2 creadas en nuestra cuenta .....	41

Figura 34. Listado de AMIs disponibles - EC2.....	42
Figura 35. Tipos de instancias EC2.....	42
Figura 36. Configuración instancia EC2.....	43
Figura 37. Configuración almacenamiento instancia EC2.....	44
Figura 38. Añadir etiquetas instancia EC2.....	44
Figura 39. Creación de un grupo de seguridad EC2.....	45
Figura 40. Resumen configuración instancia EC2.....	45
Figura 41. Key pair instancia EC2.....	46
Figura 42. Interfaz de la herramienta PuTTYgen.....	47
Figura 43. Interfaz gráfica del cliente PuTTY.....	48
Figura 44. Interfaz cliente PuTTY - Categoría SSH-Auth.....	49
Figura 45. Cuadro de diálogo PuTTY primera conexión.....	50
Figura 46. Shell conexión con la instancia EC2 desde PuTTY.....	50
Figura 47. Ventana de conexión con nuestra instancia EC2.....	51
Figura 48. Shell remota conectada a la instancia de EC2.....	52
Figura 49. Formulario de registro del servicio No-IP.....	53
Figura 50. Instalación del DUC.....	54
Figura 51. Regla tráfico entrante para No-IP.....	55
Figura 52. Comando Maven version.....	57
Figura 53. Ejecución del script start-dfs.sh.....	62
Figura 54. Interfaz Web Hadoop HDFS.....	63
Figura 55. Ejecución del script zkServer.sh start.....	65
Figura 56. Explorador de HDFS.....	70
Figura 57. Salida script de arranque de Accumulo.....	71
Figura 58. Interfaz Web Accumulo.....	71
Figura 59. Salida script de parada de Accumulo.....	72
Figura 60. Interfaz web de Apache Spark.....	75
Figura 61. Spark-Shell.....	76
Figura 62. Ejecución del contenedor Docker hello-world.....	81
Figura 63. Ejecución proyecto de prueba de sbt.....	83
Figura 64. Comando "jupyter notebook password".....	84
Figura 65. Ejecución de la función passwd() en una Shell Python.....	85
Figura 66. Solicitud de contraseña en Jupyter.....	88
Figura 67. Interfaz Jupyter - Listado de archivos.....	89
Figura 68. Lista de kernels disponibles en Jupyter.....	89
Figura 69. Interfaz de edición de notebooks de Jupyter.....	90
Figura 70. Interfaz web de Spark ejecutando Apache Torette.....	90
Figura 71. Listado de kernels ejecutándose en Jupyter.....	91
Figura 72. Ejecución ejemplo CountByDay usando spark-submit.....	105

Figura 73. Ejecución ejemplo CountByDay en Jupyter con kernel Apache Toree ...	106
Figura 74. Ejecución ejemplo ShallowJoin en Jupyter con kernel Apache Toree.....	113
Figura 75. Enlazado de GitHub en Docker Hub.....	118





## Siglas

<b>AMI</b>	Amazon Machine Image
<b>API</b>	Application Programming Interface
<b>AWS</b>	Amazon Web Services
<b>DNS</b>	Domain Name System
<b>DUC</b>	Dynamic Update Client
<b>EBS</b>	Elastic Block Store
<b>EC2</b>	Elastic Cloud Compute
<b>GDELT</b>	Global Database of Events, Language, and Tone
<b>HDFS</b>	Hadoop Distributed File System
<b>HOCON</b>	Human-Optimized Config Object Notation
<b>HQL</b>	Hibernate Query Language
<b>JAR</b>	Java ARchive
<b>JDK</b>	Java Development Kit
<b>JVM</b>	Java Virtual Machine
<b>RAM</b>	Random Access Memory
<b>RDD</b>	Resilient Distributed Datasets
<b>S3</b>	Simple Storage Service
<b>SQL</b>	Structured Query Language
<b>SSH</b>	Secure SHell
<b>VM</b>	Virtual Machine
<b>XML</b>	Extensible Markup Language
<b>YARN</b>	Yet Another Resource Negotiator

# 1 Introducción

La llegada de la sociedad de la información ha multiplicado enormemente la cantidad de datos generados y las fuentes de información disponibles para su procesamiento. En estos datos reside, en muchas ocasiones, un importante potencial económico que sólo puede extraerse a través de un adecuado análisis. Sin embargo, en un caso general, la cantidad de datos relacionados, que deben ser procesados de forma concurrente y ordenada, es tan elevada; que no puede garantizarse un resultado adecuado en un tiempo razonable. En este contexto surgen las tecnologías de Big Data.

## 1.1 Motivación

Actualmente, la mayoría de los datos que generamos llevan asociada información geoespacial que facilita el posicionamiento de un objeto o evento en un sistema de coordenadas determinado. Cada día se generan cantidades enormes de información geográfica en muchas de nuestras acciones cotidianas: hacer una foto con nuestro Smartphone, publicar en redes sociales como Facebook o Twitter, hacer uso de las aplicaciones de navegación como Google Maps o Waze, realizar una compra en Amazon, etc. Y no sólo eso, también generamos información geográfica de forma intencionada, por ejemplo, mediante las estaciones meteorológicas, medidas de los sensores de satélites, información sobre tráfico en carreteras, etc.

Esta información es muy útil para las empresas ya que les permite adaptar su modelo de negocio a las diferentes zonas geográficas conforme a la información que han analizado de estas. También se usa para personalizar la publicidad a cada usuario, de forma que si, por ejemplo, los datos del usuario muestran un cierto interés por las innovaciones en realidad virtual y sus datos geoespaciales lo sitúan frecuentemente en Madrid, sería interesante publicitarle entradas para un congreso relacionado con esta temática en Madrid. Otra utilidad de esta información es la predicción de fenómenos naturales o predicción de congestiones en el tráfico de una ciudad. Esto recibe el nombre de geomarketing y es una herramienta de marketing que permite a las empresas analizar la situación de un negocio mediante la localización de los clientes.

La utilidad de los datos geográficos es enorme y muy diversa, pero como todos los datos, precisan de un análisis para ser convertidos en información útil. Este análisis se ha llevado a cabo durante mucho tiempo con Sistemas de Información Geográfica (SIG o GIS, en su acrónimo inglés) tradicionales que, debido al gran aumento de la cantidad de datos geográficos, han tenido que evolucionar para adaptarse a las técnicas y tecnologías de Big Data.

El objetivo de este Trabajo Fin de Máster es el diseño y la implementación de un entorno para el procesamiento de datos, donde sea posible analizar flujos de información geográfica. Para ello se hará uso de técnicas de Big Data y se emplearán las soluciones tecnológicas Apache Spark y GeoMesa. Se desplegará una plataforma de procesamiento donde se combinen algunas de las tecnologías más altamente demandadas en la actualidad como GeoMesa, Apache Spark o Apache Accumulo.

## 1.2 Objetivos

Los objetivos generales que se han definido para este Trabajo Fin de Master son los siguientes:

- Implementar un entorno para el procesamiento de grandes cantidades de datos geográficos usando Apache Spark y la biblioteca GeoMesa.
- Estudiar y comprender el funcionamiento aislado y en conjunto de las tecnologías usadas en entornos de procesamiento BigData.
- Estudiar y poner en práctica las metodologías de despliegue de entornos de procesamiento BigData en la nube pública.
- Estudiar el potencial del Big Data como sistema para extraer información útil de colecciones de datos heterogéneas.
- Estudiar el uso de la biblioteca GeoMesa junto a Spark para el análisis de grandes conjuntos de datos con información geográfica.

## 1.3 Estructura del documento

En este documento se describe la implementación de un entorno para el análisis de datos geográficos usando técnicas de BigData mediante Apache Spark y la biblioteca GeoMesa.

El documento se divide en tres partes claramente diferenciadas. En una primera parte se introduce el Trabajo Fin de Master y las herramientas que compondrán nuestro entorno. La segunda parte, de contenido más técnico, describe los pasos que hemos seguido para la implementación de dicho entorno y los métodos para importar datos en nuestra BBDD o ejecutar análisis sobre estos. La tercera y última parte supone la conclusión del trabajo.

Concretamente se distinguen los siguientes capítulos en este documento:

1. **Introducción:** Se describe la motivación que impulsa a usar técnicas de BigData aplicadas a datos geográficos y los objetivos generales que pretendemos conseguir con el trabajo.

2. **Servicios y tecnologías utilizadas:** Descripción de las principales herramientas que compondrán nuestro entorno de análisis de datos geográficos.
3. **Arquitectura del entorno:** Se ofrece una visión general de la arquitectura del entorno de procesamiento de datos geográficos que hemos desarrollado para este Trabajo Fin de Master.
4. **Implementación del entorno:** Descripción de los pasos seguidos para la implementación del entorno.
5. **Ingestión (ingest) de datos con GeoMesa-Accumulo:** Muestra una descripción de posibles métodos para importar datos a nuestra base de datos GeoMesa-Accumulo, ejemplificando cada uno de los métodos.
6. **Ejecución de aplicaciones GeoMesa-Spark en el entorno:** Se presentan los distintos procedimientos que tenemos disponibles en nuestro entorno para ejecutar un análisis una vez hemos desarrollado el código del mismo. También se explica el código de varios ejemplos que hemos ejecutado para probar nuestro entorno.
7. **Automatización de workers de Spark:** Se detalla el proceso de automatización del despliegue de workers de Spark que se ha seguido para complementar el desarrollo de nuestra plataforma de análisis de datos geográficos.
8. **Conclusiones:** Se plantean las conclusiones obtenidas de la realización del proyecto y del estudio de técnicas de BigData.



## 2 Servicios y tecnologías utilizadas

Durante este capítulo describiremos las tecnologías usadas para el desarrollo de nuestro entorno de procesamiento de grandes cantidades (Big Data) de datos geográficos.

Como podemos apreciar en la lista de las tecnologías usadas, para el desarrollo de este Trabajo Fin de Master se han precisado de herramientas muy novedosas en la actualidad, lo que conlleva que estén en continuo cambio y crecimiento. Esto ha supuesto que durante el desarrollo del proyecto se haya precisado de una continua búsqueda de información y resolución de problemas, destacando los problemas derivados de la compatibilidad de versiones entre las distintas herramientas que deben trabajar conjuntamente.

### 2.1 Amazon Web Services

Amazon Web Services (1) (AWS abreviado) es una filial de Amazon.com que ofrecen una gran variedad de servicios de computación en la nube bajo demanda a través de una única plataforma web.



Figura 1. Logo Amazon Web Services

Para acceder a la plataforma de Amazon Web Services, desde la que se gestionan todos los servicios, podemos usar cualquier navegador web junto a nuestra cuenta de Amazon. Todos los servicios son facturados en función del uso, pero la forma de uso por la que es medida la facturación varía de un servicio a otro.

Amazon Web Services cuenta con una capa gratuita durante el primer año que nos da acceso a ciertos recursos muy limitados de AWS. En nuestro caso esta capa gratuita no ha sido suficiente y hemos tenido que recurrir al programa “AWS Educate”, que nos da la posibilidad de obtener créditos por valor de 100\$ canjeables en la mayoría de servicios ofertados en AWS. Para ello debemos inscribirnos en el programa con una cuenta de correo de nuestra institución educativa, rellenar una solicitud en la que debemos indicar, entre otras cosas, el motivo para solicitar los créditos y el ID de la

cuenta a la que se destinarán dichos créditos y esperar a que dicha solicitud sea aceptada por el equipo de Amazon.

### 2.1.1 Amazon EC2

Entre los servicios de computación en la nube ofrecidos por Amazon Web Services se encuentra Amazon Elastic Cloud Compute (2) (abreviado Amazon EC2), sobre el que hemos ejecutado todos los componentes de nuestro Trabajo Fin de Máster.

Amazon Elastic Compute Cloud (3) es un servicio web que nos permite ejecutar aplicaciones en la nube pública de AWS de una forma sencilla y rápida. Para ello EC2 nos brinda la oportunidad de desplegar servidores virtuales que nos proporcionan capacidad de cómputo para nuestros proyectos.

Una instancia de Amazon EC2 (4) es cada uno de estos servidores desplegados haciendo uso de este servicio. El usuario de EC2 puede incrementar o decrementar la capacidad de la instancia según necesite en cuestión de minutos, usando la interfaz web de EC2 o la API de este servicio. De hecho, es posible desarrollar aplicaciones para escalar automáticamente nuestras instancias o simplemente definir políticas de auto-escalado para nuestras instancias.

Para usar EC2, el cliente crea una Amazon Machine Image (AMI), que es una “imagen” que contiene un sistema operativo, aplicaciones y configuraciones. Esta AMI se sube a Amazon Simple Storage Service (Amazon S3) y se registra en EC2, creando un identificador de AMI. Una vez hecho esto, el usuario puede lanzar nuevas máquinas virtuales (instancias) basadas en esta AMI. Otra posibilidad es lanzar una instancia basada en una AMI proporcionada por AWS, la comunidad de usuarios o AWS Marketplace.

Los datos sólo permanecen en una instancia EC2 mientras esta se está ejecutando, sin embargo, podemos usar un volumen Amazon Elastic Block Store (Amazon EBS) para persistir estos datos y Amazon S3 para realizar backups.

Amazon EC2 proporciona diferentes tipos de instancias según su tamaño (CPU, memoria y almacenamiento) y estructura de tarificación. Estos tipos son diseñados para ajustarse a las diferentes necesidades presupuestarias y de cómputo. Las instancias bajo demanda permiten a los clientes disponer tantos recursos como precisen y pagar por ellos según las horas que se usen. Las instancias reservadas proporcionan un descuento en el precio de los recursos a cambio de comprometerse mediante un contrato a usar estos recursos durante la duración indicada. También tenemos las instancias de host dedicados en la que disponemos de espacio en un servidor físico dedicado. Para este proyecto se han utilizado instancias del primer tipo.

## 2.2 Apache Maven

Maven (5) (6) es una herramienta Open Source de la fundación Apache que nos permite simplificar el proceso de compilación y generación de ejecutables para nuestros proyectos. Con esta herramienta se buscó ofrecer una forma estándar de compilar los proyectos, una definición clara de en qué consiste nuestro proyecto, una facilidad para publicar información de nuestro proyecto y una forma de compartir JARs entre varios proyectos. El resultado fue una herramienta capaz de automatizar el proceso de compilación y gestionar cualquier proyecto basado en Java.



Figura 2. Logo Apache Maven

La potencia de una herramienta de automatización del proceso de compilación como Maven está en el tiempo que ahorra en esta tarea. Sin una herramienta de este tipo, el desarrollador invierte mucho tiempo en aprender las peculiaridades de cada proyecto: analizar que partes del código deben ser compiladas, que librerías usa el código, donde hay que incluirlas, que dependencias de compilación tiene el proyecto, etc. Otro aspecto en el que nos ahorra tiempo es en la ejecución de pruebas.

Sin embargo, Maven es más que una herramienta que automatiza la compilación del código ya que cuenta con otras funcionalidades. De hecho, Maven es capaz de gestionar completamente nuestro proyecto software, desde que comprobamos que nuestro proyecto sea correcto, hasta que realizamos el despliegue de nuestra aplicación.

Para esto, Maven define ciclos de vida (6), una lista de fases con identificador que pueden ser usadas para dar un orden al proceso. Para llevar a cabo alguna de estas fases en nuestro código, nos basta con usar el comando “mvn” seguido del nombre de la fase que deseamos ejecutar. La ejecución es en cadena, de forma que para ejecutar la fase que hemos especificado, se ejecuta previamente el resto de fases que le preceden. El ciclo por defecto contiene las siguientes fases:

1. Validar (**validate**) que el proyecto es correcto y toda la información necesaria está disponible.
2. Generar cualquier código fuente (**generate-sources**) necesario para su inclusión en la compilación.
3. Procesar el código fuente (**process-sources**).



4. Generar los recursos (**generate-resources**) para inclusión en el paquete.
5. Copiar y procesar los recursos (**process-resources**) en su directorio destino, listos para empaquetar.
6. Compilar (**compile**) el código fuente de nuestro proyecto.
7. Generar cualquier código fuente de test (**generate-test-sources**) necesario para la compilación.
8. Procesar el código fuente de los test (**process-test-source**).
9. Crear los recursos necesarios para las pruebas (**generate-test-resources**).
10. Copiar y procesar los recursos para las pruebas (**process-test-resources**) en su directorio destino.
11. Compilar las pruebas (**test-compile**) en su directorio destino.
12. Probar (**test**) el código fuente usando un framework de pruebas unitarias.
13. Tomar el código compilado y empaquetarlo (**package**) en un formato distribuible como por ejemplo JAR.
14. Procesar y desplegar el paquete, si es necesario, en un entorno donde podamos ejecutar las pruebas de integración (**integration-test**).
15. Ejecutar cualquier comprobación para verificar (**verify**) que el paquete es válido y cumple los criterios de calidad.
16. Instalar (**install**) el paquete en el repositorio local para usarlo como dependencia en otros proyectos.
17. Copiar el paquete en un repositorio remoto para compartirlo con otros desarrolladores y proyectos.

La gestión de dependencias y versiones se facilita mediante el uso de Maven. Para ello, cada proyecto cuenta con un fichero XML (pom.xml) en el que se describe como debe ser compilado el proyecto, sus dependencias, el orden de compilación, directorios y plugins requeridos.

Maven es capaz de descargar estas dependencias Java o plugins de forma dinámica de uno o más repositorios y guardarlos en una cache o repositorio local. Este repositorio local también puede ser actualizado con artefactos creados por proyectos locales. Gracias a los plugins de Maven, es posible extender su funcionalidad a otros lenguajes de programación como C#, Ruby, Scala y otros.

El uso que le hemos dado a Maven en este proyecto es compilar algunas librerías que hemos necesitado para realizar el desarrollo de nuestro entorno de procesado de datos geográficos con Big Data, además de generar paquetes JAR con dependencias de algún ejemplo que hemos ejecutado para probar dicho entorno.

## 2.3 Apache Hadoop

Apache Hadoop (8) (9) es un framework que nos permite realizar procesamiento y almacenamiento distribuido de grandes conjuntos de datos a lo largo de un cluster de computadores usando para ello modelos de programación simples.



Figura 3. Logo Apache Hadoop

El framework está diseñado para poder escalar desde pocos servidores hasta miles de máquinas, cada una de las cuales pone a disposición del cluster su capacidad de cómputo y de almacenamiento.

En lugar de confiar en el hardware para ofrecer alta disponibilidad, Hadoop está diseñada para detectar y manejar los fallos en la capa de aplicación, por lo que ofrece un servicio con alta disponibilidad sobre un cluster de computadores, cada uno de los cuales puede ser propenso a fallos.

Como todo framework software, Hadoop se compone de varios módulos funcionales. Como mínimo, Hadoop usa el módulo Hadoop Common como kernel para ofrecer las librerías esenciales del framework. Entre el resto de componentes podemos destacar los siguientes módulos:

- **Hadoop Distributed Files System (HDFS).** Este módulo cumple la funcionalidad de sistema de archivos distribuido. Es capaz de almacenar datos a lo largo de miles de servidores para lograr ofrecer un alto ancho de banda entre los diferentes nodos.
- **Hadoop Yet Another Resource Negotiator (YARN).** Este módulo proporciona funciones de planificación de trabajos y gestión de los recursos de un cluster.
- **Hadoop MapReduce.** Este módulo consiste en un sistema basado en YARN para el procesamiento en paralelo y distribuido de grandes conjuntos de datos. Proporciona un modelo de programación utilizado para abordar el procesamiento de grandes cantidades de datos distribuidos consistente en el mapeado (Map) de los datos y la posterior reducción (Reduce) a un resultado de dichos datos.

Además de estos módulos, existen otros que no vamos a mencionar por no ser relevantes para nuestro proyecto. En la siguiente figura podemos ver la arquitectura principal de Hadoop.

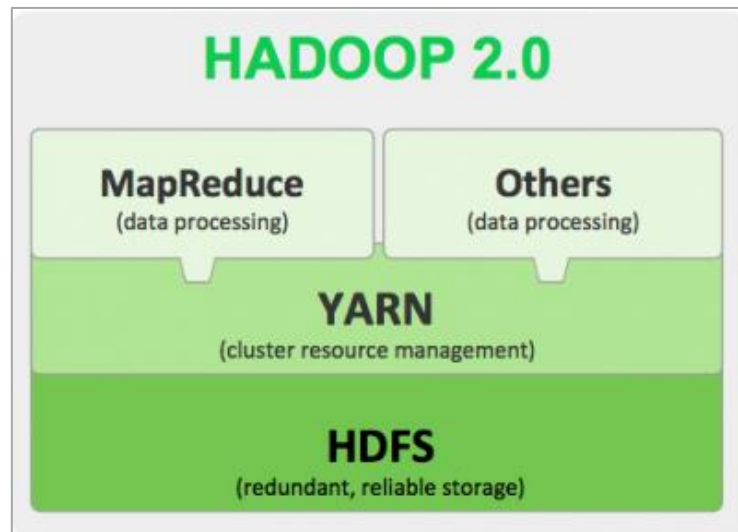


Figura 4. Arquitectura de Hadoop

En este proyecto haremos uso principalmente del módulo HDFS como sistema de archivos para Apache Accumulo. También usaremos, en menor medida, MapReduce como ejemplo de método de ingestión de datos en Geomesa-Accumulo.

### 2.3.1 HDFS

Hadoop Distributed File System (HDFS) (10) es un sistema de archivos distribuido diseñado para ejecutarse sobre hardware relativamente barato.



Figura 5. Logo Hadoop HDFS

HDFS tiene muchas similitudes con los sistemas de archivos distribuidos existentes, pero son más significativas las diferencias que tiene con estos. HDFS es altamente tolerante a fallos y está diseñado para desplegarse en hardware de bajo coste. Proporciona un alto rendimiento en el acceso a datos de aplicación y es altamente adecuado para aplicaciones que manejan grandes conjuntos de datos.

Para conseguir estas características, HDFS se basa en una arquitectura master/slave (maestro/esclavo). Un clúster HDFS consta de un único NameNode, un servidor

máster que gestiona el namespace del sistema de ficheros y controla el acceso a los ficheros por parte de los clientes. Además, consta de un cierto número de DataNodes, normalmente uno por nodo en el clúster, que gestionan el almacenamiento correspondiente a los nodos sobre los que se ejecutan. En la siguiente figura se muestra la arquitectura de HDFS:

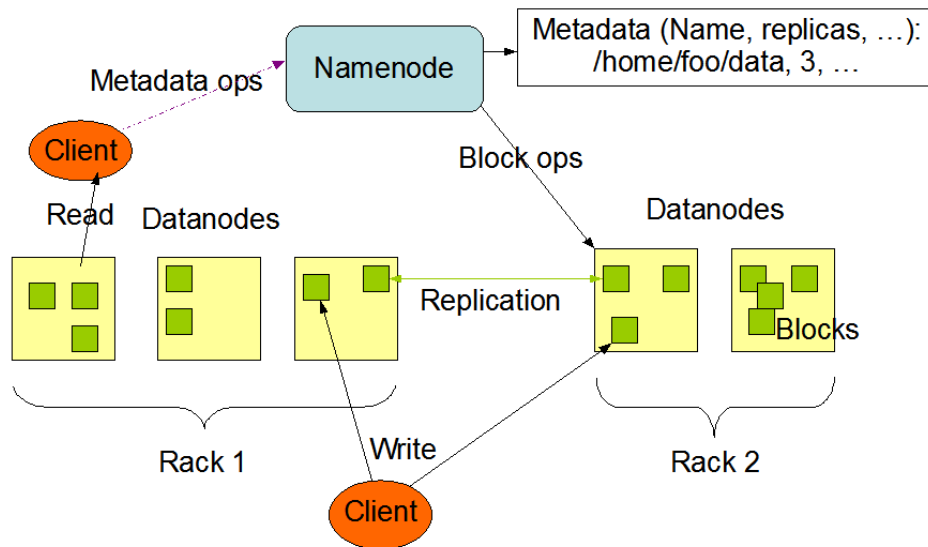


Figura 6. Arquitectura HDFS.

HDFS expone un namespace de un sistema de ficheros y permite que los datos de los usuarios se almacenen en ficheros. Internamente, un fichero se divide en uno o más bloques y estos bloques se almacenan en un conjunto de DataNodes. El NameNode ejecuta operaciones del namespace del sistema de archivos como por ejemplo abrir, cerrar y renombrar archivos y directorios. También determina el mapeado de bloques con DataNodes. Los DataNodes son responsables de servir las peticiones de lectura y escritura de los clientes. También son responsables de las operaciones de creación, borrado y replicado de bloques.

La existencia de un único NameNode en el cluster simplifica la arquitectura del sistema. El NameNode mantiene todos los metadatos HDFS del sistema y este está diseñado de forma que los datos de usuario nunca pasan por el NameNode.

Tanto NameNode como DataNodes son piezas de software diseñadas para ejecutarse en máquinas de bajo coste. Están escritas en Java, por lo que cualquier sistema capaz de ejecutar Java puede ejecutar un NameNode o DataNode. El uso del lenguaje Java significa que HDFS puede desplegarse en un amplio rango de máquinas. Un despliegue típico consta de una máquina dedicada que ejecuta únicamente el NameNode y cada una del resto de máquinas ejecuta una instancia del DataNode.

## 2.4 Apache Zookeeper

Zookeeper (11) (12) es un servicio open source de coordinación de alto rendimiento para aplicaciones distribuidas desarrollado por Apache. Entre otras cosas, ofrece servicios de gestión de configuraciones, naming, sincronización, servicios de grupo, consenso, elección de líder, etc.



Figura 7. Logo Apache Zookeeper

Zookeeper es ideal para distribuir servicios en clusters con un gran número de nodos, consiguiendo dotar al servicio de tolerancia a fallos, alta disponibilidad mediante redundancia en los nodos y alta escalabilidad. Además, permite a los desarrolladores implementar las funciones de coordinación mediante una API.

Cuando tenemos un número de nodos gestionados por Zookeeper, este categoriza los nodos en líder (o coordinador) y seguidores. Los nodos seguidores se comunican con el líder para sincronizarse, manteniendo una copia exacta del estado actual del líder, de forma que, si el nodo líder cae, Zookeeper se encarga de cambiar a otro nodo líder manteniendo la disponibilidad del sistema. Es importante precisar que las peticiones por parte de los clientes pueden ser enviadas a cualquier nodo (líder o seguidor).

A continuación, explicamos algunas de las características más destacables de Zookeeper:

- **Simple.** Zookeeper permite a los procesos distribuidos coordinarse entre ellos a través de un espacio de nombres jerárquico y compartido el cual se organiza de forma similar a un sistema de archivos estándar. Este espacio de nombres consiste en registros de datos (llamados znodes) y son similares a ficheros y directorios. Al contrario que un sistema de archivos tradicional, el cual está diseñado para almacenamiento, Zookeeper mantiene los datos en memoria principal por lo que consigue alto rendimiento y baja latencia. Este alto rendimiento ofrecido por Zookeeper le permite ser usado en sistemas distribuidos de gran tamaño.

- **Replicado.** Al igual que los procesos distribuidos que coordina, Zookeeper en sí está destinado a ser replicado en un conjunto de hosts.

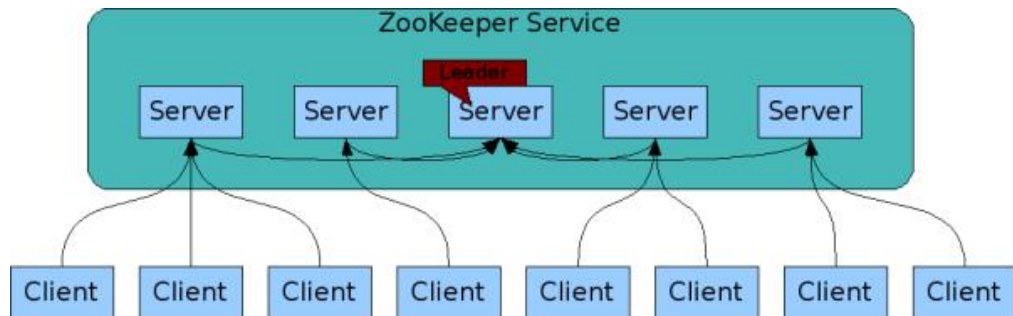


Figura 8. Arquitectura servicio Zookeeper

Los servidores que componen el servicio Zookeeper deben conocerse entre ellos. Mantienen una imagen de estado en memoria, junto con un registro de transacciones e instantáneas en un almacén persistente. Mientras la mayoría de los servidores estén disponibles, el servicio Zookeeper estará disponible.

Los clientes se conectan a un único servidor Zookeeper, manteniendo una conexión TCP a través de la cual envían peticiones y reciben respuestas y eventos. Si esta conexión se interrumpe, el cliente se conectará a un servidor diferente.

- **Ordenado.** Zookeeper sella cada actualización con un número que refleja el orden de todas las transacciones realizadas. Las operaciones subsiguientes pueden utilizar el orden para implementar abstracciones de nivel superior, como primitivas de sincronización.
- **Rápido.** Zookeeper es especialmente rápido en cargas de trabajo “read-dominant”, es decir, que predominen las peticiones de lectura frente a las escrituras (aproximadamente en una proporción de 10:1).

## 2.5 Apache Accumulo

Apache Accumulo (13) (14) es un software de almacenamiento estructurado altamente escalable basado en BigTable de Google. Accumulo está escrito en Java y opera a través de Hadoop Distributed File System (HDFS) y Zookeeper.



Figura 9. Logo Apache Accumulo

Accumulo implementa eficientemente el almacenamiento y la recuperación de datos estructurados, incluyendo consultas por rangos, y permite utilizar tablas Accumulo como entrada y salida para trabajos MapReduce.

Entre sus características podemos destacar las siguientes:

- **Cell-level security.** Accumulo extiende el modelo de datos de Bigtable, añadiendo un nuevo elemento, llamado “Column Visibility”, a la clave. Este elemento almacena una combinación lógica de etiquetas de seguridad que deben satisfacerse en el momento de la consulta para que la clave y el valor asociado se devuelvan como parte de la petición del usuario. Esto nos permite almacenar datos con diferentes requisitos de seguridad en la misma tabla y permite a los usuarios consultar sólo aquellos datos para los que están autorizados.
- **Server-side programming.** Apache Accumulo proporciona un mecanismo de programación en servidor llamado “Iteradores” que permite a los usuarios realizar procesamiento adicional en el servidor Tablet (explicaremos a continuación la arquitectura y componentes de Accumulo). El rango de operaciones que pueden ser aplicadas es equivalente a aquellas que pueden ser implementadas dentro de una función MapReduce, que produce un valor agregado para varios pares clave-valor.

### 2.5.1 Modelo de datos

Accumulo proporciona un modelo de datos (15) enriquecido respecto al proporcionado por los almacenamientos de clave-valor, pero sin llegar a ser una base de datos completamente relacional. Los datos se representan como pares clave-valor, donde la clave y el valor están compuestos por los elementos mostrados en la siguiente figura:

Key				Value	
Row ID	Column				Timestamp
	Family	Qualifier	Visibility		

Figura 10. Modelo de datos de Accumulo

Accumulo ordena las claves según sus elementos y lexicográficamente en orden ascendente. La excepción se encuentra en las marcas temporales (“el campo Timestamp” de la clave) que se ordenan en orden descendente de forma que las versiones posteriores de una misma clave aparezcan antes en una exploración

secuencial. Las tablas de Accumulo consisten en un conjunto de pares clave-valor ordenados.

### 2.5.2 Arquitectura

Accumulo es un sistema de almacenamiento y recuperación de datos distribuido y, como tal, consta de varios componentes, algunos de los cuales se ejecutan en varios servidores individuales. Gran parte del trabajo que realiza Accumulo, consiste en mantener ciertas propiedades de los datos, tales como organización, disponibilidad e integridad, a través de muchas máquinas.

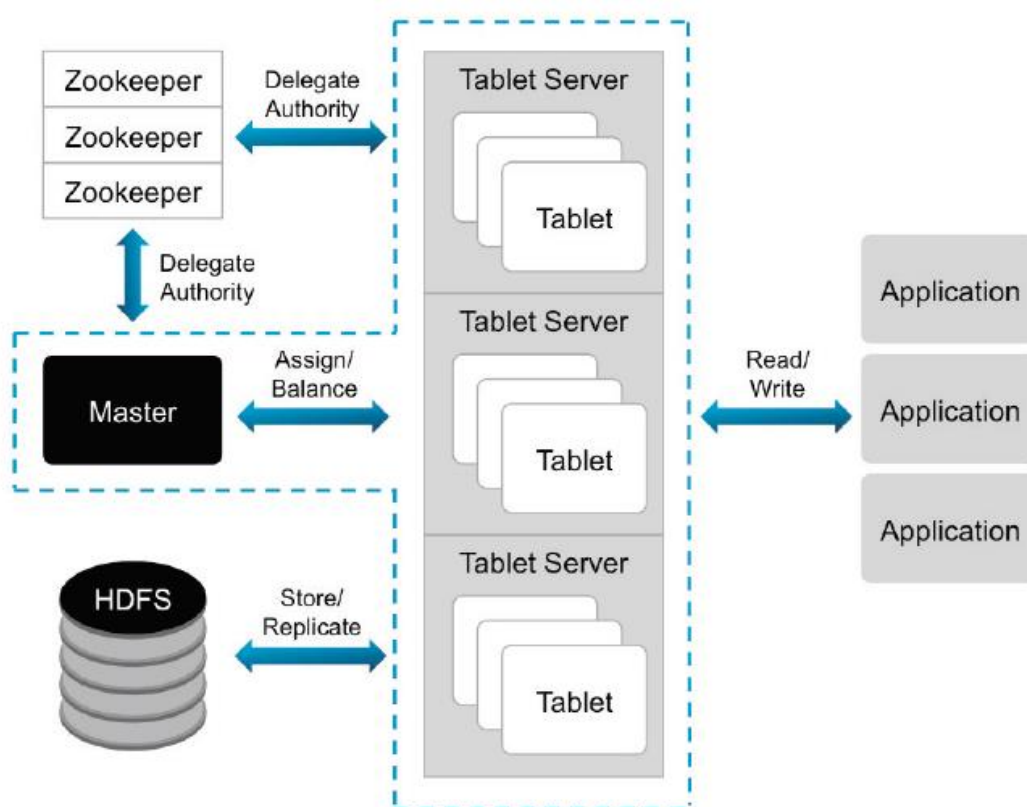


Figura 11. Arquitectura Apache Accumulo

Una instancia de Accumulo incluye un número variable de TabletServers y Clientes, un servidor Master y un proceso recolector de basura (Garbage Collector process). A continuación, pasaremos a explicar cada uno de los componentes de esta arquitectura.

#### *TabletServer*

Cada uno de los TabletServers se encargan de gestionar un subconjunto de todas las tablets (particiones de tablas). Esto incluye la recepción de peticiones de escritura de los clientes, la persistencia de estas escrituras en un registro de escritura anticipada, ordenar nuevos pares clave-valor en memoria, la escritura periódica de los pares clave-valor ordenados en un nuevo fichero en HDFS y responder a las peticiones de lectura de los clientes.



Los TabletServers también realizan la tarea de recuperar tablets perdidos por pertenecer a un servidor que ha fallado. Para ello, vuelven a aplicar las escrituras encontradas en el registro de escritura anticipada.

### *Garbage Collector*

Los procesos de Accumulo comparten entre ellos archivos almacenados en HDFS. Periódicamente, el recolector de basura identifica los archivos que no son necesarios por ningún proceso y los elimina. Es posible ejecutar varios recolectores de basura para proporcionar soporte hot-standby. Entre los procesos disponibles se llevará a cabo una elección de líder para seleccionar una única instancia de estos procesos para estar activo.

### *Master*

El servidor Master es el responsable de detectar y responder a fallos en los TabletServers. Intenta balancear la carga entre los diferentes TabletServers asignando cuidadosamente las tablets y solicitando a los TabletServers liberar memoria ocupada por tablets cuando sea necesario. El Master también asegura que cada una de las tablets estén asignadas a un TabletServer, y gestiona las peticiones de creación, alteración y borrado de tablas. Además, también es el encargado de coordinar el inicio del sistema, el apagado seguro y de la recuperación de los cambios en los registros de escritura anticipada cuando los TabletServers fallan.

Es posible ejecutar varios servidores Master al mismo tiempo. Las instancias que se estén ejecutando elegirán entre ellas un solo Master y el resto serán copias de seguridad para el caso en que el Master elegido fallase.

### *Monitor*

El componente Monitor es una aplicación web que proporciona una gran cantidad de información sobre una instancia. Esta aplicación web muestra gráficas y tablas que contienen información sobre velocidades de lectura/escritura, tasas de acierto/error de cache, e información sobre tablas de Accumulo. Además, el componente Monitor sirve como primer punto de entrada cuando intentamos depurar un problema de Accumulo, ya que nos mostrará los problemas de alto nivel además de errores agregados de todos los nodos del cluster.

### *Client*

Accumulo incluye bibliotecas para cliente que se enlaza a todas las aplicaciones que hacen uso de esta. La biblioteca cliente contiene lógica para buscar servidores que gestionan una tablet en particular y para comunicarse con TabletServers para escribir y recuperar pares clave-valor.

### 2.5.3 Gestión de datos

Accumulo almacena los datos en tablas, las cuales son particionadas en tablets. Los tablets se conforman de un conjunto de filas de forma que todas las columnas y valores de una fila particular se encuentran en el mismo tablet. El maestro asigna los tablets a un TabletServer a la vez. Esto permite que las transacciones a nivel de fila se lleven a cabo sin utilizar bloqueo distribuido o algún otro mecanismo de sincronización complicado. A medida que los clientes insertan y consultan datos, y cuando se añaden y eliminan máquinas del clúster, el Master migra los tablets para asegurarse de que permanezcan disponibles y que la carga de ingestión y consulta esté equilibrada en el clúster.

En la siguiente figura podemos ver un ejemplo de distribución de datos en la base de datos Accumulo.

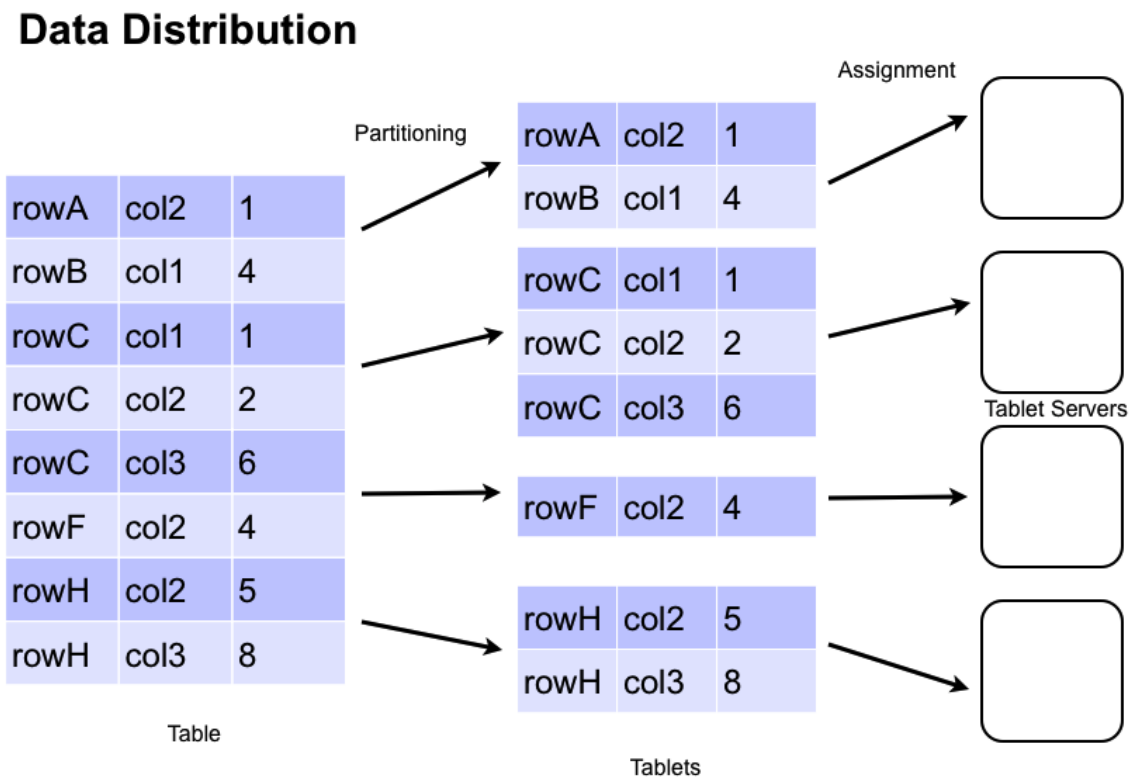


Figura 12. Distribución de datos en Accumulo

Es posible observar lo que ya comentamos en el párrafo anterior: si una fila tiene varias columnas, al particionar la tabla en tablets, todas las columnas de esta fila se insertan en el mismo tablet y, por tanto, se asigna a un mismo TabletServer.

## 2.6 Apache Spark

Apache Spark (16) (17) (18) es un framework open-source de computación en cluster basado en la velocidad, facilidad de uso y análisis sofisticado. Originalmente fue

desarrollado por la Universidad de California para posteriormente pasar a ser gestionado y desarrollado por la Apache Software Foundation. Spark proporciona una interfaz para programar clústeres enteros con paralelismo implícito de datos y tolerancia a fallos.



Figura 13. Logo Apache Spark

Proporciona APIs de alto nivel en Java, Scala, Python y R, y un motor optimizado que da soporte a grafos de ejecución general. También soporta un rico conjunto de herramientas de alto nivel incluyendo Spark SQL para SQL y procesamiento de datos estructurados, MLlib para Machine Learning, GraphX para procesamiento de grafos y Spark Streaming.

En términos de velocidad, Spark extiende el modelo MapReduce de Hadoop para soportar eficientemente más tipos de computaciones, incluyendo consultas interactivas y procesamiento en streaming. A la hora de procesar grandes conjuntos de datos la velocidad es muy importante, tanto que supone la diferencia entre explorar datos interactivamente o esperar minutos e incluso horas. Una de las principales características que ofrece Spark es la habilidad de ejecutar computaciones en memoria, pero el sistema es también más eficiente que MapReduce de Hadoop en el caso de aplicaciones complejas que se ejecutan en disco.

En términos de generalidad, Spark está diseñado para cubrir un amplio rango de cargas de trabajo que previamente requerían sistemas distribuidos separados, incluyendo aplicaciones batch, algoritmos iterativos, consultas interactivas, y streaming. Para soportar estas diferentes cargas de trabajo en un mismo motor, Spark hace que sea fácil y económico combinar diferentes tipos de procesamiento, lo cual es necesario a menudo en entornos de análisis de datos en producción. Además, esto reduce la carga de gestión asociada a mantener herramientas separadas.

### 2.6.1 Stack unificado

Apache Spark contiene múltiples componentes (19) estrechamente integrados. En su núcleo, Spark es un motor computacional el cual es responsable de programar,

distribuir y monitorizar aplicaciones consistentes en múltiples tareas computacionales repartidas en varios workers, o un cluster. Debido a que el motor central de Spark es rápido y de propósito general, da soporte a múltiples componentes de alto nivel especializadas en diferentes tareas, como SQL o Machine Learning. Estos componentes están diseñados para interoperar, permitiéndonos combinarlos como cualquier biblioteca en un proyecto software.

Esta filosofía de integración estrecha entre componentes tiene varios beneficios. En primer lugar, todas las bibliotecas y componentes de nivel superior en el stack se benefician de las mejoras de las capas inferiores. Por ejemplo, cuando el motor central de Spark añade una optimización, las bibliotecas de SQL y Machine Learning también se benefician automáticamente de esta mejora. En segundo lugar, los costes asociados a ejecutar este stack unificado se minimizan porque en lugar de ejecutar un número indeterminado de sistemas software independientes, necesitamos ejecutar sólo uno. Estos costes incluyen desarrollo, mantenimiento, testeo, soporte y otros. Esto también significa que cada vez que añadimos un nuevo componente a este stack, cada organización que use Spark, será capaz inmediatamente de probar este nuevo componente.

Por último, una de las mayores ventajas de la integración estrecha es la capacidad de crear aplicaciones que combinan perfectamente diferentes modelos de procesamiento. Por ejemplo, en Spark podemos escribir una aplicación que use Machine Learning para clasificar datos en tiempo real, obteniéndolos de fuentes en streaming. Simultáneamente, los analistas pueden consultar los datos resultantes, también en tiempo real, vía SQL. Además, estos datos pueden ser solicitados desde clientes de diferente índole. Todo esto manteniendo un solo sistema.

En la siguiente figura podemos ver el stack unificado de Spark que hemos comentado.

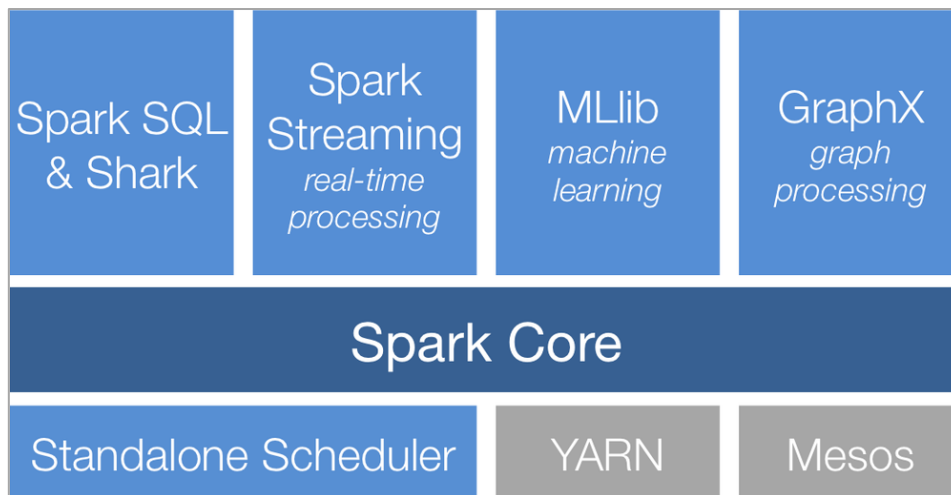


Figura 14. Stack de Apache Spark

### ***Spark Core***

Es el núcleo de Spark y contiene la funcionalidad básica de este, incluyendo componentes para programación de tareas, gestión de memoria, recuperación de fallos, interacción con sistemas de almacenamiento y más. Spark Core contiene también la API que define los RDDs (Resilient Distributed Datasets), que son la principal abstracción de programación de Spark. Un RDD representa una colección de elementos distribuidos a través de un cluster y que pueden ser manipulados en paralelo. Spark Core proporciona varias APIs para construir y manipular estas colecciones.

### ***Spark SQL***

Spark SQL es un paquete de Spark para trabajar con datos estructurados. Permite consultar datos a través de SQL o su variante HQL y soporta varias fuentes de datos. Spark SQL permite a los desarrolladores mezclar consultas SQL con las manipulaciones de RDDs en Python, Java y Scala, todo ello dentro de una sola aplicación, combinando así SQL con análisis complejos. Spark SQL fue añadido a Spark en su versión 1.0.

### ***Spark Streaming***

Spark Streaming es un componente de Spark que habilita el procesamiento de flujos de datos en tiempo real. Como ejemplos de estos flujos de datos podemos señalar los archivos de registros generados por los servidores en producción, o colas de mensajes que contienen actualizaciones de estado publicados por usuarios de un servicio web. Spark Streaming proporciona una API para manipular flujos de datos que se aproxima a la API RDD de Spark Core, facilitando a los programadores el aprendizaje del proyecto y moverse entre aplicaciones que manipulan datos almacenados en memoria, en disco o que llegan en tiempo real. Spark Streaming fue diseñado para proporcionar el mismo grado de tolerancia a fallos, rendimiento y escalabilidad que Spark Core.

### *MLlib*

Spark proporciona una biblioteca llamada MLlib y que contiene funcionalidades comunes de Machine Learning. MLlib proporciona múltiples tipos de algoritmos de Machine Learning, incluyendo clasificación, regresión, clustering y filtrado colaborativo, además de funcionalidades de soporte como evaluación de modelos e importación de datos. También ofrece algunas primitiva Machine Learning de más bajo nivel, incluyendo un algoritmo genérico de optimización por gradiente descendente. Todos estos métodos están diseñados para ser escalables y poder usar toda la capacidad de computo de un cluster.

Debido, en gran parte, a la arquitectura basada en memoria de Spark, MLlib es hasta nueve veces más rápido que la implementación basada en disco usada por Apache Mahout (según los benchmarks hechos por los desarrolladores de MLlib y antes de que Mahout ganara una interfaz Spark), y escala mejor que Vowpal Wabbit.

### *GraphX*

GraphX es una biblioteca para manipular grafos (por ejemplo, grafos de amistad en redes sociales) y realizar computaciones en paralelo sobre estos grafos. Al igual que Spark Streaming y Spark SQL, GraphX extiende la API RDD de Spark, permitiéndonos crear grafos con propiedades arbitrarias. GraphX también nos proporciona varios operadores para manipular grafos y una biblioteca de algoritmos comunes para grafos.

### *Gestores de Cluster*

En modo distribuido, Spark utiliza una arquitectura master/slave con un coordinador central y muchos workers distribuidos. El coordinador central recibe el nombre de driver. El driver se comunica con un número potencialmente grande de workers distribuidos llamados executors. El driver se ejecuta en su propio proceso Java y cada executor es un proceso Java separado. La siguiente figura representa esta arquitectura.

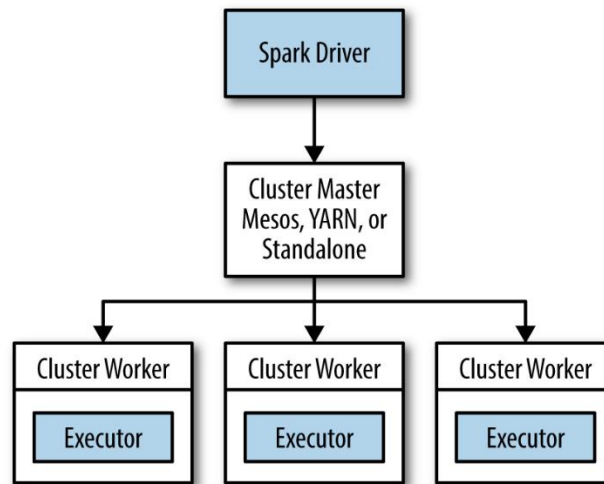


Figura 15. Componentes de una aplicación Spark distribuida

Spark está diseñado para escalar eficientemente de uno a varios miles de nodos. Para lograr esto y maximizar la flexibilidad, Spark puede ejecutarse sobre varios gestores de cluster, incluyendo Hadoop YARN, Apache Mesos, y un gestor de cluster simple incluido en Spark y que recibe el nombre de Standalone Scheduler (planificador independiente). Si tan sólo tenemos la necesidad de instalar Spark en un conjunto pequeño de máquinas, este último gestor proporciona una forma fácil para comenzar en el uso de Spark. Si ya tenemos un cluster gestionado por Hadoop YARN o Mesos, el soporte de Spark para estos gestores permite que nuestras aplicaciones puedan ser ejecutadas en estos.

## 2.7 GeoMesa

GeoMesa (20) (21) es una base de datos espaciotemporal, distribuida y open-source construida sobre una serie de sistemas distribuidos de almacenamiento de datos en la nube, incluyendo Accumulo, HBase, Cassandra, y Kafka. Aprovechando una estrategia de indexación altamente paralelizada, GeoMesa tienen como objetivo proporcionar a Accumulo (y otros sistemas distribuidos de almacenamiento) consultas espaciales y manipulación de datos.



Figura 16. Logo GeoMesa

Más concretamente, GeoMesa es un conjunto de herramientas de código abierto con licencia Apache que permite realizar análisis geoespaciales a gran escala en entornos cloud y sistemas de computación distribuidos, permitiéndonos gestionar y analizar los enormes conjuntos de datos que el IoT, las redes sociales, y las aplicaciones móviles generan para tomar ventaja de estos.

Para conseguir esto, GeoMesa proporciona persistencia de datos espaciotemporales sobre bases de datos distribuidas y orientadas a columnas como Accumulo, HBase y Cassandra. Esto permite un acceso rápido a estos datos a través de consultas que aprovechan al máximo las propiedades geográficas para especificar la distancia y el área. GeoMesa también proporciona soporte para el procesamiento de datos espaciotemporales en tiempo real haciendo uso de Apache Kafka.

Las características de GeoMesa incluyen la capacidad de:

- Almacenar desde gigabytes a petabytes de datos espaciotemporales.
- Servir decenas de millones de puntos geográficos en segundos.
- Ingerir datos a una velocidad superior a 10000 registros por segundo por cada nodo.
- Escalar horizontalmente con facilidad, es decir, agregar más servidores para añadir más capacidad.
- Soportar análisis de datos usando Apache Spark.

### 2.7.1 Arquitectura

Como ya hemos dicho, GeoMesa (22) soporta varias tecnologías de almacenamiento de datos basadas en la nube, incluyendo Apache Accumulo, Apache HBase, y Google Cloud Bigtable, así como Apache Kafka para streaming de datos. Apache Storm nos permite definir fuentes de información y manipulaciones para permitir procesamiento distribuido por lotes de datos en streaming con GeoMesa, y los entornos GeoMesa también pueden tomar ventaja de Apache Spark para hacer análisis a gran escala de datos almacenados y en streaming.



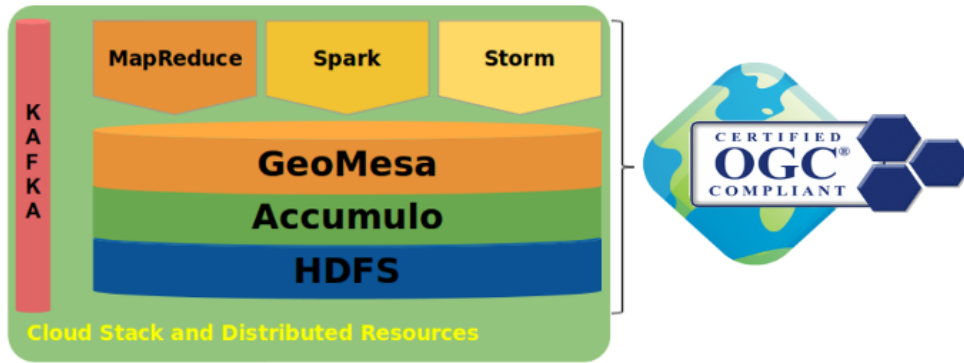


Figura 17. Arquitectura Geomesa

### Integración con GeoMesa

Para exponer los datos geoespaciales que almacena para ser usados por sus aplicaciones, GeoMesa implementa las interfaces de GeoTools para proporcionar acceso HTTP a los siguientes estándares de Open Geospatial Consortium: Web Feature Service (WFS), Web Mapping Service (WMS), Web Processing Service (WPS) y Web Coverage Service (WCS).

Es posible usar múltiples frameworks para el streaming y la ingestión por lotes de datos. Entre ellos se incluyen las herramientas de línea de comandos de GeoMesa, tareas map-reduce con Apache Hadoop, y topologías en tiempo real con Apache Storm. La siguiente figura muestra una posible arquitectura de ingestión:

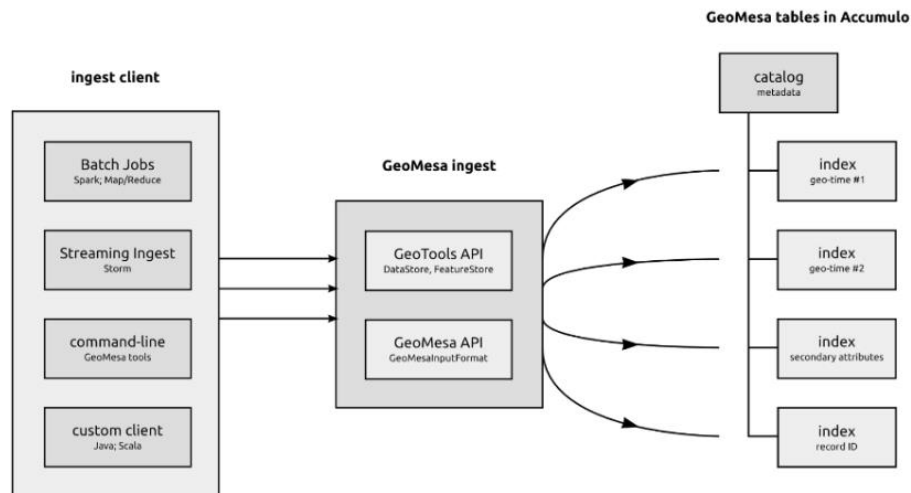


Figura 18. Posible arquitectura de ingestión de datos con GeoMesa

Así mismo, la siguiente figura muestra una posible arquitectura para las peticiones en la cual las APIs de GeoMesa y GeoTools median el uso de los iteradores de Apache Accumulo para clientes externos de consulta.

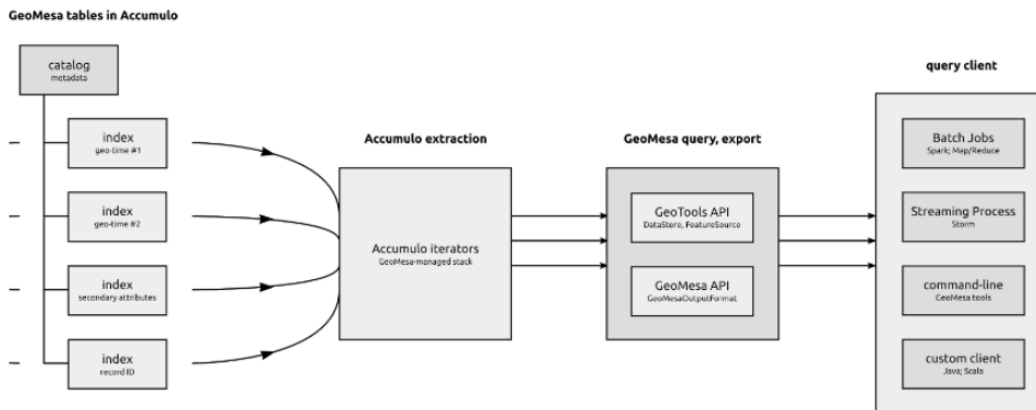


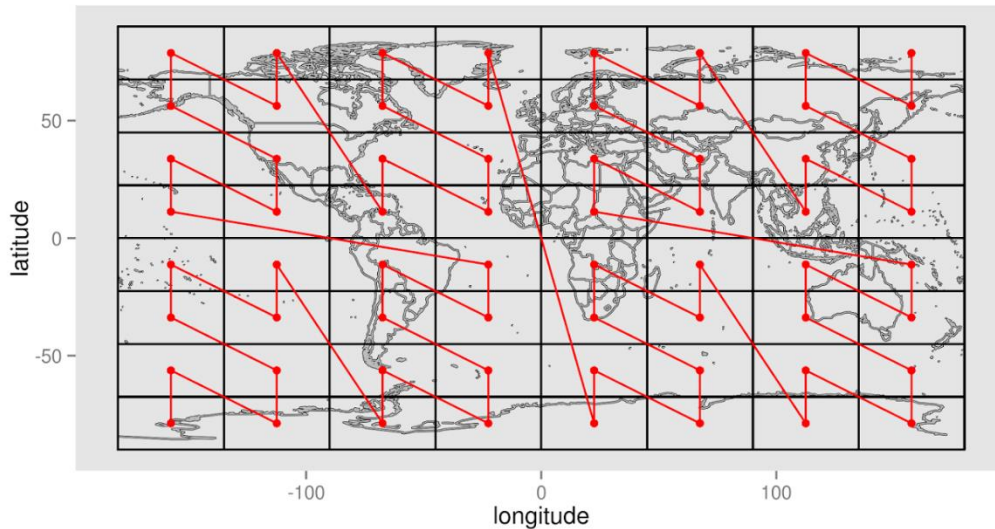
Figura 19. Posible arquitectura para peticiones de datos con GeoMesa

### Almacenamiento clave-valor y Z-curves

Las bases de datos que GeoMesa usa para almacenamiento son bases de datos de clave-valor, un tipo de base de datos NoSQL en el cual cada registro se almacena y recupera usando un identificador único para ese registro conocido como clave. Accumulo, HBase y Google Cloud Bigtable ordenan estas claves y pueden almacenarlas en cualquier número de nodos (servidores).

Cuando usamos una base de datos clave-valor, un buen diseño de las propias claves puede traducirse en aplicaciones más eficientes. A diferencia de las base de datos relacionales, donde las claves son frecuentemente números enteros secuenciales, el almacenamiento clave-valor normalmente usa la clave para representar una característica por la que los datos son consultados frecuentemente. Por ejemplo, imaginemos una base de datos de pedidos de clientes indexada por el número de pedido. De esta forma, cuando un cliente solicita un registro por número de pedido, la base de datos va directamente a esa clave y devuelve como resultado el registro correspondiente a ese pedido.

Esto es una simplificación de cómo funcionan este tipo de base de datos, pero también podemos explicar el principio fundamental de GeoMesa en términos de claves y valores. Para almacenar datos espaciotemporales, necesitamos crear una clave que represente la localización espaciotemporal de un registro. GeoMesa usa este sistema para almacenar localizaciones como puntos distribuidos sobre una línea especial que visita todos los sectores de un mapa, como la línea roja mostrada en la siguiente figura:



**Figura 20. Representación de una Z-curve de GeoMesa**

La curva roja se conoce como Z-curve. Esta línea visita cada celda del mapa exactamente una vez, estableciendo un orden único entre las celdas. Estas curvas pueden trabajar con mayor o menor resolución (tamaño de las celdas) dependiendo de nuestra configuración.

Cada punto de esta curva se le puede asignar un valor secuencial, permitiendo a GeoMesa representar lo que debería haber sido un par latitud-longitud como un simple número entero. Esto es ideal para representar datos bidimensionales con una clave de una única dimensión. Además, estas Z-curves pueden adaptarse para n dimensiones, permitiendo a GeoMesa linealizar n dimensiones de datos en una sola dimensión.

### ***Indexación en GeoMesa***

El principio básico de indexación en GeoMesa es representar las tres dimensiones de longitud, latitud y tiempo con una Z-curve tridimensional, usando los valores de los puntos de esta curva como clave. Esto le permite almacenarlo en un registro de una base de datos clave-valor con una clave que representa las tres dimensiones de datos que se usa con más frecuencia para las peticiones.

La estructura real de la clave es más compleja que un simple par clave-valor. En la siguiente figura podemos ver una representación detallada de la estructura de esta clave cuando usamos Accumulo como sistema de almacenamiento.

KEY						VALUE	
ROW			COLUMN		TIMESTAMP	VIZ	Byte-encoded <b>SimpleFeature</b>
			COLUMN FAMILY	COLUMN QUALIFIER			
Epoch Week 2 bytes	<b>Z3(x,y,t)</b> 8 bytes	Unique ID (such as UUID)	"F"	-	-	Security tags	

Figura 21. Estructura de un registro almacenado por GeoMesa en Accumulo

Esta estructura es igual a la que ya comentamos al hablar de Apache Accumulo destacando el uso de la codificación de la Z-curve tridimensional para el identificador de la fila, y el uso de SimpleFeature (un registro espacial) en el campo valor. La estructura de esta clave puede ser ajustada dependiendo de los datos, pero esta es la que se ofrece por defecto.

## 2.8 Jupyter Notebook

Jupyter Notebook (23) (24) es una aplicación web que nos permite crear y compartir documentos que pueden contener código ejecutable, ecuaciones, visualizaciones y texto. Estos documentos reciben el nombre de notebooks.



Figura 22. Logo Jupyter

Más concretamente, podemos decir que Jupyter es un entorno de computación interactivo que permite a los usuarios editar y ejecutar este tipo de documentos que hemos llamado notebooks.

El principal uso de estos notebooks es el de servir como una herramienta que permite la ejecución interactiva de código fuente, intercalando entre este código explicaciones en forma de texto y resultados (textuales o gráficos). Además, estos notebooks pueden ser compartidos entre usuarios simplemente enviando el archivo que representa el notebook a compartir.

Jupyter combina tres componentes para su correcto funcionamiento: Notebook Web Application, Kernels y notebooks.

### 2.8.1 Notebook Web Application

Se trata de una aplicación web cliente-servidor que permite la edición y ejecución de notebooks a través de un navegador web. Puede ser ejecutada en local, de forma que no necesita acceso a internet, o puede instalarse en un servidor remoto y acceder a través de Internet.

Además de editar y ejecutar notebooks, esta aplicación web nos permite obtener los resultados de la ejecución en formatos de representación enriquecido como HTML, LaTeX, PNG, SVG, PDF, etc. También es posible insertar celdas de texto entre las celdas que contengan código ejecutable para posibles explicaciones sobre este. Esto incluye la inclusión de ecuaciones matemáticas usando sintaxis LaTeX.

Esta aplicación web también cuenta con un *dashboard*, consistente en un panel de control que nos muestra los archivos locales para abrir nuestros notebooks o para los kernels que se están ejecutando.

### 2.8.2 Kernel

Un kernel es un “motor computacional” que ejecuta el código contenido en un notebook. Cada kernel es capaz de ejecutar código en un lenguaje específico y existen kernels para una gran variedad de lenguajes de programación. El kernel por defecto ejecuta código Python. La aplicación web de Jupyter proporciona una forma sencilla de elegir que kernel se usa para cada notebook.

Cuando abrimos un Notebook en la aplicación web, el kernel asociado se ejecuta automáticamente. Cuando ejecutamos el notebook, el kernel lleva a cabo las operaciones de computación y produce los resultados. Cada uno de estos kernels se comunica con la aplicación web y nuestro navegador web enviando mensajes en formato JSON sobre un protocolo de mensajes basados en *WebSockets*.

### 2.8.3 Notebook

Como ya hemos comentado, los notebooks son documentos producidos por la aplicación Jupyter Notebook, los cuales contienen las entradas y salidas de una sesión de computación interactiva y elementos de representación enriquecidos (texto, ecuaciones, imágenes, HTML, gráficas, etc...). Los notebooks son al mismo tiempo documentos legibles por humanos que contienen descripción de los análisis y los resultados, y documentos ejecutables que pueden llevar a cabo análisis de datos.

Cuando ejecutamos la aplicación web de Jupyter en nuestro ordenador o en un servidor remoto, los notebooks son simplemente archivos con extensión “.ipynb”. Esto

nos permite organizar los notebooks en directorios y compartirlos con otros desarrolladores además de tener la posibilidad de usar un software de control de versiones como Git.

## 2.9 Apache Toree

Apache Toree (25) es un *middleware* cuyo objetivo es proporcionar una interfaz a las aplicaciones interactivas para conectarse y usar Spark.



Figura 23. Logo Apache Toree

El proyecto intenta proporcionar a las aplicaciones la habilidad de enviar a Apache Spark paquetes JAR o fragmentos de código para su ejecución. Apache Toree implementa el último protocolo de mensajes de Jupyter por lo que puede ser fácilmente enlazado con nuestro ecosistema Jupyter para realizar exploraciones de datos rápidas e interactivas.

Cuando está enlazado con Jupyter u otra aplicación similar, Apache Toree hace las funciones de kernel, intermediando entre Jupyter y nuestro cluster de Spark. Como puede verse en la figura que se muestra a continuación, Apache Toree mantiene un *SparkContext* en lugar de mantenerlo la aplicación cliente que hace uso de Toree. Este *SparkContext* se crea por defecto al iniciarse el kernel Apache Toree sin necesidad de indicárselo.

Cuando el cliente solicita que se evalúe cierta función de Spark, esta petición se envía realmente al kernel Apache Toree y este la reenvía al cluster de Spark, devolviendo los resultados a la aplicación cliente. Una vez visto los resultados, el usuario puede modificar el código y volver a realizar la consulta sin necesidad de crear un nuevo *SparkContext*.

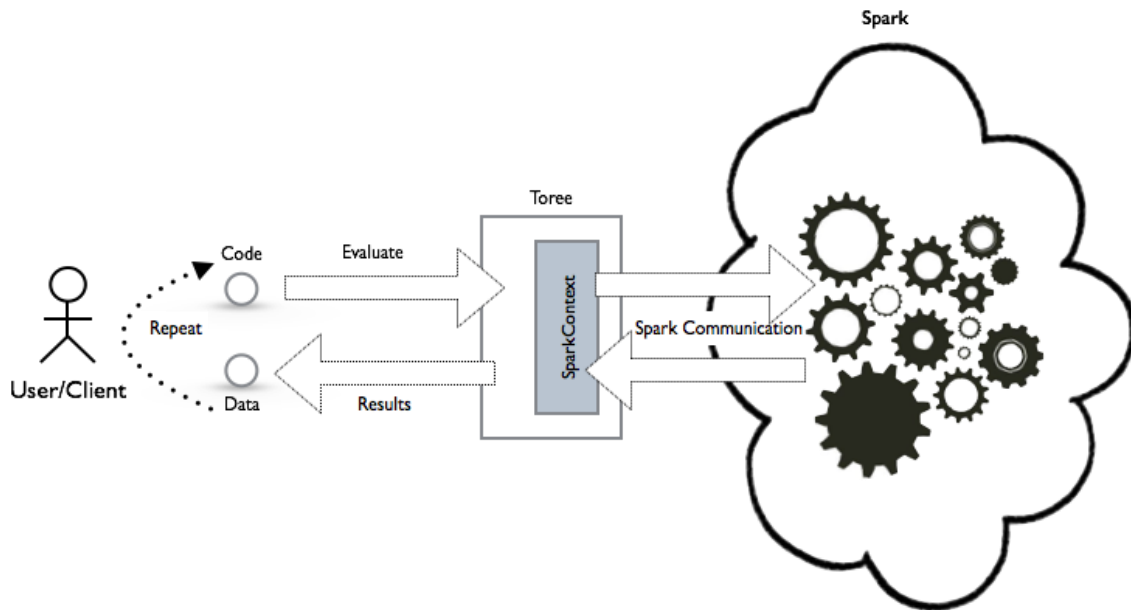


Figura 24. Interacción aplicación cliente - Apache Toree -Apache Spark

## 2.10 Docker

Docker (26) (27) es la plataforma líder en contenedores software. Los desarrolladores usan Docker para eliminar los problemas del tipo “en mi ordenador funcionaba” cuando colaboran en la elaboración de código con compañeros. Los operadores usan Docker para ejecutar y gestionar aplicaciones en contenedores aislados. Las empresas usan Docker para construir procesos de desarrollo de software ágiles para enviar nuevas características de forma más rápida, más segura y más confiable tanto para aplicaciones Linux como Windows.

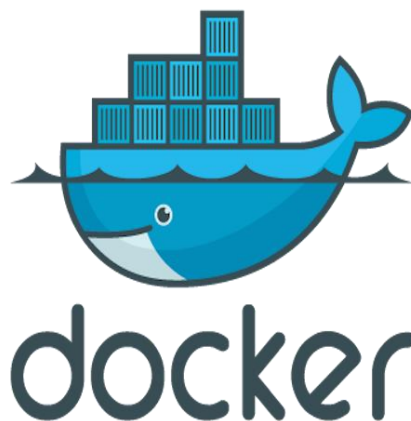


Figura 25. Logo Docker

Más concretamente, Docker es un proyecto open-source que automatiza el despliegue de aplicaciones dentro de contenedores software. Docker proporciona una capa adicional de abstracción y automatización de la virtualización a nivel de sistema operativo en Linux, Mac OS y Windows. Docker usa las características de aislamiento

de recursos del kernel de Linux, tales como cgroups y namespaces, y un sistema de archivos que permite a contenedores independientes ejecutarse en una única instancia de Linux, evitando la sobrecarga de iniciar y mantener máquinas virtuales.

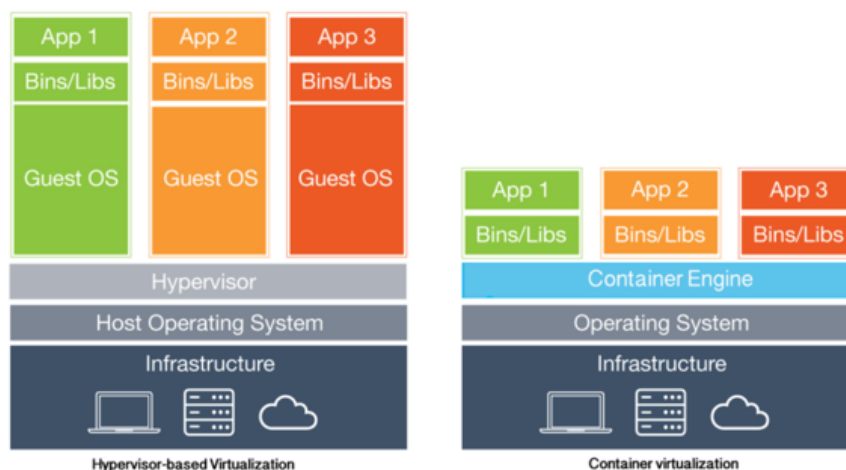


Figura 26. Arquitectura de virtualización basada en hypervisor y basada en contenedores

En la figura anterior vemos una comparativa entre las arquitecturas de visualización basadas en *hypervisor* (máquinas virtuales) y basadas en contenedores.

Las máquinas virtuales (VMs) son una abstracción del hardware físico que convierte un servidor en varios. El *hypervisor* permite ejecutar varias máquinas virtuales en un único servidor. Cada máquina virtual incluye una copia completa de un sistema operativo, una o más aplicaciones, los binarios necesarios y bibliotecas, ocupando decenas de GBs de almacenamiento. Además, suelen demorar más a la hora del arranque.

Los contenedores por su parte son una abstracción en la capa de aplicación que empaqueta el código y las dependencias juntos. Podemos ejecutar múltiples contenedores en la misma máquina y compartir el kernel de nuestro sistema operativo con otros contenedores, cada uno de ellos ejecutándose como procesos aislados en el espacio de usuario. Los contenedores ocupan menos espacio de almacenamiento que las VMs (las imágenes de contenedores normalmente ocupan decenas de MBs), y su arranque es casi inmediato.

Para crear una imagen de Docker, se usan ficheros de descripción de imagen llamados Dockerfiles. Estos ficheros de descripción toman como base una imagen creada previamente y describen las modificaciones a realizar sobre esta para obtener la nueva imagen. La ventaja de usar Dockerfiles es que existe un proceso de compilación automática que nos asegura que tenemos la última versión disponible. Esto es bueno



desde la perspectiva de la seguridad ya que nos permite asegurarnos de que no estamos instalando software vulnerable.

## 2.11 Ansible

Ansible (28) (29) es una plataforma de automatización open-source que destaca por su sencillez de configuración y su gran utilidad. Ansible puede ayudarnos con la gestión de configuración, despliegue de aplicaciones y automatización de tareas. También permite realizar tareas de orquestación en infraestructuras IT, donde tenemos que ejecutar tareas en secuencia y crear una cadena de eventos los cuales deben ser llevados a cabo en diferentes servidores o dispositivos.



Figura 27. Logo Ansible

Un ejemplo podría ser si tenemos un grupo de servidores web tras un balanceador de carga. Ansible puede actualizar los servidores web uno por uno y mientras se actualiza, puede eliminar el servidor web del balanceador de carga y deshabilitarlo en nuestro sistema de monitorización Nagios.

Al contrario que otras plataformas de automatización, Ansible no usa un agente en el host remoto. En su lugar, Ansible usa SSH el cual se asume que está instalado en todos los sistemas que queremos gestionar. Además, está escrito en Python el cuál debe ser instalado en los hosts remotos. Esto significa que no tenemos que instalar nada en los sistemas que queremos gestionar antes de usar Ansible, podemos ejecutar Ansible en cualquiera de nuestras máquinas y, desde el punto de vista del cliente, no existe conocimiento de ningún servidor Ansible. Existen otros requisitos, por ejemplo, si queremos hacer algo relacionado con Git en un host remoto, debemos instalar previamente Git en nuestro host remoto, pero esto puede llevarse a cabo por medio de Ansible.

### 2.11.1 Ansible Playbooks

La verdadera fortaleza de Ansible reside en sus Playbooks. Un playbook es como una receta o un manual de instrucciones que le dice a Ansible que hacer cuando se conecta a cada host. Los Playbooks están escritos en YAML, que es un lenguaje del tipo

de XML, pero legible por humanos y con la ventaja de tener una curva de aprendizaje muy exponencial.

Podríamos tener un playbook que configurase nuestros servidores en base a una configuración base, de forma que todos usen una configuración sshd correcta y autenticación central. Además, podríamos usar roles para grupos específicos de servidores (por ejemplo, un grupo para servidores web, otro para servidores de base de datos y otro para servidores de monitorización).



### 3 Arquitectura del entorno

La finalidad de este capítulo es dar una visión general de la arquitectura del entorno de procesamiento de datos geográficos que hemos desarrollado para este Trabajo Fin de Master formada por los componentes que hemos presentado en el capítulo anterior. Así mismo, también se pretende añadir una capa a esta visión general que nos muestre como los distintos componentes interactúan entre sí para ofrecer a los usuarios la funcionalidad del entorno.

Como se puede apreciar si nos fijamos en el índice de contenidos de este documento, el desarrollo inicial de nuestro entorno de procesamiento de datos geográficos se ha llevado a cabo en una única instancia de Amazon EC2, concretamente una instancia creada a partir de una AMI de Ubuntu 16.04. Una vez se terminó el desarrollo inicial y la fase de pruebas en la que ejecutamos algunos ejemplos para probar el correcto funcionamiento del entorno, se llevó a cabo una fase de mejora en que se automatizó el despliegue de workers de Spark para conseguir una mejora instantánea de nuestra capacidad de cómputo.

Centrándonos ya en la parte central de nuestro entorno, pasamos a describir los componentes instalados sobre Ubuntu 16.04 en la instancia de Amazon EC2. Para facilitar la explicación y comprensión de nuestro entorno, hemos diseñado el diagrama mostrado en la figura que se adjunta a continuación.

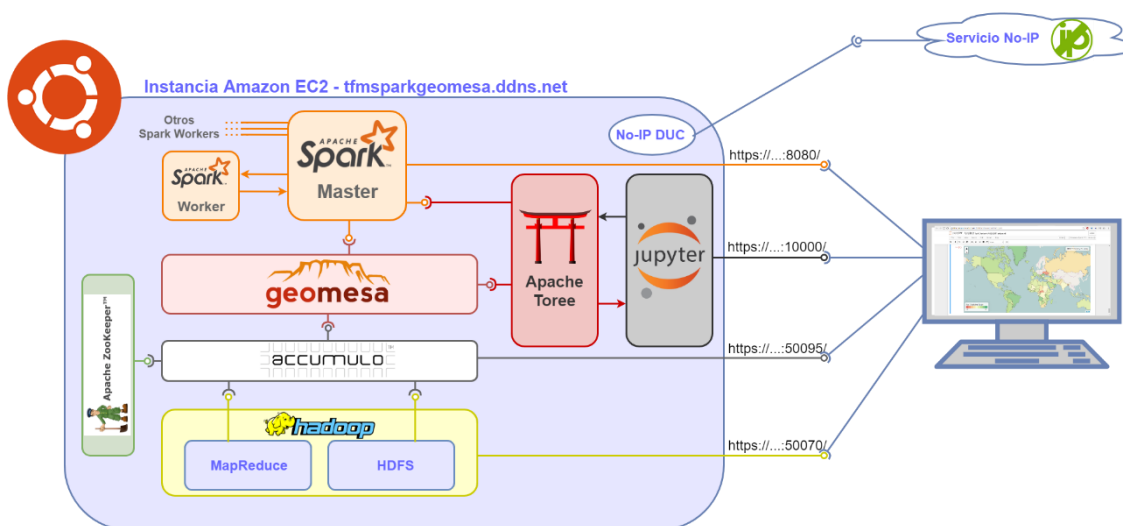


Figura 28. Esquema general de arquitectura del entorno

En esta figura se aprecian los componentes instalados en nuestra instancia de Amazon EC2 y las relaciones de uso entre ellos. Para comenzar podemos fijarnos en el único componente aislado del resto, No-IP DUC. Se trata de un cliente de actualización

automática de DNS. Se encarga de actualizar la IP asociada al nombre DNS que le hemos dado a la instancia (tfmsparkgeomesa.ddns.net) haciendo uso del servicio gratuito No-IP.

Uno de los ejes centrales de nuestro entorno es GeoMesa, ya que como comentamos en el capítulo anterior, nos brinda la funcionalidad de indexar nuestros datos geográficos usando índices tridimensionales convertidos a una sólo dimensión, lo que nos permite realizar consultas muy rápidas. Como también comentamos en el capítulo anterior, GeoMesa hace uso de Apache Accumulo como base de datos NoSQL, por lo que existe comunicación entre el framework GeoMesa y la base de datos Accumulo.

Accumulo, al mismo tiempo utiliza como sistema de ficheros Hadoop HDFS y cuenta con una funcionalidad para importar datos haciendo uso de Hadoop MapReduce. Accumulo también hace uso de Apache Zookeeper como coordinador de sus distintos componentes y nodos. Por tanto, en nuestro entorno, Accumulo hace uso de las interfaces de Apache Zookeeper, Hadoop HDFS y MapReduce.

Como ya sabemos, parte de las librerías de GeoMesa nos ofrecen la funcionalidad de aplicar funciones de Apache Spark a nuestros datos geográficos, por lo que Spark y GeoMesa interactúan entre ellos. El master de Spark a su vez, tendrá un worker ejecutándose en nuestra instancia y un número indefinido que se ejecutará en otras instancias.

Los usuarios de nuestro entorno podrán usar los notebooks de Jupyter para ejecutar código Spark Geomesa interactivo. Los notebooks de este tipo se ejecutarán sobre un kernel Apache Toree que hará uso de las APIs de GeoMesa o de Spark directamente.

Como se aprecia en la figura anterior, los componentes Hadoop, Accumulo, Spark y Jupyter cuentan con una interfaz web a la que tenemos acceso a través de Internet usando un navegador web.

Una vez habíamos completado la implementación del entorno en nuestra instancia de EC2 y las pruebas asociadas, se decidió realizar una mejora en este, ya que no tiene mucha utilidad ejecutar Spark con un solo worker. Es por ello que se decide implementar un método de despliegue automático de workers de Spark. En la siguiente figura se muestra un diagrama que describe el funcionamiento de este método.

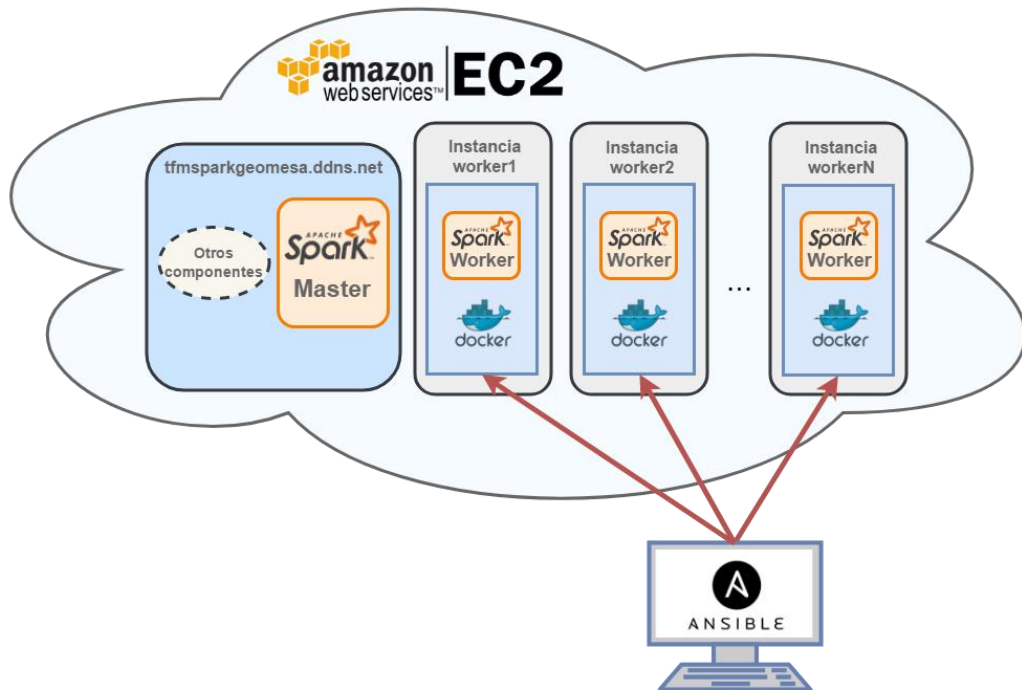


Figura 29. Diagrama automatización despliegue de workers de Spark

Hemos creado una imagen de Docker a partir de la cual es posible crear un contenedor que ejecuta un worker de Spark que se conecta automáticamente al master de nuestro entorno. Para desplegar con más facilidad el contenedor Docker en nuestras instancias de Amazon EC2, hemos creado un playbook de Ansible que instala todo lo necesario para poder ejecutar en cualquier instancia un contenedor Docker, descarga nuestra imagen y ejecuta un contenedor a partir de ella.

Con esto concluye la visión general de nuestro entorno. A partir de aquí, los siguientes son capítulos más técnicos en los que explicamos el proceso de despliegue de nuestro entorno, el método para importar datos con GeoMesa, los distintos métodos de ejecución que existen y, también, explicamos con más profundidad la automatización del despliegue de workers.



## 4 Implementación del entorno

Una vez terminada la fase de investigación de este proyecto (gran parte de esta se realizó en paralelo a la fase de desarrollo), comenzamos con este capítulo la fase de desarrollo llevada a cabo durante el transcurso de este proyecto.

### 4.1 Adquisición de recursos de computación en AWS

Durante esta sección, describiremos los pasos necesarios para lanzar una instancia de Amazon EC2 que usaremos de base para desplegar el resto de herramientas que componen nuestro entorno de ejecución.

Inicialmente se decidió desplegar todas las herramientas en una misma instancia ya que el procedimiento para desplegarlos en varias instancias es prácticamente el mismo y no podíamos disponer de muchas instancias de Amazon EC2 debido a nuestra limitación de presupuesto.

Esto conllevó que tuviésemos que elegir para nuestra tarea una instancia de mayor potencia de cómputo, en concreto una instancia del tipo t2.large cuyas especificaciones son: 2 CPUs virtuales, 8 GiB de memoria Ram y almacenamiento EBS. El precio de este tipo de instancias bajo demanda en la región “UE (Frankfurt)” es de \$0.108 por hora, como podemos observar en la siguiente tabla de precios (30) para instancias que ejecutan un sistema operativo Linux en dicha región:

	vCPU	ECU	Memoria (GiB)	Almacenamiento de la instancia (GB)	Uso de Linux/UNIX
<b>Uso general – Generación actual</b>					
t2.nano	1	Variable	0.5	Solo EBS	\$0.0068 por hora
t2.micro	1	Variable	1	Solo EBS	\$0.014 por hora
t2.small	1	Variable	2	Solo EBS	\$0.027 por hora
t2.medium	2	Variable	4	Solo EBS	\$0.054 por hora
t2.large	2	Variable	8	Solo EBS	\$0.108 por hora
t2.xlarge	4	Variable	16	Solo EBS	\$0.216 por hora
t2.2xlarge	8	Variable	32	Solo EBS	\$0.432 por hora
m4.large	2	6.5	8	Solo EBS	\$0.129 por hora
m4.xlarge	4	13	16	Solo EBS	\$0.257 por hora
m4.2xlarge	8	26	32	Solo EBS	\$0.513 por hora
m4.4xlarge	16	53.5	64	Solo EBS	\$1.026 por hora
m4.10xlarge	40	124.5	160	Solo EBS	\$2.565 por hora

Figura 30. Precios y características instancias Amazon EC2



En nuestro caso, el factor determinante a la hora de elegir el tipo t2.large fue la memoria RAM. Evidentemente, para entornos de producción en los que se ejecuten aplicaciones que procesen grandes cantidades de datos geográficos no nos vale con una sola instancia, sino que debemos desplegar un cluster para poder aprovechar toda la potencia de Spark.

#### 4.1.1 Creación y configuración de una instancia EC2

Existen diferentes métodos para lanzar una instancia de Amazon EC2 (31). En nuestro caso, el método elegido es lanzarla desde la interfaz web de AWS usando una AMI proporcionada por AWS.

Lo primero que debemos hacer es identificarnos con nuestra cuenta de AWS para acceder a la consola y pulsar en la esquina superior izquierda sobre “Services”, de esta forma se nos mostrará una lista de los servicios ofrecidos por AWS además de un buscador para agilizar nuestra tarea. El servicio que nos ocupa en este momento es EC2, por lo que lo seleccionamos.

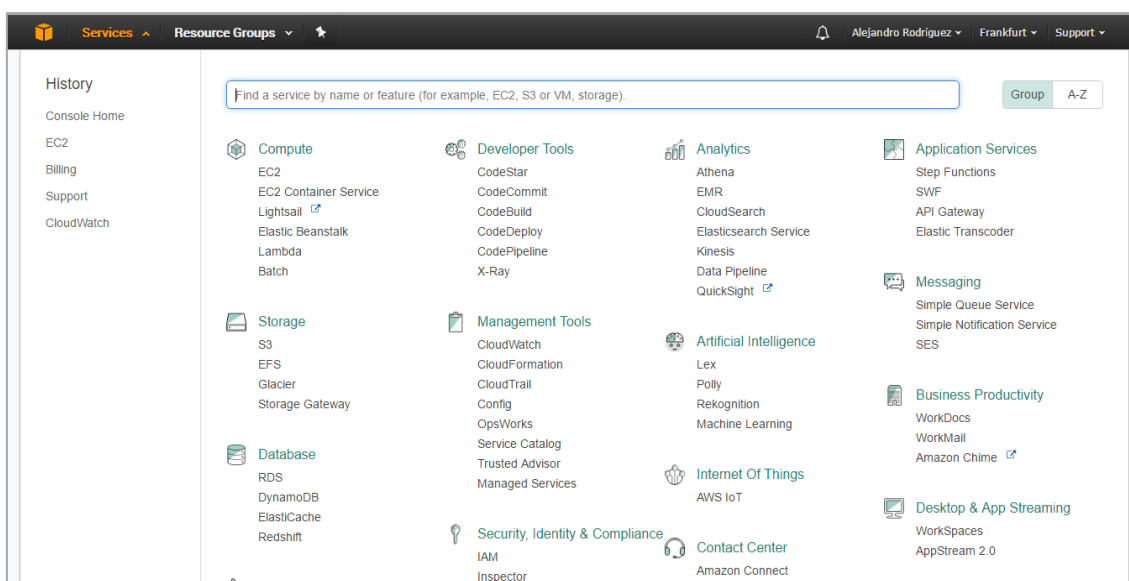


Figura 31. Listado de servicios de AWS

A continuación, se nos mostrará la consola de Amazon EC2 en la que vemos un listado de recursos de EC2 que estamos usando en la región actual. La región actual se muestra, y puede ser cambiada, en la barra superior a la derecha, junto a nuestro nombre. Como podemos observar, en el panel lateral izquierdo se nos muestran los enlaces a las utilidades más interesantes de EC2.

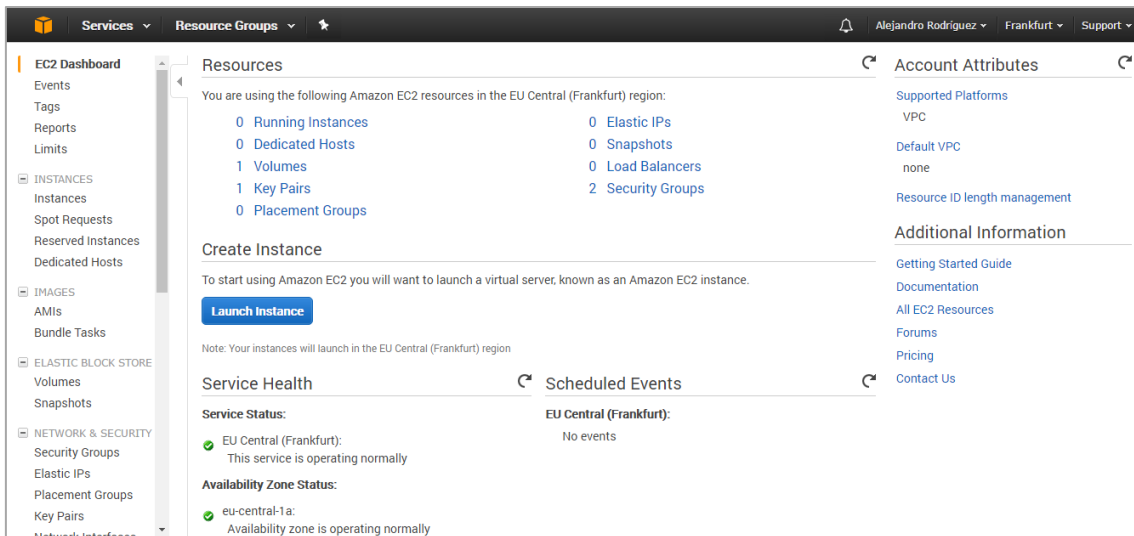


Figura 32. Consola de Amazon EC2

Si pulsamos sobre “Instances” nos aparece un listado de las instancias que hayamos creado hasta el momento y tendremos la oportunidad de cambiar el estado de nuestras instancias, ver la información asociada a nuestras instancias, o lanzar una nueva. Precisamente esto último es lo que nos ocupa durante esta subsección por lo que el siguiente paso será pulsar sobre “Launch Instance”.

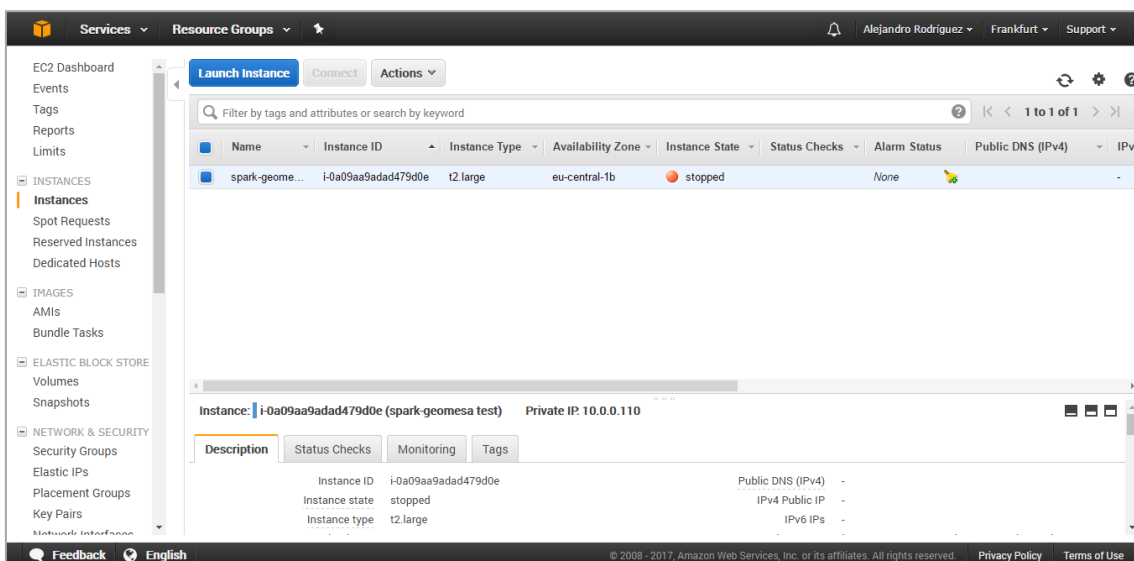


Figura 33. Listado de instancias EC2 creadas en nuestra cuenta

Al pulsar sobre “Launch Instance”, se nos abrirá una ventana donde pasaremos a configurar nuestra nueva instancia de Amazon EC2. El primer paso es seleccionar en el panel izquierdo el tipo de AMI que queremos usar. Como se puede observar en la siguiente figura, las opciones disponibles son:

- **Quick Start.** Una selección de AMIs populares para su rápido acceso.
- **My AMIs.** AMIs privadas de nuestra propiedad o compartidas con nosotros.

- **AWS Marketplace.** Tienda online donde podemos adquirir AMIs.
- **Community AMI.** AMIs privadas que han compartido la comunidad de miembros de AWS.

En nuestro caso, seleccionaremos una AMI del tipo “Quick Start”, concretamente la AMI con nombre “Ubuntu Server 16.04 LTS (HVM)”, que aparece en la parte baja de la siguiente figura.

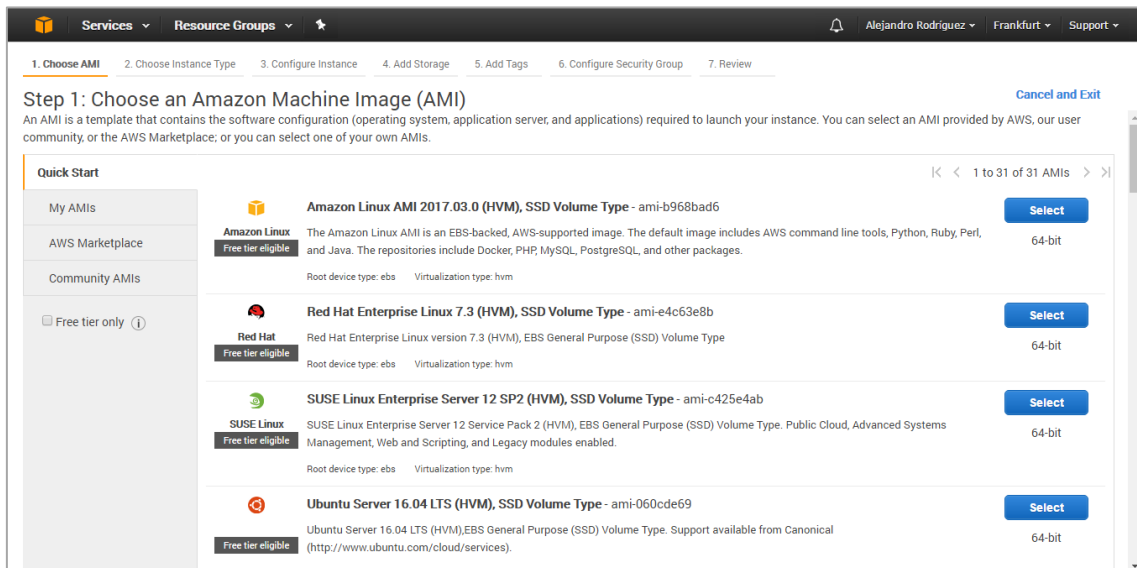


Figura 34. Listado de AMIs disponibles - EC2

Seguidamente, se nos muestra la página con el segundo paso: seleccionar un tipo de instancia. Como se puede observar en la siguiente figura, el único tipo seleccionable para la capa gratuita es la instancia t2.micro que cuenta con una vCPU y 1 GiB de RAM. Como ya hemos comentado, este tipo de instancia no es suficiente para nuestro proyecto y hemos usado una instancia del tipo t2.large (2 vCPU, 8 GiB RAM).

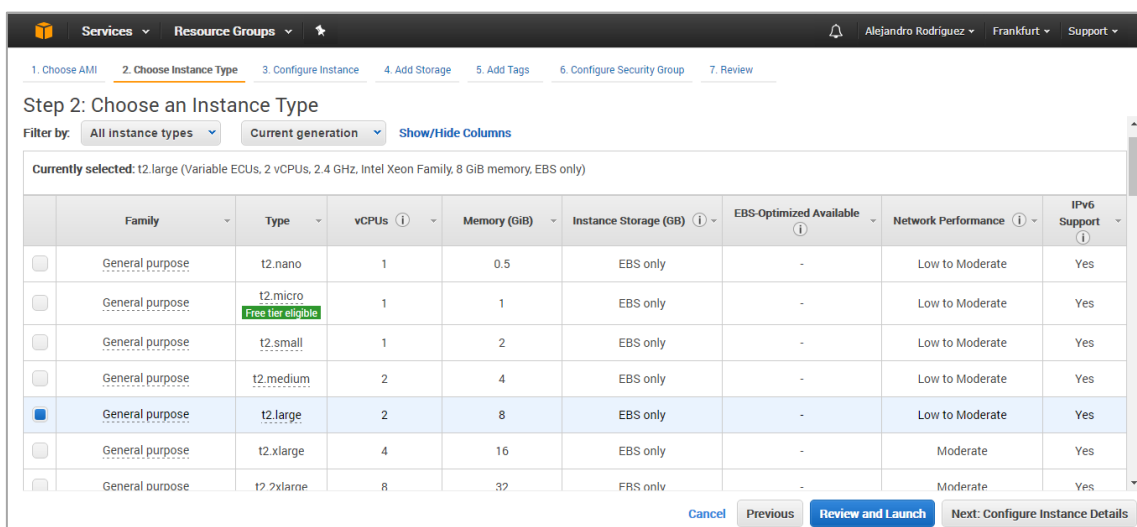


Figura 35. Tipos de instancias EC2

Una vez seleccionado el tipo de instancia, pulsamos en “Next: Configure Instance Details”. Una vez hecho esto, nos aparecerá la ventana mostrada en la siguiente figura. En ella tenemos la posibilidad de especificar la configuración de nuestra instancia. Los campos más relevantes son los siguientes:

- **Number of Instances.** Podemos desplegar simultáneamente varias instancias iguales.
- **Network.** Podemos seleccionar a que red virtual se conecta nuestra instancia.
- **Subnet.** En este campo indicamos la subred interna a la que queremos conectarnos.
- **Shutdown behaviour.** Comportamiento de nuestra instancia al apagarse.
- **Network interfaces.** Podemos crear y configurar interfaces de red virtuales para nuestra instancia.

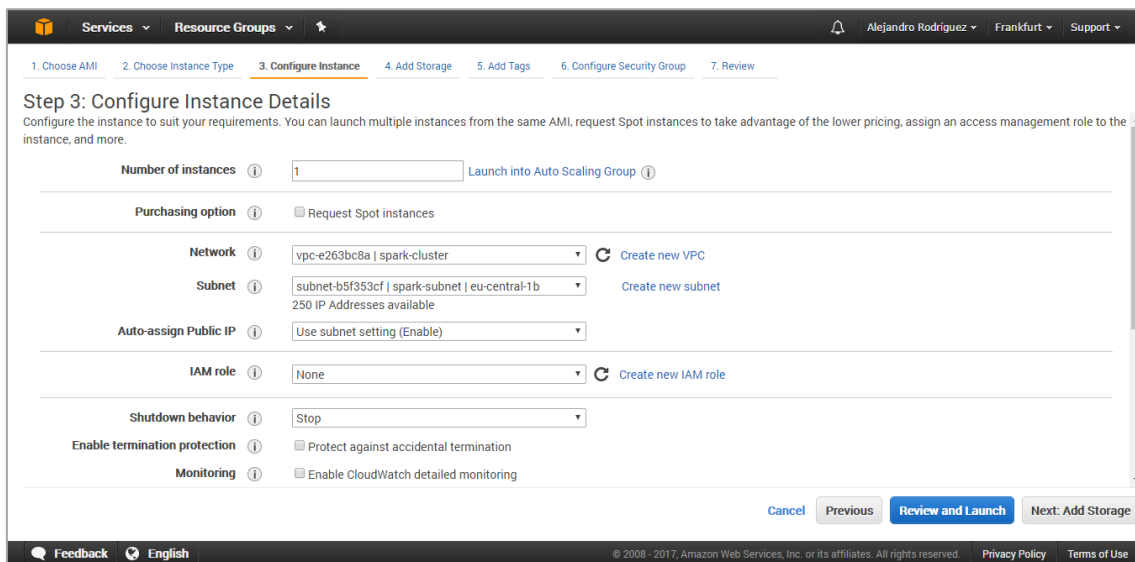


Figura 36. Configuración instancia EC2

Una vez tenemos la configuración básica de nuestra instancia, pulsamos sobre “Next: Add Storage” y se nos abrirá la página de configuración del almacenamiento para nuestra instancia. En nuestro caso, ha sido necesario un volumen de 30 GiB de espacio y del tipo “General Purpose SSD (GP2)”. Inicialmente se creó un volumen de 16 GiB, que llegados a un punto del despliegue nos fue insuficiente, por lo que tuvimos que aumentar el espacio de este volumen mediante el procedimiento que se detalla en la documentación de EC2. Una vez configuremos nuestro almacenamiento, pulsamos en “Next: Add Tags”.

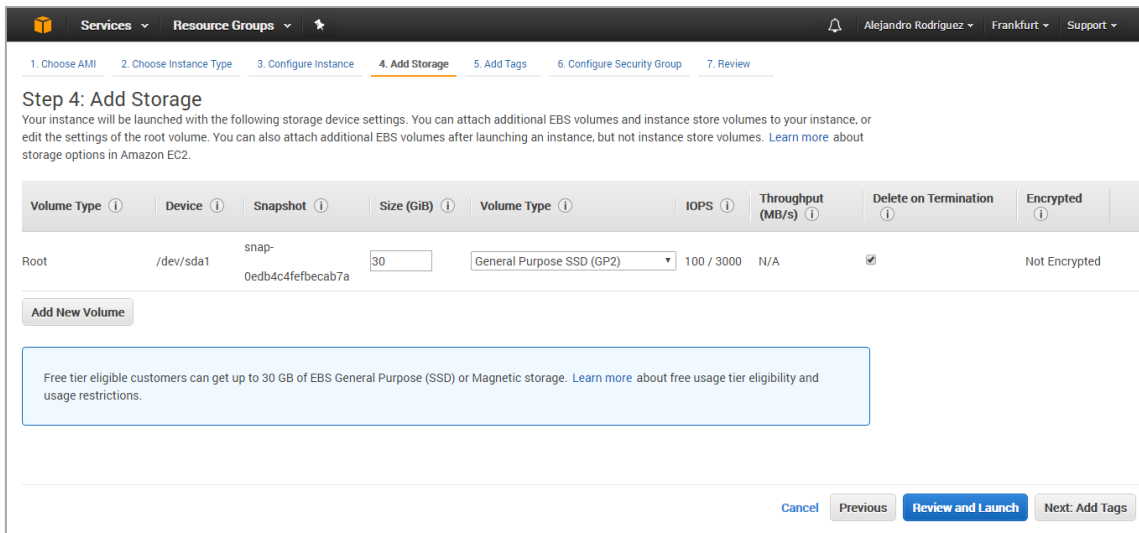


Figura 37. Configuración almacenamiento instancia EC2

En la siguiente pantalla se nos ofrece la posibilidad de etiquetar nuestra instancia. Si pulsamos sobre “Add Tag”, añadimos una etiqueta clave-valor que nos puede servir para identificar nuestra nueva instancia entre otras.

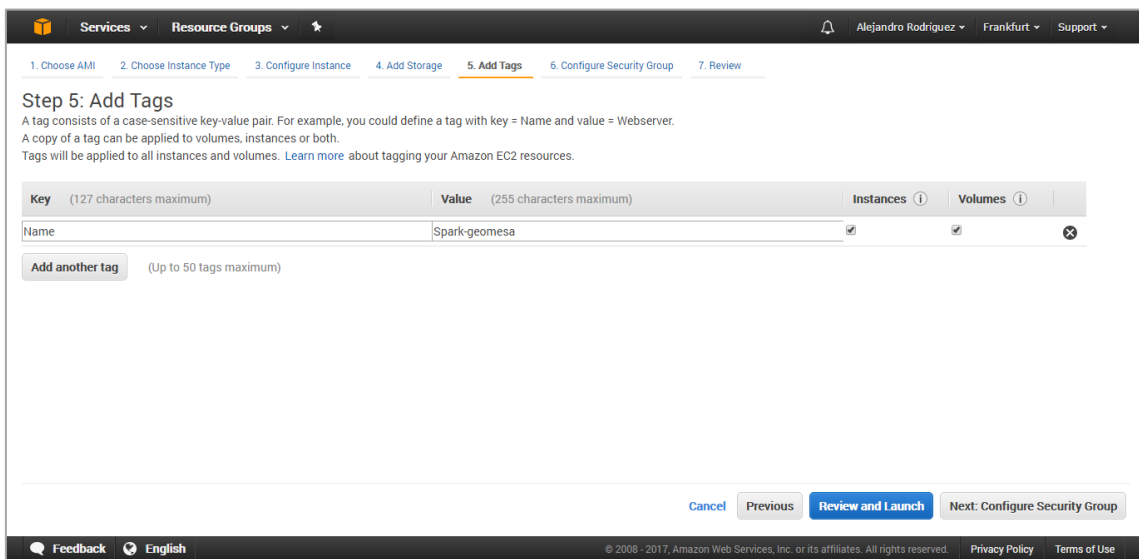


Figura 38. Añadir etiquetas instancia EC2

En el último paso previo a la creación de la instancia, deberemos configurar el grupo de seguridad, es decir, un conjunto de reglas que nos permiten especificar que puertos de la instancia estarán accesibles y desde donde. Tenemos la posibilidad de seleccionar un grupo existente o crear uno nuevo especificando las reglas. Este grupo de seguridad se puede modificar con posterioridad, por lo que es mejor añadir las reglas según sean necesarias.

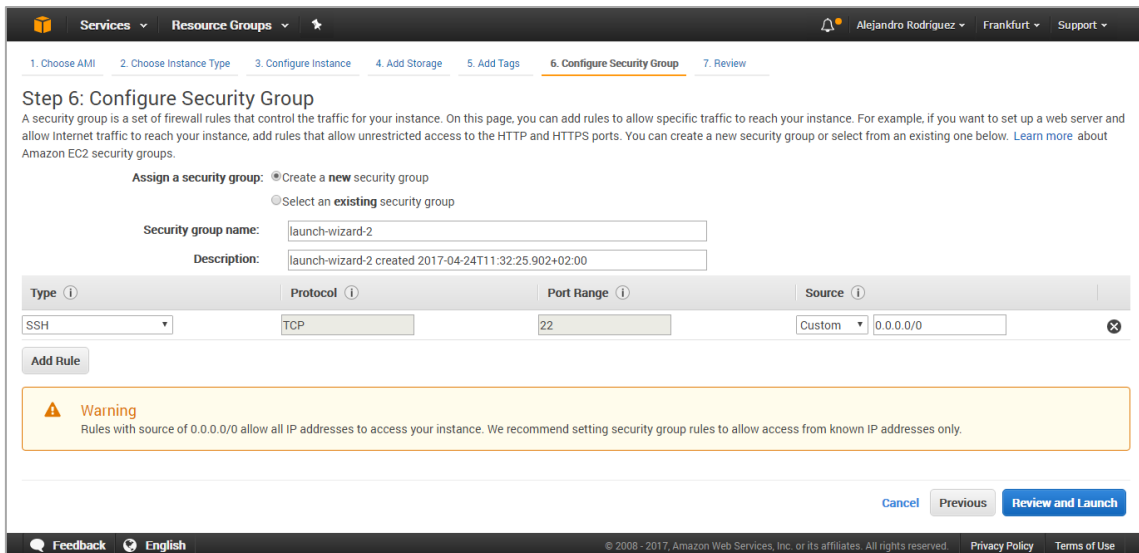


Figura 39. Creación de un grupo de seguridad EC2

Para finalizar, pulsamos sobre “Review and Launch” y se nos mostrará un resumen de la configuración realizada para nuestra nueva instancia, el cual podemos observar en la siguiente figura.

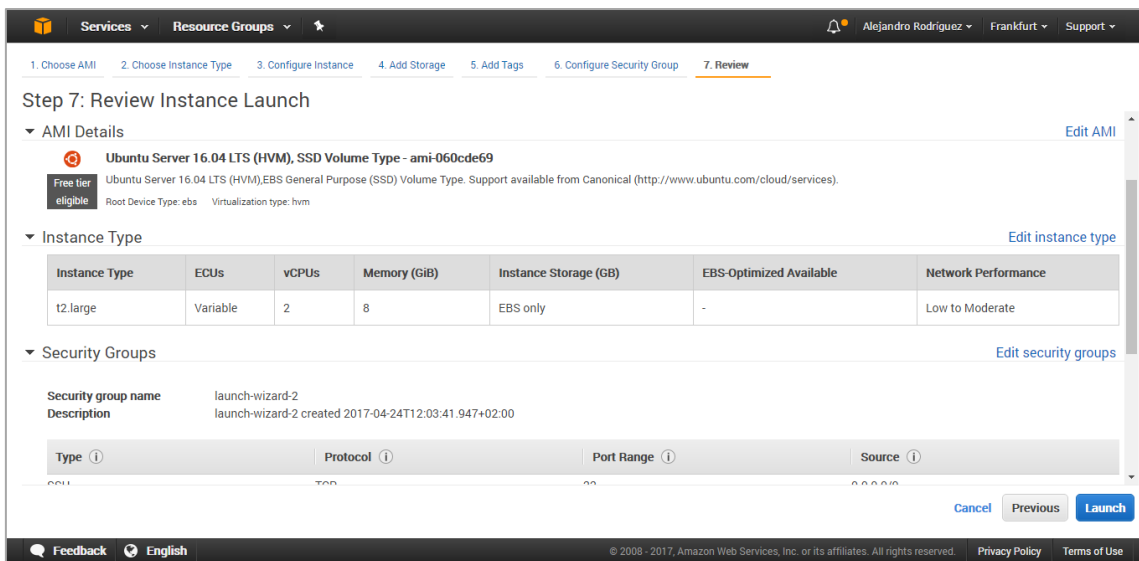


Figura 40. Resumen configuración instancia EC2.

Sobre este resumen debemos comprobar que toda la configuración mostrada es correcta y una vez verificada, pulsamos en “Launch” para finalizar el proceso de creación de la instancia, aunque previamente nos saldrá un cuadro emergente que nos pedirá que seleccionemos un par de claves existentes o que creamos uno nuevo. Este par de claves consiste en una clave pública que almacena AWS y una clave privada que almacenamos nosotros, y que nos servirán para conectarnos a nuestra instancia por SSH.

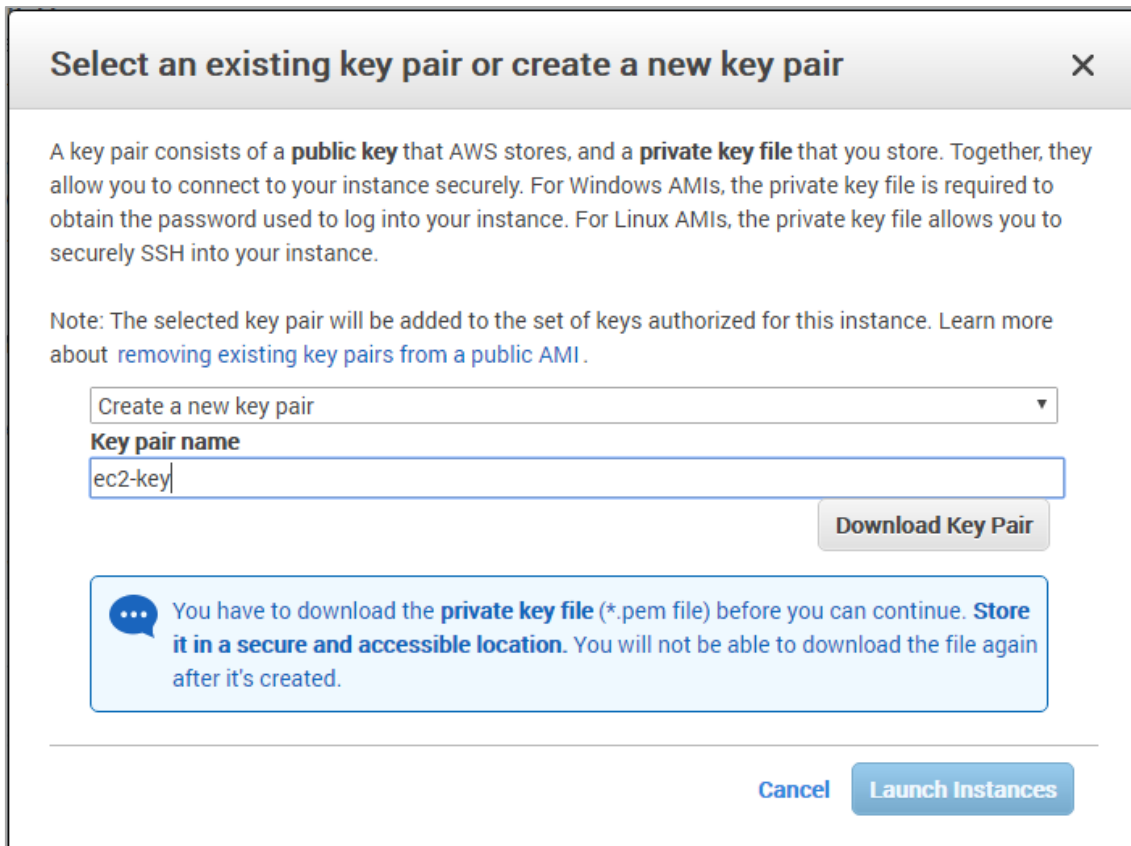


Figura 41. Key pair instancia EC2

Debemos seleccionar un “key pair” existente o crear uno nuevo dándole un nombre y pulsando en “Download Key Pair”. Es muy importante que guardemos bien el archivo con extensión “pem”. Para finalizar pulsamos en “Launch Instance”.

#### 4.1.2 Conexión con la instancia

Una vez lanzado nuestra instancia, podemos conectarnos a ella y usarla. Justo al lanzar la instancia, el estado de esta es “pending”. Cuando nuestra instancia pasa al estado “running”, ya habrá comenzado a arrancarse. Quizás tengamos que esperar un pequeño periodo de tiempo antes de poder conectarnos a esta. Nuestra instancia recibirá un nombre DNS público para que podamos conectarnos a ella a través de internet y, además, recibirá un nombre DNS privado para que otras instancias dentro de nuestra red privada virtual puedan conectarse a ella.

Durante el desarrollo de este TFM se han llevado a cabo dos métodos diferentes para la conexión con las instancias de EC2: un método para conectarnos desde Ubuntu 16.04 y otro para conectarnos desde Windows 10.

##### *Conexión desde Windows 10*

A continuación, vamos a detallar las instrucciones para conectarnos a nuestra instancia de EC2 desde Windows usando PuTTY (32), un cliente SSH gratuito para Windows.

Como requisitos previos tenemos los siguientes:

- Instalar el programa PuTTY, el cual es gratuito y puede descargarse desde su página web.
- Obtener el nombre DNS de nuestra instancia. Se puede obtener en la consola de EC2, accediendo a la descripción de nuestra instancia.
- Tener localizado la clave secreta (el archivo con extensión “.pem”) que descargamos al crear nuestra instancia.
- Asegurarnos de que nuestro grupo de seguridad permite el tráfico SSH entrante a nuestra instancia.

El cliente PuTTY no soporta de forma nativa el formato de nuestra clave privada (.pem) generada por EC2. Sin embargo, PuTTY tiene una herramienta llamada PuTTYgen que puede convertir nuestra clave privada al formato requerido por PuTTY (.ppk). Por tanto, lo primero que vamos a hacer es convertir el formato de nuestra clave privada.

El primer paso es iniciar la herramienta PuTTYgen. Para ello, vamos al menú de inicio y hacemos uso del buscador para encontrar la herramienta. Nos aparecerá la ventana que se muestra en la siguiente figura.

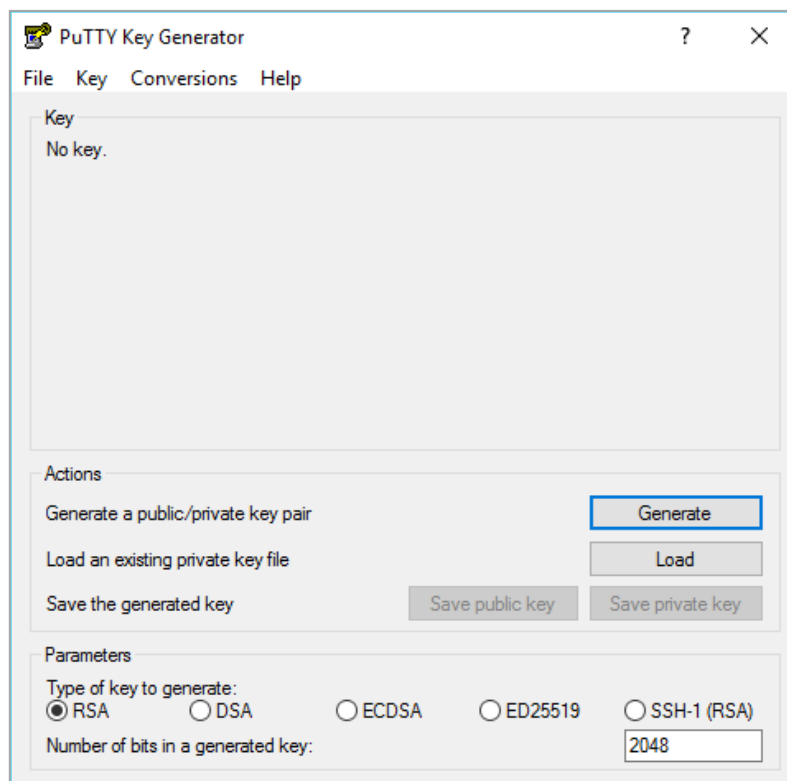


Figura 42. Interfaz de la herramienta PuTTYgen



En el tipo de clave a generar, debemos seleccionar RSA, como ya está seleccionado en la figura anterior. Además, debemos indicar que queremos generar 2048 bits.

El siguiente paso es pulsar en “Load” y seleccionar nuestra clave privada generada por EC2 al crear nuestra instancia. Por defecto, PuTTYgen sólo muestra los archivos con extensión “.ppk”, por lo que debemos indicarle que nos muestre todos los tipos de archivos. Seleccionamos nuestra clave privada y pulsamos en “Open”. Nos aparecerá un cuadro de diálogo que nos indica que la clave se ha cargado correctamente.

Por último, pulsamos en “Save private key”, elegimos un nombre para nuestra clave privada y pulsamos en “Save”. Con esto ya tenemos nuestra clave privada en el formato que precisa el software PuTTY.

Una vez tenemos nuestra clave privada en el formato correcto, podemos pasar a configurar PuTTY para realizar correctamente la conexión con nuestra instancia EC2. Lo primero es iniciar nuestro cliente PuTTY realizando una búsqueda en el buscador del menú inicio. La interfaz del cliente PuTTY es la mostrada en la siguiente figura:

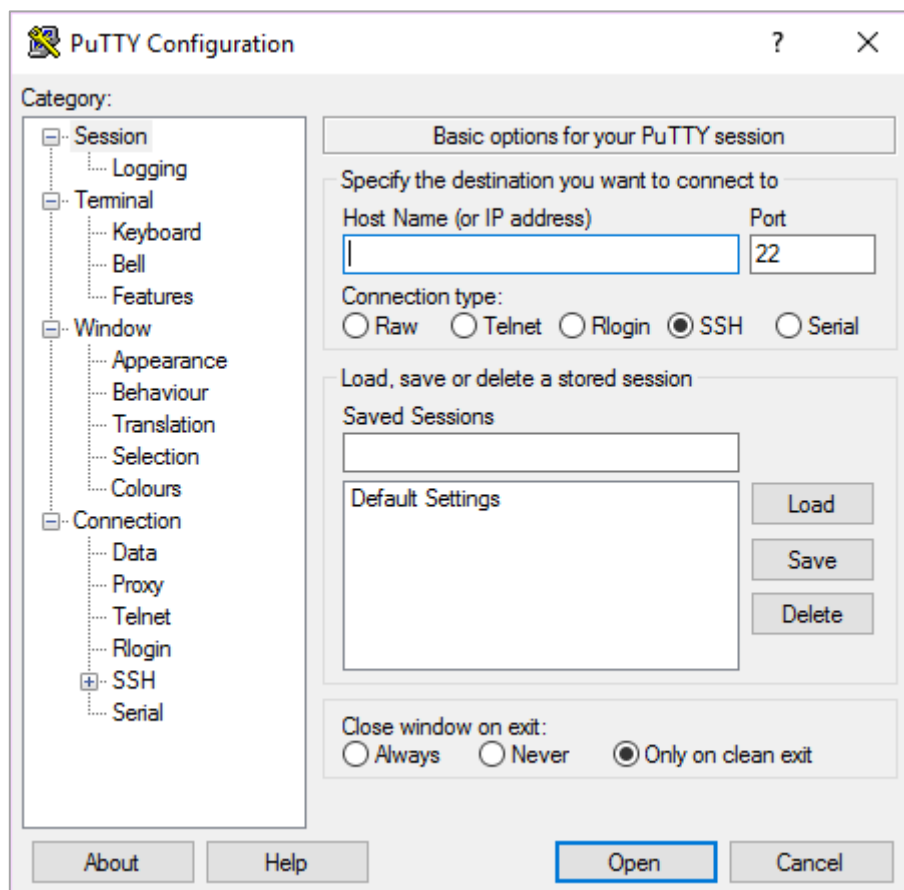


Figura 43. Interfaz gráfica del cliente PuTTY

En la categoría “Session”, que es la que se muestra por defecto, debemos introducir nuestro “Host Name” en la forma “nombre\_de\_usuario@nombre\_dns\_publico”, donde

el nombre de usuario por defecto para la AMI que hemos usado es “ubuntu”. También debemos asegurarnos de que el puerto sea el 22 y que el tipo de conexión seleccionado sea SSH.

Una vez tenemos lista la configuración de la categoría “Session”, debemos acceder mediante el panel lateral a la categoría “Connection → SSH → Auth”. En esta categoría debemos seleccionar nuestra clave privada en el formato “.ppk”. Para ello pulsamos en “Browse” y navegamos en nuestro sistema de directorios hasta nuestra clave privada.

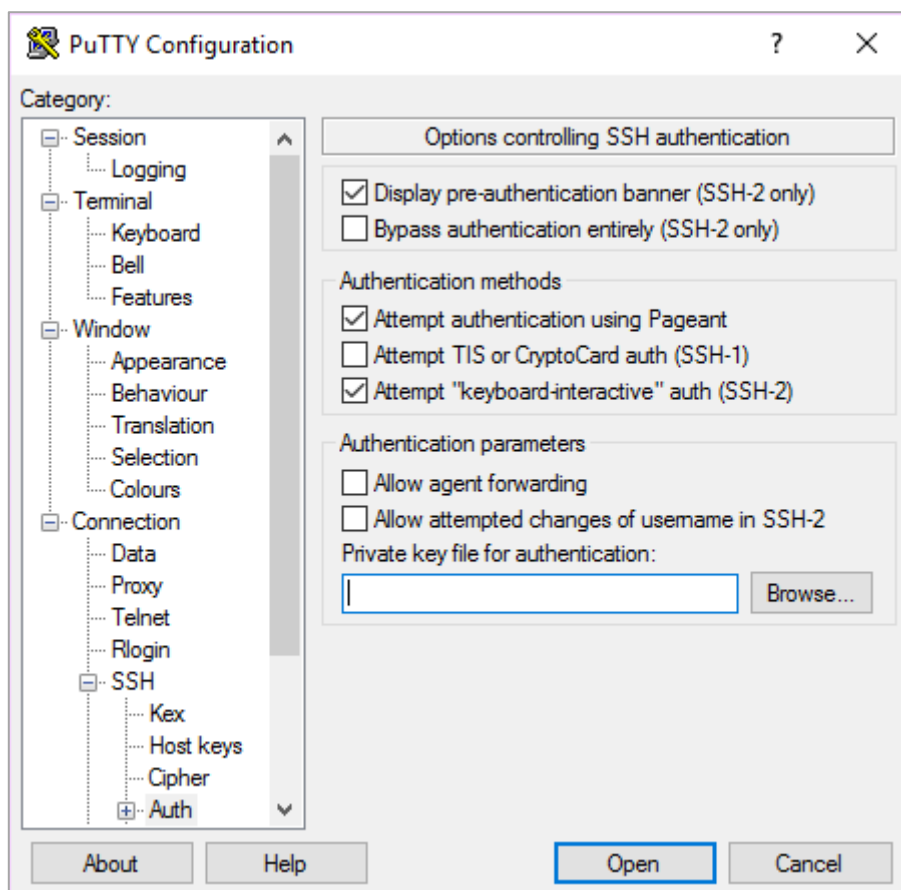


Figura 44. Interfaz cliente PuTTY – Categoría SSH-Auth

Ya tenemos todo configurado para realizar la conexión con nuestra instancia, pero antes es recomendable volver a la categoría “Session” y guardar la configuración para volver a utilizarla en conexiones posteriores. Para ello le damos un nombre a nuestra configuración y pulsamos sobre “Save”.

Por último, podemos realizar la conexión con nuestra instancia seleccionando la opción “Open”. La primera vez que nos conectemos a nuestra instancia, nos aparecerá el cuadro de diálogo representado en la siguiente figura.

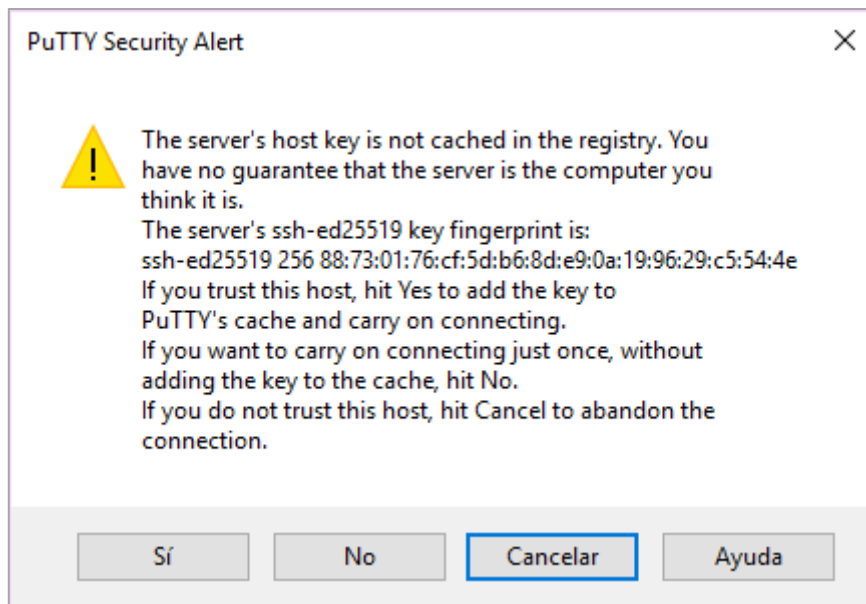


Figura 45. Cuadro de diálogo PuTTY primera conexión

En este cuadro de diálogo se nos indica que no tenemos garantía de que el host con el que nos conectamos sea el que creemos que es y nos pregunta si confiamos en este host. Hacemos click en “Sí” y nos aparecerá la Shell correspondiente a la conexión que hemos creado y que se muestra en la siguiente figura.

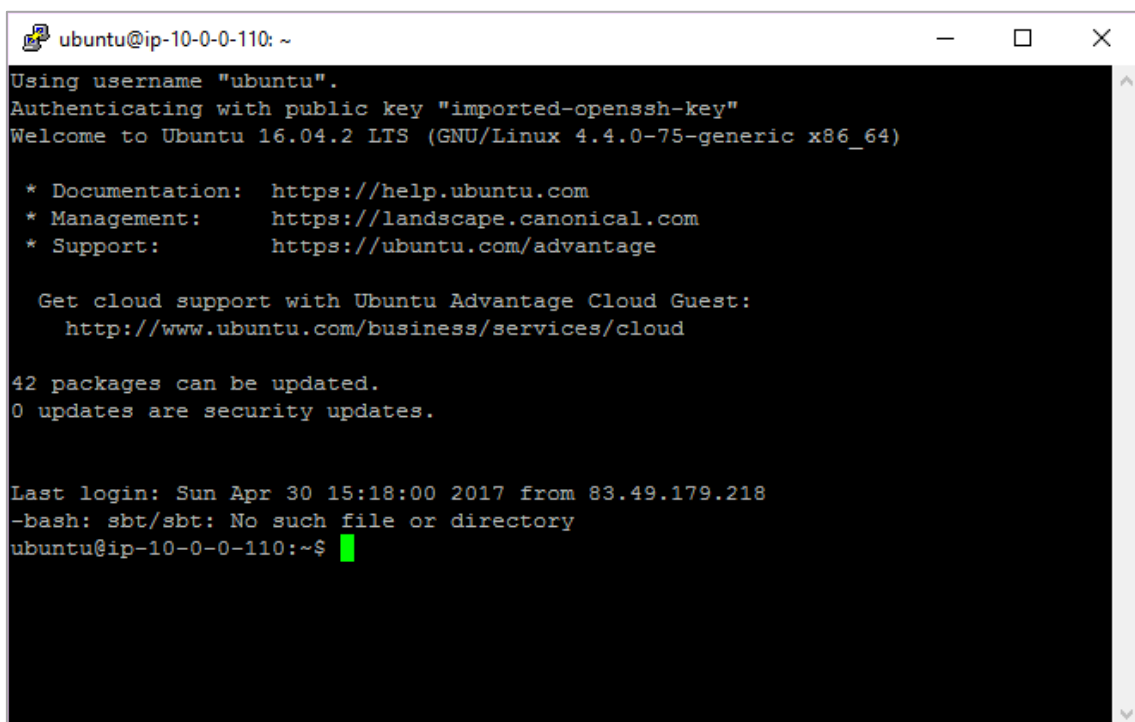


Figura 46. Shell conexión con la instancia EC2 desde PuTTY

El problema con este método de conexión y con el método de conexión desde Ubuntu, es que debemos cambiar la configuración de nuestra conexión ya que el DNS público cambia cada vez que apagamos y volvemos a encender nuestra instancia de

EC2. Esto es lo que se intenta solucionar en la siguiente sección, haciendo uso del servicio No-IP.

### *Conexión desde Ubuntu 16.04*

La conexión desde Ubuntu es muy sencilla ya que sólo necesitamos un terminal bash y tener instalado un cliente SSH. En caso de no tener instalado un cliente SSH podemos instalarlo haciendo uso del siguiente comando:

```
$ sudo apt install -y openssh-client
```

También es necesario que nuestro grupo de seguridad permita el tráfico SSH entrante hacia nuestra instancia de EC2.

Una vez cumplimos los requisitos previos, tenemos que acceder a nuestra consola de EC2, pulsar en "Connect" y nos aparecerá la ventana mostrada en la siguiente figura.

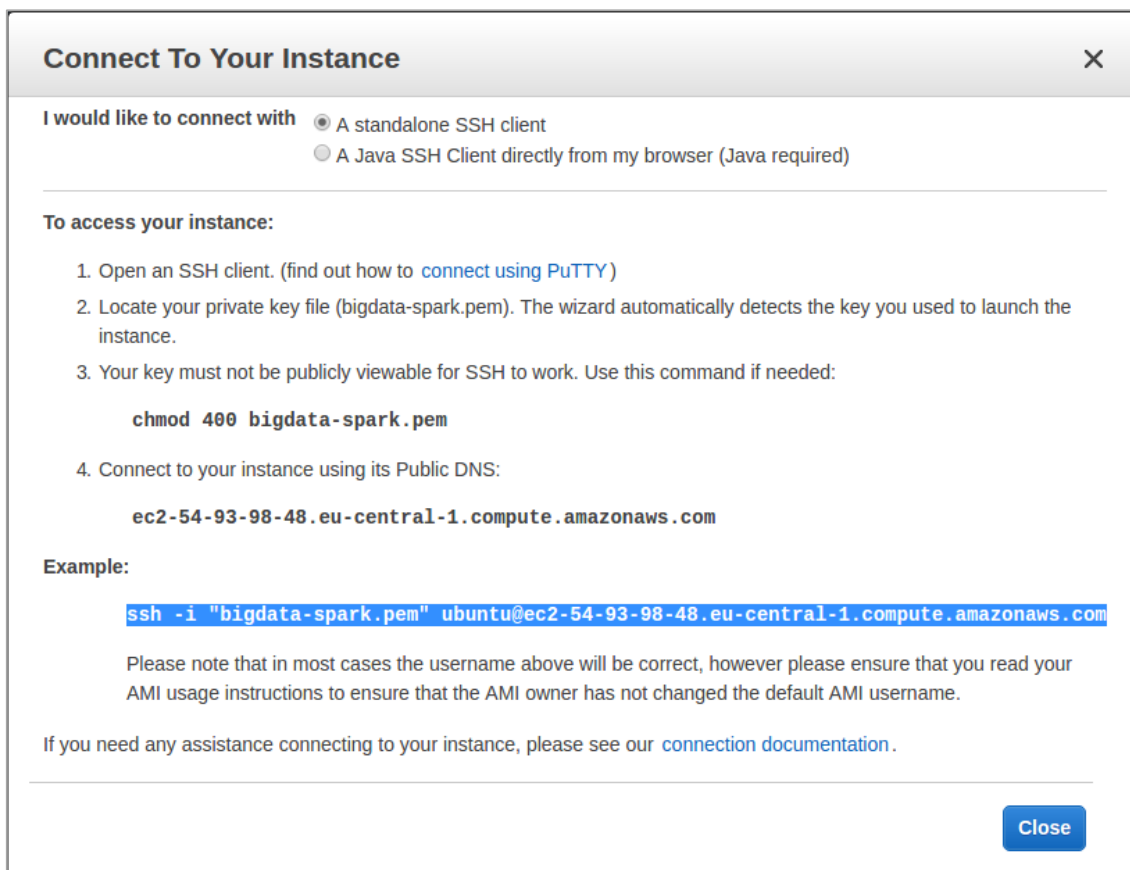


Figura 47. Ventana de conexión con nuestra instancia EC2

Esta ventana nos muestra información relevante para la conexión con nuestra instancia de EC2. Debemos copiar el comando seleccionado en la captura que hemos adjuntado. Una vez hecho esto, abriremos una terminal en la carpeta donde tengamos nuestra clave privada descargada al crear nuestra instancia de EC2.

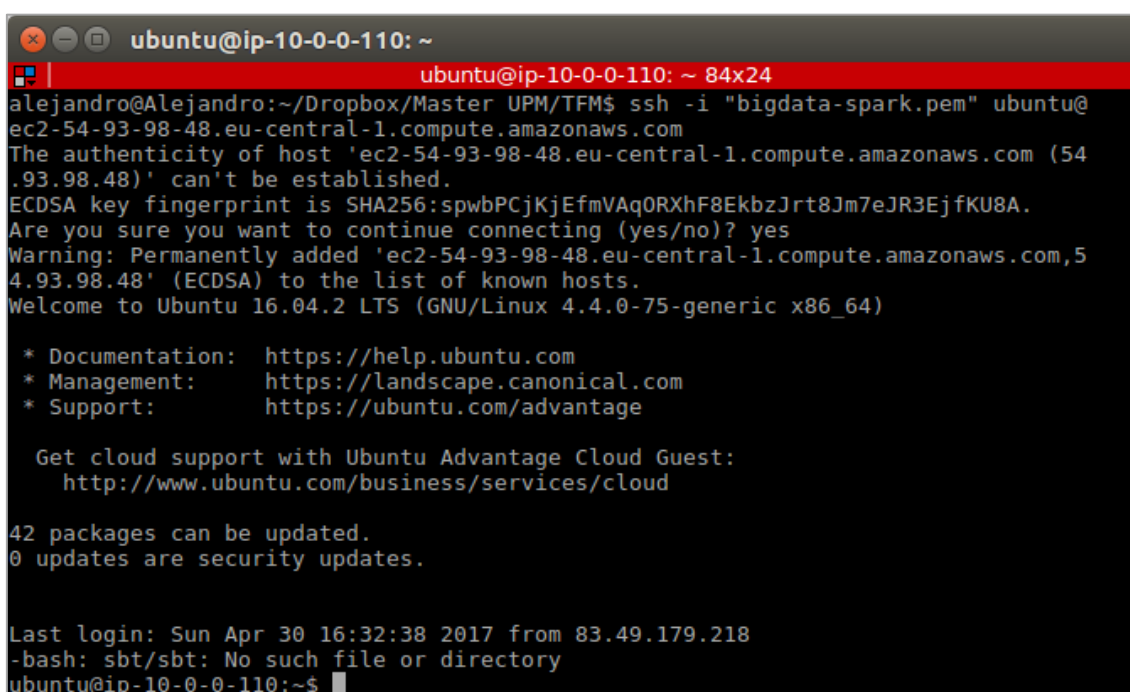
En la terminal ejecutaremos el siguiente comando para cambiar los permisos de nuestra clave privada:

```
$ chmod 400 archivo_clave_privada.pem
```

Y una vez tenemos permiso para usar nuestra clave, podemos ejecutar el comando que copiamos anteriormente para conectarnos a nuestra instancia. Este comando tiene la siguiente estructura:

```
$ ssh -i "archivo_clave_privada.pem" ubuntu@dns_publico_ec2
```

Tras ejecutar este comando nos habremos conectado a nuestra instancia de EC2 y estaremos ante una shell remota.



```
ubuntu@ip-10-0-0-110: ~
alejandro@Alejandro:~/Dropbox/Master UPM/TFM$ ssh -i "bigdata-spark.pem" ubuntu@
ec2-54-93-98-48.eu-central-1.compute.amazonaws.com
The authenticity of host 'ec2-54-93-98-48.eu-central-1.compute.amazonaws.com (54
.93.98.48)' can't be established.
ECDSA key fingerprint is SHA256:spwbPCjKjEfmVAqORXhF8EkbzJrt8Jm7eJR3EjfkU8A.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-54-93-98-48.eu-central-1.compute.amazonaws.com,5
4.93.98.48' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-75-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

Get cloud support with Ubuntu Advantage Cloud Guest:
http://www.ubuntu.com/business/services/cloud

42 packages can be updated.
0 updates are security updates.

Last login: Sun Apr 30 16:32:38 2017 from 83.49.179.218
-bash: sbt/sbt: No such file or directory
ubuntu@ip-10-0-0-110:~$
```

Figura 48. Shell remota conectada a la instancia de EC2

## 4.2 Instalación del servicio No-IP

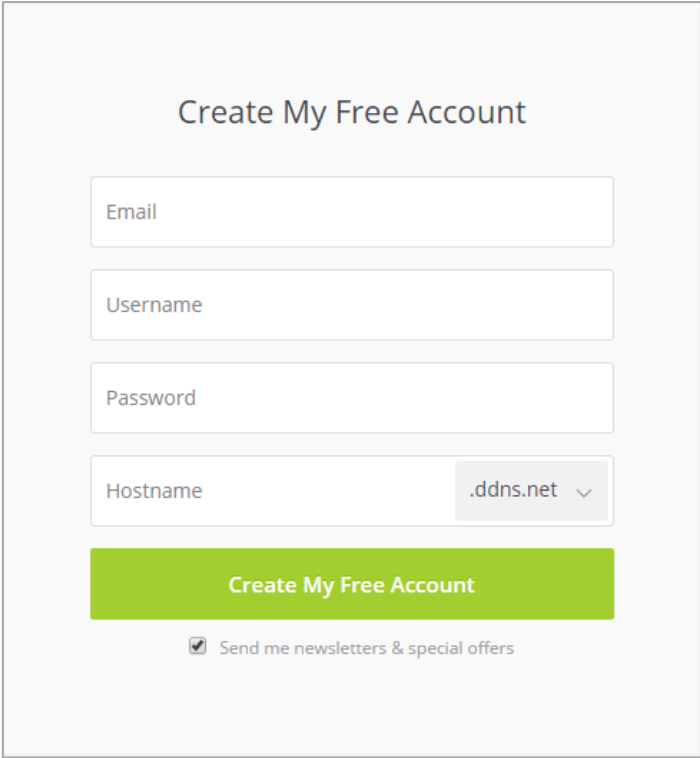
Como ya hemos comentado con anterioridad, Amazon EC2 cambia el nombre DNS público y la dirección IP pública de una instancia cada vez que esta es reiniciada. Aunque es posible usar el nombre DNS que se nos asigna para conectarnos a nuestra instancia o para acceder a las interfaces web de los servicios que vamos a instalar, es muy molesto que este nombre DNS cambie cada vez que reiniciamos la instancia y, además, este nombre DNS tiene una longitud considerable.

Para solucionar los posibles problemas que esto conlleva, podemos utilizar un proveedor de DNS dinámico junto con nuestra instancia EC2, y configurar la instancia para actualizar las direcciones asociadas con un nombre de DNS público cada vez que

la instancia se arranque. En este caso hemos decidido usar el servicio No-IP que nos ofrece hasta tres dominios públicos gratuitos durante el primer mes de uso.

#### 4.2.1 Registrar un hostname

Para registrar un dominio en el servicio No-IP, debemos registrarnos en su página web (<https://www.noip.com/sign-up>) rellenando el formulario que se muestra en la siguiente figura.



The image shows a registration form titled "Create My Free Account". It contains four input fields: "Email", "Username", "Password", and "Hostname". The "Hostname" field has a dropdown menu currently showing ".ddns.net". Below the fields is a green button labeled "Create My Free Account". At the bottom, there is a checkbox labeled "Send me newsletters & special offers" which is checked.

Figura 49. Formulario de registro del servicio No-IP

Deberemos introducir nuestro email, un nombre de usuario, una contraseña y el primer nombre DNS que deseamos crear. Tras rellenar el formulario pulsamos sobre "Create My Free Account" y ya tendremos nuestra cuenta creada. Antes de comenzar a usar el servicio se nos solicitará que verifiquemos nuestro email mediante un correo de confirmación.

Ya tenemos un dominio preparado para nuestro host. El siguiente paso es asignar un host a este dominio. Para ello vamos a instalar un cliente en nuestro host que actualizará la dirección IP de nuestra instancia en el servicio No-IP cada vez que esta cambie.

#### 4.2.2 Instalación del Dynamic Update Client (DUC)

A continuación, tenemos que instalar el DUC de No-IP para Linux. Podremos hacerlo de una forma muy sencilla mediante el terminal, tras conectarnos a la instancia correspondiente. Todos los comandos que tendremos que introducir, necesitarán

permisos de superusuario, por lo que podemos hacer login como usuario root, o bien, utilizar sudo con todos los comandos.

En primer lugar, tenemos que descargarnos el código del DUC mediante la ejecución de los siguientes comandos:

```
$ cd /usr/local/src/
```

```
$ wget http://www.no-ip.com/client/linux/noip-duc-linux.tar.gz
```

Ahora, extraemos el fichero comprimido que acabamos de descargar y comenzamos la instalación. Los comandos a utilizar son los siguientes:

```
$ tar -xvzf noip-duc-linux.tar.gz
```

```
$ cd noip-2.1.9-1/
```

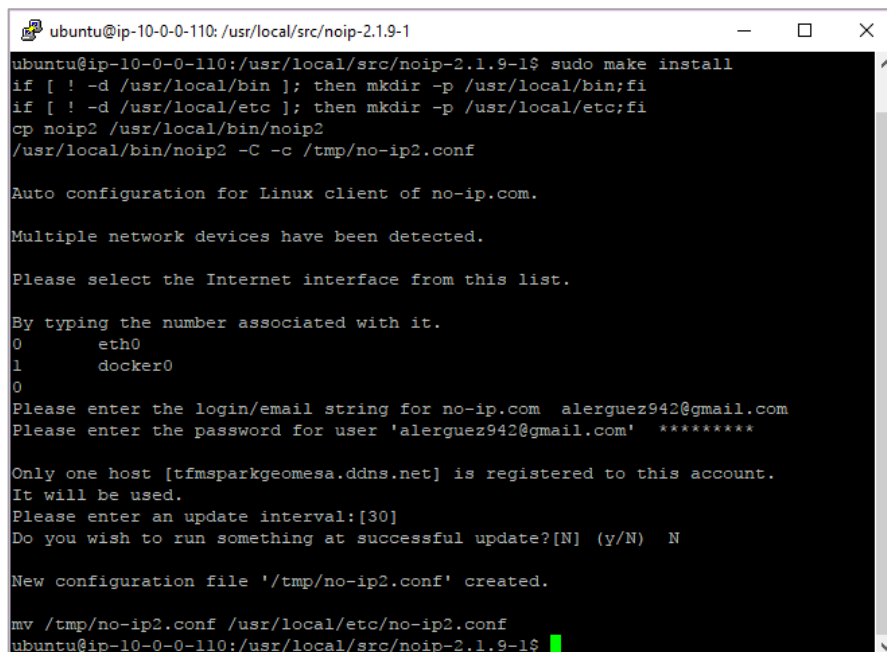
```
$ make install
```

Para que la llamada al comando "make install" tenga éxito es necesario tener instalados en nuestro sistema los componentes make y gcc.

```
$ apt-get install -y make
```

```
$ apt-get install -y gcc
```

Por último, tendremos que completar la información que se nos solicita por pantalla al ejecutar el comando "make install". En la captura que se muestra a continuación podemos ver la información requerida.



```
ubuntu@ip-10-0-0-110: /usr/local/src/noip-2.1.9-1
ubuntu@ip-10-0-0-110:/usr/local/src/noip-2.1.9-1$ sudo make install
if [ ! -d /usr/local/bin ]; then mkdir -p /usr/local/bin;fi
if [ ! -d /usr/local/etc ]; then mkdir -p /usr/local/etc;fi
cp noip2 /usr/local/bin/noip2
/usr/local/bin/noip2 -C -c /tmp/no-ip2.conf

Auto configuration for Linux client of no-ip.com.

Multiple network devices have been detected.

Please select the Internet interface from this list.

By typing the number associated with it.
0      eth0
1      docker0
0

Please enter the login/email string for no-ip.com  alerguez942@gmail.com
Please enter the password for user 'alerguez942@gmail.com'  *****

Only one host [tfmsparkgeomesa.ddns.net] is registered to this account.
It will be used.
Please enter an update interval:[30]
Do you wish to run something at successful update?[N] (y/N)  N

New configuration file '/tmp/no-ip2.conf' created.

mv /tmp/no-ip2.conf /usr/local/etc/no-ip2.conf
ubuntu@ip-10-0-0-110:/usr/local/src/noip-2.1.9-1$
```

Figura 50. Instalación del DUC

Como podemos ver en la siguiente figura, se pedirá el login/email y la contraseña del servicio No-IP. A continuación, en caso de que exista más de un hostname registrado en nuestra cuenta tendremos que elegir cuál de ellos usar, junto con un intervalo de actualización del hostname (que será el plazo en días, tras el que se nos preguntará si queremos seguir conservando el hostname o no), y si queremos ejecutar algo tras estas actualizaciones.

Al terminar la configuración se nos indica la ruta al fichero de configuración que podemos modificar para cambiar cualquier configuración del servicio No-IP.

Con esto ya podemos utilizar el servicio No-IP en nuestra instancia, pero antes, debemos asegurarnos de que nuestro grupo de seguridad permite el tráfico entrante en el puerto 8245 TCP. Para ello, añadimos una nueva regla de seguridad para permitir el tráfico TCP entrante usando dicho puerto desde cualquier dirección IP.

Type ⓘ	Protocol ⓘ	Port Range ⓘ	Source ⓘ
Custom TCP Rule	TCP	8245	0.0.0.0/0

Figura 51. Regla tráfico entrante para No-IP

Para arrancar el DUC, debemos invocar el siguiente comando:

```
$ /usr/local/bin/noip2
```

Con esto conseguiremos contactar con el servicio No-IP y actualizar nuestra IP en los servidores DNS.

El siguiente y último paso es hacer que este comando se ejecute cada vez que nuestra instancia se enciende.

#### 4.2.3 Ejecución automática del DUC

Para que se ejecute el DUC al iniciar nuestra instancia, tenemos un método muy sencillo consistente en añadir el comando a ejecutar en el archivo `/etc/rc.local` y configurar `systemd` para que ejecute el servicio `rc.local` al inicio. Para ello, comenzamos por añadir la siguiente línea al fichero `/etc/rc.local` (antes de la sentencia `exit 0`):

```
/usr/local/bin/noip2
```

A continuación, configuramos `systemd` para que ejecute el servicio `rc.local` al inicio:

```
sudo systemctl enable rc-local.service
```

Y hacemos el archivo `/etc/rc.local` ejecutable:

```
sudo chmod a+x /etc/rc.local
```



## 4.3 Instalaciones previas

Durante este apartado mostraremos los comandos usados para la instalación de los componentes secundarios que componen los prerequisites comunes a los componentes principales.

### 4.3.1 JDK 8

Para algunos de nuestros componentes, como por ejemplo Geomesa, es necesaria la versión 8 del JDK. Para instalar dicha versión es suficiente con ejecutar el siguiente comando:

```
$ sudo apt-get install -y openjdk-8-jdk
```

Una vez terminada la instalación, debemos asegurarnos de que la variable `JAVA_HOME` contiene el directorio de instalación del JDK 8. Para ello podemos modificar el archivo `/home/Ubuntu/.bashrc` para que se cree la variable con el valor adecuado cada vez que iniciamos sesión en nuestra instancia con el usuario “ubuntu” (usuario por defecto). Para ello, introducimos la siguiente línea al final de dicho fichero:

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
```

Con esto ya tendríamos completamente funcional nuestra instalación del JDK 8.

### 4.3.2 Git

Otra importante herramienta que nos servirá para obtener el código de algunos componentes o librerías es Git. Instalarlo es tan sencillo como ejecutar el siguiente comando:

```
$ sudo apt-get install -y git
```

### 4.3.3 Apache Maven

Esta herramienta como ya hemos comentado en secciones anteriores nos servirá para compilar código de librerías o ejemplos que ejecutemos a lo largo de nuestro proyecto.

La instalación de Apache Maven (33) consiste en un proceso simple basado en la extracción de un fichero comprimido y añadir la carpeta con los binarios a la variable `PATH`. Previamente debemos asegurarnos de que hemos instalado el JDK 8 y hemos dado el valor correspondiente a la variable `JAVA_HOME`.

El primer paso consiste en visitar la web de descargas de Apache Maven (<http://maven.apache.org/download.cgi>) y copiar el link de descarga de la versión actual (en nuestro caso, en el momento de descargarlo esta era la versión 3.3.9). Podemos descargar indistintamente el archivo comprimido en formato zip o en tar.gz.

Para realizar la descarga de los binarios de Maven en nuestra instancia, ejecutamos el siguiente comando:

```
$ curl -O https://archive.apache.org/dist/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.tar.gz
```

El siguiente paso es descomprimir el archivo que acabamos de descargar:

```
$ tar -xzvf apache-maven-3.3.9-bin.tar.gz
```

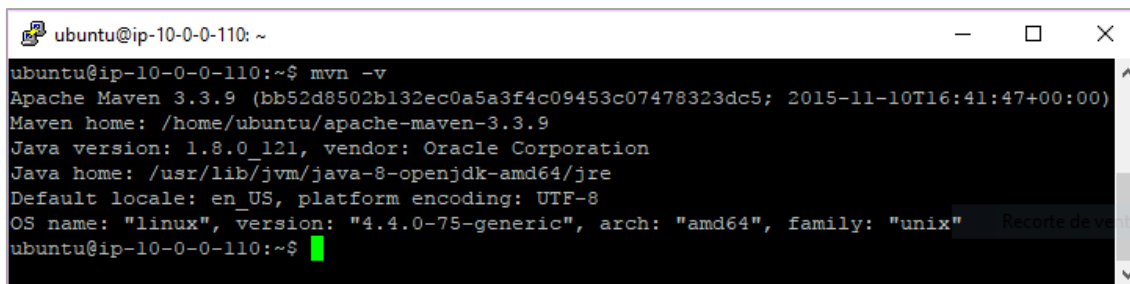
O en el caso de haber descargado el archivo en formato zip:

```
$ unzip apache-maven-3.3.9-bin.tar.gz
```

Una vez terminada la instalación, debemos añadir el directorio `bin`, ubicado dentro de la carpeta que hemos descomprimido y que contiene los binarios de Maven, a la variable `PATH`. Para ello podemos modificar el archivo `/home/Ubuntu/.bashrc` para que se actualice la variable con el valor adecuado cada vez que iniciamos sesión en nuestra instancia con el usuario "ubuntu" (usuario por defecto). Para ello, introducimos la siguiente línea al final de dicho fichero:

```
export PATH=/home/ubuntu/apache-maven-3.3.9/bin:$PATH
```

Con esto ya tendríamos completamente funcional nuestra instalación del Maven. Para comprobarlo podemos ejecutar el comando `mvn -v` en nuestra Shell. El resultado debe ser similar al siguiente:



```
ubuntu@ip-10-0-0-110: ~  
ubuntu@ip-10-0-0-110:~$ mvn -v  
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T16:41:47+00:00)  
Maven home: /home/ubuntu/apache-maven-3.3.9  
Java version: 1.8.0_121, vendor: Oracle Corporation  
Java home: /usr/lib/jvm/java-8-openjdk-amd64/jre  
Default locale: en_US, platform encoding: UTF-8  
OS name: "linux", version: "4.4.0-75-generic", arch: "amd64", family: "unix"  
ubuntu@ip-10-0-0-110:~$
```

Figura 52. Comando Maven version

## 4.4 Instalación Apache Hadoop

Como ya hemos comentado, es necesaria la instalación de Hadoop (34) ya que la base de datos Accumulo usa HDFS como sistema de ficheros distribuido.

Como prerrequisitos deberemos tener instalado el JDK 8 (véase subsección 4.3.1), SSH y Rsync (para gestionar los componentes remotos). Si no tenemos instalado en nuestra instancia alguno de estas dos últimas herramientas, basta con ejecutar los siguientes comandos:

```
$ sudo apt-get install -y ssh
$ sudo apt-get install -y rsync
```

#### 4.4.1 Configurar conexión SSH sin contraseña

Para que Hadoop pueda funcionar correctamente, necesita tener conexión SSH con sus nodos sin necesidad de ser preguntado por una contraseña. Es decir, Hadoop debe ser capaz de establecer la conexión SSH del mismo modo que nosotros lo hacemos cuando nos conectamos a nuestra instancia, usando una clave privada contenida en un archivo.

El primer paso es generar una clave RSA usando `ssh-keygen`:

```
$ ssh-keygen -P ''
```

Cuando se nos pregunte por el archivo donde queremos guardar la clave, presionamos ENTER para seleccionar el valor por defecto. Una vez hemos generado nuestra clave, tenemos que añadirla al fichero de claves autorizadas (`~/.ssh/authorized_keys`):

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

El último paso es añadir “localhost” y “0.0.0.0” a la lista de hosts conocidos. La forma más sencilla de hacerlo es ejecutar el comando `ssh`:

```
$ ssh localhost
```

Nos aparecerá un mensaje informándonos de que no es posible garantizar la autenticidad del host al que queremos conectar y preguntando si queremos conectarnos igualmente. El mensaje será similar al siguiente:

```
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is
SHA256:PIWrIbUGCuGWSZ4Biq+5KU7Y7JXqiaJjL47LaGhJq3s.
Are you sure you want to continue connecting (yes/no)?
```

Debemos escribir “yes” y pulsar ENTER. Tras esto nos conectaremos a nuestro propio host usando SSH y se nos habrá añadido “localhost” a la lista de hosts conocidos, por lo que la próxima conexión será inmediata. Podemos salir de la sesión usando el comando `exit`.

Repetimos el mismo proceso con el host “0.0.0.0” y habremos terminado la configuración de SSH.

#### 4.4.2 Descarga e instalación

Antes de proceder a la descarga de Hadoop, vamos a crear un directorio para contener las descargas que realicemos durante el proceso de desarrollo de nuestro Trabajo Fin de Máster. También cambiaremos el directorio actual a el nuevo directorio:

```
$ mkdir -p ~/Downloads
```

```
$ cd ~/Downloads
```

Ya podemos pasar a realizar la descarga de Hadoop. Para ello, buscamos el link de descarga en la página de descargas de Hadoop (<http://www.apache.org/dyn/closer.cgi/hadoop/common/>) y usamos la herramienta curl:

```
$ curl -O http://apache.uvigo.es/hadoop/common/hadoop-2.7.3/hadoop-2.7.3.tar.gz
```

A continuación, descomprimos el archivo descargado en el directorio personal:

```
$ tar -xvzf hadoop-2.7.3.tar.gz -C ~
```

Una vez terminado esto, debemos añadir los directorios bin y sbin, ubicados dentro de la carpeta que hemos descomprimido y que contienen los binarios y los scripts de Hadoop, a la variable PATH. Para ello podemos modificar el archivo /home/Ubuntu/.bashrc para que se actualice la variable con el valor adecuado cada vez que iniciamos sesión en nuestra instancia con el usuario "ubuntu" (usuario por defecto). Para ello, introducimos la siguiente línea al final de dicho fichero:

```
export PATH=/home/ubuntu/hadoop-2.7.3/bin:/home/ubuntu/hadoop-2.7.3/sbin:$PATH
```

Por último, debemos crear la variable HADOOP\_HOME que contenga el directorio base de Hadoop. Modificamos el archivo anterior con la siguiente línea:

```
export HADOOP_HOME=/home/ubuntu/hadoop-2.7.3
```

#### 4.4.3 Configuración

En esta subsección comentaremos los cambios a realizar para poder usar HDFS y poder acceder a su interfaz web desde nuestro equipo.

El primer paso es modificar la configuración de Java por defecto que trae Hadoop. Para ello, tenemos que modificar el scrip hadoop-env.sh. En nuestro caso hemos usado la herramienta de edición en línea de comandos vim. Si queremos instalar esta herramienta:

```
$ sudo apt-get install -y vim
```

Para editar el script que hemos comentado:

```
$ vim ~/hadoop-2.7.3/etc/hadoop/hadoop-env.sh
```

Debemos buscar la línea en la que se exporta la variable `JAVA_HOME` y modificarla para que apunte al directorio que contiene nuestra instalación de Java (mismo directorio que hemos especificado en `.bashrc`). En nuestro caso:

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
```

Por defecto, Hadoop genera muchos logs de nivel de depuración que pueden llegar a ser molestos. Para detener este comportamiento, buscamos en el fichero anterior la línea que exporta la variable `HADOOP_OPTS` y la cambiamos a:

```
export HADOOP_OPTS="$HADOOP_OPTS -XX:-PrintWarnings  
-Djava.net.preferIPv4Stack=true"
```

Guardamos y cerramos el archivo. De nuevo usamos el editor `vim` para modificar, en este caso, el archivo `core-site.xml`.

```
$ vim ~/hadoop-2.7.3/etc/hadoop/core-site.xml
```

Tenemos que añadir un bloque `<property>` con nombre `fs.defaultFS`. El valor de este bloque debe contener el hostname y el puerto del NameNode (en nuestro caso `localhost` y el puerto es el `9000`). Ignorando los comentarios, el fichero debe quedar con el siguiente contenido:

```
<?xml version="1.0" encoding="UTF-8"?>  
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>  
<configuration>  
  <property>  
    <name>fs.defaultFS</name>  
    <value>hdfs://localhost:9000</value>  
  </property>  
</configuration>
```

Guardamos y cerramos el archivo. Pasamos a editar el archivo `hdfs-site.xml`:

```
$ vim ~/hadoop-2.7.3/etc/hadoop/hdfs-site.xml
```

Este fichero contiene la configuración de Hadoop HDFS. Necesitamos agregar las siguientes propiedades:

- **dfs.replication:** Esta propiedad especifica cuantas veces es replicado un bloque por Hadoop. Por defecto, Hadoop crea 3 réplicas para cada bloque. Para empezar, vamos a poner esta propiedad a 1 ya que no estamos creando un cluster en principio.

- **dfs.name.dir:** Esta propiedad especifica una localización en el sistema de ficheros local donde el NameNode almacena la tabla de nombres. Tenemos que cambiarla debido a que, por defecto, Hadoop almacena esta tabla en /tmp. Vamos a usar `hdfs_storage/name` para almacenar dicha tabla.
- **dfs.data.dir:** Esta propiedad especifica una localización en el sistema de ficheros local donde los DataNodes deben almacenar sus bloques. De nuevo, Hadoop usa /tmp por defecto, por lo que debemos cambiarla. Vamos a usar `hdfs_storage/data` para almacenar los bloques.

Una vez añadidas estas propiedades e ignorando los comentarios, el fichero debe quedar como el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.name.dir</name>
    <value>hdfs_storage/name</value>
  </property>
  <property>
    <name>dfs.data.dir</name>
    <value>hdfs_storage/data</value>
  </property>
</configuration>
```

De nuevo, guardamos y cerramos el archivo. El último archivo por editar es `mapred-site.xml`:

```
$ vim ~/hadoop-2.7.3/etc/hadoop/mapred-site.xml
```

Este archivo contiene la configuración de MapReduce. En concreto, debemos agregar la propiedad `mapred.job.tracker` que contiene el hostname y puerto donde se ejecuta el rastreador de trabajos MapReduce. En nuestro caso el archivo ha quedado como sigue:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:9001</value>
  </property>
```

```
</configuration>
```

#### 4.4.4 Ejecución de HDFS

Antes de ejecutar Hadoop por primera vez, debemos inicializar el NameNode. Para ello, primero debemos cambiar el directorio de trabajo actual al directorio base de Hadoop:

```
$ cd ~/hadoop-2.7.3/
```

Este paso es importante ya que según la configuración que hemos usado, Hadoop creará el directorio `hdfs_storage` en el directorio actual.

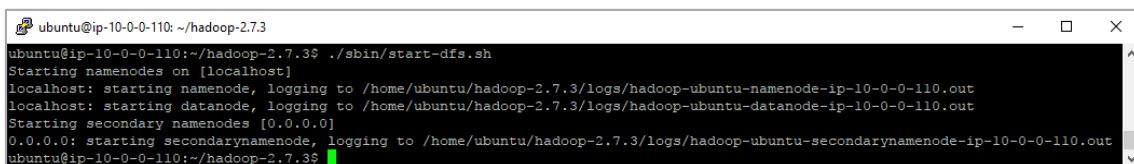
A continuación, podemos usar el siguiente comando para inicializar el NameNode:

```
$ ./bin/hdfs namenode -format
```

Obtendremos varios mensajes de salida que nos dirán como va avanzando el proceso y finalizará. Una vez finalizado, podemos pasar a ejecutar Hadoop HDFS. Para ello podemos usar el script `start-dfs.sh`:

```
$ ./sbin/start-dfs.sh
```

La salida de este script es la mostrada en la siguiente figura:



```
ubuntu@ip-10-0-0-110: ~/hadoop-2.7.3
ubuntu@ip-10-0-0-110:~/hadoop-2.7.3$ ./sbin/start-dfs.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/ubuntu/hadoop-2.7.3/logs/hadoop-ubuntu-namenode-ip-10-0-0-110.out
localhost: starting datanode, logging to /home/ubuntu/hadoop-2.7.3/logs/hadoop-ubuntu-datanode-ip-10-0-0-110.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /home/ubuntu/hadoop-2.7.3/logs/hadoop-ubuntu-secondarynamenode-ip-10-0-0-110.out
ubuntu@ip-10-0-0-110:~/hadoop-2.7.3$
```

Figura 53. Ejecución del script `start-dfs.sh`

En la salida se puede apreciar que este script nos arranca en local un NameNode y un DataNode y además arranca un NameNode secundario. También vemos los ficheros de registro donde se almacenan los logs de cada componente.

Una vez termine la ejecución del script, tendremos operativo nuestro sistema Hadoop HDFS. Podemos acceder a su interfaz web usando la ip de nuestra instancia y el puerto 50070 en un navegador web. En nuestro caso podemos usar el nombre DNS que configuramos en la sección 4.2 ([tfmsparkgeomesa.ddns.net](http://tfmsparkgeomesa.ddns.net)). Dicha interfaz se muestra en la siguiente figura:

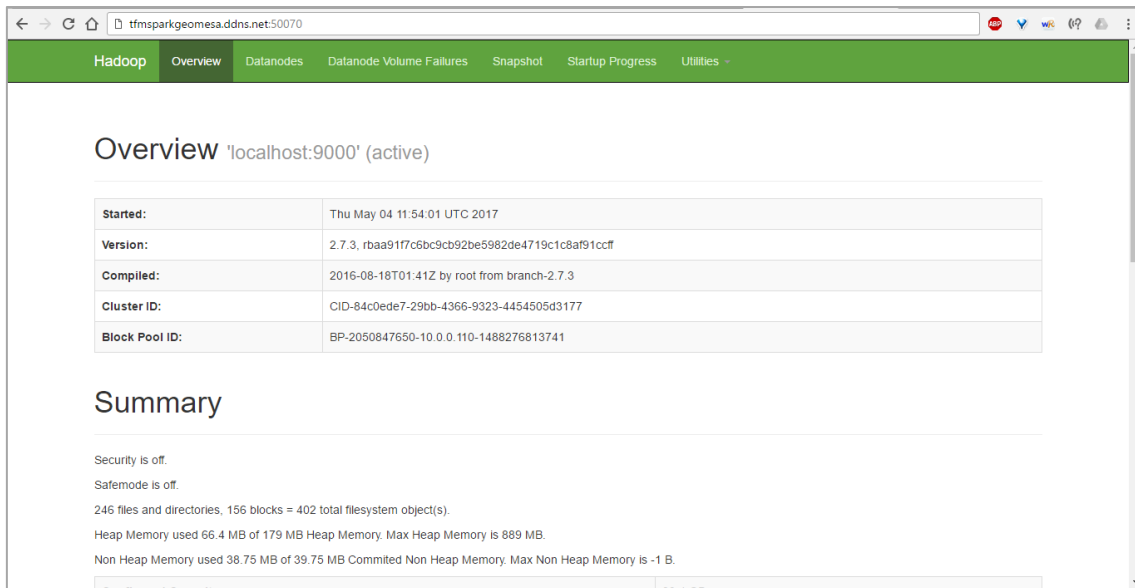


Figura 54. Interfaz Web Hadoop HDFS.

Para parar la ejecución de Hadoop HDFS, contamos con un script equivalente llamado `stop-dfs.sh`:

```
$ ./sbin/stop-dfs.sh
```

## 4.5 Instalación Apache Zookeeper

En esta sección mostraremos el procedimiento para instalar Zookeeper (35) en nuestra instancia Ubuntu 16.04 y la configuración que se ha usado para el desarrollo del proyecto.

### 4.5.1 Descarga e instalación

El primer paso será descargar la versión actual de Zookeeper, que se puede descargar desde el siguiente enlace: <http://apache.rediris.es/zookeeper/>. En el momento de escribir esta memoria, la versión actual de Zookeeper es la 3.4.9.

Para ello vamos a cambiar el directorio actual al directorio de descargas que creamos previamente:

```
$ cd ~/Downloads
```

Podemos usar, como en ocasiones anteriores, la herramienta `curl` para la descarga del fichero comprimido que contiene el software de Zookeeper:

```
$ curl -O http://apache.rediris.es/zookeeper/zookeeper-3.4.9/zookeeper-3.4.9.tar.gz
```

A continuación, descomprimos el archivo descargado en el directorio personal:

```
$ tar -xvzf zookeeper-3.4.9.tar.gz -C ~
```



Al descomprimir se nos crea un directorio con el mismo nombre que el archivo comprimido. Dentro de este encontramos varios directorios, entre los que cabe destacar los siguientes:

- El directorio `bin` contiene varios scripts entre los que destaca “zkServer.sh”, que nos servirá para arrancar y parar el proceso servidor de Zookeeper.
- El directorio `conf` contiene las configuraciones útiles para Zookeeper, como por ejemplo la configuración de la librería log4j (para crear logs desde una aplicación escrita en java) y la configuración del propio servidor de Zookeeper. Cuando iniciemos el servidor Zookeeper, se buscará por defecto en esta carpeta el archivo “zoo.cfg” que debe contener la configuración de Zookeeper.

Debemos añadir el directorio `bin` a la variable `PATH` y la variable `ZOOKEEPER_HOME`. Para ello volveremos a modificar el archivo `/home/Ubuntu/.bashrc` para que se actualice la variable con el valor adecuado cada vez que iniciamos sesión en nuestra instancia con el usuario “ubuntu” (usuario por defecto). Introducimos las siguientes líneas al final de dicho fichero:

```
export PATH=/home/ubuntu/zookeeper-3.4.9/bin:$PATH
export ZOOKEEPER_HOME=/home/ubuntu/zookeeper-3.4.9
```

Con esto hemos concluido la descarga e instalación de Zookeeper.

#### 4.5.2 Configuración

Como hemos dicho, la configuración de Zookeeper se encuentra en el archivo `zookeeper-3.4.9/conf/zoo.cfg`, a no ser que especifiquemos otro archivo de configuración en el momento de la ejecución.

En este fichero de configuración podemos encontrar dos tipos de línea:

- **Comentarios.** Son líneas que tienen la función de dar algún tipo de información a la persona que lee el archivo de configuración, pero no son procesadas por Zookeeper. Van precedidas del carácter ‘#’.
- **Líneas de configuración.** Son las líneas que procesa Zookeeper al iniciarse. La sintaxis es la siguiente: `nombre_propiedad=valor_propiedad`

A continuación, mostramos el contenido del fichero de configuración de Zookeeper usado para el desarrollo de este proyecto:

```
# The number of milliseconds of each tick
tickTime=2000
```

```
# The number of ticks that the initial
# synchronization phase can take
initLimit=5

# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=2

forceSync=no

# the directory where the snapshot is stored.
dataDir=/home/ubuntu/zookeeper-3.4.9/data

# the port at which the clients will connect
clientPort=2181

# the maximum number of client connections.
# increase this if you need to handle more clients
maxClientCnxns=0

# The number of snapshots to retain in dataDir
autopurge.snapRetainCount=5

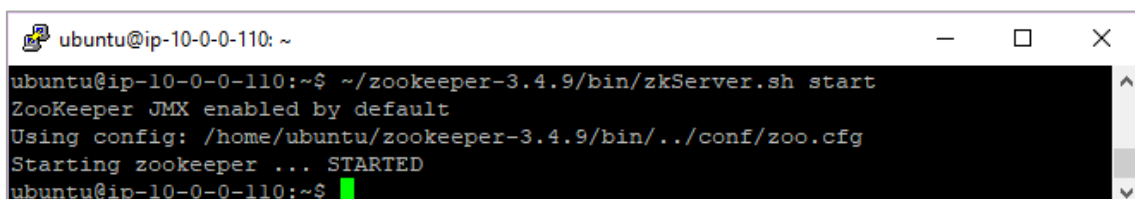
# Purge task interval in hours
# Set to "0" to disable auto purge feature
autopurge.purgeInterval=24
```

### 4.5.3 Ejecución

Para ejecutar el servidor Zookeeper debemos hacer uso del script `zkServer.sh` ubicado en el directorio `bin` dentro de la instalación de ZooKeeper. Dicho script podemos usarlo de la siguiente forma para arrancar Zookeeper:

```
$ ~/zookeeper-3.4.9/bin/zkServer.sh start
```

En caso de que la configuración sea correcta y el servicio inicie satisfactoriamente debemos obtener la siguiente salida:

A terminal window titled 'ubuntu@ip-10-0-0-110: ~' with standard window controls. The terminal shows the command `~/zookeeper-3.4.9/bin/zkServer.sh start` being executed. The output is: `ZooKeeper JMX enabled by default`, `Using config: /home/ubuntu/zookeeper-3.4.9/bin/./conf/zoo.cfg`, and `Starting zookeeper ... STARTED`. The prompt returns to `ubuntu@ip-10-0-0-110:~$` with a green cursor.

```
ubuntu@ip-10-0-0-110:~$ ~/zookeeper-3.4.9/bin/zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /home/ubuntu/zookeeper-3.4.9/bin/./conf/zoo.cfg
Starting zookeeper ... STARTED
ubuntu@ip-10-0-0-110:~$
```

Figura 55. Ejecución del script `zkServer.sh start`

Para parar la ejecución de Zookeeper, podemos usar el mismo script con la opción `stop`:

```
$ ~/zookeeper-3.4.9/bin/zkServer.sh stop
```

## 4.6 Instalación Apache Accumulo

El siguiente paso en el despliegue de nuestro entorno es la instalación de la base de datos Apache Accumulo (36) (37). Como ya dijimos en la sección dedicada a Accumulo, esta se ejecuta haciendo uso de HDFS como sistema de archivos y Zookeeper para gestión de configuraciones y consenso. Es por esto por lo que previamente a esta sección debemos haber instalado Hadoop y Zookeeper (secciones 4.4 y 4.5 respectivamente).

### 4.6.1 Descarga e instalación

Para comenzar debemos descargar los binarios de Accumulo, buscando el enlace de descarga en su página web (<https://accumulo.apache.org/downloads/>). En este caso debemos descargar la versión 1.7.2 de Accumulo ya que es la versión compatible con la versión actual de Geomesa.

Para ello vamos a cambiar el directorio actual al directorio de descargas que creamos previamente:

```
$ cd ~/Downloads
```

Usaremos de nuevo la herramienta `curl` para la descarga del fichero comprimido que contiene los binarios de Accumulo:

```
$ curl -O http://apache.rediris.es/accumulo/1.7.2/accumulo-1.7.2-bin.tar.gz
```

A continuación, descomprimos el archivo descargado en el directorio personal:

```
$ tar -xvzf accumulo-1.7.2-bin.tar.gz -C ~
```

Al descomprimir se nos crea un directorio con el mismo nombre que el archivo comprimido. Dentro de este encontramos varios directorios, entre los que cabe destacar los siguientes:

- El directorio `bin` contiene varios scripts que nos servirán para arrancar los diferentes componentes de Accumulo e interactuar con estos.
- El directorio `conf` contiene los ficheros de configuración de Accumulo. Dentro de este se encuentra el directorio `examples` que, como su propio nombre indica, contiene varios ejemplos de archivos de configuración de Accumulo para diferentes tamaños de memoria.

Debemos añadir el directorio `bin` a la variable `PATH`. Para ello volveremos a modificar el archivo `/home/Ubuntu/.bashrc` para que se actualice la variable con el

valor adecuado cada vez que iniciamos sesión en nuestra instancia con el usuario “ubuntu” (usuario por defecto). Introducimos la siguiente línea al final de dicho fichero:

```
export PATH=/home/ubuntu/accumulo-1.7.2/bin:$PATH
```

También debemos añadir la variable ACCUMULO\_HOME que contendrá el directorio base de Accumulo. Para ello introducimos la siguiente línea en el fichero anterior:

```
export ACCUMULO_HOME=/home/ubuntu/accumulo-1.7.2
```

#### 4.6.2 Configuración

Accumulo trae consigo configuraciones de ejemplo para servidores con distintos tamaños de memoria: 512 MB, 1 GB, 2 GB y 3 GB. En nuestro caso y teniendo en cuenta que la memoria de nuestra instancia es de 8 GB hemos tomado como base los archivos de configuración del ejemplo para 2 GB.

El primer paso será, copiar la configuración elegida al directorio `conf` que se encuentra en el directorio base de Accumulo. Para ello usamos el siguiente comando:

```
$ cp ~/accumulo-1.7.2/conf/examples/2GB/standalone/*  
~/accumulo-1.7.2/conf/
```

Ahora vamos a personalizar esta configuración para que se adapte a nuestras necesidades.

Para comenzar modificaremos el script `accumulo-env.sh`, disponible en la carpeta `~/accumulo-1.7.2/conf/`:

```
$ vim ~/accumulo-1.7.2/conf/accumulo-env.sh
```

Por defecto, el proceso Accumulo Monitor sólo está accesible desde la interfaz de red local de nuestra instancia. Para ser capaces de acceder a esta interfaz web a través de Internet, tenemos que poner el valor de la variable `ACCUMULO_MONITOR_BIND_ALL` a `true`. De esta forma conseguiremos que el proceso Monitor esté accesible en todas las interfaces.

Buscamos la línea del fichero `accumulo-env.sh` que cambia el valor de esta variable y que estará comentada. Quitamos la marca de comentario (`#`) para que tenga efecto en la configuración. Debería quedar de la siguiente forma:

```
export ACCUMULO_MONITOR_BIND_ALL="true"
```

Si dejamos este fichero tal y como está ahora mismo, hemos experimentado problema provocados por falta de memoria en la JVM (Java Virtual Machine) que ejecuta ciertos procesos de Accumulo. En concreto los procesos que han sufrido esta falta de memoria son “tserver”, “gc” y “monitor”.

Para solucionar esto, tenemos que cambiar en el fichero anterior las opciones de memoria de la JVM. Para ello se definen en el fichero variables que contienen las opciones para cada componente: ACCUMULO\_TSERVER\_OPTS, ACCUMULO\_MONITOR\_OPTS, ACCUMULO\_GC\_OPTS y ACCUMULO\_MASTER\_OPTS.

Debemos modificar las líneas donde se exportan dichas variables para que queden como las siguientes:

```
test -z "$ACCUMULO_TSERVER_OPTS" && export
ACCUMULO_TSERVER_OPTS="${POLICY} -Xmx768m -Xms768m "

test -z "$ACCUMULO_MASTER_OPTS" && export
ACCUMULO_MASTER_OPTS="${POLICY} -Xmx512m -Xms256m"

test -z "$ACCUMULO_MONITOR_OPTS" && export
ACCUMULO_MONITOR_OPTS="${POLICY} -Xmx256m -Xms64m"

test -z "$ACCUMULO_GC_OPTS" && export ACCUMULO_GC_OPTS="-Xmx256m
-Xms128m"
```

El cambio se encuentra en las opciones de la JVM del tipo “-XmxMAXm”. Esta opción indica la memoria máxima (en MB) que puede reservar para el “montón” (heap) de la JVM que ejecuta el proceso. Si superamos ese valor el proceso acabará y recibiremos un error al intentar reservar más memoria nuestro proceso. La otra opción (“-XmsMINm”) indica la memoria mínima (en MB) que será reservada para dicho fin.

Una vez hecho este cambio, podemos pasar a guardar y cerrar el fichero. El siguiente archivo que debemos editar es ~/accumulo-1.7.2/conf/accumulo-site.xml:

```
$ vim ~/accumulo-1.7.2/conf/accumulo-site.xml
```

Los procesos de Accumulo se comunican unos con otros usando una clave secreta. Esta debe cambiarse previamente a la inicialización de Accumulo usando una cadena de caracteres segura. Debemos buscar la propiedad `instance.secret` en el archivo anterior y cambiar su valor. En nuestro caso vamos a usar “123ARC” como clave ya que estamos en un entorno de pruebas, sin embargo, si estuviésemos en producción deberíamos usar una clave más compleja. La propiedad en el archivo XML anterior debe quedar como se indica a continuación:

```
<property>
  <name>instance.secret</name>
  <value>PASS1234</value>
  <description>
    A secret unique to a given instance that all servers
    must know in order to communicate with one another.
  </description>
</property>
```

A continuación, debemos indicar a Accumulo donde debe almacenar los datos dentro del HDFS. Para ello vamos a usar la propiedad `instances.volumes` indicando en su valor que queremos usar el directorio `/accumulo` dentro de nuestro almacenamiento HDFS. La propiedad debe quedar como sigue:

```
<property>
  <name>instance.volumes</name>
  <value>hdfs://localhost:9000/accumulo</value>
  <description>
    comma separated list of URIs for volumes.
  </description>
</property>
```

Por último, antes de guardar y cerrar el archivo `accumulo-site.xml`, debemos buscar la propiedad `trace.token.property.password` y cambiar su valor por algo seguro. Debemos recordar este valor ya que nos será necesario en futuros pasos. En nuestro caso, como estamos en un entorno de pruebas, hemos usado la contraseña "123456", pero en un entorno de producción deberíamos usar una clave más compleja. La propiedad debe quedar de la siguiente forma en el fichero:

```
<property>
  <name>trace.token.property.password</name>
  <value>123456</value>
</property>
```

Guardamos el fichero y lo cerramos.

Ya hemos terminado con la configuración. El siguiente paso consiste en inicializar Accumulo, es decir, que Accumulo cree los ficheros necesarios para funcionar en el HDFS. Para ello haremos uso del script `accumulo` disponible en el directorio `bin` de Accumulo.

Antes de ejecutar el script que nos inicializará Accumulo, debemos ejecutar HDFS y Zookeeper, ya que Accumulo hace uso de estas dos herramientas y si no se están ejecutando obtendremos un error. Una vez hecho esto, ejecutamos el siguiente comando:

```
$ ~/accumulo-1.7.2/bin/accumulo init
```

Se nos preguntará por un nombre para la instancia de Accumulo. Podemos usar cualquier nombre, pero en nuestro caso hemos usado “TFM”. También se nos preguntará por una clave, la cual debemos introducir la misma que usamos para la propiedad `trace.token.property.password` en el archivo `accumulo-site.xml`.

Una vez terminado el proceso de inicialización, ya tenemos nuestra base de datos Accumulo totalmente operativa y lista para funcionar. Podemos comprobar que todo el proceso de inicialización se ha desarrollado correctamente accediendo a la interfaz web de Hadoop HDFS a través de un navegador. Dentro de esta interfaz web seleccionaremos en la barra superior “Utilities → Browse the file system” y se nos mostrará un explorador del sistema de ficheros HDFS. Como se puede apreciar en la siguiente figura, se ha creado el directorio `/accumulo` tras el proceso de inicialización.

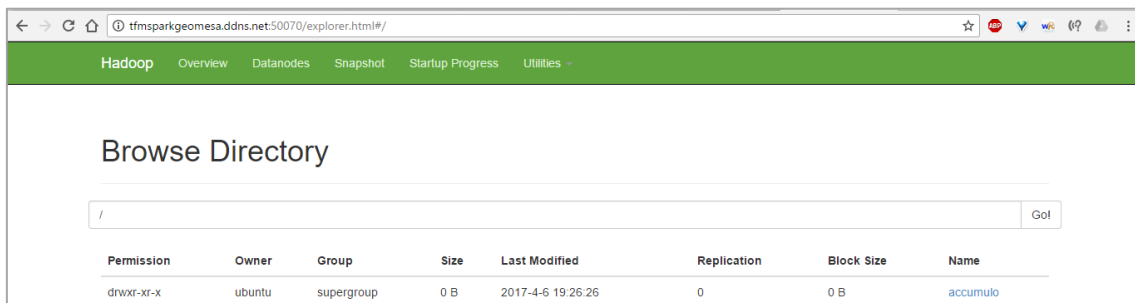


Figura 56. Explorador de HDFS

### 4.6.3 Ejecución

Una vez realizados los pasos previos, podemos pasar a ejecutar Accumulo. Para ello podemos usar el script `start-all.sh`:

```
$ ~/accumulo-1.7.2/bin/start-all.sh
```

La salida de este script es la mostrada en la siguiente figura:

```

ubuntu@ip-10-0-0-110: ~
ubuntu@ip-10-0-0-110:~$ accumulo-1.7.2/bin/start-all.sh
Starting monitor on localhost
WARN : Max open files on localhost is 1024, recommend 32768
Starting tablet servers .... done
Starting tablet server on localhost
WARN : Max open files on localhost is 1024, recommend 32768
2017-05-11 10:21:33,415 [fs.VolumeManagerImpl] WARN : dfs.datanode.synconclose s
et to false in hdfs-site.xml: data loss is possible on hard system reset or powe
r loss
2017-05-11 10:21:33,428 [server.Accumulo] INFO : Attempting to talk to zookeeper
2017-05-11 10:21:33,614 [server.Accumulo] INFO : ZooKeeper connected and initial
ized, attempting to talk to HDFS
2017-05-11 10:21:33,754 [server.Accumulo] INFO : Connected to HDFS
Starting master on localhost
WARN : Max open files on localhost is 1024, recommend 32768
Starting garbage collector on localhost
WARN : Max open files on localhost is 1024, recommend 32768
Starting tracer on localhost
WARN : Max open files on localhost is 1024, recommend 32768
ubuntu@ip-10-0-0-110:~$

```

Figura 57. Salida script de arranque de Accumulo

En la salida se puede apreciar que este script nos ejecuta todos los componentes de los que hemos hablado en la sección dedicada a la arquitectura de Accumulo. Además, podemos ver ciertos mensajes “WARNING” que nos indican que el número máximo de archivos abiertos en nuestro sistema es inferior al recomendado. Podemos hacer caso omiso a este mensaje ya que estamos en un entorno de pruebas y no vamos a dar al sistema mucha carga de trabajo.

Una vez termine la ejecución del script, tendremos ejecutándose nuestra base de datos Accumulo. Podemos acceder a su interfaz web usando la ip de nuestra instancia (o el nombre DNS) y el puerto 50095 en un navegador web.

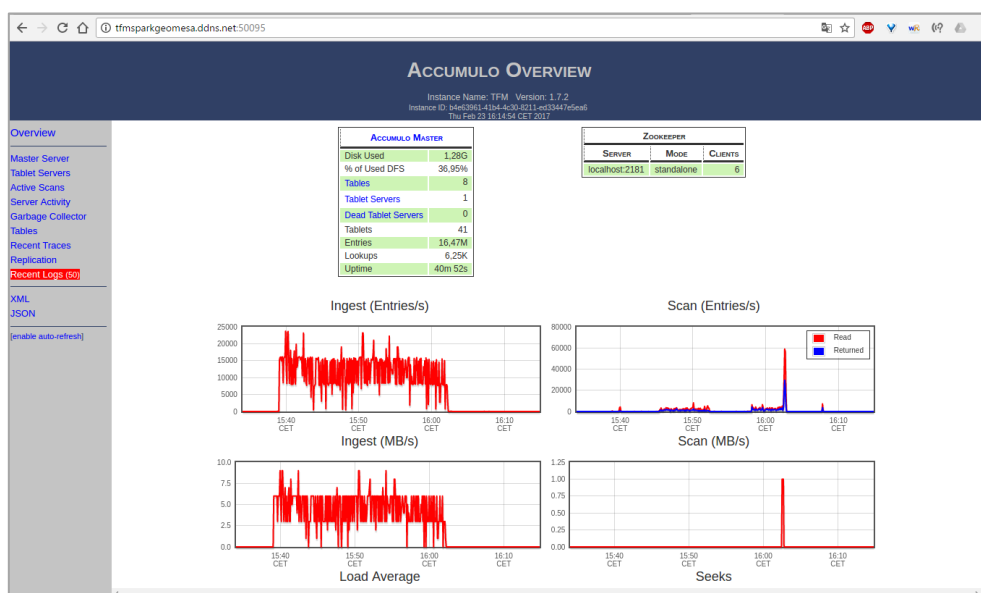


Figura 58. Interfaz Web Accumulo

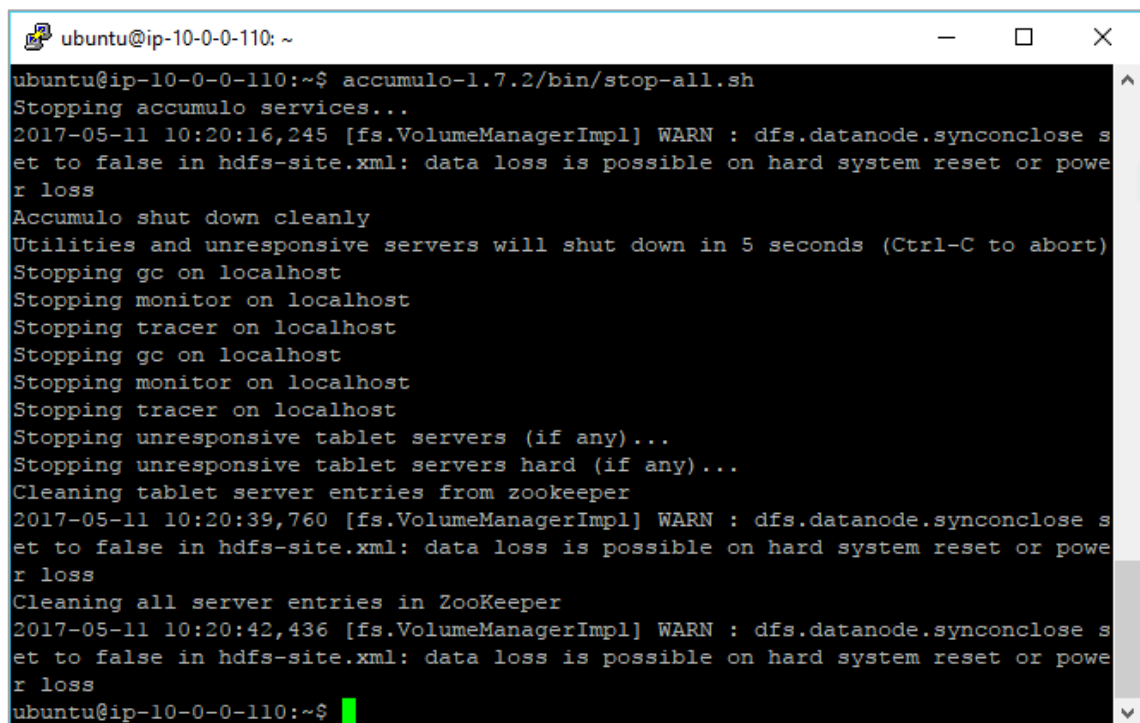


En la figura anterior se muestra dicha interfaz web. Como vemos la interfaz nos muestra mucha información interesante, empezando por las tablas superiores que contienen datos como uso del disco, tablas, tablets, etc. A continuación, se nos muestra gráficas sobre ingestión (ingest) y escaneos de datos, carga de trabajo, compactaciones y aciertos en la cache.

Para parar la ejecución de Accumulo, contamos con un script equivalente llamado `stop-all.sh`:

```
$ ~/accumulo-1.7.2/bin/stop-all.sh
```

La salida de este script, si todos los componentes son detenidos correctamente, será la mostrada en la siguiente figura.



```
ubuntu@ip-10-0-0-110: ~  
ubuntu@ip-10-0-0-110:~$ accumulo-1.7.2/bin/stop-all.sh  
Stopping accumulo services...  
2017-05-11 10:20:16,245 [fs.VolumeManagerImpl] WARN : dfs.datanode.synconclose s  
et to false in hdfs-site.xml: data loss is possible on hard system reset or powe  
r loss  
Accumulo shut down cleanly  
Utilities and unresponsive servers will shut down in 5 seconds (Ctrl-C to abort)  
Stopping gc on localhost  
Stopping monitor on localhost  
Stopping tracer on localhost  
Stopping gc on localhost  
Stopping monitor on localhost  
Stopping tracer on localhost  
Stopping unresponsive tablet servers (if any)...  
Stopping unresponsive tablet servers hard (if any)...  
Cleaning tablet server entries from zookeeper  
2017-05-11 10:20:39,760 [fs.VolumeManagerImpl] WARN : dfs.datanode.synconclose s  
et to false in hdfs-site.xml: data loss is possible on hard system reset or powe  
r loss  
Cleaning all server entries in ZooKeeper  
2017-05-11 10:20:42,436 [fs.VolumeManagerImpl] WARN : dfs.datanode.synconclose s  
et to false in hdfs-site.xml: data loss is possible on hard system reset or powe  
r loss  
ubuntu@ip-10-0-0-110:~$
```

Figura 59. Salida script de parada de Accumulo

En la figura puede observarse como se paran todos los componentes de Accumulo eliminando previamente los ficheros temporales.

## 4.7 Instalación Apache Spark

Llegamos al momento de la instalación de nuestro motor de computación distribuida: Apache Spark (38). Como veremos durante esta sección, en nuestro caso la instalación es muy similar a los componentes anteriores ya que podemos usar la versión de Spark precompilado para Hadoop 2.7 disponible en la página web de este componente.

### 4.7.1 Descarga e instalación

Apache Spark tiene unos requisitos mínimos, con lo cual tendremos que instalar una serie de paquetes para que funcione correctamente: Java, Python y Scala. Java ya lo tenemos instalado por lo que sólo será necesario instalar el resto. Para ello usaremos los siguientes comandos:

```
$ sudo apt-get install python2.7
$ sudo apt-get install python-minimal
$ sudo apt-get install Scala
```

Debemos asegurar que la versión de Scala instalada es la 2.11 ya que tanto Apache Spark como Geomesa hacen uso de esta versión. Además debemos añadir al fichero `~/.bashrc` una línea para exportar la variable `SCALA_HOME` y otra para añadir el directorio `bin` de Scala a la variable `PATH`. Estas líneas son las siguientes:

```
export SCALA_HOME=/usr/share/Scala
export PATH=$SCALA_HOME/bin:$PATH
```

Una vez hecho esto, podemos proceder a descargar Apache Spark. Lo podemos hacer con `curl`. El enlace de descarga lo podemos obtener de la página de descargas oficial de Apache Spark (<http://spark.apache.org/downloads.html>). En nuestro caso hemos descargado la versión de Spark 2.1.0, con el paquete “Pre-built for Hadoop 2.7 and later”.

Para ello vamos a cambiar el directorio actual al directorio de descargas que creamos previamente y procederemos con la descarga:

```
$ cd ~/Downloads
$ curl -O https://archive.apache.org/dist/spark/spark-2.1.0/spark-2.1.0-bin-hadoop2.7.tgz
```

A continuación, descomprimos el archivo descargado en el directorio personal:

```
$ tar -xvzf spark-2.1.0-bin-hadoop2.7.tgz -C ~
```

Al descomprimir se nos crea un directorio con el mismo nombre que el archivo comprimido. Este directorio contiene entre otros los directorios `bin` y `sbin` que contienen respectivamente binarios y scripts necesarios para Spark.

Debemos añadir estos directorios a la variable `PATH`. Para ello volveremos a modificar el archivo `/home/Ubuntu/.bashrc`. Introducimos la siguiente línea al final de dicho fichero:

```
export PATH=/home/ubuntu/spark-2.1.0-bin-hadoop2.7/bin:  
/home/ubuntu/spark-2.1.0-bin-hadoop2.7/sbin:$PATH
```

También debemos añadir la variable `SPARK_HOME` que contendrá el directorio base de Spark. Para ello introducimos la siguiente línea en el fichero anterior:

```
export SPARK_HOME=/home/ubuntu/spark-2.1.0-bin-hadoop2.7
```

#### 4.7.2 Configuración

La configuración que utilizará Apache Spark a la hora de ejecutar una aplicación mediante el comando `bin/spark-submit`, se encuentra en el fichero de configuración que se encuentra en el directorio `/conf`: `spark-defaults.conf`. Podemos modificar este fichero para configurar algunos parámetros de interés para la ejecución de nuestras aplicaciones.

El fichero está estructurado como un conjunto de parámetros formados por una pareja clave-valor. Nos basta con especificar el nombre de la variable que queremos modificar y el nuevo valor que queramos. Algunos de los parámetros que podemos modificar son los siguientes:

- **spark.app.name:** nombre de la aplicación. Aparecerá en la interfaz de usuario y en los logs.
- **spark.driver.cores:** número de cores que usará el proceso driver (worker).
- **spark.driver.maxResultSize:** límite en el tamaño total de resultados serializados por todas las particiones por cada acción de Spark.
- **spark.driver.memory:** cantidad de memoria usada por el proceso driver (worker).
- **spark.executor.memory:** cantidad de memoria usada por el proceso ejecutor (worker).

Para nuestro caso, añadiremos al fichero `spark-defaults.conf` las siguientes líneas:

```
spark.driver.memory 2g  
  
spark.driver.maxResultSize 2g  
  
spark.executor.memory 2g
```

Con esto, configuramos el valor de estas variables en 2GB (el valor por defecto para las tres es de 1GB). Si existiera algún conflicto entre estos valores y las capacidades de los workers (por ejemplo, worker de 1GB de memoria), el worker en cuestión no ejecutaría la aplicación, sin llegar a mostrar ningún error por pantalla.

También se han añadido estas dos líneas para especificar que los ficheros de logs se guarden en el sistema de almacenamiento distribuido HDFS.

```
spark.eventLog.enabled true

spark.eventLog.dir hdfs://localhost:9000/user/ubuntu/spark-eventLog
```

### 4.7.3 Ejecución

En esta subsección, explicaremos como ejecutar un master o un worker de Spark y a utilizar el script `spark-submit` para enviar trabajos de ejecución a Spark.

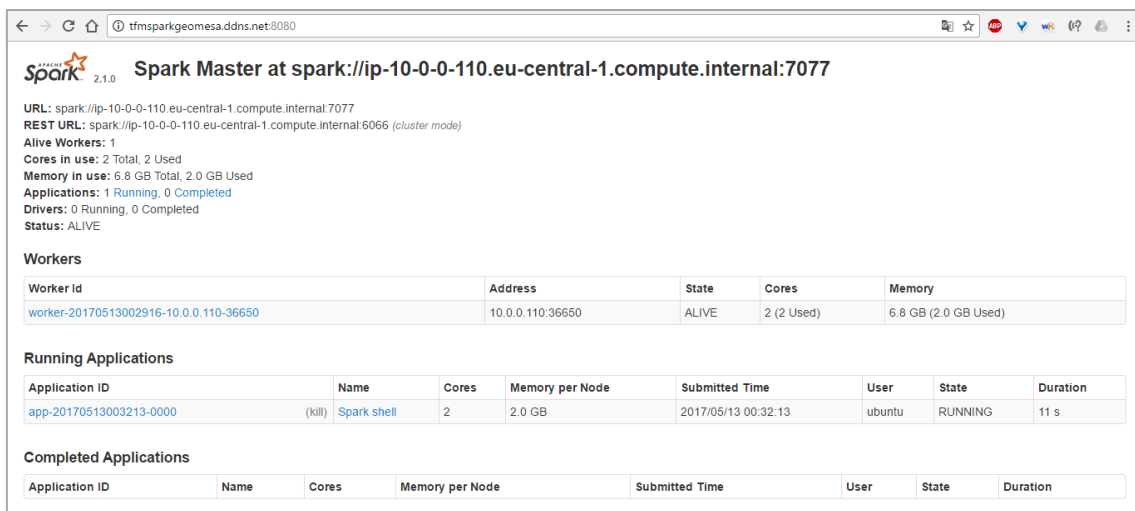
El primer paso será arrancar el máster de Spark. Para ello hacemos uso del script `start-master.sh` disponible en el directorio `sbin` de Spark:

```
$ ~/spark-2.1.0-bin-hadoop2.7/sbin/start-master.sh
```

Una vez ejecutemos este comando, ya tenemos nuestro master de Spark operativo y a la espera de workers que quieran conectarse al cluster o trabajos para ejecutar. Para ejecutar un worker y que se una a nuestro cluster podemos usar el script `start-slave.sh` pasándole la URL del master. Esta URL se puede consultar en la interfaz web del master. Es posible ejecutar dicho script de la siguiente forma:

```
$ ~/spark-2.1.0-bin-hadoop2.7/sbin/start-slave.sh spark://ip-10-0-0-110.eu-central-1.compute.internal:7077
```

Si accedemos mediante un navegador a la dirección de nuestra instancia y el puerto 8080 (<http://tfmsparkgeomesa.ddns.net:8080/>) podemos acceder a la interfaz web de Spark. En esta interfaz podemos ver los workers conectados al cluster, las aplicaciones que se están ejecutando y las aplicaciones cuya ejecución ya ha terminado. En la siguiente figura podemos ver una captura de esta interfaz web.



The screenshot shows the Apache Spark Master web interface. The title is "Spark Master at spark://ip-10-0-0-110.eu-central-1.compute.internal:7077". The interface displays the following information:

- URL:** spark://ip-10-0-0-110.eu-central-1.compute.internal:7077
- REST URL:** spark://ip-10-0-0-110.eu-central-1.compute.internal:6066 (cluster mode)
- Alive Workers:** 1
- Cores in use:** 2 Total, 2 Used
- Memory in use:** 6.8 GB Total, 2.0 GB Used
- Applications:** 1 Running, 0 Completed
- Drivers:** 0 Running, 0 Completed
- Status:** ALIVE

**Workers**

Worker Id	Address	State	Cores	Memory
worker-20170513002916-10.0.0.110-36650	10.0.0.110:36650	ALIVE	2 (2 Used)	6.8 GB (2.0 GB Used)

**Running Applications**

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20170513003213-0000	(kill) Spark shell	2	2.0 GB	2017/05/13 00:32:13	ubuntu	RUNNING	11 s

**Completed Applications**

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------

Figura 60. Interfaz web de Apache Spark

Ya tenemos nuestro cluster de Spark preparado para ejecutar aplicaciones. Para enviar una tarea a nuestro cluster, podemos usar el script `spark-submit` de la siguiente forma:

```
$ ~/spark-2.1.0-bin-hadoop2.7/bin/spark-submit \  
--master spark://dirección_IP_master:7077 \  
--class clase_a_ejecutar \  
--jars lista_de_jars_a_incluir_separados_por_coma \  
jar_que_incluye_la_clase_cuyo_método_main_se_va_a_ejecutar
```

Otra opción es ejecutar código `spark-scala` interactivamente haciendo uso de la `Spark-Shell`. Para poder hacer uso de esta Shell, debemos ejecutar el siguiente script:

```
$ spark-shell --master spark://dirección_IP_master:7077
```

En la siguiente figura se muestra una captura de dicha Shell:

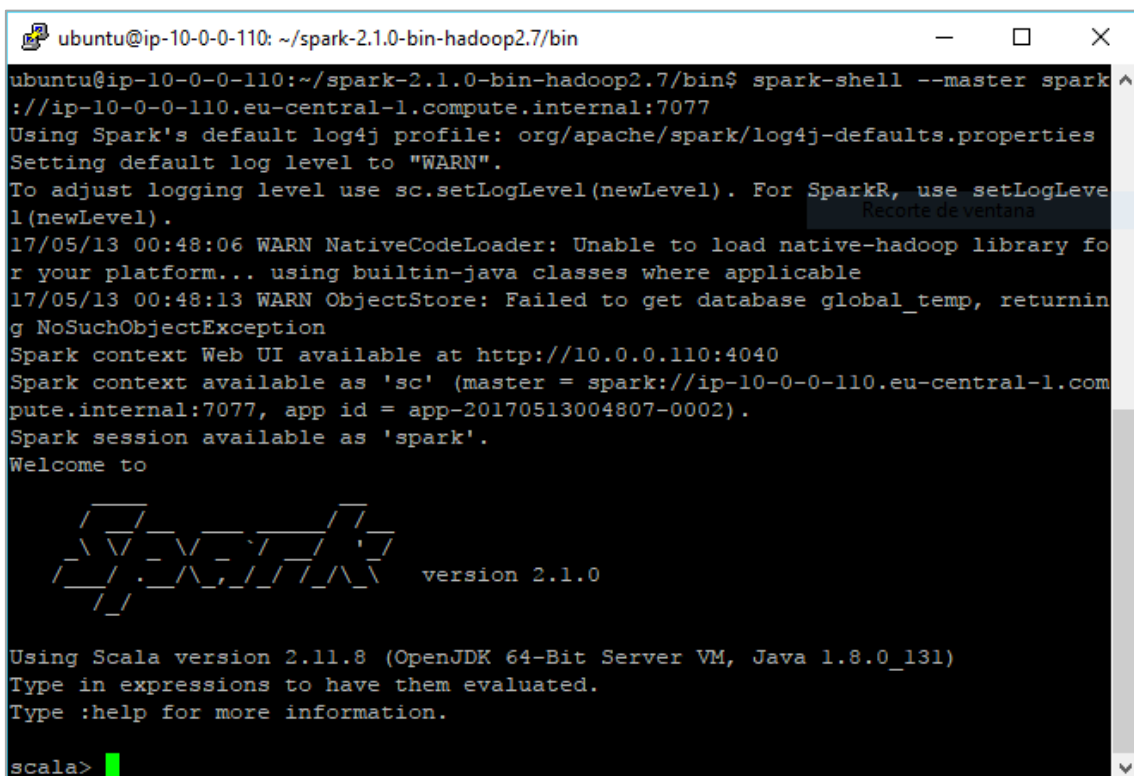


Figura 61. Spark-Shell

Por último, para parar la ejecución de un worker o master, podemos usar los scripts `stop-slave.sh` y `stop-master.sh` respectivamente:

```
$ ~/spark-2.1.0-bin-hadoop2.7/sbin/stop-slave.sh  
  
$ ~/spark-2.1.0-bin-hadoop2.7/sbin/ stop-master.sh
```

## 4.8 Instalación de GeoMesa

En esta sección mostraremos los pasos para instalar GeoMesa (39) y enlazarlo con nuestra base de datos Accumulo. Mostraremos la forma de instalación mediante descarga de binarios y mediante compilación del código fuente. Esto es necesario ya que GeoMesa está en constante cambio y a veces necesitaremos la última versión donde se corrigen ciertos bugs pero que aún no están corregido en los últimos binarios.

### 4.8.1 Instalación con binarios

El proceso para instalar GeoMesa usando los binarios disponibles en su página web es muy similar a los procesos de instalación de las anteriores herramientas. En este caso tendremos que descargar los binarios preparados para la base de datos Apache Accumulo de la web de GeoMesa (<http://www.geomesa.org/#downloads>).

Para ello vamos a cambiar el directorio actual al directorio de descargas que ya hemos usado en instalaciones anteriores:

```
$ cd ~/Downloads
```

Usaremos de nuevo la herramienta `curl` para la descarga del fichero comprimido que contiene los binarios de GeoMesa-Accumulo:

```
$ curl -O https://repo.locationtech.org/content/repositories/geomesa-releases/org/locationtech/geomesa/geomesa-accumulo-dist_2.11/1.3.1/geomesa-accumulo-dist_2.11-1.3.1-bin.tar.gz
```

A continuación, descomprimos el archivo descargado en el directorio personal:

```
$ tar -xvzf geomesa-accumulo-dist_2.11-1.3.1-bin.tar.gz -C ~
```

Al descomprimir se nos crea el directorio `geomesa-accumulo_2.11-1.3.1`. Debemos añadir el directorio `bin` a la variable `PATH`. Para ello volveremos a modificar el archivo `/home/Ubuntu/.bashrc` para que se actualice la variable con el valor adecuado cada vez que iniciamos sesión en nuestra instancia con el usuario “ubuntu” (usuario por defecto). Introducimos la siguiente línea al final de dicho fichero:

```
export PATH=/home/ubuntu/geomesa-accumulo_2.11-1.3.1/bin:$PATH
```

También debemos añadir la variable `GEOMESA_HOME` y `GEOMESA_LIB` que contendrá el directorio base de GeoMesa y el directorio con las librerías de GeoMesa respectivamente. Para ello introducimos las siguientes líneas en el fichero anterior:

```
export GEOMESA_HOME=/home/ubuntu/geomesa-accumulo_2.11-1.3.1
export GEOMESA_LIB=${GEOMESA_HOME}/lib
```

## 4.8.2 Instalación mediante compilación del código fuente

Para la instalación de GeoMesa mediante compilación del código fuente haremos uso de Git para obtener dicho código fuente y de Apache Maven para compilarlo. El primer paso por tanto será clonar el repositorio de GeoMesa usando Git:

```
$ git clone https://github.com/locationtech/geomesa.git
```

Esto nos creará la carpeta `~/geomesa` que contendrá e código fuente del proyecto GeoMesa. Cambiamos el directorio de trabajo a este último:

```
$ cd ~/geomesa
```

En este proyecto se usa Maven para la compilación y gestión de dependencias. El archivo `pom.xml` de Maven contenido en el directorio raíz de GeoMesa contiene una lista explícita de bibliotecas dependientes que se agruparán para cada módulo del programa. Para realizar la compilación se usa el siguiente comando:

```
$ mvn clean install
```

También es posible añadir la propiedad `skipTests` para saltarnos los test y terminar en un tiempo menor el proceso de compilación.

```
$ mvn clean install -DskipTests=true
```

## 4.8.3 Instalación de la biblioteca “Accumulo Distributed Runtime”

En el directorio `/home/ubuntu/geomesa-accumulo_2.11-1.3.1/dist/accumulo` podemos encontrar los archivos JAR que contienen código para servidor de Accumulo que deben estar disponibles en todas las instancias de `tabletServer` del cluster de Accumulo. Estos JARs contienen código GeoMesa y los iteradores de Accumulo necesarios para realizar consultas de datos a GeoMesa.

En dicho directorio existen dos JARs de los cuáles sólo es necesario uno de ellos. La versión de estos JARs debe coincidir con la versión del cliente de almacén de datos de GeoMesa. De lo contrario, es posible que las consultas no funcionen correctamente.

GeoMesa necesita la versión 2.1 o mayor de `commons-vfs2.jar`. Este JAR viene incluido en la versión 1.7.2 y posterior de Accumulo, pero para versiones anteriores este JAR debe ser actualizado en el directorio `$ACCUMULO_HOME/lib` de todos los servidores.

El JAR del que estamos hablando debe copiarse en el directorio `$ACCUMULO_HOME/lib/ext` en cada `tabletServer`:

```
$ cp ~/geomesa-accumulo_2.11-1.3.1/dist/accumulo/geomesa-accumulo-distributed-runtime_2.11-1.3.1.jar ~/accumulo-1.7.2/lib/ext/
```

No es necesario copiar este JAR a los servidores Master de Accumulo.

#### 4.8.4 Instalación de “Accumulo Command Line Tools”

GeoMesa trae consigo un conjunto de herramientas basadas en línea de comandos para gestionar las características de Accumulo. Estas herramientas pueden encontrarse en el directorio `~/geomesa-accumulo_2.11-1.3.1/bin`.

Para configurar dichas herramientas, debemos ejecutar el script `geomesa` ubicado en el directorio `bin`, pasándole por parámetro la opción `configure`. La salida de este comando es la siguiente:

```
$ ~/geomesa-accumulo_2.11-1.3.1/bin/geomesa configure
Warning: GEOMESA_ACCUMULO_HOME is not set, using
/home/ubuntu/geomesa-accumulo_2.11-1.3.1
Using GEOMESA_ACCUMULO_HOME as set: /home/ubuntu/geomesa-accumulo_2.11-1.3.1
Is this intentional? Y\n y
Warning: GEOMESA_LIB already set, probably by a prior
configuration.
Current value is /home/ubuntu/geomesa-accumulo_2.11-1.3.1/lib.

Is this intentional? Y\n y

To persist the configuration please update your bashrc file to
include:
export GEOMESA_ACCUMULO_HOME=/home/ubuntu/geomesa-accumulo_2.11-1.3.1
export PATH=${GEOMESA_ACCUMULO_HOME}/bin:$PATH
```

En la salida se puede apreciar que se nos indica que modifiquemos el fichero `~/ .bashrc` para añadir/modificar dos variables. Realizamos lo que nos piden, aunque es posible ahorrarnos la modificación a la variable `PATH` ya que fue modificada anteriormente con ese mismo valor.

Debido a restricciones de licencia, las dependencias para el soporte de archivos de forma (shapefile) deben ser instalados por separado. Para ello podemos ejecutar los siguientes scripts:

```
$ ~/geomesa-accumulo_2.11-1.3.1/bin/install-jai.sh
$ ~/geomesa-accumulo_2.11-1.3.1/bin/install-jline.sh
```

Para probar que todo se ha instalado de forma correcta podemos ejecutar el script `geomesa` sin argumentos y debemos obtener la ayuda para el uso de este comando.



#### 4.8.5 Actualización versión de Geomesa

Como ya hemos comentado anteriormente, GeoMesa está en constante cambio. Durante el desarrollo de nuestro proyecto nos ha ocurrido que hemos tenido que actualizar GeoMesa debido a que la nueva versión incluía cambios importantes o correcciones de bugs importantes.

El proceso para actualizar GeoMesa es sencillo, pero debemos de tener en cuenta todos los módulos que hemos instalado. Resumiendo, el proceso consistiría en los siguientes pasos:

1. Borrar el JAR copiado en el directorio `$ACCUMULO_HOME/lib/ext` ya que este estará desactualizado y tendremos que cambiarlo por la nueva versión.
2. Actualizar las variables que hemos exportado en el fichero `~/.bashrc`.
3. Eliminar la carpeta que contiene la versión anterior de GeoMesa.
4. Realizar la instalación de nuevo siguiendo los pasos descritos en las subsecciones 4.8.1, 4.8.3 y 4.8.4.

#### 4.8.6 Ejecución

El script `geomesa` que hemos utilizado anteriormente tiene distintas utilidades entre las que se encuentran la ingestión de datos por línea de comandos (lo veremos en un capítulo posterior), gestión de `featureTypes`, análisis de estadísticas de los datos almacenados, etc. En esta subsección se pretende mostrar algunos de las utilidades que hemos usado durante nuestro proyecto.

Para ver los `featureTypes` que hemos creado para un cierto catálogo podemos usar el script `geomesa` con la opción `get-type-names` y se nos mostrará la lista solicitada:

```
$ geomesa get-type-names -u root -p 123456 -c countries_catalog
```

También podemos ver el esquema de uno de los `featureTypes` que hemos creado, por ejemplo:

```
$ geomesa describe-schema -u root -p 123456 -c  
countries_catalog -f country
```

Como hemos dicho otra de las utilidades disponibles es la de sacar estadísticas simples de uno de nuestros catálogos. Los siguientes comandos nos muestran estadísticas simples e histogramas respectivamente:

```
$ geomesa stats-analyze -u root -p 123456 -c countries_catalog  
-f country
```

```
$ geomesa stats-histogram -u root -p 123456 -c  
countries_catalog -f country
```

## 4.9 Instalación Docker CE

Como prerrequisito para usar el kernel Apache Toree con Jupyter, es necesario tener instalado Docker. Para instalar Docker en nuestro sistema Ubuntu (40) hemos de ejecutar varios comandos por lo que hemos decidido escribir una sección exclusiva para Docker, al contrario que con otros prerrequisitos.

El primer paso es configurar el repositorio de Docker en nuestra instancia. Para ello hacemos uso de los siguientes comandos:

```
$ sudo apt-get -y install apt-transport-https ca-certificates curl
```

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

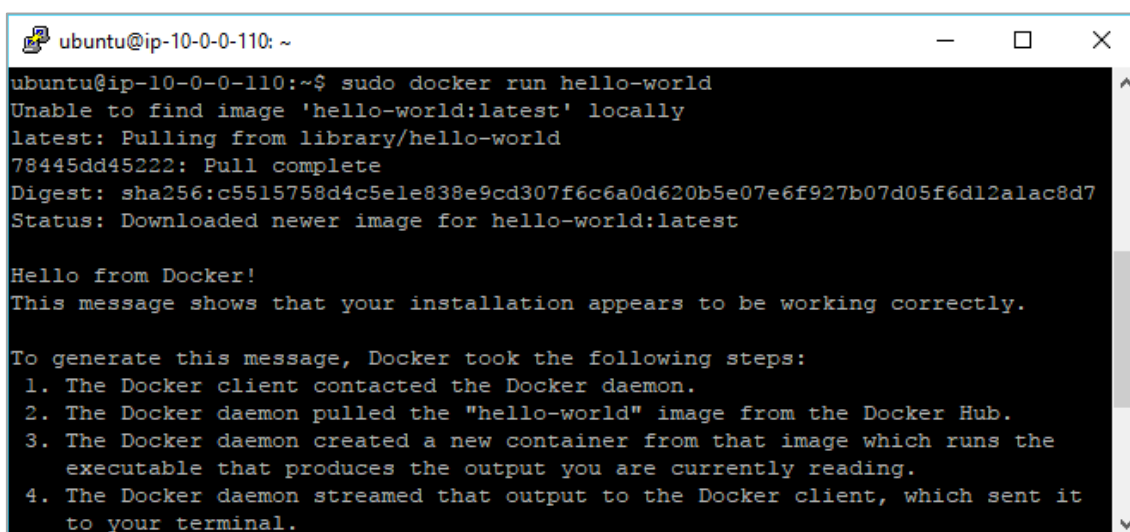
```
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

```
$ sudo apt-get update
```

A continuación, podemos pasar a la instalación de la última versión de Docker:

```
$ sudo apt-get -y install docker-ce
```

Por último, podemos comprobar si nuestra instalación ha tenido éxito ejecutando el comando “`sudo docker run hello-world`”. La salida debe ser parecida a la mostrada en la siguiente figura.

A terminal window titled 'ubuntu@ip-10-0-0-110: ~' with standard window controls. The terminal output shows the command 'sudo docker run hello-world' being executed. The output indicates that the 'hello-world:latest' image was not found locally and was pulled from the Docker Hub. The pull is complete, and the container is run, resulting in the message 'Hello from Docker!' and a confirmation that the installation is working correctly. A list of four steps is provided to explain the process: 1. Docker client contacted the daemon, 2. daemon pulled the image from Docker Hub, 3. daemon created a container from the image, and 4. daemon streamed the output to the client.

```
ubuntu@ip-10-0-0-110:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
78445ddd45222: Pull complete
Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12alac8d7
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```

Figura 62. Ejecución del contenedor Docker hello-world

## 4.10 Instalación Jupyter Notebook y Apache Toree

Como ya sabemos, Jupyter Notebook es una aplicación web que nos ofrece la posibilidad de crear documentos interactivos que pueden contener código ejecutable, visualizaciones y texto. A través del kernel Apache Toree, Jupyter puede ser usado para preparar análisis espaciotemporales en Scala y enviarlo a Spark. En esta sección, mostraremos los procesos de instalación de Jupyter (41) (42) y Apache Toree (43) (44). Además, también explicaremos la forma en la que hemos conectado estos componentes.

### 4.10.1 Requisitos previos

Para el uso conjunto de Jupyter y Apache Toree tendremos que tener instaladas previamente las herramientas Git, Make, Python, Docker, Pip y sbt.

De estas herramientas ya hemos explicado en secciones anteriores como instalar Git y Docket. En el resto de esta subsección se explicará brevemente el proceso de instalación de las herramientas restantes.

#### *Make*

Normalmente las distribuciones de Ubuntu vienen con la herramienta de compilación Make instalada. Si ese no es nuestro caso, podemos ejecutar el siguiente comando para instalarla:

```
$ sudo apt-get install make
```

#### *Python*

En el caso de no tener instalado Python 2.7 debemos ejecutar el siguiente comando:

```
$ sudo apt-get install build-essential python-dev
```

#### *Pip*

Para instalar pip en nuestra instancia usamos los siguientes comandos:

```
$ sudo apt-get install python-pip
```

```
$ pip install --upgrade pip
```

#### *sbt*

sbt es una herramienta para compilar proyectos escritos en Scala, Java y otros lenguajes. Para instalarlo en nuestro sistema vamos a usar los siguientes comandos:

```
$ echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee -a /etc/apt/sources.list.d/sbt.list
```

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 2EE0EA64E40A89B84B2DF73499E82A75642AC823
```

```
$ sudo apt-get update
```

```
$ sudo apt-get install sbt
```

Para comprobar la correcta instalación de sbt podemos crear un proyecto de prueba y ejecutarlo. Para ello usamos el siguiente comando:

```
$ sbt new sbt/Scala-seed.g8
```

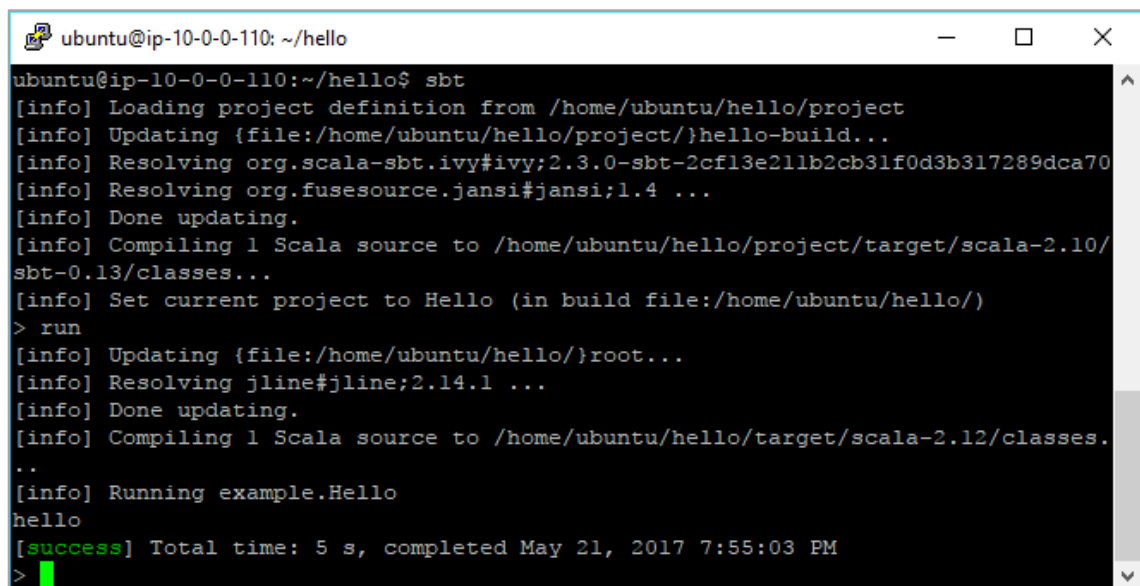
Se nos preguntará un nombre para el proyecto. Cuando termine la creación del nuevo proyecto, cambiamos el directorio actual al nuevo directorio que se ha creado. Una vez hecho esto podemos ejecutar el proyecto mediante los siguientes comandos:

```
$ cd hello
```

```
$ sbt
```

```
> run
```

La salida debe ser la siguiente:



```
ubuntu@ip-10-0-0-110: ~/hello
ubuntu@ip-10-0-0-110:~/hello$ sbt
[info] Loading project definition from /home/ubuntu/hello/project
[info] Updating {file:/home/ubuntu/hello/project/}hello-build...
[info] Resolving org.scala-sbt.ivy#ivy;2.3.0-sbt-2cf13e211b2cb31f0d3b317289dca70
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Compiling 1 Scala source to /home/ubuntu/hello/project/target/scala-2.10/sbt-0.13/classes...
[info] Set current project to Hello (in build file:/home/ubuntu/hello/)
> run
> run
[info] Updating {file:/home/ubuntu/hello/}root...
[info] Resolving jline#jline;2.14.1 ...
[info] Done updating.
[info] Compiling 1 Scala source to /home/ubuntu/hello/target/scala-2.12/classes.
..
[info] Running example.Hello
hello
[success] Total time: 5 s, completed May 21, 2017 7:55:03 PM
>
```

Figura 63. Ejecución proyecto de prueba de sbt

Tras esto, podemos eliminar la carpeta creada para este proyecto de prueba y comenzar con la instalación de Jupyter.

#### 4.10.2 Instalación Jupyter Notebook

Una vez tenemos instalado el gestor de paquetes de Python pip, la instalación de Jupyter es muy sencilla haciendo uso de este software. Basta con ejecutar el siguiente comando:

```
$ pip install -upgrade jupyter
```

Con esto ya tenemos disponible Jupyter mediante el comando `jupyter`.

### 4.10.3 Configuración Jupyter Notebook

Por defecto, un servidor de notebooks Jupyter se ejecuta localmente en 127.0.0.1:8888 y sólo es accesible desde nuestra propia instancia. Podemos acceder al servidor de notebooks desde un navegador web usando `http://127.0.0.1:8888`, pero como nuestra instancia de Amazon EC2 no cuenta con una interfaz gráfica, esto no es posible.

Es por ello que debemos cambiar la configuración para que Jupyter este accesible desde cualquier interfaz de nuestra instancia, incluyendo las interfaces con dirección IP pública. En esta subsección vamos a describir el proceso para realizar lo dicho anteriormente y hacer que Jupyter nos solicite una contraseña para poder usarlo, de esta forma añadimos cierta seguridad y no todo el mundo puede usar nuestro servidor.

El primer paso es verificar si tenemos un fichero de configuración de notebooks, con el nombre `jupyter_notebook_config.py`. La localización por defecto para este fichero de configuración es nuestro directorio de Jupyter: `~/ .jupyter`.

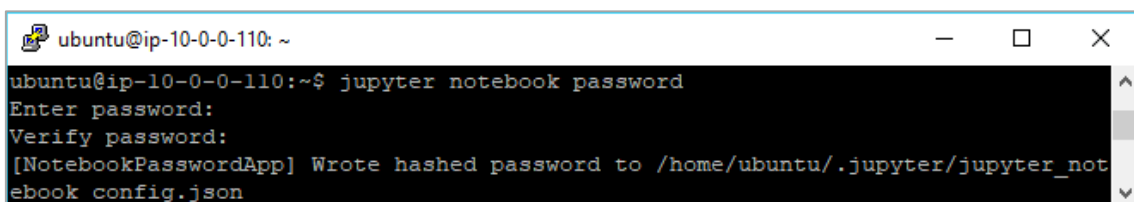
En caso de no existir, podemos crear un fichero de configuración para el servidor de notebooks usando el siguiente comando:

```
$ jupyter notebook --generate-config
```

El siguiente paso es indicar una contraseña para nuestro servidor de notebooks. Para ello podemos usar el siguiente comando:

```
$ jupyter notebook password
```

Como se puede observar en la siguiente figura, que representa una captura de la salida del comando anterior, se nos solicita la nueva contraseña y la verificación una vez la introducimos.



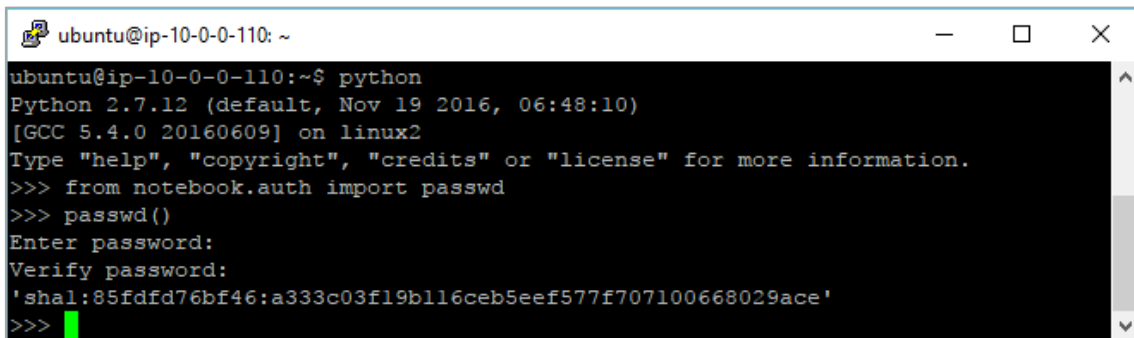
```
ubuntu@ip-10-0-0-110: ~  
ubuntu@ip-10-0-0-110:~$ jupyter notebook password  
Enter password:  
Verify password:  
[NotebookPasswordApp] Wrote hashed password to /home/ubuntu/.jupyter/jupyter_notebook_config.json
```

Figura 64. Comando “`jupyter notebook password`”.

El siguiente paso es modificar el fichero de configuración que hemos generado anteriormente, pero antes necesitamos obtener el hash de la contraseña que acabamos de indicar para el acceso a Jupyter. Para ello, una forma sencilla de hacerlo es invocando la función `passwd()` en una Shell Python:

```
$ Python
>>> from notebook.auth import passwd
>>> passwd()
```

Esta función nos pedirá que introduzcamos nuevamente la contraseña y la verifiquemos. Debemos introducir la misma contraseña y nos devolverá el hash (SHA1) de esta, como podemos observar en la siguiente figura.



```
ubuntu@ip-10-0-0-110: ~
ubuntu@ip-10-0-0-110:~$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from notebook.auth import passwd
>>> passwd()
Enter password:
Verify password:
'sha1:85fdfd76bf46:a333c03f19b116ceb5eef577f707100668029ace'
>>>
```

Figura 65. Ejecución de la función passwd() en una Shell Python

Debemos copiar la cadena de caracteres que nos devuelve la función para usarla a continuación en la modificación de nuestro fichero de configuración:

```
$ vim ~/.jupyter/jupyter_notebook_config.py
```

En este fichero debemos insertar las líneas que vamos a ir comentando a continuación, o bien, quitar la marca de comentario y modificar las líneas existentes en el fichero.

La primera línea que debemos insertar hace referencia a la dirección IP en la que estará escuchando nuestro servidor de notebooks. Debemos indicar un asterisco para que nuestro servidor escuche en todas las interfaces:

```
c.NotebookApp.ip = '*'
```

A continuación, debemos indicar el hash de la contraseña que configuramos anteriormente:

```
c.NotebookApp.password =
u'sha1:be0d6efb8644:66f9adf40c72176c39ae670ea24da323516e0ca9'
```

También debemos indicar que queremos que se solicite a los usuarios la contraseña para poder acceder al servidor:

```
c.NotebookApp.password_required = True
```

Por último, debemos modificar el puerto donde se ejecuta la interfaz web de Jupyter, ya que el puerto 8888 ya está en uso por la interfaz web de Spark:

```
c.NotebookApp.port = 10000
```

#### 4.10.4 Instalación Toree

Para comenzar la instalación de Apache Toree, debemos clonar el repositorio del proyecto usando nuestro cliente git:

```
$ git clone https://github.com/apache/incubator-toree.git
```

A continuación, cambiamos el directorio de trabajo al del proyecto Toree y nos cambiamos de rama con git para movernos a una con soporte para Spark 2.0 y posterior:

```
$ cd incubator-toree
```

```
$ git checkout v0.2.0-dev1
```

El siguiente paso es compilar nuestro proyecto. Durante la compilación, se hace uso de Docker y se descarga una imagen de unos 5 GB de tamaño, por lo que es importante que tengamos suficiente espacio en nuestro almacenamiento secundario.

```
$ make clean
```

```
$ make release
```

Por último, debemos instalar el proyecto compilado en nuestro gestor de paquetes python, pip. Para ello ejecutamos los siguientes comandos:

```
$ sudo -H pip install --upgrade ./dist/toree-pip/toree-0.2.0.dev1.tar.gz
```

```
$ pip freeze |grep toree
```

Con esto ya tenemos Apache Toree instalado en nuestro gestor de paquetes pip y listo para enlazarlo con Jupyter.

#### 4.10.5 Enlazar Jupyter y Apache Toree

Para enlazar Jupyter con el kernel Apache Toree hemos creado un script por comodidad, ya que realmente se ejecuta tan sólo un comando, pero debido a la longitud de dicho comando, hemos decidido incluirlo en un script para poder estructurar dicho comando. El comando en cuestión es “jupyter toree install”.

A este comando hemos de pasarle los parámetros necesarios para indicarle el nombre del kernel que queremos crear en Jupyter, el directorio de instalación de Spark y las opciones que se le pasan a Spark, entre las que se encuentran la dirección del

Master de Spark y la lista de JARs que queremos tener disponibles en el cluster Spark. En este caso, la lista de JARs que hemos pasado es bastante amplia y de ahí que el comando tenga un tamaño considerable. Los JARs que hemos listado son bibliotecas de Geomesa que incluyen las clases necesarias para funcionar con Spark.

A continuación, se detalla el contenido del script mencionado anteriormente y al que hemos llamado `jupyter-toree-install.sh`:

```
#!/bin/sh

# bundled GeoMesa Accumulo Spark and Spark SQL runtime JAR
# (contains geomesa-accumulo-spark, geomesa-spark-core, geomesa-
spark-sql, and dependencies)
jars="file://$GEOMESA_ACCUMULO_HOME/dist/spark/geomesa-accumulo-
spark-runtime_2.11-1.3.1.jar"

# to use the converter or GeoTools RDD providers
jars="$jars,file:///home/ubuntu/geomesa-geomesa_2.11-
1.3.1/geomesa-spark/geomesa-spark-converter/target/geomesa-
spark-converter_2.11-1.3.1.jar"
jars="$jars,file:///home/ubuntu/geomesa-geomesa_2.11-
1.3.1/geomesa-spark/geomesa-spark-geotools/target/geomesa-spark-
geotools_2.11-1.3.1.jar"

# to work with shapefiles (requires
$GEOMESA_ACCUMULO_HOME/bin/install-jai.sh)
jars="$jars,file://$GEOMESA_ACCUMULO_HOME/lib/jai_codec-
1.1.3.jar"
jars="$jars,file://$GEOMESA_ACCUMULO_HOME/lib/jai_core-
1.1.3.jar"
jars="$jars,file://$GEOMESA_ACCUMULO_HOME/lib/jai_imageio-
1.1.jar"

# to include prepackaged converters for publically available
data sources
jars="$jars,file://$GEOMESA_ACCUMULO_HOME/lib/geomesa-
tools_2.11-1.3.1-data.jar"

# to include an interface for the Leaflet spatial visualization
library
jars="$jars,file://$GEOMESA_ACCUMULO_HOME/lib/geomesa-jupyter-
leaflet_2.11-1.3.1.jar"
jars="$jars,file:///home/ubuntu/geomesa-geomesa_2.11-
1.3.1/geomesa-accumulo/geomesa-accumulo-compute/target/geomesa-
accumulo-compute_2.11-1.3.1.jar"

jupyter toree install \
  --replace \
  --user \
  --kernel_name "Geomesa Spark 1.3.1" \
  --spark_home=${SPARK_HOME} \
  --spark_opts="--master spark://ip-10-0-0-110.eu-central-
1.compute.internal:7077 --jars $jars"
```



Con esto ya tendríamos nuestro kernel Apache Toree enlazado con nuestro servidor de notebooks Jupyter.

Sin embargo, para que todas las funciones de GeoMesa puedan funcionar adecuadamente con Spark, es necesario modificar la configuración del serializador de Spark para que este coincida con el que usa Geomesa. Para esto, debemos introducir las siguientes líneas en el fichero de configuración de Spark (`~/spark-2.1.0-bin-hadoop2.7/conf/spark-defaults.conf`):

```
spark.serializer org.apache.spark.serializer.KryoSerializer
spark.kryo.registrator
org.locationtech.geomesa.spark.GeoMesaSparkKryoRegistrator
spark.kryo.registrationRequired false
```

#### 4.10.6 Ejecución

Para la ejecución de Jupyter Notebook basta con ejecutar el siguiente comando desde cualquier directorio:

```
$ jupyter notebook
```

Con esto ya tendremos nuestro servidor corriendo en nuestra instancia y podremos acceder a este mediante un navegador web en la dirección `http://tfmsparkgeomesa.ddns.net:10000`. Al acceder a esta dirección web por primera vez, se nos solicitará la contraseña que hemos definido en nuestra configuración de Jupyter.

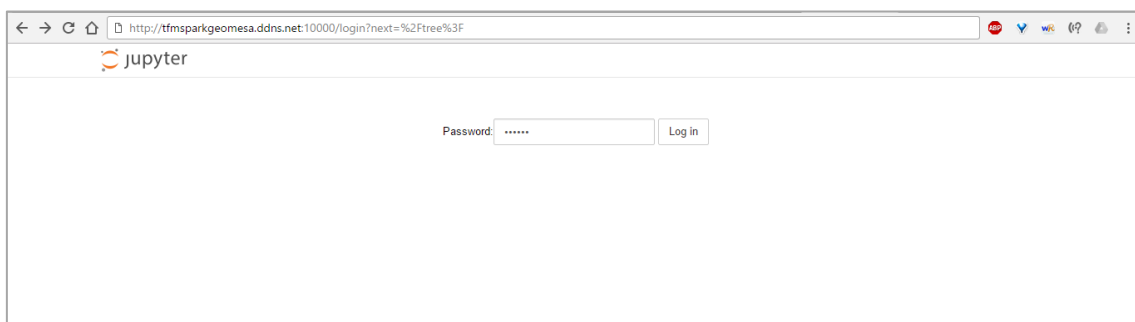


Figura 66. Solicitud de contraseña en Jupyter

Una vez introducida correctamente la contraseña, se nos mostrará la interfaz que se muestra en la siguiente figura, en la que se listan los archivos y directorios disponibles en el sistema de archivo de nuestra instancia para que podamos abrir notebooks que hayamos guardado previamente. Es importante destacar que desde esta interfaz se tiene acceso al directorio desde el que hemos ejecutado el comando de inicio de nuestro servidor Jupyter y sus subdirectorios, pero nunca los directorios superiores. Por ello,

para mejorar la seguridad, es recomendable crear un directorio que contenga los notebooks que creamos y ejecutar el comando de inicio del servidor Jupyter desde este directorio.

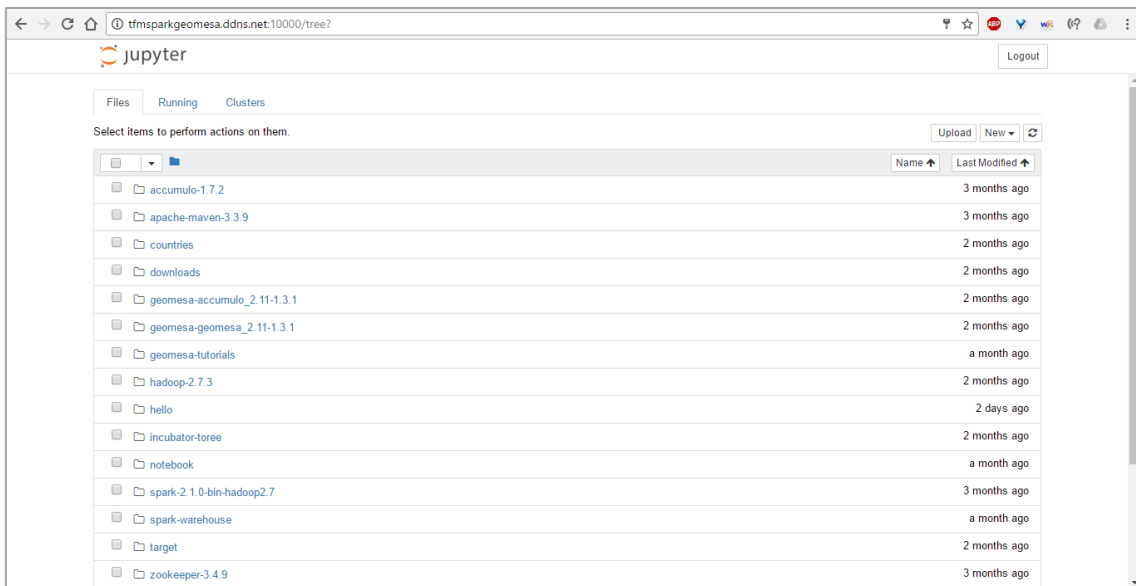


Figura 67. Interfaz Jupyter - Listado de archivos

Desde la interfaz mostrada en la figura anterior podemos crear un nuevo notebook que se ejecute sobre el kernel Apache Toree que hemos instalado previamente. Para ello debemos pulsar en el botón “New” situado en la esquina superior derecha de esta interfaz y se nos desplegará una lista con los kernels disponibles como la mostrada en la siguiente figura.

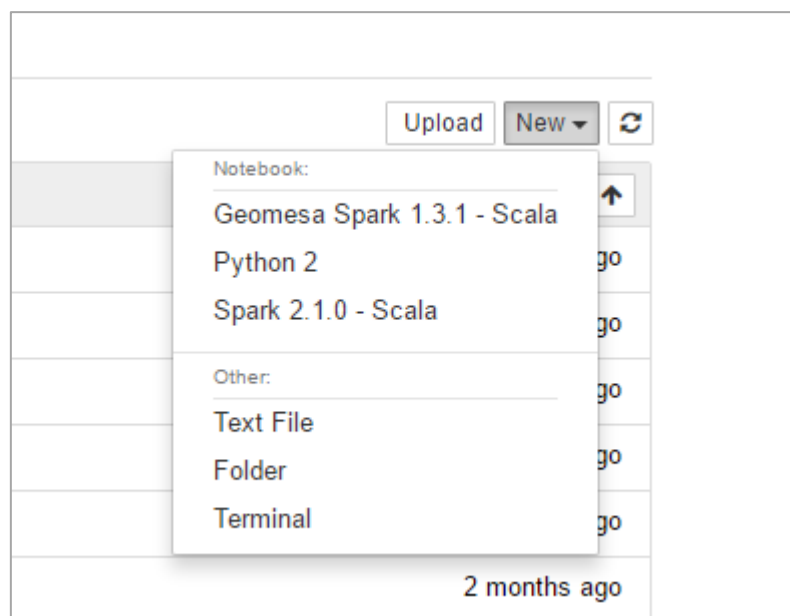


Figura 68. Lista de kernels disponibles en Jupyter

El kernel que nos interesa es el que instalamos en la subsección anterior con el nombre de “Geomesa Spark 1.3.1 – Scala”. Seleccionamos este y se nos mostrará la interfaz de edición de notebooks que se muestra en la siguiente figura.

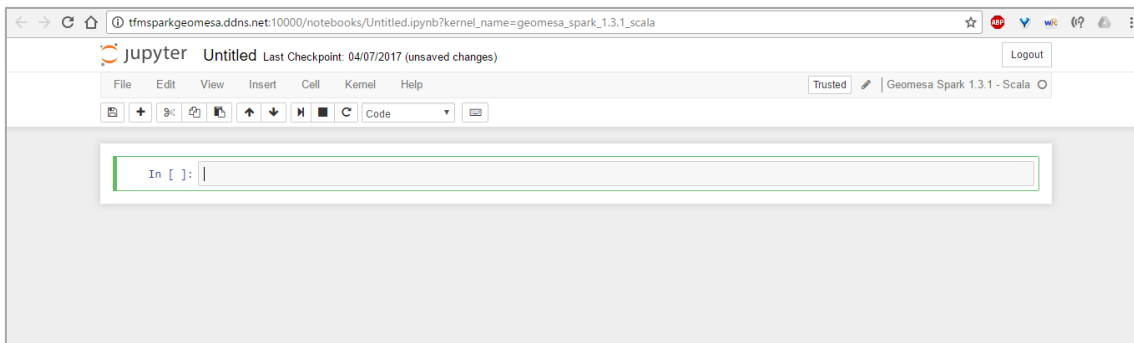


Figura 69. Interfaz de edición de notebooks de Jupyter

Antes de comenzar con la ejecución de nuestro notebook debemos esperar a que el kernel seleccionado termine de cargarse e inicializarse, lo que nos será notificado con un el mensaje “Connected” en la esquina superior derecha. Una vez ocurra esto, si accedemos a la interfaz web de Spark (que debe ejecutarse previamente), podemos ver que se está ejecutando una aplicación con el nombre “Apache Toree” y que se corresponde con nuestro kernel.

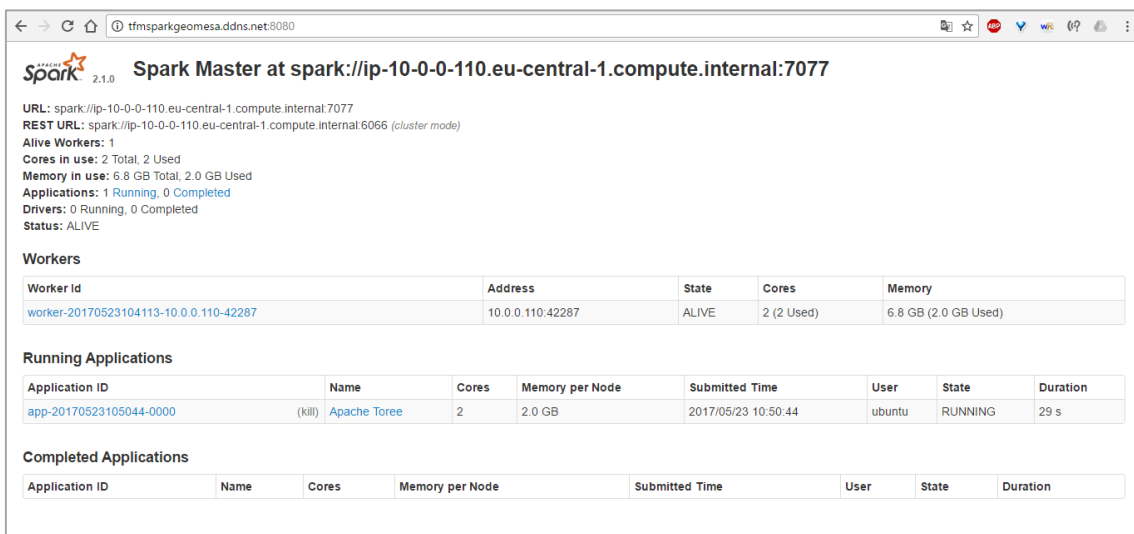


Figura 70. Interfaz web de Spark ejecutando Apache Toree

Por último, si queremos para la ejecución del kernel sobre el que se ejecuta uno de nuestros notebooks, por ejemplo, porque ya no queramos trabajar más sobre este, debemos acceder a la pestaña “Running” de la interfaz principal de Jupyter. En esta pestaña se nos mostrará un listado con los kernels que se están ejecutándose como podemos ver en la figura mostrada a continuación. Para parar la ejecución de un kernel pulsamos sobre “Shutdown”.



Figura 71. Listado de kernels ejecutándose en Jupyter

Para parar por completo el servidor Jupyter de notebooks, debemos parar el proceso creado al ejecutar el comando “jupyter notebook”. Para ello podemos usar `Ctrl-C` si el proceso se está ejecutando en *foreground* o ejecutar “`kill -9 PID`” si el proceso se ejecuta en *background*.

#### 4.11 Script para el arranque de servicios

Llegados a este punto ya tenemos nuestra infraestructura completamente funcional, aunque aún se puede mejorar, como veremos en capítulos posteriores, incluyendo automatización para el despliegue de workers de Spark que nos permitan aumentar nuestra potencia de computo. En apartados posteriores también veremos los detalles básicos para usar la infraestructura que hemos desarrollado, como ingerir datos en GeoMesa o la ejecución de algún ejemplo que haga uso de todas las herramientas.

Pero antes, teniendo en cuenta que son muchas las herramientas que tenemos que gestionar, hemos decidido que lo mejor es escribir un script que se encargue de comenzar la ejecución de todas estas herramientas en el orden adecuado. Hemos nombrado a este script “`start-script.sh`” y su contenido es el siguiente:

```
#!/bin/bash

# Start the HDFS System
start-dfs.sh

# Start Zookeeper server
zkServer.sh start

# Start all Accumulo components
cd ~/accumulo-1.7.2/
./bin/start-all.sh

# Start Spark master
cd ~/spark-2.1.0-bin-hadoop2.7/
./sbin/start-master.sh

# Start a Spark worker and link it to the master
./sbin/start-slave.sh spark://ip-10-0-0-110.eu-central-
1.compute.internal:7077
```

```
# Start the Jupyter notebook server  
cd ~/notebook/  
jupyter notebook &
```

Como se puede observar, el script básicamente va ejecutando de forma secuencial los comandos para la ejecución que hemos comentado para cada componente en las secciones anteriores.

## 5 Ingestión (ingest) de datos con GeoMesa-Accumulo

Una vez terminada la implementación de nuestro entorno para el análisis de datos geográficos, nuestro siguiente objetivo consiste en dar una idea general sobre los pasos a seguir para su utilización, es decir, la ejecución de código Spark que haga uso de GeoMesa para llevar a cabo un análisis de datos geográficos.

Antes de proceder a ejecutar cualquier análisis, es requisito indispensable el tener disponible los datos a analizar en nuestra instalación de GeoMesa. En este caso hemos instalado GeoMesa sobre Accumulo por lo que haciendo uso de GeoMesa tendremos que importar los datos que vamos a analizar a nuestra base de datos Accumulo siguiendo el principio básico de indexación de GeoMesa (representar las tres dimensiones de longitud, latitud y tiempo con un solo valor). Para ello debemos representar los datos mediante un clave con la estructura que ya vimos en la subsección 2.7.1.

En la documentación de GeoMesa, a este proceso se le da el nombre de ingestión (ingest) de datos y consiste básicamente en importar los datos en nuestra base de datos Accumulo usando la estructura de índice que hemos comentado. Para realizar esta tarea existen diferentes procedimientos de los cuales vamos a mostrar dos de ellos a partir de ejemplos que hemos llevado a cabo durante el desarrollo de este Trabajo Fin de Máster. En concreto nuestros dos métodos consisten en importar programáticamente los datos mediante técnicas MapReduce, lo que nos da mayor libertad a la hora de definir la estructura de los datos importados, o importar los datos haciendo uso de herramientas de línea de comandos de GeoMesa que fueron instaladas al mismo tiempo.

### 5.1 Ingestión de datos usando técnicas MapReduce

El primer procedimiento que vamos a comentar consiste en la ingestión de datos mediante técnicas MapReduce. Para ello vamos a importar a nuestra instancia de Accumulo datos provenientes de la base de datos GDELT (Global Database of Events, Language, and Tone).

El Proyecto GDELT (45) monitorea las noticias de televisión, noticias impresas y noticias web de todo el mundo en más de 100 idiomas e identifica a las personas, lugares, organizaciones, temas, fuentes, emociones, imágenes y eventos que impulsan nuestra sociedad global cada segundo de cada día, creando una plataforma abierta y gratuita para la computación en todo el mundo. La base de datos contiene estos eventos actualizados cada día. Pueden consultarse los atributos asociados a los eventos

en su documentación, concretamente esta información puede encontrarse en el enlace: [http://data.gdeltproject.org/documentation/GDELT-Data\\_Format\\_Codebook.pdf](http://data.gdeltproject.org/documentation/GDELT-Data_Format_Codebook.pdf).

A continuación, vamos a describir el proceso para obtener un subconjunto de estos datos y usar procesamiento MapReduce para importar estos datos en nuestra base de datos Accumulo.

Podemos descargar los datos de GDELT en el siguiente enlace: <http://data.gdeltproject.org/events/index.html>. Queremos descargar sólo un subconjunto de los datos ya que la base de datos GDELT almacena eventos desde 1979 y descargar todos estos eventos nos supondría tener que procesar demasiados eventos, algo que no queremos para este ejemplo. Podemos descargar por ejemplo el subconjunto de eventos ocurridos en febrero de este mismo año. Para ello vamos a crear un directorio (~gdelt) donde descargar estos datos y vamos a usar la herramienta `wget` para obtenerlos. También usaremos `md5sum` para verificar el hash de los datos descargados:

```
$ mkdir ~/gdelt && cd ~/gdelt
$ wget http://data.gdeltproject.org/events/md5sums
$ for file in `cat md5sums | cut -d' ' -f3 | grep '^201702'`;
do wget http://data.gdeltproject.org/events/$file ; done
$ md5sum -c md5sums 2>&1 | grep '^201702'
```

Esto nos descargará un archivo comprimido “.zip” por cada día de febrero de 2017 y cada uno de estos contendrá un archivo con extensión “.csv” conteniendo los eventos ocurridos el día indicado en el nombre del zip. Debemos descomprimir estos archivos y unificarlo en un solo archivo con extensión “.tsv” almacenado en nuestra instancia HDFS. Para ello podemos usar el siguiente comando:

```
$ (ls -l *.zip | xargs -n 1 zcat) | hadoop fs -put -
/gdelt/uncompressed/gdelt.tsv
```

Con esto ya tenemos accesibles los datos para pasar a aplicar nuestro procesamiento MapReduce. En concreto vamos a aplicar un procesamiento MapReduce disponible como tutorial en la página de GeoMesa.

El siguiente paso será obtener el código de dicho tutorial. Para ello podemos clonar el repositorio de GitHub:

```
$ git clone https://github.com/geomesa/geomesa-tutorials.git
$ cd geomesa-tutorials
```

Tenemos que asegurarnos de que las versiones de GeoMesa, Accumulo, Zookeeper y Hadoop especificadas en el fichero `~/geomesa-tutorials/pom.xml` se corresponden con las versiones que tenemos instaladas. Si las versiones son correctas, podemos pasar a compilar el tutorial que vamos a ejecutar:

```
$ cd geomesa-examples-gdelt
$ mvn clean install
```

Con esto podemos pasar a la ejecución de nuestro trabajo de ingestión MapReduce:

```
hadoop jar geomesa-examples-gdelt/target/geomesa-examples-
gdelt-1.3.0.0-m3-SNAPSHOT.jar \
  com.example.geomesa.gdelt.GDELTIngest \
  -instanceId TFM \
  -zookeepers localhost:2181 \
  -user root -password 123456 \
  -tableName gdelt -featureName event \
  -ingestFile hdfs:///gdelt/uncompressed/gdelt.tsv
```

Dependiendo de la cantidad de datos que hayamos descargado de la base de datos GDELT, esta ejecución puede llegar a tardar varias horas en completarse. En concreto, para la cantidad de datos especificada en este documento, el tiempo de ejecución fue cercano a las dos horas.

### 5.1.1 Estructura del código MapReduce

A continuación, pasaremos a explicar el código del tutorial que acabamos de ejecutar ya que este sigue la estructura general que debemos seguir en caso de querer ingerir otros datos haciendo uso de MapReduce.

GeoTools utiliza un `SimpleFeatureType` para representar el esquema de las `SimpleFeatures` individuales creadas a partir de los datos de GDELT. Podemos crear fácilmente un esquema de este tipo usando la clase `org.geotools.data.DataUtilities`.

Podemos definir la cadena de esquema como una lista separada por comas de descriptores de atributo de la forma `"Nombre_atributo:tipo_atributo"`, por ejemplo `"Year:Integer"`. Algunos atributos pueden tener un tercer término con una indicación extra, por ejemplo `"Geom:Point:srid=4236"`, y además, el atributo de geometría por defecto se precede por un asterisco. Por ejemplo, una cadena de esquema completa para un `SimpleFeatureType` que describe una ciudad con un punto de latitud/longitud, un nombre y una población puede ser `"*geom:Point:srid=4326,cityname:String,population:Integer"`.



En el ejemplo que hemos ejecutado, se crea el SimpleFeatureType para los eventos de GDELT de la siguiente forma:

```
static List<String> attributes = Lists.newArrayList(  
    "GLOBALEVENTID:Integer",  
    "SQLDATE:Date",  
    "MonthYear:Integer",  
    "Year:Integer",  
    "FractionDate:Float",  
    //...  
    "*geom:Point:srid=4326"  
);  
String spec = Joiner.on(",").join(attributes);  
SimpleFeatureType featureType = DataUtilities.createType(name,  
spec);
```

Como vemos en el fragmento de código anterior, primero creamos una cadena de esquema para definir el SimpleFeatureType asociado a los eventos de GDELT y después, usamos esta cadena junto a la clase `org.geotools.data.DataUtilities` para crear dicho tipo.

Tras crear el SimpleFeatureType de GDELT, tenemos que decirle a Geomesa que campo debe usar como índice temporal. Para ello debemos usar la siguiente línea:

```
featureType.getUserData().put(SimpleFeatureTypes.DEFAULT_DATE_  
KEY, "SQLDATE");
```

Por último, antes de enviar el trabajo MapReduce a nuestro cluster Hadoop, debemos crear el nuevo SimpleFeatureType en nuestra instancia de GeoMesa:

```
DataStore ds = DataStoreFinder.getDataStore(dsConf);  
ds.createSchema(featureType);
```

Cuando configuremos nuestro trabajo MapReduce debemos especificar la clase *Mapper*. En el método `setup` de dicha clase debemos crear un objeto `SimpleFeatureBuilder` que actuará como constructor del SimpleFeatureType que creamos en la fase de inicialización. El contenido de dicho método es el siguiente:

```
public void setup(Mapper<LongWritable, Text, Text,  
SimpleFeature>.Context context) throws IOException,  
InterruptedException {  
    super.setup(context);  
  
    String featureName =  
context.getConfiguration().get(GDELTIngest.FEATURE_NAME);  
    try {  
        SimpleFeatureType featureType =  
GDELTIngest.buildGDELTFeatureType(featureName);  
        featureBuilder = new SimpleFeatureBuilder(featureType);  
    } catch (Exception e) {
```

```

        throw new IOException("Error setting up feature type",
e);
    }
}

```

La entrada al método `map` es una sola línea del archivo con extensión “.tsv” que tenemos disponible en nuestro sistema HDFS y que contiene los eventos que hemos obtenido de GDELT. Los archivos de este tipo almacenan de forma que cada entrada se representa por una línea y cada atributo de esta línea se separa del anterior por una tabulación. Dividimos la línea por las tabulaciones y extraemos los atributos de los datos. Además, debemos obtener los campos latitud y longitud para establecer la geometría predeterminada de nuestra `SimpleFeature`. A continuación, mostramos el código que realiza estas acciones:

```

String[] attributes = value.toString().split("\\t", -1);
featureBuilder.reset();
featureBuilder.addAll(attributes);

Double lat = Double.parseDouble(attributes[LATITUDE_COL_IDX]);
Double lon =
Double.parseDouble(attributes[LONGITUDE_COL_IDX]);
Geometry geom = geometryFactory.createPoint(new
Coordinate(lon, lat));
SimpleFeature simpleFeature =
featureBuilder.buildFeature(attributes[ID_COL_IDX]);
simpleFeature.setDefaultGeometry(geom);

```

`GeoTools` proporciona las conversiones más comunes para la mayoría de los tipos de datos y algunos formatos de fecha. Sin embargo, cualquier atributo que no se convierta automáticamente en la clase especificada debe añadirse explícitamente al `SimpleFeature`. Un ejemplo de ello puede ser el campo “SQLDATE”, el cuál mostramos cómo añadirlo a continuación:

```

simpleFeature.setAttribute("SQLDATE",
formatter.parse(attributes[DATE_COL_IDX]));

```

Una vez hemos creado nuestro objeto `SimpleFeature` que contiene todos los datos asociados a un evento de GDELT, lo último que debemos hacer antes de acabar nuestra función `map` es escribirlo en el contexto de salida para que de esta forma se almacene en nuestra instancia de `Accumulo`:

```

context.write(new Text(), simpleFeature);

```

En resumen, cada línea del fichero con extensión “.tsv” debe ser parseada para obtener los atributos del evento. Con estos atributos se debe crear un objeto `SimpleFeature` que debe ser almacenado en nuestra instancia de `Accumulo`. Todas estas acciones se llevan a cabo en la función `map` (no se hace uso de la función `reduce`) por lo que pueden ser paralelizadas.

## 5.2 Ingestión de datos usando GeoMesa Command Line Tools

Con las herramientas de línea de comandos de GeoMesa vienen ciertas funcionalidades para facilitar la ingestión de datos. Estas funcionalidades incluyen ingestión de ficheros con formatos de texto delimitado (TSV, CSV), JSON, XML y Avro. Además, también permite la ingestión de los llamados *shapefiles* o archivos de formas.

Normalmente, para hacer funcionar estas herramientas, debemos especificar un *converter* y un *SimpleFeatureType*. El *converter* no es más que un parseador encargado de analizar cada línea y cargar los atributos en variables para después poder incluirlos en un *SimpleFeatureType* que como ya sabemos es un conjunto de atributos que representas una fila en la base de datos Accumulo. *Converters* y *SimpleFeatureTypes* son especificados en formato HOCON (Human-Optimized Config Object Notation) y pueden ser referenciados en las herramientas de línea de comandos de GeoMesa mediante las opciones “-s” y “-C” respectivamente.

Para definir nuevos *converters* y *SimpleFeatureTypes* podemos especificarlos en el fichero `application.conf` dentro del directorio de configuración de las herramientas de línea de comando de GeoMesa. También existe la opción de especificar *converter* y *SimpleFeatureType* por separado en ficheros independientes. Si decidimos usar la primera forma, para referenciar *converter* y *SimpleFeatureType* debemos indicar el nombre que le hemos dado en su especificación. Si por el contrario usamos la segunda forma, debemos especificar el fichero donde se definen para poder referenciarlos.

Para poner en evidencia la sencillez de este método frente al método visto en la sección anterior, vamos a recurrir a un ejemplo que hemos tomado de la documentación de GeoMesa.

Imaginemos que tenemos el siguiente archivo con formato CSV y con nombre `example.csv`:

```
FID,Name,Age,LastSeen,Friends,Lat,Lon
23623,Harry,20,2015-05-06,"Will, Mark, Suzan",-100.236523,23
26236,Hermione,25,2015-06-07,"Edward, Bill, Harry",40.232,-
53.2356
3233,Severus,30,2015-10-23,"Tom, Riddle, Voldemort",3,-62.23
```

Para ingerir este archivo, podemos definir un *SimpleFeatureType* en el archivo `/tmp/renegades.sft` de la siguiente forma:

```
geomesa.sfts.renegades = {
  attributes = [
    { name = "fid", type = "Integer", index = false }
    { name = "name", type = "String", index = true }
    { name = "age", type = "Integer", index = false }
  ]
}
```

```

    { name = "lastseen", type = "Date", index = true }
    { name = "friends", type = "List[String]", index = true }
    { name = "geom", type = "Point", index = true, srid = 4326,
default = true }
  ]
}

```

Como vemos, es realmente sencillo definir un SimpleFeatureType que represente los atributos de nuestros datos. El siguiente paso sería definir un *converter* en el fichero `/tmp/renegades.convert` por ejemplo:

```

geomesa.converters.renegades-csv = {
  type = "delimited-text"
  format = "CSV"
  options {
    skip-lines = 0
  }
  id-field = "toString($fid)"
  fields = [
    { name = "fid", transform = "$1::int" }
    { name = "name", transform = "$2::string" }
    { name = "age", transform = "$3::int" }
    { name = "lastseen", transform = "date('YYYY-MM-dd', $4)" }
    { name = "friends", transform = "parseList('string', $5)" }
    { name = "lon", transform = "$6::double" }
    { name = "lat", transform = "$7::double" }
    { name = "geom", transform = "point($lon, $lat)" }
  ]
}

```

Con esto ya tendríamos todo lo necesario para proceder a ingerir nuestro fichero de ejemplo. Para importar los datos a nuestra instancia de Accumulo bastaría con ejecutar el comando “`geomesa ingest`” con los parámetros adecuados:

```

$ geomesa ingest -u root -p 123456 \
  -z localhost:2181 \
  -c example_catalog -i TFM \
  -s /tmp/renegades.sft \
  -C /tmp/renegades.convert ./example.csv

```

Con esto ya habremos conseguido nuestro objetivo y podremos pasar al análisis de los datos que hemos ingerido. Como puede observarse, este procedimiento es bastante más sencillo que el definido en la sección anterior, sin embargo, este último será necesario ante un tipo de fichero de datos que no esté contemplado por las herramientas de línea de comandos de GeoMesa.

### 5.2.1 Ingestión de Shapefile

Como ya hemos comentado antes, también está soportada la ingestión de *Shapefile*. Un *Shapefile* es un formato vectorial de almacenamiento digital donde se guarda la

localización de elementos geográficos, su forma geométrica y los atributos asociados a estos. Es un formato multiarchivo, es decir está generado por varios ficheros informáticos.

En concreto, hemos ingerido durante el desarrollo de este Trabajo Fin de Máster un *Shapefile* que contiene polígonos que representan las fronteras de los diferentes países del mundo. Esto es interesante ya que podemos tener un conjunto de datos geográficos indexados por sus coordenadas, pero sin ningún atributo que nos indique a que país pertenecen esas coordenadas. Combinando el conjunto de datos geográficos anterior con el conjunto de datos que nos describe los límites de los países, somos capaces de clasificar cada evento por país y obtener estadísticas de cada país.

Para ello el primer paso es descargar y descomprimir dicho *Shapefile*:

```
$ curl -O
http://thematicmapping.org/downloads/TM_WORLD_BORDERS-0.3.zip
$ unzip TM_WORLD_BORDERS-0.3.zip
$ rm TM_WORLD_BORDERS-0.3.zip
```

Como hemos dicho el *Shapefile* es un formato multiarchivo, por lo que al descomprimir se nos crearán cuatro archivos con distintas extensiones. El siguiente paso es pasar estos archivos a nuestro almacenamiento HDFS. Este paso es opcional, pero de esta forma vemos que también es posible ingerir datos del sistema de archivos HDFS mediante las herramientas de línea de comandos de GeoMesa:

```
$ hdfs dfs -put TM_WORLD_BORDERS-0.3.* /countries/
```

Por último, podemos proceder a importar este conjunto de datos a nuestra base de datos Accumulo mediante el siguiente comando el cual es muy similar al visto en el ejemplo anterior:

```
$ geomesa ingest -u root -p 123456 \
-z localhost:2181 -i TFM \
-c countries_catalog -f country \
hdfs:///countries/TM_WORLD_BORDERS-0.3.shp
```

## 6 Ejecución de aplicaciones GeoMesa-Spark en el entorno

Llegados a este punto, ya tenemos disponibles datos en nuestra instancia de Accumulo siguiendo el principio de indexación de GeoMesa. Estos datos están preparados y disponibles para realizar cualquier análisis sobre ellos, usando técnicas Big Data combinando Spark y Geomesa en el entorno de análisis de datos geográficos que hemos desplegado.

Nuestro objetivo para este capítulo es mostrar los distintos procedimientos que tenemos disponibles en nuestro entorno para ejecutar un análisis una vez hemos desarrollado el código del mismo. Así mismo, explicaremos de forma general el código de algunos ejemplos que hemos ejecutado para probar el entorno desplegado en el transcurso de este Trabajo Fin de Máster.

Principalmente existen dos métodos de ejecución de análisis en nuestro entorno de análisis de datos geográficos. El primero de ellos es una ejecución de un trabajo Spark normal usando para ello el script `spark-submit`. El otro método consiste en una ejecución interactiva usando Jupyter en combinación con el kernel Apache Toree.

### 6.1 Ejecución mediante `spark-submit`

La forma más sencilla de enviar un trabajo a nuestro cluster Spark es creando un JAR con dependencias incluidas, de esta forma no es necesario usar la opción `--jars` para incluir cada dependencia.

En nuestro caso, hemos usado Maven como software de compilación. Además, Maven nos facilita enormemente la gestión de dependencias y la creación de JARs con dependencias incluidas.

Todas las dependencias del proyecto que vamos a ejecutar deben ser especificadas en su fichero `pom.xml`, pero además para la creación de un JAR con dependencias incluidas, debemos añadir lo siguiente en el mismo fichero:

```
<project ...>
  ...
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-assembly-plugin</artifactId>
        <version>3.0.0</version>
        <configuration>
          <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
          </descriptorRefs>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

```

    <executions>
      <execution>
        <id>make-assembly</id>
        <phase>package</phase>
        <goals>
          <goal>single</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
</plugins>
</build>
...
</project>

```

Con esto habremos configurado el “*Maven Assembly Plugin*” para que nos cree el paquete JAR con dependencias incluidas que hemos comentado anteriormente. Basta con ejecutar el siguiente comando en la carpeta en la que se encuentre el archivo `pom.xml` para obtener dicho JAR:

```
$ maven package -DskipTests=true
```

Con esto obtendremos el JAR en el directorio `target` de nuestro proyecto sin realizar los tests que hayamos definido. Para hacer lo mismo pasando los tests, debemos quitar la opción `-DskipTests=true`.

Si tenemos nuestro entorno correctamente desplegado y todos sus componentes ejecutándose, podemos pasar a enviar nuestro análisis a Spark mediante el script `spark-submit`:

```
$ spark-submit --master spark://master_ip:7077 \
  --class package.subpackage.class_name \
  jar-with-dependencies.jar
```

## 6.2 Ejecución en Jupyter-Toree

La ejecución de un análisis con Jupyter usando el kernel Apache Toree es muy sencilla. Simplemente debemos crear, desde la interfaz web de Jupyter, un nuevo notebook que se ejecute sobre el kernel Apache Toree que hemos instalado previamente. El proceso a seguir se encuentra detallado en la subsección 4.8.6 de este mismo documento.

Cuando usamos el kernel Apache Toree, se nos crea por defecto un contexto de Spark (`SparkContext`) asociado a nuestro cluster de Spark y que está disponible en la variable “`sc`”. Es por ello por lo que no necesitamos crear otro contexto y podemos usar las APIs de Spark o de Geomesa basándonos en el contexto por defecto.

Las dependencias deben estar disponibles en nuestro kernel Toree para que este pueda hacer que estas estén disponibles en nuestro cluster de Spark. Para ello, a la hora de la instalación del kernel Apache Toree en Jupyter, debemos indicar los paquetes JAR que contienen las clases usadas dentro de nuestro código. Debido a esto, es posible que debamos crear un kernel Apache Toree ligeramente diferente de una aplicación a otra, dependiendo de las dependencias de cada una.

## 6.3 Ejemplos

En esta sección explicaremos de forma general el código de dos ejemplos que hemos ejecutado para probar el correcto funcionamiento de nuestro entorno. Ambos ejemplos están basados en los datasets que importamos en nuestra base de datos Accumulo mediante GeoMesa en el capítulo anterior.

### 6.3.1 CountByDay

El primer ejemplo, cuyo nombre es *CountByDay* (46), utiliza los eventos de GDELT que hemos importado en el capítulo anterior para llevar a cabo un recuento de eventos por día. Es decir, se clasifican los eventos según el día que se produjeron y se cuenta cuantos eventos se han producido cada día.

El código de este ejemplo está disponible en el repositorio de GitHub de ejemplos de Geomesa (<https://github.com/geomesa/geomesa-tutorials.git>) aunque debe ser modificado para adaptarlo a los parámetros de nuestro entorno. Podemos clonar dicho repositorio usando el comando “\$ git clone”.

A continuación, pasamos a describir el código de este ejemplo.

Lo primero es configurar nuestros parámetros de conexión: instancia, usuario, contraseña y nombre de la tabla de Accumulo, además de la instancia de Zookeeper. Almacenamos estos parámetros en un Map de Scala:

```
val params = Map(  
  "instanceId" -> "TFM",  
  "zookeepers" -> "localhost",  
  "user"       -> "root",  
  "password"   -> "123456",  
  "tableName"  -> "gdelt")
```

Seguidamente debemos definir un filtro para seleccionar un subconjunto de eventos de GDELT de los que tenemos disponibles en nuestra base de datos GeoMesa Accumulo. Debemos crear un string que defina una caja basada en coordenadas y que nos servirá de filtro espacial y otro string que defina un intervalo temporal que se ajuste a los datos que tenemos disponibles. Tras esto, definimos en un string el filtro ECQL que defina un filtro espacial usando el que hemos creado previamente e indicando el nombre del atributo que guarda la información espacial. También debe



incluir el filtro temporal previo aplicado al atributo que guarda la información temporal mediante la sentencia “during”.

```
val typeName = "event"
val geom      = "geom"
val date      = "SQLDATE"

val bbox      = "-80, 35, -79, 36"
val during    = "2017-02-01T00:00:00.000Z/2017-02-27T12:00:00.000Z"

val filter    = s"bbox($geom, $bbox) AND $date during $during"
```

Usando los parámetros que creamos en el primer paso, podemos crear un gestor para nuestra base de datos GeoMesa Accumulo en forma de objeto del tipo `AccumuloDataStore`. Este objeto nos servirá para gestionar peticiones a la base de datos:

```
val ds =
DataStoreFinder.getDataStore(params).asInstanceOf[AccumuloDataStore]
```

Y creamos una *query* usando el filtro ECQL:

```
val q = new Query(typeName, ECQL.toFilter(filter))
```

El siguiente paso sería crear nuestro contexto asociado a nuestro cluster de Spark. Este paso puede no ser necesario dependiendo de si vamos a ejecutar este ejemplo en Jupyter o no.

```
val conf = new SparkConf(true).setAppName("CountByDay")
val sc   = new SparkContext(GeoMesaSpark.init(conf), ds)
```

El objeto `GeoMesaSpark` proporcionado por el módulo `geomesa-spark-core` utiliza el estándar SPI para encontrar una implementación de la interfaz `SpatialRDDProvider`. En este caso, será una instancia de `AccumuloSpatialRDDProvider` del módulo `geomesa-accumulo-spark`, que conectará a Accumulo con los parámetros proporcionados. Para obtener este objeto:

```
val spatialRDDProvider = GeoMesaSpark(params)
```

Con esto ya podemos inicializar un RDD usando la *query* creada con anterioridad:

```
val rdd = spatialRDDProvider.rdd(new Configuration, sc,
params, q)
```

Finalmente, llevamos a cabo nuestras operaciones que consisten en extraer el campo `SQLDATE` de cada evento y truncarlo a la resolución del día.

```

val dayAndFeature = rdd.mapPartitions { iter =>
    val df = new SimpleDateFormat("yyyyMMdd")
    val ff = CommonFactoryFinder.getFilterFactory2
    val exp = ff.property(dateField)
    iter.map { f =>
(df.format(exp.evaluate(f).asInstanceOf[java.util.Date]), f)
}
}

```

Por último, agrupamos por día y contamos el número de eventos ocurridos en cada grupo. También presentamos el resultado por pantalla:

```

val countByDay = dayAndFeature.map( x => (x._1,
1)) .reduceByKey(_ + _)
countByDay.collect().foreach(println)

```

A continuación, incluimos una captura del resultado de ejecución usando el script spark-submit:

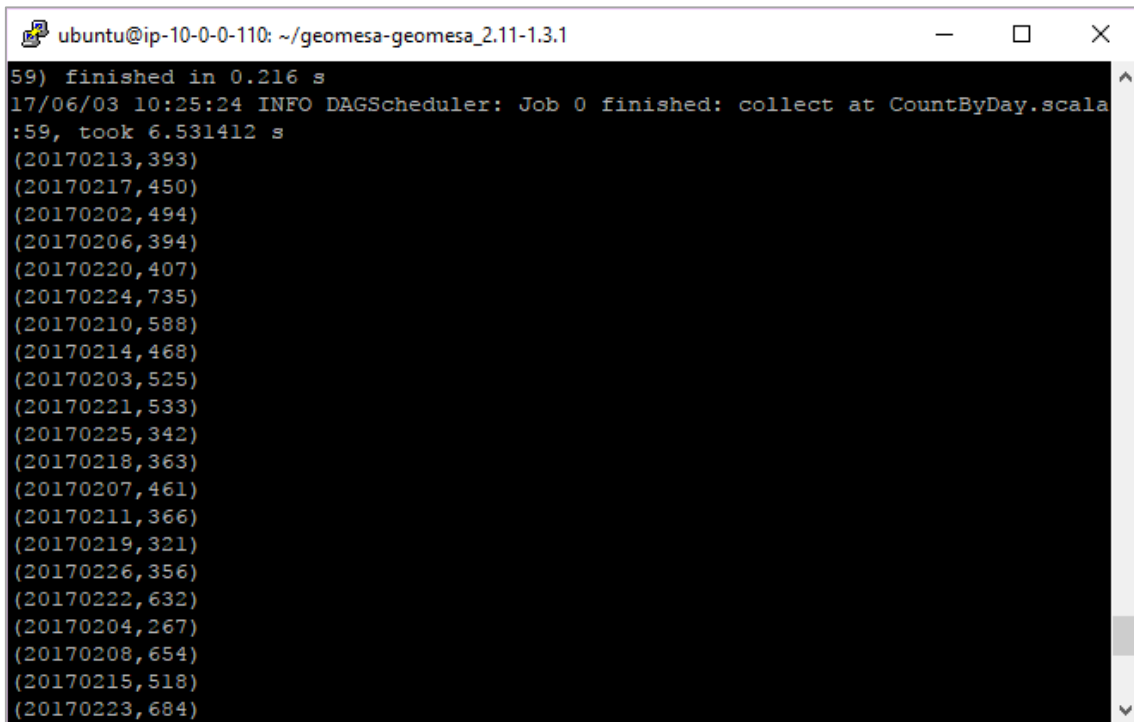


Figura 72. Ejecución ejemplo CountByDay usando spark-submit

También incluimos captura de la ejecución del código anterior en un notebook de Jupyter corriendo sobre kernel Apache Toree:

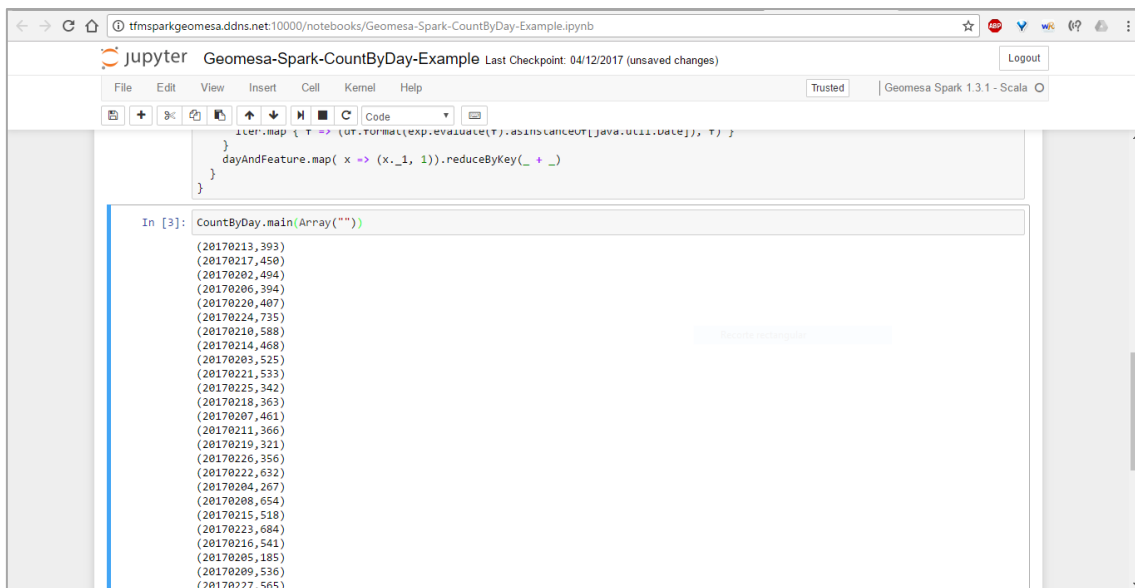


Figura 73. Ejecución ejemplo CountByDay en Jupyter con kernel Apache Toree

### 6.3.2 ShallowJoin

El siguiente ejemplo, que recibe el nombre de ShallowJoin (47), realiza una operación consistente en imponer un pequeño conjunto de datos geospaciales en un conjunto de datos de un tamaño mucho mayor. En esencia estamos realizando un *inner join* basándonos en las características geospaciales y después agregamos sobre el resultado. Todo esto se hace usando computación distribuida basada en Spark.

En este ejemplo, nuestro conjunto de datos obtenidos de GDELT tienen atributos que incluyen puntos geográficos para cada evento, pero no tenemos conocimiento directo sobre en que país tuvo lugar el evento. Vamos a realizar un *join* entre este conjunto de datos y el conjunto de datos importado en la subsección 5.2.1, que contiene las formas de los países, para calcular estadísticas por países.

El código de este ejemplo está disponible en el repositorio de GitHub de ejemplos de Geomesa (<https://github.com/geomesa/geomesa-tutorials.git>) aunque debe ser modificado para adaptarlo a los parámetros de nuestro entorno y renovar el uso de APIs de GeoMesa ya que está un poco desfasado y no funciona con la versión actual de GeoMesa. Podemos clonar dicho repositorio usando el comando “\$ git clone”.

El primer paso, al igual que en el ejemplo anterior, consiste en configurar los parámetros de conexión con nuestra base de datos (instancia, usuario, contraseña y nombre de la tabla de Accumulo, además de la instancia de Zookeeper) e inicializar un gestor de base de datos específico para cada uno de los dos conjuntos de datos que hemos comentado:

```

val countriesDsParams = Map(
  "instanceId" -> "TFM",
  "zookeepers" -> "localhost",

```

```

    "user"          -> "root",
    "password"     -> "123456",
    "tableName"   -> "countries_catalog")

val gdeltDsParams = Map(
  "instanceId" -> "TFM",
  "zookeepers" -> "localhost",
  "user"       -> "root",
  "password"   -> "123456",
  "tableName"  -> "gdelt")

val countriesDs =
  DataStoreFinder.getDataStore(countriesDsParams).asInstanceOf[AccumuloDataStore]
val gdeltDs =
  DataStoreFinder.getDataStore(gdeltDsParams).asInstanceOf[AccumuloDataStore]

```

A continuación, deberíamos iniciar un *SparkContext*, sin embargo, como vamos a ejecutar este ejemplo en Jupyter con kernel Apache Toree, ya tenemos un *SparkContext* inicializado por defecto. Pasamos entonces a obtener una instancia de *SpatialRDDProvider* para cada uno de los *datasets* que hemos comentado:

```

val rddProviderCountries = GeoMesaSpark(countriesDsParams)
val rddProviderGdelt     = GeoMesaSpark(gdeltDsParams)

```

Seguidamente, inicializamos los RDDs para cada una de las dos fuentes. Si queremos trabajar con un subconjunto de datos de GDELT podemos especificar un filtro a la hora de crear la *query* (ver subsección 6.3.1):

```

val countriesRdd: RDD[SimpleFeature] =
  rddProviderCountries.rdd(new Configuration(), sc,
  countriesDsParams, new Query("country"))
val gdeltRdd: RDD[SimpleFeature] = rddProviderGdelt.rdd(new
  Configuration(), sc, gdeltDsParams, new Query("event"))

```

Para llevar a cabo nuestro objetivo, tenemos que enviar nuestro dataset que contiene las formas de los continentes a cada una de las particiones que se han hecho de nuestro conjunto de datos de eventos GDELT. Esto podemos realizarlo usando el *broadcast* de Spark, el cual serializa los datos y los envía a cada uno de los nodos en nuestro cluster. De esta forma sólo se copia una vez por tarea. Notesé también que recolectamos el RDD de los continentes en un *Array* antes de enviarlo a los nodos. Esto es así ya que Spark no permite hacer *broadcast* de RDDs, y debido al pequeño tamaño de nuestro conjunto de datos, podemos recolectarlo en nuestro nodo Master sin riesgo de exceder la capacidad de memoria del mismo:

```

val broadcastedCover = sc.broadcast(coveringSet.collect)

```

Con el conjunto de datos de los continentes disponible en cada partición, podemos iterar sobre los eventos GDELT y clasificarlos según el país en el que ocurrieron. En la función *mapPartitions*, el objeto *iter* es un iterador que recorre todos los elementos de una partición (en este caso *SimpleFeature*). En el código mostrado a continuación, transformamos cada iterador y lo almacenamos en un nuevo RDD:

```

val keyedData = data.mapPartitions { iter =>

```

```

import org.locationtech.geomesa.utils.geotools.Conversions._

iter.flatMap { sf =>
  // Iterate over covers until a match is found
  val it = broadcastedCover.value.iterator
  var container: Option[String] = None

  while (it.hasNext) {
    val cover = it.next()
    // If the cover's polygon contains the feature,
    // or in the case of non-point geoms, if they intersect,
    set the container
    if (cover.geometry.intersects(sf.geometry)) {
      container =
Some(cover.getAttribute(key).asInstanceOf[String])
    }
  }
  // return the found cover as the key
  if (container.isDefined) {
    Some(container.get, sf)
  } else {
    None
  }
}
}

```

Nuestro nuevo RDD es de tipo `RDD[(String, SimpleFeature)]` y puede ser usado en una operación `reduceByKey` de Spark, pero primero, necesitamos crear un `SimpleFeatureType` para representar los datos agregados. Antes debemos recorrer mediante un bucle los tipos de los atributos de un evento de GDELT para decidir que campos se pueden agregar:

```

val countableTypes = Seq("Integer", "Long", "Double")
val typeNames = data.first.getType.getTypes.toIndexedSeq.map{t
=> t.getBinding.getSimpleName.toString}

val countableIndices = typeNames.indices.flatMap { index =>
  val featureType = typeNames(index)
  // Only grab countable types, skipping the ID field
  if ((countableTypes contains featureType) && index != 0) {
    Some(index, featureType)
  } else {
    None
  }
}.toArray
val countable = sc.broadcast(countableIndices)

```

Con estos campos, podemos crear un `SimpleFeatureType` para almacenar sus valores medios y totales. No tiene sentido agregar campos identificadores o campos que ya son valores medios.

```

val sftBuilder = new SftBuilder()
sftBuilder.stringType(key)
sftBuilder.multiPolygon("geom")
sftBuilder.intType("count")
val featureProperties = data.first.getProperties.toSeq

```

```

countableIndices.foreach{ case (index, clazz) =>
  val featureName = featureProperties.apply(index).getName
  clazz match {
    case "Integer" => sftBuilder.intType(s"total_${featureName}")
    case "Long" => sftBuilder.longType(s"total_${featureName}")
    case "Double" =>
sftBuilder.doubleType(s"total_${featureName}")
  }
  sftBuilder.doubleType(s"avg_${featureProperties.apply(index).g
etName}")
}
val coverSft = SimpleFeatureTypes.createType("aggregate",
sftBuilder.getSpec)

```

Con esto ya podemos comenzar la fase de agregación o reducción. Para ello primero debemos enviar el `SimpleFeatureType`, que creamos anteriormente, a cada uno de nuestros nodos para que puedan crear y serializar `SimpleFeature` de este tipo:

```

GeoMesaSparkKryoRegistrar.register(Seq(coverSft))
val newSfts =
sc.broadcast(GeoMesaSparkKryoRegistrar.typeCache.values.map {
sft =>
  (sft.getTypeName, SimpleFeatureTypes.encodeType(sft))
}).toArray)

keyedData.foreachPartition { iter =>
  newSfts.value.foreach { case (name, spec) =>
    val newSft = SimpleFeatureTypes.createType(name, spec)
    GeoMesaSparkKryoRegistrar.putType(newSft)
  }
}

```

Ya podemos pasar a realizar la operación `reduceByKey` que comentamos anteriormente sobre el RDD que creamos de tipo `RDD[(String, SimpleFeature)]`. Esta operación de Spark toma un par de elementos del RDD con la misma clave aplicando una función dada y remplazándolos por el resultado. En este ejemplo, tenemos tres casos de reducción:

- Ninguno de los elementos se ha agregado aún en uno de un nuevo tipo.
- Los dos elementos ya se han agregado en uno de un nuevo tipo.
- Uno de los elementos simples ya se ha agregado (pero no ambos).

Por simplicidad y brevedad vamos a mostrar sólo el primer caso, pero los otros dos siguen un patrón similar:

```

// Loop over the countable properties and sum them for both
gdelt simple features
countable.value.foreach { case (index, clazz) =>
  val propA = featurePropertiesA(index)
  val propB = featurePropertiesB(index)
  val valA = if (propA == null) 0 else propA.getValue
  val valB = if (propB == null) 0 else propB.getValue

  val sum = (valA, valB) match {
    case (a: Integer, b: Integer) => a + b

```

```

        case (a: java.lang.Long, b: java.lang.Long) => a + b
        case (a: java.lang.Double, b: java.lang.Double) => a + b
        case _ => throw new Exception("Couldn't match countable
type.")
    }
    // Set the total
    if( propA != null)
        aggregateFeature.setAttribute("total_" +
propA.getName.toString, sum)
}
aggregateFeature.setAttribute("count", new Integer(2))
aggregateFeature

```

Con los valores totales y el número de eventos calculado, podemos pasar a calcular los valores medios para cada campo. También, mientras iteramos, podemos añadir el nombre del país y su geometría a cada característica. Para hacer esto, primero enviamos a nuestros nodos un mapa que asocie nombre del país con su geometría:

```

aggregateFeature.setAttribute("count", new Integer(2))
aggregateFeature

val countryMap: scala.collection.Map[String, Geometry] =
    countriesRdd.map { sf =>
        (sf.getAttribute("NAME").asInstanceOf[String] ->
sf.getAttribute("the_geom").asInstanceOf[Geometry])
    }.collectAsMap

val broadcastedCountryMap = sc.broadcast(countryMap)

```

Tras esto, podemos transformar el RDD agregado en uno con los valores medios y las geometrías añadidos:

```

val averaged = aggregated.mapPartitions { iter =>
    import
org.locationtech.geomesa.utils.geotools.Conversions.RichSimpleFe
ature

    iter.flatMap { case ( sf) =>
        if (sf.getType.getTypeName == "aggregate") {
            sf.getProperties.foreach { prop =>
                val name = prop.getName.toString
                if (name.startsWith("total_")) {
                    val count = sf.get[Integer]("count")
                    val avg = (prop.getValue) match {
                        case (a: Integer) => a / count
                        case (a: java.lang.Long) => a /
count
                        case (a: java.lang.Double) => a /
count
                        case _ => throw new
Exception(s"couldn't match $name")
                    }

                    sf.setAttribute("avg_" + name.substring(6),
avg)

```

```

        }
    }
    Some(sf)
  } else {
    None
  }
}
}
}

```

Llegados a este punto, hemos creado un nuevo `SimpleFeatureType` que representa los datos agregados y un RDD de `SimpleFeatures` de este tipo. El código anterior puede ser compilado y enviado a Spark mediante `spark-submit`, pero si lo usamos en un notebook de Jupyter con el kernel Apache Toree, el RDD puede mantenerse en memoria e incluso modificarlo rápidamente mientras actualizamos continuamente las visualizaciones.

Aunque existen muchas formas de visualizar los datos de un RDD, para este ejemplo se usa una biblioteca JavaScript, llamada Leaflet, que nos permite crear mapas interactivos e integrarlos fácilmente con Jupyter. Para usar esta biblioteca, podemos instalarla a través de las herramientas *nbextensions* de Jupyter, o colocar el código HTML mostrado a continuación en una celda de nuestro Notebook:

```

%%HTML
<link rel="stylesheet"
href="http://cdn.leafletjs.com/leaflet/v0.7.7/leaflet.css" />
<script
src="http://cdn.leafletjs.com/leaflet/v0.7.7/leaflet.js"></scrip
t>

```

Para poder tomar los datos almacenados en el RDD en un JavaScript ejecutado en cliente podemos exportar dicho RDD como un GeoJSON. Para hacer esto, usamos la característica `AddDeps` de Toree para añadir la dependencia “GeoTool GeoJSON” sobre la marcha:

```

%AddDeps org.geotools gt-geojson 14.1 --transitive --repository
http://download.osgeo.org/webdav/geotools

```

Podemos transformar el RDD de `SimpleFeatures` en un RDD de strings, recolectar estos strings, unirlos, y almacenarlos en un fichero.

```

import org.geotools.geojson.feature.FeatureJSON
import java.io.StringWriter

// Convert simple features to their GeoJson string
representation
val geoJsonWriters = averaged.mapPartitions{ iter =>
  val featureJson = new FeatureJSON()

  val strRep = iter.map{ sf =>
    featureJson.toString(sf)
  }
  // Join all the features on this partition
  Iterator(strRep.mkString(", "))
}

```



```
// Collect these strings and joining them into a json array
val geoJsonString = geoJsonWriters.collect.mkString("[", ",", ",",""]")

import java.io.File
import java.io.FileWriter
val jsonFile = new File("aggregateGdeltEarthJuly.json")
val fw = new FileWriter(jsonFile)
fw.write(geoJsonString)
fw.close
```

Para modificar el DOM del documento HTML desde una celda de Jupyter, debemos configurar un “Mutation Observer” para responder adecuadamente a los cambios asíncronos. Adjuntamos este observador al elemento que hace referencia a la celda desde la que se ejecuta el código JavaScript. Dentro de este observador instanciamos un mapa de la biblioteca Leaflet.

```
(new MutationObserver(function() {

    // Initialize the map
    var map = L.map('map').setView([35.4746,-44.7022],3);
    // Add the base layer
    L.tileLayer("http://{s}.tile.osm.org/{z}/{x}/{y}.png").addTo
o(map);

    this.disconnect()
})) .observe(element[0], {childList: true})
```

Usando la biblioteca Leaflet creamos una capa a partir de los datos del archivo GeoJSON que hemos creado.

```
var rawFile = new XMLHttpRequest();
rawFile.onreadystatechange = function () {
    if(rawFile.readyState === 4) {
        if(rawFile.status === 200 || rawFile.status == 0) {
            var allText = rawFile.response;
            var gdeltJson = JSON.parse(allText)
            L.geoJson(gdeltJson).addTo(map);
            // Css override
            $('svg').css("max-width","none")
        }
    }
}
rawFile.open("GET", "aggregateGdelt.json", false);
rawFile.send()
```

Existen muchas formas de dar un formato a esta capa, como por ejemplo colorear los polígonos dependiendo de los atributos. En este ejemplo se colorea cada país dependiendo del valor medio de la escala Goldstein. Esta escala representa el impacto potencial teórico en la estabilidad de un país ocasionado por los eventos ocurridos en el intervalo temporal abarcado por nuestros datos.

El resultado de este ejemplo es un mapa similar al mostrado en la siguiente figura. Dependiendo de la cantidad de eventos de GDELT que hayamos importado a nuestra base de datos, la ejecución puede demorarse horas e incluso días.

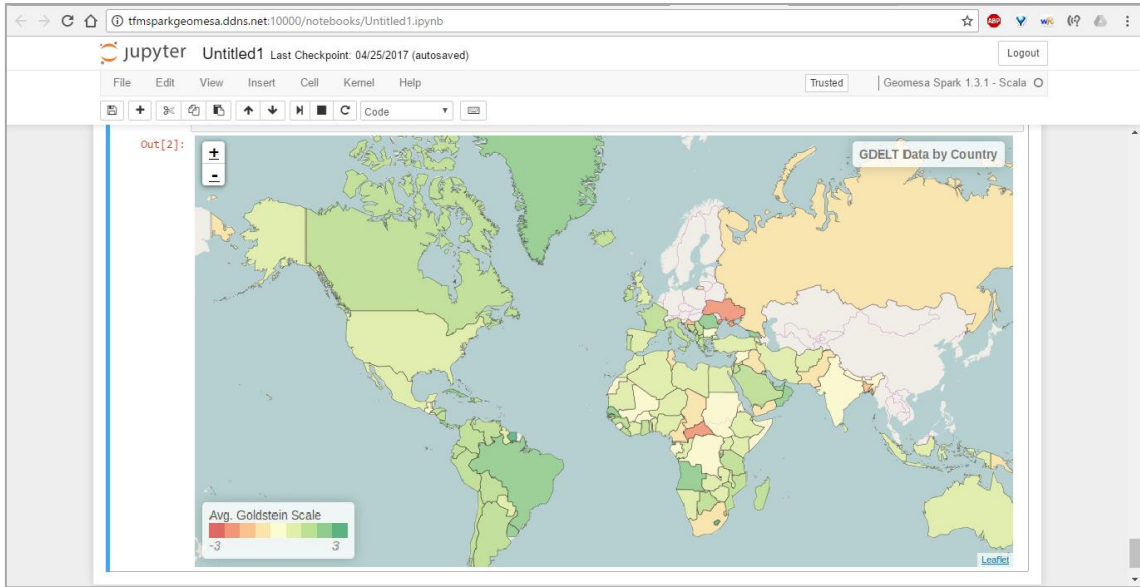


Figura 74. Ejecución ejemplo ShallowJoin en Jupyter con kernel Apache Toree



## 7 Automatización de workers de Spark

Durante este capítulo se detallará el proceso de automatización del despliegue de workers de Spark que se ha seguido para complementar el desarrollo de nuestra plataforma de análisis de datos geográficos. Para ello se ha hecho uso conjunto de las herramientas Docker y Ansible de las que ya hablamos en el capítulo 2.

### 7.1 Docker

Para realizar una automatización del escenario, se creará un Dockerfile que permitirá la creación de contenedores con la imagen especificada en el Dockerfile, que descargará los paquetes necesarios, instalará Apache Spark y arrancará un worker, conectado al máster automáticamente. Así, será posible crear y arrancar nuevos workers únicamente ejecutando un nuevo contenedor con la imagen previamente descrita.

#### 7.1.1 Dockerfile

El primer paso para crear nuestra imagen será escribir un Dockerfile (48) con las especificaciones que deseemos. En nuestro caso, el objetivo será instalar y arrancar Apache Spark, por lo que habrá que descargar los paquetes que necesite Spark para funcionar.

En primer lugar, creamos un nuevo directorio, donde ubicaremos nuestro Dockerfile, y creamos un fichero vacío:

```
$ mkdir spark-worker  
  
$ cd spark-worker  
  
$ touch Dockerfile
```

A continuación, se muestra el contenido del Dockerfile. Podemos acceder a este desde el enlace <https://hub.docker.com/r/alerguez/spark-worker/~dockerfile/>.

```
FROM ubuntu:16.04  
  
MAINTAINER alejandro.rodriguez.calzado@alumnos.upm.es  
cayetano.rodriguez.medina@alumnos.upm.es  
  
RUN apt-get update \  
    && apt-get install -y openjdk-8-jdk \  
    && apt-get install -y python2.7 \  
    && apt-get install -y python-minimal \  
    && apt-get install -y scala
```

```

# JAVA
ENV JAVA_HOME /usr/lib/jvm/java-1.8.0-openjdk-amd64

WORKDIR /opt

# Download Spark 2.1.0 pre-build for Hadoop 2.1 and later
ADD http://d3kbcqa49mib13.cloudfront.net/spark-2.1.0-bin-hadoop2.7.tgz./

RUN tar -xvzf spark-2.1.0-bin-hadoop2.7.tgz \

    && rm spark-2.1.0-bin-hadoop2.7.tgz

WORKDIR /opt/spark-2.1.0-bin-hadoop2.7

CMD ./bin/spark-class org.apache.spark.deploy.worker.Worker
spark://tfmsparkgeomesa.ddns.net:7077

```

La primera línea determina la imagen que debe usarse como base, para ello se usa la palabra clave `FROM`. La siguiente línea informa sobre el contacto de los autores del Dockerfile (palabra clave `MAINTAINER`). Esta línea no es ejecutada por Docker, simplemente es informativa.

A continuación, con la palabra clave `RUN` se describen los comandos que deben ejecutarse en la imagen base. Básicamente se trata de instalar los paquetes requeridos por Spark para funcionar (Java, Python y Scala). Como ya sabemos, es necesario añadir la variable `JAVA_HOME`. En el Dockerfile, esto se hace con la palabra clave `ENV`.

Usaremos la palabra clave `WORKDIR` para cambiarnos de directorio (similar al comando `cd`). Lo hacemos para descargar Spark en el directorio `/opt`. La descarga puede hacerse con la palabra clave `ADD`. La directiva `ADD` acepta dos parámetros (origen y destino), y funciona de la siguiente manera: copia un fichero origen (en el host, no el contenedor) en un directorio destino del contenedor. Si se especifica una URL en lugar de un directorio origen, lo que hará será descargar el contenido de la URL en el directorio destino.

Posteriormente, volvemos a usar `RUN` para descomprimir el fichero que acabamos de descargar y para eliminar este fichero.

Para finalizar, volvemos a cambiar de directorio para posicionarnos en el recién creado, y usamos la palabra clave `CMD` para especificar los comandos que queremos ejecutar en el contenedor una vez esté creado. En nuestro caso, queremos arrancar un worker y enlazarlo al máster.

### 7.1.2 Docker Image

Una vez tenemos nuestro Dockerfile, el siguiente paso será crear una imagen a partir de él. Esto podemos hacerlo con el comando:

```
$ docker build -t alerguez/spark-worker:latest .
```

Este comando busca un fichero llamado Dockerfile en el directorio actual. Se usa la etiqueta `latest` para utilizar la última versión del Dockerfile. Podemos facilitar la tarea de creación de la imagen descargándonos la imagen base de Docker Hub, de modo que la encuentre localmente y no tenga que buscarla.

```
$ docker pull ubuntu:16.04
```

El comando “`docker build`” lee el Dockerfile y procesa, una a una, las instrucciones para crear una imagen llamada `alerguez/spark-worker`.

Con esto, ya tenemos una imagen que podemos ejecutar en un contenedor. Para comprobar que se ha creado correctamente, podemos consultar las imágenes existentes en nuestro ordenador con el comando:

```
$ docker images
```

### 7.1.3 Docker Hub: subir y bajar imágenes

Ahora, podemos subir la imagen que hemos creado a Docker Hub . En primer lugar, tendremos que hacer *login* en Docker Hub desde línea de comandos:

```
$ docker login
```

Se nos pedirá un usuario y contraseña.

Finalmente, podemos subir la imagen con el comando:

```
$ docker push alerguez/spark-worker
```

Igualmente, si queremos descargarnos la imagen, podemos hacerlo de la siguiente forma:

```
$ docker pull alerguez/spark-worker
```

Esto no es estrictamente necesario, ya que, si no tenemos la imagen en nuestra máquina local, al intentar ejecutar un contenedor con la imagen, Docker la descargará automáticamente de DockerHub.

### 7.1.4 Docker Hub: Automate Build

En el subapartado anterior hemos explicado el proceso de creación de una imagen de forma manual. Utilizando Docker Hub, es posible realizar este mismo proceso de

forma automática. Para hacer esto, necesitaremos enlazar la cuenta de Docker Hub con GitHub. Esto podemos hacerlo accediendo a los siguientes menús en la web de Docker Hub:

Profile > Settings > Linked Accounts & Services.

El “Automate Build” se basa en la integración con el código existente en el repositorio de GitHub. Nuestro código está en el repositorio <https://github.com/aleroedcal/spark-worker>. En Docker Hub, pulsamos sobre Create > Create Automate Build. Ahí tendremos que elegir el proyecto de GitHub que queremos automatizar.

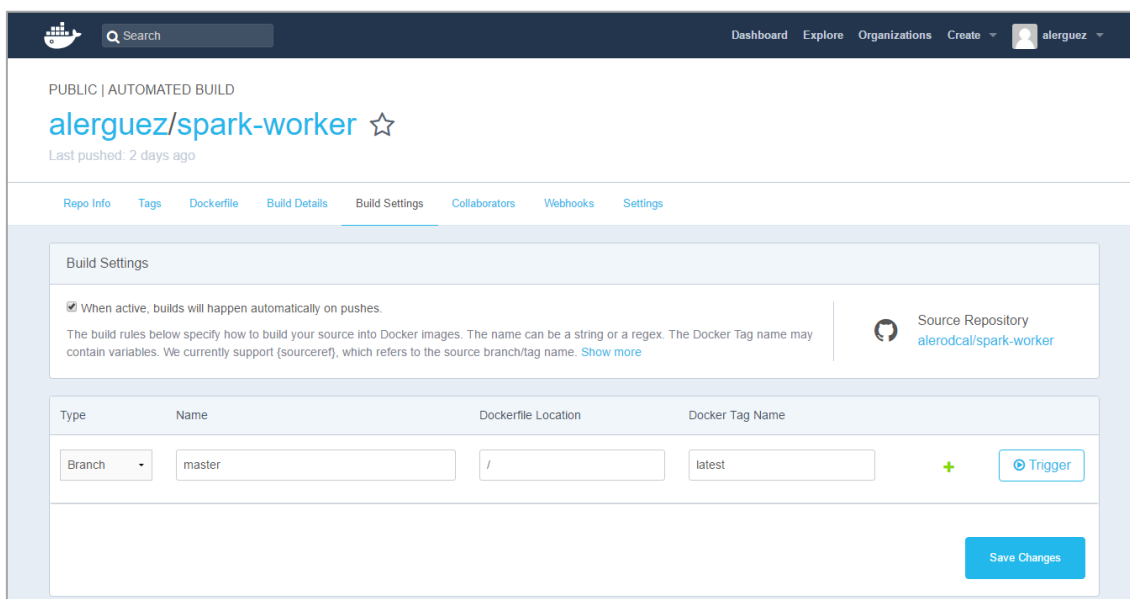


Figura 75. Enlazado de GitHub en Docker Hub

En la figura anterior, podemos ver la configuración realizada sobre esta imagen automatizada. Hemos activado la opción de actualizar automáticamente tras cada *push* en el repositorio de GitHub, para que los cambios se actualicen de manera inmediata y automática. También hemos especificado la rama sobre la que se debe hacer la automatización (rama master), y la etiqueta a usar (por defecto, latest).

La primera vez que creamos un “Automated Build”, Docker Hub crea la imagen. Tras unos minutos podremos ver la imagen en el dashboard.

### 7.1.5 Ejecución de un contenedor

Ya disponemos de la imagen que hemos diseñado para crear nuevos workers de Spark. Lo único que nos queda es ejecutar la imagen en un contenedor de Docker.

```
$ docker run -d -P --name worker --net=host alerguez/spark-worker:latest
```

Usamos la opción “-d” para ejecutar el contenedor en *background*, de modo que no muestre la salida en primer plano. La opción “-P” publica todos los puertos del host, realizando un enlace entre los puertos del contenedor y el host. Especificamos el nombre del contenedor con la opción “--name”. Y, por último, la opción “--net=host” es muy importante, ya que permite conectividad completa entre el contenedor y el host, de modo que se permite la comunicación entre el contenedor y el resto de Internet, atravesando el host sobre el que se ejecuta el contenedor.

## 7.2 Ansible

Una vez tenemos la imagen de Docker preparada para correr un worker de Spark en cualquier instancia, el siguiente paso sería conectarnos a cada instancia, instalar Docker, descargar la imagen del repositorio DockerHub, crear y ejecutar un contenedor a partir de esta imagen. Esto puede ser una tarea pesada si tenemos que desplegar muchos workers, por lo que decidimos añadir un nivel más de automatización mediante una herramienta como Ansible.

Como ya hemos comentado con anterioridad, Ansible es una herramienta open-source desarrollada en Python y que podemos definir como un motor de orquestación para la automatización de tareas. Ansible nos permitirá desplegar, de forma simultánea en cualquier número de instancias, workers de Spark a partir de la imagen de Docker que hemos creado. Las dos grandes ventajas que hemos encontrado en esta herramienta frente a otras de la misma índole, como Chef, es que se conecta a los clientes por SSH de forma paralela, por lo que la conexión es muy sencilla y que no es necesario instalar ningún tipo de agente en los clientes.

### 7.2.1 Instalación de Ansible

Como hemos dicho Ansible no necesita instalar ningún agente en los nodos que van ser gestionados, siendo el único requisito que los nodos cuenten con una versión de Python superior a la 2.5. En caso de que esta versión no venga por defecto en las máquinas gestionadas, no es necesario conectarse a cada una de ellas para instalarlo ya que Ansible cuenta con el módulo “raw” que nos permite ejecutar comandos de la Shell en los nodos gestionados. Este módulo no necesita de Python para realizar su cometido, por lo que puede ser usado para instalarlo y, de hecho, lo hemos usado para ello.

La otra parte de la arquitectura de Ansible la compone la máquina de gestión, que es la encargada de enviar las órdenes a los nodos gestionados. El único requisito para la instalación de Ansible en ésta es Python versión 2.6 o 2.7. En nuestro caso hemos usado un ordenador con Ubuntu 16.04. A continuación, pasamos a detallar los comandos ejecutados para la instalación de Ubuntu en este sistema:



```
$ sudo apt-get install software-properties-common
$ sudo apt-add-repository ppa:ansible/ansible
$ sudo apt-get update
$ sudo apt-get install ansible
```

### 7.2.2 Conexión con los nodos

Para que Ansible sea capaz de conectarse a los nodos y enviar órdenes, el único requisito es tener conectividad mediante SSH con ellos. Para ello es necesario realizar algunos pasos para que nuestro sistema sepa que clave privada debe usar para conectarse a cada nodo sin necesidad de especificar una clave de acceso.

En nuestro caso hemos usado la misma clave privada para todas las instancias de Amazon Web Services. El primer paso es copiar la clave en formato “pem” en el directorio `~/.ssh/`.

```
$ cp bigdata-spark.pem ~/.ssh/
```

A continuación, debemos modificar el fichero de configuración de SSH (`~/.ssh/config`) para añadir los parámetros de cada nodo. Por cada nodo debemos añadir a este fichero las siguientes líneas:

```
Host <IDENTIFICADOR-NODO>
    Hostname <DNS-PUBLICO>
    User <USUARIO>
    IdentityFile <RUTA-AL-FICHERO-PEM/NOMBRE-PEM>
```

Una vez hecho esto debemos añadir al fichero `/etc/ansible/hosts` el identificador que usamos en el archivo anterior para cada uno de los nodos. Los nodos se engloban en un grupo (role) para poder referirnos a todos ellos simultáneamente como objetivos de nuestro playbook. El formato que debemos usar en este fichero es el siguiente:

```
[<NOMBRE-GRUPO>]
<IDENTIFICADOR-NODO-1>
<IDENTIFICADOR-NODO-2>
...
<IDENTIFICADOR-NODO-N>
```

Debido a que los nombres DNS que asigna Amazon Web Services automáticamente a sus instancias son demasiado largos, debemos añadir al fichero de configuración de Ansible (/etc/ansible/ansible.cfg) la siguiente línea tras la sentencia “[ssh\_connection]”:

```
Control_path = %(directory)s/%%h-%%r
```

Con esto, Ansible ya es capaz de enviar las órdenes de nuestro playbook a todos los nodos gestionados. Para ejecutar un playbook basta con usar el siguiente comando en el directorio donde esté almacenado dicho playbook:

```
$ ansible-playbook worker-playbook.yml
```

### 7.2.3 Playbook

Para la automatización del despliegue mediante Ansible, hemos escrito un playbook que se encarga de instalar en los nodos todas las dependencias necesarias para ejecutar un contenedor de Docker basado en la imagen que hemos creado. El contenido de este playbook se muestra a continuación:

```
---
- hosts: spark-workers
  remote_user: ubuntu
  gather_facts: False
  environment:
    PYTHONPATH: "{{ lookup('env', 'PYTHONPATH')
  }}:/home/ubuntu/.local/lib/python2.7/site-
  packages:/usr/local/lib/python2.7/dist-
  packages:/usr/local/lib/python2.7/site-packages"

  tasks:
    - name: Install python-minimal
      raw: test -e /usr/bin/python || (apt-get -y update &&
apt-get install -y python-minimal)
      become: yes
      become_method: sudo

    - name: Install (or update) docker.io
      apt:
        name: docker.io
        state: latest
        update_cache: yes
      become: yes
      become_method: sudo

    - name: Install python-pip
      apt:
        name: python-pip
        state: latest
        update_cache: yes
      become: yes
      become_method: sudo
```

```

- name: Install certain python modules for docker
  pip:
    name: "{{ item.name }}"
    version: "{{ item.version }}"
    state: present
  with_items:
    - { name: docker, version: 2.0.0 }
    - { name: docker-py, version: 1.10.6 }

- name: Pull alerguez/spark-worker image
  docker_image:
    name: alerguez/spark-worker
  become: yes
  become_method: sudo

- name: Create a data container
  docker_container:
    name: spark-worker
    image: alerguez/spark-worker
    detach: yes
    network_mode: host
    published_ports:
      - all
  become: yes
  become_method: sudo
...

```

El `playbook` comienza indicando el grupo de los definidos en `/etc/ansible/hosts` que van a ejecutar este `playbook`, mediante la palabra reservada `hosts`. Seguidamente, mediante la sentencia `remote_user` podemos indicar el usuario que ejecutará estas acciones en los `hosts` y usando la sentencia `environment` podemos definir variables de entorno que estarán disponibles durante la ejecución del `playbook`. En este caso hemos definido la ruta a la instalación de Python y sus módulos.

Una vez definidos los parámetros generales, con la sentencia `tasks`, comienza una lista de tareas que componen lo que realmente se ejecutará en los nodos con este `playbook`. Cada tarea se le asigna un nombre mediante la sentencia `name` y hace uso de un módulo de Ansible según la tarea. Las sentencias `become` y `become_method` sirven para ejecutar la tarea como usuario con privilegios.

La primera tarea consiste en instalar Python que, como dijimos, es el único requisito para los nodos gestionados mediante Ansible. La instalación se ejecuta mediante el módulo `raw` que nos permite ejecutar comandos de la Shell sin necesidad de tener Python instalado. La tarea comprueba si Python está instalado y en caso de que no lo esté ejecuta el comando `apt-get update` y lo instala mediante el comando `apt-get install`.

La siguiente tarea se encarga de instalar la última versión de Docker en los nodos gestionados para ello hace uso del módulo apt de Ansible. Previamente a la instalación se realiza un “apt update” .

El módulo que vamos a usar para ejecutar el contenedor de Docker necesita de algunos módulos de Python en los nodos gestionados. Para la instalación de estos módulos se usa la herramienta pip. Esta herramienta se instala mediante el módulo apt en la siguiente tarea.

A continuación, en la tarea con nombre “Install certain python modules for docker” vamos a instalar los dos módulos de Python necesarios: Docker y Docker-py. Para ello se hace uso del módulo pip. El nombre y la versión de módulos se indica mediante el uso de ítems. Con la directiva “with\_items” se indican con que ítems deben ejecutarse la tarea.

Con esto ya tenemos Docker en nuestros nodos y somos capaces de gestionarlos mediante los módulos de Ansible diseñados para ello. En la tarea “Pull alerguez/spark-worker image” se hace uso del módulo docker\_image para descargar de DockerHub en todos los nodos la imagen que hemos creado.

Una vez tenemos disponible la imagen en los nodos, el último paso es ejecutar un contenedor Docker a partir de esta imagen. Esto se realiza en la última tarea mediante el módulo docker\_container.

Este playbook se ha probado con hasta 10 nodos simultáneamente, consiguiendo en cuestión de poco más de 2 minutos, que los 10 nodos se uniesen correctamente al cluster de Spark, listos para ejecutar las tareas que el master ordene. Gracias a que las tareas se realizan en paralelo en todos los nodos, Ansible tarda lo mismo si ejecutamos el playbook en un nodo que en varios, lo que demuestra la gran ventaja que supone la gestión de nodos mediante esta herramienta.



## 8 Conclusiones

En esta memoria se ha intentado plasmar el resultado de varios meses de investigación, aprendizaje, superación personal, trabajo y, en definitiva, esfuerzo. A lo largo de este periodo he podido aprender sobre tecnologías relacionadas con BigData como Spark, GeoMesa, Hadoop y otras que sin duda serán de gran ayuda en un futuro próximo.

Como se ha descrito en este documento, en el transcurso de este Trabajo Fin de Master se ha realizado el diseño y la implementación de un entorno que cuenta con las herramientas necesarias para llevar a cabo análisis de grandes conjuntos de datos geográficos.

La primera fase de este proyecto consistió esencialmente en realizar investigación sobre las diferentes herramientas que se usan actualmente en el ámbito de BigData, para así poder decidir que herramientas usar y diseñar la arquitectura de nuestro entorno de forma que todos sus componentes se compenetren entre ellos para cumplir los objetivos de este proyecto.

Una vez estudiados los componentes de nuestro entorno, se procedió con la fase de implementación del mismo. Durante esta fase hemos desarrollado nuestra capacidad de administración de entornos Linux y más concretamente entornos Linux en la nube pública. Los principales problemas encontrados son debido a incompatibilidades entre las versiones de los diferentes componentes.

Una vez teníamos nuestro entorno implementado, nos dedicamos a probarlo y enterderlo en más profundidad. Para ello nos basamos en distintos tutoriales y ejemplos disponibles tanto en la documentación de GeoMesa como en otras webs. También tuvimos que estudiar el uso del lenguaje de programación Scala y de las principales APIs de Spark, para entender el código de los diferentes ejemplos.

Para finalizar la parte práctica de nuestro trabajo, realizamos una automatización para el despliegue de workers de Spark en instancias de Amazon EC2. Para ello se han usado tecnologías punteras en este ámbito, como son Docker y Ansible. El uso de tecnologías novedosas ha sido la tónica general a lo largo de este proyecto, con todo lo que ello implica: cambios constantes, pocos ejemplos o referencias para consultar e incompatibilidad entre versiones.

En definitiva, podemos decir que se han cumplido los objetivos que se propusieron al inicio de este Trabajo Fin de Master lo que produce en el autor de este documento una gran satisfacción personal.

Al finalizar el proyecto, podemos concluir que GeoMesa, como tecnología BigData de análisis de datos geográficos, es una herramienta muy potente pero aún se encuentra en una fase temprana de su desarrollo por lo que debe mejorar ciertos aspectos. Uno de estos aspectos es definir una herramienta propia y open-source para la representación de datos geográficos, de forma que no tengamos que recurrir a herramientas de terceros que pueden presentar dificultades en su integración.

## 8.1 Líneas de desarrollo

La principal línea de desarrollo, que podría suponer el tema de otro Trabajo Fin de Master, sería diseñar una aplicación que, usando las herramientas del entorno desarrollado, realice un análisis de algún conjunto de datos y se integre con alguna herramienta de visualización con más funcionalidades de las ofrecidas por Jupyter.

Otra posibilidad sería integrar algún componente como Apache Storm o Apache Kafka para facilitar la ingestión y análisis de datos geográficos en *streaming*. Esto posibilita el análisis de datos provenientes de algunas webs que nos facilitan datos en tiempo real de, por ejemplo, posición de aviones en el espacio aéreo o posición de barcos sobre las distintas rutas marítimas.

Por último, proponemos como línea de desarrollo la automatización del despliegue de nodos de otros componentes de nuestro entorno, como por ejemplo nodos de Hadoop HDFS, Accumulo o Zookeeper.

## Bibliografía

1. **Wikipedia.** Amazon Web Services - Artículo Wikipedia. [En línea] [https://en.wikipedia.org/wiki/Amazon\\_Web\\_Services](https://en.wikipedia.org/wiki/Amazon_Web_Services).
2. **Amazon Web Services.** Introducción Amazon EC2. [En línea] <https://aws.amazon.com/es/ec2/>.
3. **Rouse, Margaret.** Amazon EC2 (Elastic Compute Cloud). [En línea] <http://searchaws.techtarget.com/definition/Amazon-Elastic-Compute-Cloud-Amazon-EC2>.
4. **Amazon Web Services.** Documentación Amazon EC2. [En línea] [http://docs.aws.amazon.com/es\\_es/AWSEC2/latest/UserGuide/concepts.html](http://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/concepts.html).
5. **The Apache Software Foundation.** What is Maven? [En línea] <https://maven.apache.org/what-is-maven.html>.
6. **Wikipedia.** Apache Maven - Artículo Wikipedia. [En línea] [https://en.wikipedia.org/wiki/Apache\\_Maven](https://en.wikipedia.org/wiki/Apache_Maven).
7. **The Apache Software Foundation.** Introduction to the Build Lifecycle. [En línea] <https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>.
8. **Rouse, Margaret.** What is Hadoop? *TechTarget*. [En línea] <http://searchcloudcomputing.techtarget.com/definition/Hadoop>.
9. **The Apache Software Foundation.** Apache Hadoop Documentation. [En línea] <http://hadoop.apache.org/>.
10. —. HDFS Architecture Guide. *Apache Hadoop Documentation*. [En línea] [https://hadoop.apache.org/docs/r1.2.1/hdfs\\_design.html](https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html).
11. —. ZooKeeper: A Distributed Coordination Service for Distributed Applications. *Zookeeper Documentation*. [En línea] <https://zookeeper.apache.org/doc/r3.4.10/zookeeperOver.html>.
12. **Wikipedia.** Apache Zookeeper. *Artículo Wikipedia*. [En línea] [https://es.wikipedia.org/wiki/Apache\\_ZooKeeper](https://es.wikipedia.org/wiki/Apache_ZooKeeper).
13. **The Apache Software Foundation.** Apache Accumulo User Manual. [En línea] [https://accumulo.apache.org/1.8/accumulo\\_user\\_manual.html](https://accumulo.apache.org/1.8/accumulo_user_manual.html).



14. **Wikipedia.** Apache Accumulo. *Artículo Wikipedia.* [En línea] [https://en.wikipedia.org/wiki/Apache\\_Accumulo](https://en.wikipedia.org/wiki/Apache_Accumulo).
15. **Rinaldi, Billie, Cordova, Aaron y Wall, Michael.** *Accumulo: Application Development, Table Design, and Best Practices.* Sebastopol : O'Reilly Media, Inc., 2015. ISBN: 9781491947098.
16. **The Apache Software Foundation.** Spark Overview. *Soark Documentation.* [En línea] <https://spark.apache.org/docs/2.1.0/index.html>.
17. **Databricks.** What is Apache Spark? [En línea] <https://databricks.com/spark/about>.
18. **Wikipedia.** Apache Spark. *Artículo Wikipedia.* [En línea] [https://en.wikipedia.org/wiki/Apache\\_Spark](https://en.wikipedia.org/wiki/Apache_Spark).
19. **Zaharia, Matei, y otros.** *Learning Spark.* Sebastopol : O'Reilly Media, Inc., 2015. 978-1449359034.
20. **Wikipedia.** GeoMesa. *Artículo Wikipedia.* [En línea] <https://en.wikipedia.org/wiki/GeoMesa>.
21. **GeoMesa.** Introduction to GeoMesa. *GeoMesa Documentation.* [En línea] <http://www.geomesa.org/documentation/user/introduction.html>.
22. —. GeoMesa Architecture Overview. *GeoMesa Documentation.* [En línea] <http://www.geomesa.org/documentation/user/architecture.html>.
23. **Jupyter Team.** The Jupyter Notebook Website. [En línea] <http://jupyter.org/>.
24. —. What is the Jupyter Notebook? *Jupyter Documentation.* [En línea] <http://jupyter-notebook.readthedocs.io/en/latest/examples/Notebook/What%20is%20the%20Jupyter%20Notebook.html>.
25. **The Apache Software Foundation.** Apache Toree Documentation. [En línea] <https://toree.apache.org/>.
26. **Docker Team.** What is Docker? [En línea] <https://www.docker.com/what-docker>.
27. **Wikipedia.** Docker (software). *Artículo Wikipedia.* [En línea] [https://en.wikipedia.org/wiki/Docker\\_\(software\)](https://en.wikipedia.org/wiki/Docker_(software)).

28. **Networklore Team.** What is Ansible? [En línea] <https://networklore.com/ansible/>.
29. **Ansible Team.** How Ansible Works. [En línea] <https://www.ansible.com/how-ansible-works>.
30. **Amazon Web Services.** Amazon EC2 Pricing - Documentación Amazon EC2. [En línea] [https://aws.amazon.com/ec2/pricing/on-demand/?nc1=h\\_ls](https://aws.amazon.com/ec2/pricing/on-demand/?nc1=h_ls).
31. —. Launching an Instance - Documentación Amazon EC2. [En línea] [http://docs.aws.amazon.com/es\\_es/AWSEC2/latest/UserGuide/launching-instance.html](http://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/launching-instance.html).
32. —. Connecting to Your Linux Instance from Windows Using PuTTY. *Documentación AWS.* [En línea] [http://docs.aws.amazon.com/es\\_es/AWSEC2/latest/UserGuide/putty.html](http://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/putty.html).
33. **The Apache Software Foundation.** Installing Apache Maven. *Maven Documentation.* [En línea] <http://maven.apache.org/install.html>.
34. —. Hadoop: Setting up a Single Node Cluster. *Apache Hadoop Documentation.* [En línea] <https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html>.
35. —. ZooKeeper Getting Started Guide. *Zookeeper Documentation.* [En línea] <https://zookeeper.apache.org/doc/r3.4.10/zookeeperStarted.html>.
36. **A., Hathy.** How To Install the Big-Data Friendly Apache Accumulo NoSQL Database on Ubuntu 14.04. *DigitalOcean.* [En línea] 19 de Marzo de 2015. <https://www.digitalocean.com/community/tutorials/how-to-install-the-big-data-friendly-apache-accumulo-nosql-database-on-ubuntu-14-04>.
37. **Walch, Mike, Turner, Keith y Elser, Josh.** Installing Accumulo. *Github.* [En línea] <https://github.com/apache/accumulo/blob/master/INSTALL.md>.
38. **The Apache Software Foundation.** Spark Standalone Mode. *Spark Documentation.* [En línea] <https://spark.apache.org/docs/2.1.0/spark-standalone.html>.
39. **GeoMesa.** Installing GeoMesa Accumulo. [En línea] <http://www.geomesa.org/documentation/user/accumulo/install.html>.
40. **Docker.** Docker Community Edition for Ubuntu. [En línea] <https://store.docker.com/editions/community/docker-ce-server-ubuntu>.

41. **Jupyter Team.** Installing Jupyter Notebook. *Jupyter Documentation*. [En línea] <http://jupyter.readthedocs.io/en/latest/install.html>.
42. —. Running a notebook server. *Jupyter Documentation*. [En línea] [http://jupyter-notebook.readthedocs.io/en/latest/public\\_server.html](http://jupyter-notebook.readthedocs.io/en/latest/public_server.html).
43. **GeoMesa.** Deploying GeoMesa Spark with Jupyter Notebook. *GeoMesa Documentation*. [En línea] <http://www.geomesa.org/documentation/current/user/spark/jupyter.html>.
44. **Hughes, James y Zimmerman, Matt.** How to install the Scala Spark (Apache Toree) Jupyter kernel with GeoMesa support. [En línea] <https://geomesa.atlassian.net/wiki/display/GEOMESA/How+to+install+the+Scala+Spark+%28Apache+Toree%29+Jupyter+kernel+with+GeoMesa+support>.
45. **GDEL T Team.** GDEL T Data Format Codebook. [En línea] [http://data.gdel tproject.org/documentation/GDEL T-Data\\_Format\\_Codebook.pdf](http://data.gdel tproject.org/documentation/GDEL T-Data_Format_Codebook.pdf).
46. **GeoMesa.** GeoMesa Spark: Basic Analysis. [En línea] <http://www.geomesa.org/documentation/current/tutorials/spark.html>.
47. —. GeoMesa Spark: Aggregating and Visualizing Data. [En línea] <http://www.geomesa.org/documentation/current/tutorials/shallow-join.html>.
48. **Butler, Tim.** What is a Dockerfile? [En línea] 25 de Junio de 2015. <https://www.conetix.com.au/blog/what-is-a-dockerfile>.
49. **Wikipedia.** Amazon Elastic Compute Cloud - Artículo Wikipedia. [En línea] [https://en.wikipedia.org/wiki/Amazon\\_Elastic\\_Compute\\_Cloud](https://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud).
50. **sbt.** Installing sbt on Linux. *sbt Documentation*. [En línea] <http://www.scala-sbt.org/0.13/docs/Installing-sbt-on-Linux.html>.
51. **The Apache Software Foundation.** Apache Toree Quick Start. [En línea] <https://toree.apache.org/docs/current/user/quick-start/>.
52. **Brown, Jason y Zimmerman, Matt.** Building and installing Toree. [En línea] <https://geomesa.atlassian.net/wiki/display/GEOMESA/Building+and+installing+Toree>.
53. **Jupyter Team.** Jupyter Notebook Quick Start Guide. *Jupyter Documentation*. [En línea] [http://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what\\_is\\_jupyter.html](http://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html).

54. **Horstmann, Cay S.** *Scala for the Impatient*. s.l. : Addison-Wesley, 2016. 978-0134540566.

55. **Wampler, Dean y Payne, Alex.** *Programming Scala*. Sebastopol : O'Reilly, 2014. 978-1491949856.