Universidad de La Laguna

Escuela Superior de Ingeniería y Tecnología
Sección de Ingeniería Informática

Trabajo de Fin de Grado

Despliegue de un clúster Spark sobre Docker para Big Data

Deploying a Spark cluster on Docker for Big Data

Sergio Martín Santana

La Laguna, 5 de septiembre de 2016

D. **Marcos Colebrook Santamaría**, con N.I.F. 43.787.808-V, Profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **Carlos J. Pérez González**, con N.I.F. 45.452.719-G, Profesor Asociado adscrito al Departamento de Matemáticas, Estadística e Investigación Operativa de la Universidad de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

Despliegue de un clúster Spark sobre Docker para Big Data

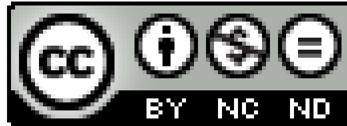
ha sido realizada bajo su dirección por D. **Sergio Martín Santana**, con N.I.F. 54.110.144-E.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 5 de septiembre de 2016.

Agradecimientos

A mis padres, Fausto y Nieves, a mi novia Silvia, porque posiblemente sin su apoyo jamás podría haber logrado escribir este documento y lo que ello representa. A mi familia en general, por enseñarme grandes valores y que con constancia todo es posible. También no quiero olvidarme, del personal docente parte de este proyecto, Marcos Colebrook y Carlos Javier, gracias por vuestro tiempo, apoyo e ideas. Asimismo, agradecer al personal del Centro de cálculo, en especial a Pedro González, por su ayuda. Y, por último, a todas aquellas personas que han participado activamente en mi vida, a todos ellos, Gracias.

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional.

Resumen

El objetivo de este trabajo ha sido realizar un estudio preliminar de las herramientas utilizadas (*Docker* y *Spark*), con la finalidad de crear un punto de partida para que alguien sin experiencia previa en el campo pueda entender cómo funciona y tener una pequeña idea de que beneficios tendrían asociadas estas herramientas.

Posteriormente se ha construido un sistema completo para Big Data, creando un clúster *Spark* 2.0.0 fácilmente escalable usando *Docker* 1.12. En los posteriores capítulos se explicará detenidamente como llevar este proceso acabo desde cero por cualquier usuario. A su vez se proporciona una visualización de los resultados obtenidos desde la web de *Spark*, y unas pequeñas comparativas de rendimiento.

En los últimos apartados, los apéndices, se añaden los códigos fuente de los Scripts, y ficheros necesarios para dicha implementación.

Palabras clave: *Docker*, *Spark*, *Big Data*, Virtualización, Contenedores, Sistemas Operativos, Administración de Sistemas.

Abstract

The aim of this work was to perform a study of the tools used (Docker and Spark), in order to create a starting point for someone who does not have previous experience in the field, so it would allow him/her to understand how it works and have a little notion of the benefits that would be associated with these tools.

Later on, we have built a complete system for Big Data, deploying a cluster Spark 2.0.0. This system can be easily scaled using Docker 1.12. In subsequent chapters we will explain how any user can take this process from scratch. At the same time, we show some results obtained from the website of Spark, and we perform a small benchmark.

In the final sections, the source scripts and files needed for that implementation codes are shown.

Keywords: *Docker, Spark, Virtualization, Containers, OS, Management Information Systems.*

Índice General

Capítulo 1. Introducción **6**

1.1 Big Data ... 6

1.1.1 ¿Qué es Big Data? ... 6

1.1.2 Las "Uves" del Big Data ... 6

1.2 Apache Spark ... 8

1.2.1 ¿Qué es Spark? ... 8

1.2.2 ¿Por qué *Spark*? ... 8

1.2.3 Apache *Spark* 2.0 ... 9

1.3 Docker ... 10

1.3.1 ¿Qué es Docker? ... 10

1.3.2 Beneficios ... 10

1.3.3 Componentes de Docker ... 11

1.3.4 Virtualización y Docker: ¿es lo mismo? ... 12

1.3.5 Una clase de Docker. Comandos útiles ... 13

Capítulo 2. Análisis de la problemática **15**

2.1 Descripción del Problema ... 15

2.2 Antecedentes ... 15

2.2.1 DSBox ... 15

2.3 Estado del Arte ... 15

2.3.1 Stratio ... 15

2.4 Objetivos ... 16

2.5 Fases ... 16

2.5.1 Análisis del caso de estudio ... 16

2.5.2 Despliegue Solución en Docker v1.11 ... 17

2.5.3 Despliegue Solución en Docker v1.12 ... 17

2.5.4 Optimización y creación de los benchmark ... 17

2.6 Esquema Hardware ... 17

2.7 Esquema Software 18

Capítulo 3. Solución Basada en Docker 1.11 **19**

3.1 Conceptos importantes en Docker 1.11 19

3.2 Despliegue de la solución (v1.11) 19

Capítulo 4. Solución Basada en Docker 1.12 **23**

4.1 Novedades introducidas en Docker 1.12 23

4.2 Despliegue de la solución (v1.12) 23

4.2.1 Usando *Tasks as a Service* 23

4.2.2 Usando *Containers as a Service* 26

Capítulo 5. Ejecución de la solución. **27**

5.1 Pruebas de ejecución de la solución 27

Capítulo 6. *Benchmarks* (Comparativas) **34**

6.1 Spark 1.6 vs Spark 2.0 34

6.2 Comparativa entre tiempo de ejecución y número de nodos (contenedores) 35

Capítulo 7. Conclusiones y líneas futuras **38**

7.1 Conclusiones. 38

7.2 Líneas Futuras. 38

Capítulo 8. Summary and Conclusions **40**

Capítulo 9. Presupuesto **41**

9.1 Presupuesto Detallado 41

Apéndice A. Bash **42**

A.1. encender.sh 42

A.2. salir.sh 43

A.3. link2swarm.sh 43

A.4. ini-cluster.sh 44

A.6. restart.sh 45
A.8. launchSpark.sh 46
A.9. verservice.sh 46
A.10. rmservice.sh 47
A.11. runSpark.sh 47
A.12. wake.sh 48

Apéndice B. Docker **49**
B.1 Servicios 49
B.1.1 Master 49
B.1.2 Worker 51
B.2. Contenedores 53
B.2.1 Master 53
B.2.2 Worker 55

Apéndice C. Spark **58**

Bibliografía **59**

Índice de figuras

Figura 1.1 Las 5 “uves” del Big Data. 7
Figura 1.2 Estructura (Virtualización Convencional y Docker) 12
Figura 2.1 “Docker 1.12 is now Available” 16
Figura 2.2 Esquema Hardware de la solución 18
Figura 2.3 Esquema Software de la Solución 18
Figura 3.1. Esquema físico resultante en Docker 1.11 21
Figura 5.1. Visualizador con una instancia del Servicio Master 27
Figura 5.2 Visualizador con 5 Instancias del servicio Worker. 28
Figura 5.3 Visualizador con 11 Instancias del servicio Worker 29
Figura 5.4 Visualizador con 4 Instancias del servicio Worker. 29
Figura 5.5 Pantalla inicio *Spark*, muestra los nodos (y sus estados) y las aplicaciones (activas e histórico). 30
Figura 5.6 Pantalla Principal de la aplicación ejecutada en *Spark* 30
Figura 5.7 Información general de los *Jobs* 31
Figura 5.8 Información detallada de un Job en particular, y los escenarios activos. 31
Figura 5.9 Detalles de un escenario específico. 32
Figura 5.10 Detalles de la ejecución en Bash (I) 32
Figura 5.11 Detalles de la ejecución en Bash (II) 33
Figura 6.1 Definición de la función Benchmark. 34
Figura 6.2 Ejecución en Spark 1.6.2. 34
Figura 6.3 Ejecución en Spark 2.0.0. 35
Figura 6.4 Comparativa Clúster. Gráfica 37

Índice de tablas

Tabla 1.1 Comandos Docker (v1.11 y posteriores) 13

Tabla 1.2 Comandos Docker Swarm (v1.12).. 14

Tabla 1.3 Comandos Docker service (v1.12) 14

Tabla 1.4 Comandos Docker node (v1.12) 14

Tabla 6.1 Benchmark de las versiones de Spark 35

Tabla 6.2 Comparativa Clúster. Tabla de Tiempos.. 36

Tabla 9.1. Tabla Presupuesto detallado.. 41

Capítulo 1. Introducción

1.1 Big Data

1.1.1 ¿Qué es Big Data?

Por Big Data nos referimos al tratamiento y análisis de grandes conjuntos de datos los cuales son tan desproporcionadamente grandes que resulta imposible tratarlos con las herramientas de bases de datos y analíticas convencionales. La tendencia se encuadra en un entorno que no nos suena para nada extraño: la proliferación de páginas web, aplicaciones de imagen y vídeo, redes sociales, dispositivos móviles, apps, sensores, internet de las cosas, etc. capaces de generar, según IBM, más de 3 exabytes al día, hasta el punto de que el 90% de los datos del mundo han sido creados durante los últimos dos años.

Hablamos de un entorno absolutamente relevante para muchos aspectos, desde el análisis de fenómenos naturales como el clima hasta, por supuesto, el ámbito empresarial. Y es precisamente en ese ámbito donde las empresas desarrollan su actividad donde está surgiendo un interés que convierte a Big Data en algo así como "*the next buzzword*", o simplemente, el nuevo término de moda. En un momento en que la mayoría de los directivos nunca se han sentado delante de una simple página de Google Analytics y se sorprenden poderosamente cuando ven lo que es capaz de hacer, llega un grupo de herramientas diseñadas para que el análisis de datos masivos pueda tener sentido.

1.1.2 Las "Uves" del Big Data

Son un conjunto de propiedades o dimensiones de los datos que caracterizan el concepto "Big Data".

El **volumen** se refiere a la cantidad de datos, la **variedad** se refiere a los diferentes de tipos de datos que se estudian, y la **velocidad** se refiere a la velocidad de procesamiento de datos.

Estas tres dimensiones fueron introducidas por el analista de Gartner, Doug Laney en 2001 en un artículo de investigación de MetaGroup [1]. Posteriormente, se han ido proponiendo nuevas dimensiones de los datos llegando (de acuerdo a algunos autores) a las 5 dimensiones; además de las citadas anteriormente hay que añadir: **valor** potencial de los datos y la **veracidad**, exactitud o integridad de los mismos. Estas dos últimas dimensiones, aun estando aceptadas por algunos autores, no definen a los datos y únicamente son cuestiones a las que aspira cualquier conjunto de datos, por lo que no los tendremos en cuenta como dimensión, igual que promueve D. Laney en un artículo de noviembre de 2013 [15].

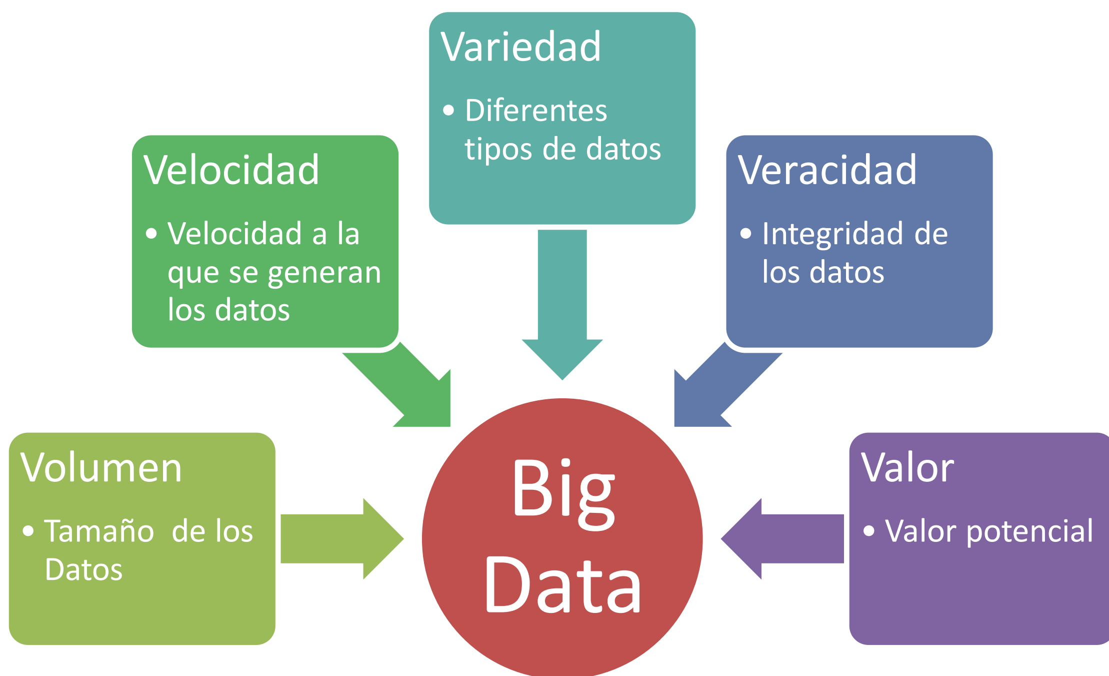


Figura 1.1 Las 5 “uves” del Big Data.

1.2 Apache Spark

1.2.1 ¿Qué es Spark?

Desarrollado en el lenguaje Scala, *Spark* es una plataforma de computación de código abierto para análisis y procesos avanzados que tiene muchas ventajas sobre *Hadoop*. Desde el principio *Spark* fue diseñado para soportar en memoria algoritmos iterativos que se pudiesen desarrollar sin escribir un conjunto de resultados cada vez que se procesaba un dato. Esta habilidad para mantener todo en memoria es una técnica de computación de alto rendimiento aplicado al análisis avanzado, la cual permite que *Spark* tenga unas velocidades de procesamiento altísimas.

Spark tiene un *framework* integrado para implementar análisis avanzados que incluye la librería *MLlib*, el motor de grafos *GraphX*, *Spark Streaming*, y la herramienta de consulta *Shark*. Esta plataforma asegura a los usuarios la consistencia en los resultados a través de distintos tipos de análisis.

1.2.2 ¿Por qué *Spark*?

En los siguientes puntos se justifican las ventajas de *Spark* frente a *Hadoop*:

- **Mucha más potencia que *Hadoop***: Spark es un *framework* de análisis distribuido en memoria y nos permite ir más allá de las operaciones en *batch* de *Hadoop MapReduce*.
- ***Spark* es rápido, muy rápido:** *Spark* puede ejecutar análisis de varios órdenes de magnitud más rápido que los despliegues de *Hadoop* existentes. Esto significa una mayor interactividad, una experimentación más rápida y mayor productividad para los analistas.
- ***Spark* puede coexistir con tu arquitectura de Big Data:** Se ha invertido bastante en *clusters* con *Hadoop*. Para *Spark* esto no es un problema ya que puede coexistir con las instalaciones existentes de *Hadoop* y añadir nuevas funcionalidades.
- ***Spark* entiende SQL:** Esto permite realizar consultas sobre datos estructurados dentro de Spark utilizando tanto SQL como la API *Dataset* del propio *framework*.

- ***Spark* mima a los desarrolladores:** *Spark* se ha programado con el lenguaje *Scala* que es un nuevo lenguaje funcional y orientado a objetos. Gracias a *Scala* son capaces de programar de manera muy concisa y fluida soluciones que antes requerían cientos de líneas. Además, se puede programar en Python, R e incluso en Java.
- ***Spark* empieza a ser el motor de *Big Data*:** Ahora mismo *Apache Spark* forma muchos proyectos de Big Data y grandes empresas como IBM, Microsoft, Amazon o Google, lo integran con sus productos de *Big Data.*

1.2.3 Apache *Spark* 2.0

La versión 1.0 salió hace ahora 2 años y, en este lapso de tiempo, ha habido tantos seguidores como detractores de la plataforma. En la versión 2.0 el equipo de desarrollo de *Spark* pretende utilizar todo lo que se ha aprendido en este periodo de tiempo, replicando en esta versión lo que gusta a los usuarios y mejorar activamente los puntos débiles que se comentaban en la comunidad.

Hay que mencionar que las tres principales características de *Apache Spark* 2.0 son: más fácil, más rápido, más inteligente. *Spark* 2.0 es más fácil e intuitivo que su predecesor, esto viene dado por la optimización tanto de SQL como de las API. La parte de SQL se resuelve con la introducción de un nuevo analizador. En las API las modificaciones han sido sustanciales y vienen explicadas con detalle en este artículo de *Databricks* [6].

Spark es más rápido que un compilador, y es que la versión 2.0 incluye el motor *Tungsten*, el cual se basa en las ideas de los compiladores modernos, así como en las bases de datos MPP (Procesamiento Masivo en Paralelo), lo que se aplica al procesamiento de datos. La idea principal es emitir *bytecode* optimizado en tiempo de ejecución, el cual ejecuta toda la consulta en una función eliminando las llamadas a funciones virtuales y mejorando el rendimiento de uso de registros de la CPU para los datos intermedios.

Spark es más inteligente puesto que el *Structured Streaming* API de *Spark* 2.0 es una novedosa forma de usar el *streaming*, que implementa la idea de que la manera más simple de calcular las respuestas sobre los *streams* de datos es la de no tener que razonar sobre el hecho de que es realmente un *stream.* Esto se lleva a cabo con la visión del *Structured Streaming* de utilizar

el optimizador *Catalyst* para saber cuándo es posible transformar de forma transparente un código estático en una ejecución incremental que funciona sobre datos dinámicos e infinitos (*stream*). Cuando se visualiza sobre estas lentes estructuradas de datos (como una tabla discreta o una tabla infinita), el *streaming* se simplifica bastante.

1.3 Docker

1.3.1 ¿Qué es Docker?

Docker es un proyecto de software libre que permite automatizar el despliegue de aplicaciones dentro de contenedores. Le permite empaquetar una aplicación en una unidad estandarizada para el desarrollo de software, que contiene todo lo que necesita para funcionar: código, tiempo de ejecución, herramientas y bibliotecas del sistema, etc.

Docker usa recursos del sistema totalmente aislado, es decir, permite asignar recursos (tiempo de CPU, la memoria del sistema, el ancho de banda de la red, etc.) o combinaciones de estos recursos entre los procesos que se ejecutan en un sistema.

1.3.2 Beneficios

Ligero

El peso de este sistema no tiene comparación con cualquier otro sistema más convencional que estemos acostumbrados a usar. Por ejemplo, cualquier imagen de Ubuntu que queramos usar en otro equipo pesará entorno a 1Gb si contamos únicamente con la instalación limpia del sistema. En cambio, con el uso de docker, un Ubuntu con Apache y una aplicación web, pesa alrededor de 180Mb, lo que nos demuestra un significativo ahorro a la hora de almacenar diversos contenedores que podamos desplegar con posterioridad.

Abierto

Docker es igual a decir portable, es decir, cualquier contenedor Docker podremos desplegarlo en otro sistema (que soporte esta tecnología), con lo que

nos ahorraremos el tener que instalar en este nuevo entorno todas aquellas aplicaciones que normalmente usemos.

Autosuficiencia

Un contenedor Docker no contiene todo un sistema completo, sino únicamente aquellas librerías, archivos y configuraciones necesarias para desplegar las funcionalidades que contenga. Asimismo, Docker se encarga de la gestión del contenedor y de las aplicaciones que contenga.

Además, su ligereza es un aspecto ideal para el usuario final, puesto que incluso en equipos moderadamente antiguos se desenvuelve prácticamente igual que el sistema anfitrión, además de ofrecernos un entorno similar a Git para, a base de "capas", controlar cada cambio que se haga en la máquina virtual o contenedor.

1.3.3 Componentes de Docker

Contenedores

Un contenedor es simplemente un proceso para el sistema operativo que, internamente, contiene la aplicación que queremos ejecutar y todas sus dependencias. La aplicación contenida solamente tiene visibilidad sobre el sistema de ficheros virtual del contenedor y utiliza indirectamente el kernel del sistema operativo principal para ejecutarse.

Imágenes

Quizá la parte más atractiva de Docker es la introducción del concepto de imágenes, una especie de contenedores ya preparados que sirven de base para nuevas imágenes o para poner directamente en marcha servicios. Para poder crear estas imágenes o modificarlas Docker pone a nuestra disposición Dockerfiles, donde usando algunas palabras clave podremos customizar un sistema operativo adaptado a nuestras necesidades.

Repositorios

También conocidos como Registros Docker, contienen imágenes creadas por los usuarios y puestas a disposición del público. Podemos encontrar repositorios públicos y totalmente gratuitos, o repositorios privados donde tendremos que comprar las imágenes que necesitemos. Estos registros

permiten desarrollar o desplegar aplicaciones de forma simple y rápida en base a plantillas, reduciendo el tiempo de creación o implementación de aplicaciones o sistemas.

Servicios y tareas

En la última versión de Docker se incluyen dos nuevos conceptos: servicio y tarea. Una tarea es un contenedor en ejecución, por lo tanto, es un sistema operativo completo que realiza trabajo efectivo (es decir consume recursos), y sería similar a un "proceso" para el sistema operativo. Por otra parte, se agrega también el término servicio, el cual no es más que un conjunto de tareas iguales.

1.3.4 Virtualización y Docker: ¿es lo mismo?

Como hemos ido indicando a lo largo de este capítulo, a priori docker es un paso adelante en la administración de sistemas, pero diferente a la alternativa de la virtualización convencional.

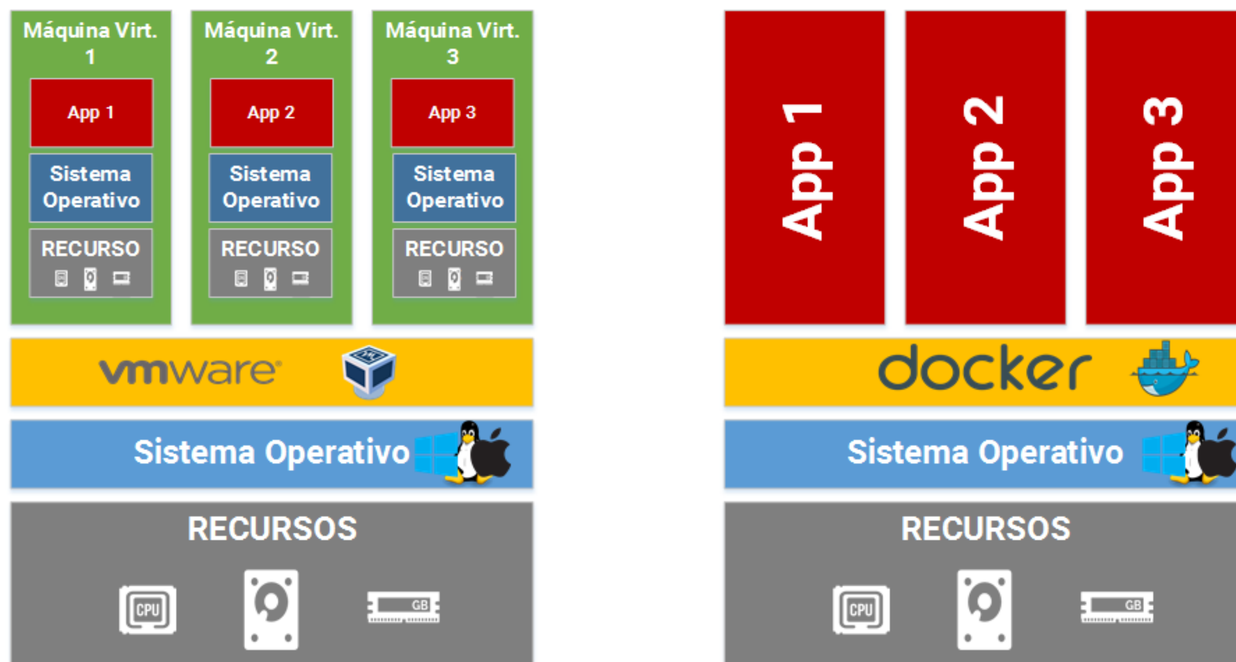


Figura 1.2 Estructura (Virtualización Convencional y Docker)

La virtualización convencional contiene (por cada máquina creada) los siguientes elementos: la aplicación, los binarios y librerías necesarias, y todo un sistema operativo invitado. Frente a esto vemos que Docker simplifica bastante el esquema ya que los contenedores incluyen la aplicación y todas

sus dependencias, pero comparten el núcleo con otros contenedores funcionando como procesos aislados en el espacio de usuario del sistema operativo anfitrión. Estos contenedores no están vinculados a ninguna infraestructura específica: se ejecutan en cualquier ordenador, en cualquier infraestructura, y en cualquier "nube".

1.3.5 Una clase de Docker. Comandos útiles

En la siguiente Tabla 1.1 se muestran los comandos básicos de Docker.

Comando	**Descripción**	**Referencia**
ps	Muestra los contenedores que están corriendo, si le pasamos -a nos mostrará también los contenedores que hemos detenido.	Enlace
attach	Se conecta a un contenedor que está corriendo, podemos hacerlo por el nombre o por el ID de contenedor	Enlace
start	Inicia un contenedor detenido. Lo podemos hacer por el Nombre o el ID	Enlace
stop	Detiene un contenedor	Enlace
inspect	Muestra informaciones de bajo nivel del contenedor o la imagen.	Enlace
export	Exporta el contenido del sistema de archivo de un contenedor a un archivo tar	Enlace
search	Busca una imagen en el registro de Docker	Enlace
pull	Descarga una imagen del registro de Docker	Enlace
images	Muestra las imágenes que tenemos disponible localmente	Enlace
rmi	Elimina una o más imagen	Enlace
build	Crea una imagen a partir de un DockerFile	Enlace
commit	Crea una imagen a partir de un contenedor	Enlace
push	Envía un repositorio a DockerHub	Enlace
run	Ejecuta un contenedor en el sistema Docker	Enlace

Tabla 1.1 Comandos Docker (v1.11 y posteriores)

Comando	Descripción	Referencia
init	Inicia un cluster Swarm (Crea nodo Master)	Enlace
join	Une un nodo a un cluster	Enlace
leave	Saca un nodo del cluster	Enlace
update	Actualiza el cluster	Enlace

Tabla 1.2 Comandos Docker Swarm (v1.12)

Comando	Descripción	Referencia
create	Crea un nuevo servicio	Enlace
inspect	Muestra información de un servicio	Enlace
ls	Lista los servicios activos	Enlace
rm	Elimina un servicio (y contenedores asociados)	Enlace
scale	Escala un servicio (tanto aumenta como reduce)	Enlace
ps	Lista las tareas asociadas a un servicio	Enlace
update	Actualiza un servicio	Enlace

Tabla 1.3 Comandos Docker service (v1.12)

Comando	Descripción	Referencia
inspect	Muestra información de un nodo	Enlace
ls	Lista nodos	Enlace
promote	Convierte uno o más nodos en Master	Enlace
rm	Elimina un nodo	Enlace
tasks	Lista las tareas activas de un nodo	Enlace
update	Actualiza un nodo	Enlace

Tabla 1.4 Comandos Docker node (v1.12)

Capítulo 2. Análisis de la problemática

2.1 Descripción del Problema

El problema a resolver en este proyecto es desarrollar y desplegar un clúster que permita crear un sistema de Big Data, de manera automática por parte de cualquier usuario. Para ello, se han hecho uso de varias herramientas ya comentadas: *Docker* y *Apache Spark*.

2.2 Antecedentes

2.2.1 DSBox

Un antecedente a este TFG es un proyecto en *github* denominado *DSbox* [11]. El resultado pretende ser el mismo, aunque desde otro enfoque y despliegue. Este proyecto utiliza la herramienta *Vagrant*, en sustitución de Docker, la cual es una herramienta para la creación y configuración de entornos de desarrollo virtualizados. Entre los posibles inconvenientes se podría citar aquellos asociados a la virtualización convencional, ya descritos en el capítulo anterior.

2.3 Estado del Arte

Existen trabajos similares por la novedad de la temática y herramientas utilizadas.

2.3.1 Stratio

La empresa española Stratio publicó un proyecto en *DockerHub* referente a *Spark* [12]. Entendemos que no es exactamente igual dado que la creación de un clúster *Spark* por medio de su propuesta puede ser un poco tedioso, y nuestra implementación mejora este aspecto, ya que es fácilmente escalable.

2.4 Objetivos

Al inicio de este proyecto propusimos como objetivo principal inicial realizar un *benchmarking* de *Spark*, mediante una comparativa entre lanzar un mismo trabajo modificando algunos parámetros: como el número de *cores*, la cantidad de memoria RAM asignada, o el número de nodos presentes al clúster.

Por otra parte, decidimos realizar el despliegue del clúster con la herramienta Docker debido al interés mostrado por el equipo responsable del Centro de Cálculo, lugar donde realizaríamos el despliegue de prueba.

Desafortunadamente, el realizar el despliegue sobre Docker supuso prácticamente todo el tiempo del proyecto, aunque el rendimiento valía la pena. Por ello, designamos como objetivo principal el despliegue de un clúster *Spark* sobre Docker 1.12, versión que pasó a producción el 29 de Julio de 2016.

Figura 2.1 "Docker 1.12 is now Available"

2.5 Fases

2.5.1 Análisis del caso de estudio

En una primera fase del proyecto, dedicamos tiempo a intentar entender las soluciones a implementar para ver qué problemas o errores podrían surgir en las siguientes fases. Por ello, consultamos diversas fuentes sobre Docker, *Spark* y administración de sistemas en general.

2.5.2 Despliegue Solución en Docker v1.11

Una vez tomada la primera prueba de contacto con Docker comenzamos a realizar nuestras propias imágenes Docker, creando los *Dockerfiles* y Scripts para realizar el despliegue del servicio. Una vez lanzado el servicio y realizadas algunas pruebas para determinar su correcto funcionamiento, damos por finalizada esta fase.

2.5.3 Despliegue Solución en Docker v1.12

En el transcurso de este proyecto, desde Docker se lanzó una primera versión *"release candidate"* de una nueva versión. Tras consultar las nuevas funcionalidades adquiridas, y las mejoras de las anteriores, decidimos migrar el servicio a esta nueva versión debido a que simplificaba muchos pasos, y se añadían algunos nuevos de manera automática, como por ejemplo la seguridad interna del clúster. En esta fase creamos dos versiones estructuralmente diferentes. Un modelo seguiría la antigua filosofía de Docker con el uso de contenedores (*Containers as a Service*) y otra que utilizaría conceptos introducidos en la versión, desplegando tareas (*Tasks as a Service*).

2.5.4 Optimización y creación de los benchmark

Una vez realizado el despliegue de la solución en la versión 1.12.rc, actualizamos todos los sistemas a la versión definitiva de la herramienta 1.12.1. La cuál introducía algunos cambios que hubo que subsanar, y también en este punto empezamos la optimización de los *scripts* de *bash* y *Dockerfiles.*

2.6 Esquema Hardware

Para llevar a cabo este proyecto se ha hecho uso de los recursos cedidos por parte del Centro de Cálculo de la Escuela Superior de Ingeniería y Tecnología. El clúster cuenta con 6 Equipos, cada uno de ellos con un procesador i5 (4 *cores*) de Intel y 8GB de RAM.

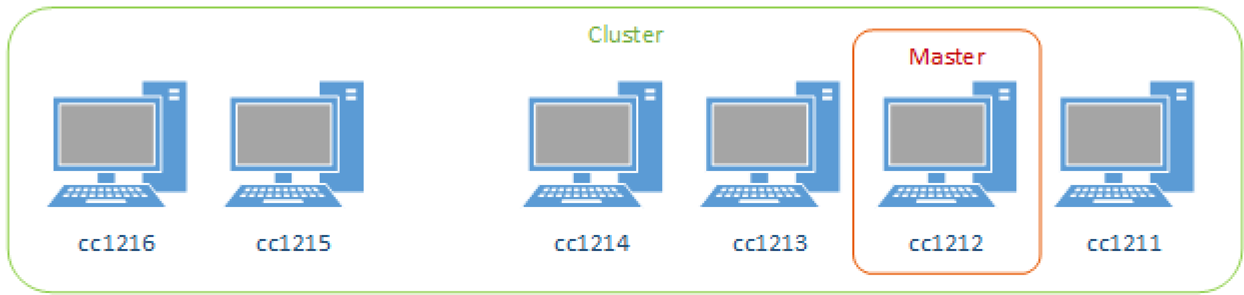


Figura 2.2 Esquema Hardware de la solución

2.7 Esquema Software

En cuanto al Software utilizado, cada equipo cuenta con un sistema operativo Ubuntu 14.04 y Docker 1.12. Y cada contenedor lanzará Apache *Spark* 2.0.0 sobre una base Ubuntu Xenial (16.04).

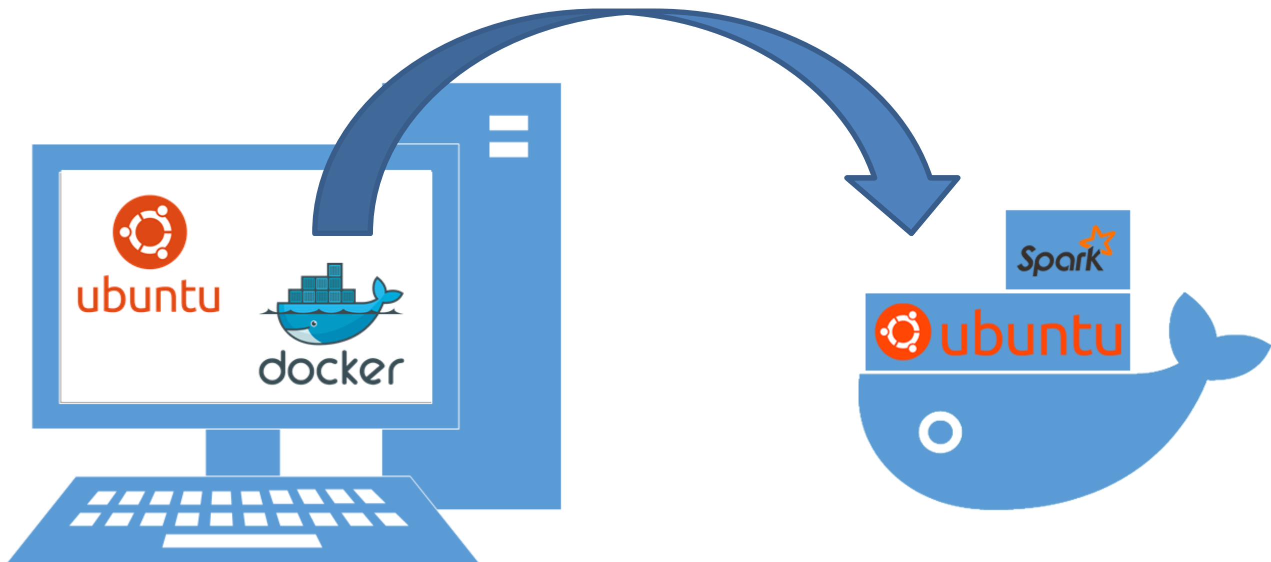


Figura 2.3 Esquema Software de la Solución

Capítulo 3. Solución Basada en Docker 1.11

3.1 Conceptos importantes en Docker 1.11

Para entender bien el despliegue realizado en esta versión de la herramienta deberemos de conocer de antemano algunos conceptos básicos.

Swarm

Swarm es una imagen proporcionada por Docker que crea en cada nodo físico un contenedor que permite la conexión de varios equipos físicos, generándose así el clúster. Esta imagen nos da la opción de declarar si un nodo es *Manager* (nodo encargado de supervisar el clúster) o *Worker* (nodo sin ninguna responsabilidad frente al sistema).

Consul

Al igual que el *Swarm*, es una imagen de Docker aunque suministrada por HashiCorp [13]. *Consul* es un centro de datos en tiempo de ejecución que proporciona el descubrimiento de servicios, configuración y capacidades de orquestación. En resumen, permite crear de manera segura, intuitiva y fácil un clúster Docker en producción o en desarrollo, así como las conexiones internas entre ellos.

3.2 Despliegue de la solución (v1.11)

En este apartado comentaremos cómo implantamos la solución propuesta al problema en la versión que nos atañe. El software que se utilizó fue Docker 1.11, Swarm, progrium/cónsul y Apache Spark 1.5.2.

Como primer paso, deberemos parar el servicio Docker de nuestros equipos, para poder configurar algunos aspectos en el demonio, o no podríamos realizar esta solución.

```
stop docker
```

Ahora podremos volver a lanzar el servicio de Docker, haciendo uso de su demonio, declarando algunos *flags* para indicarle determinadas opciones.

```
docker daemon -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock --cluster-
store=consul://<IP_CONSUL>:8500 --cluster-advertise=eth0:2375
```

Una vez completado el paso anterior para cada uno de los nodos físicos del clúster, pasaremos a declarar en el nodo, y solo en él, con IP igual a <IP_CONSUL>, la imagen de *Consul* y para ello haremos uso del siguiente comando.

```
docker run -d -p 8500:8500 --name=consul progrium/consul -server -bootstrap
```

Una vez configurado el Docker *Daemon* y creado el *Consul*, pasaremos a definir el clúster *Swarm*. Para ello deberemos de crear un esquema y decidir cuantos *Managers* y *Workers* declararemos. En nuestro caso, por simplificar, crearemos 2 nodos *Manager* y 2 nodos *Worker*.

```
//CREANDO MANAGER 0
docker run -d -p 4000:4000 --name=manager0 swarm manage -H :4000 --replication --
advertise <IP_MASTER0>:4000 consul://<IP_CONSUL>:8500
//CREANDO MANAGER 1
docker run -d -p 4000:4000 --name=manager1 swarm manage -H :4000 --replication --
advertise <IP_MANAGER1>:4000 consul://<IP_CONSUL>:8500
//CREANDO NODE 0
docker run -d --name=node0 swarm join --advertise=<IP_NODE0>:2375
consul://<IP_CONSUL>:8500
//CREANDO NODE 1
docker run -d --name=node1 swarm join --advertise=<IP_NODE1>:2375
consul://<IP_CONSUL>:8500
```

Tras ejecutar en cada máquina cada comando tendremos un sistema algo similar a este, siempre que hayamos seguido los pasos anteriores:

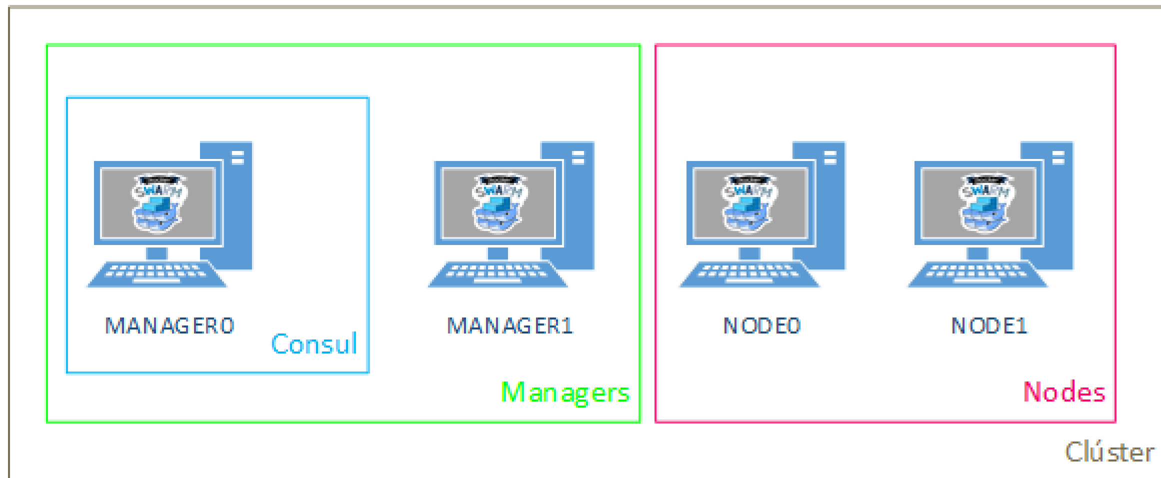


Figura 3.1. Esquema físico resultante en Docker 1.11

También podremos ver un este esquema, aunque no tan gráfico, si accedemos a: `<IP_CONSUL:PORT>/ui/#/dc1/kv/docker/swarm/leader/edit`

Ahora deberemos crear una red *Overlay*, que nos permite poder arrancar contenedores en el *Swarm* y que estos simulen estar en una misma red, para que no existan problemas con su visibilidad.

```
docker network create -d overlay --subnet=192.168.100.0/24 --ip-range=192.168.100.0/24 mynet
```

Con este último paso, ya tenemos creado y configurado por completo todo el sistema, ya solo nos quedaría arrancar contenedores. Para ello utilizaremos el comando Docker run.

```
docker -H :4000 run --net=mynet -dti ubuntu:latest
```

Para nuestro problema deberemos utilizar una imagen básica con Ubuntu y Spark, y posteriormente configurarla.

```
// EN SPARK_MASTER
export SPARK_LOCAL_IP= `ip a ls dev eth0 | sed -n "s,.*inet  *\([^/]*\)/.*,\1,p"`
export SPARK_MASTER_IP=192.168.100.3 // (Siempre que coincida Manager0
                                     //  y Spark_Master)
cd /opt/spark/sbin
./start-master.sh
```

```
// EN SPARK_SLAVE
export SPARK_LOCAL_IP= `ip a ls dev eth0 | sed -n "s,.*inet  *\([^/]*\)/.*,\1,p"`
export SPARK_MASTER_IP=192.168.100.3 // (Siempre que coincida Manager0
                                     //  y Spark_Master)
cd /opt/spark/sbin
./start-slave.sh spark://$SPARK_MASTER_IP:7077
```

Capítulo 4.
Solución Basada en Docker 1.12

4.1 Novedades introducidas en Docker 1.12

Con el lanzamiento de Docker 1.12, se incluyeron algunos aspectos nuevos como los comandos *service* y *node*, sobre los que hemos basado estas soluciones.

Por otra parte, se han "perdido" otros aspectos ya que en esta nueva versión no es necesario declarar explícitamente el *Consul* y lo hace Docker por nosotros. Además facilita muchísimo la seguridad del clúster y hace que su escalado y orquestación se reduzca en tiempo y espacio, y por ende, en dificultad.

4.2 Despliegue de la solución (v1.12)

Para solucionar este problema hemos optado por diseñar dos modelos conceptualmente dispares, pero funcionalmente esto debería ser trasparente para el usuario, *Container as a Service*, donde respetamos la antigua filosofía de Docker y utilizamos el comando `docker run`, y por otro lado un modelo *Task as a Service*, donde hacemos uso del comando *docker service*.

4.2.1 Usando *Tasks as a Service*

Para llevar a cabo esta configuración deberemos crear previamente nuestro clúster. Esto, como veremos, se simplifica bastante respecto a Docker v1.11. Para crearlo ejecutaremos en cada nodo el comando pertinente de la lista a continuación:

```
docker swarm init --advertise-addr <IP_MANAGER> --listen-addr <IP_MANAGER>:2377
```

Con esta línea ya tendríamos inicializado el manager del clúster, olvidándonos de configurar el *daemon* o de inicializar el *Consul*; únicamente le indicamos que se quede escuchando en una IP y por un puerto a todo aquel que quiera ingresar.

Para unirnos al clúster Docker nos exigirá incluir un *token*, que funcionará como contraseña. Ahora procederemos a unir el resto de nodos a nuestro clúster, ejecutando el siguiente comando en cada uno de ellos.

```
docker swarm join-token -q worker

docker swarm join <IP_MANAGER>:2377 --token <Token>
```

Podremos indicar al unirnos que queremos que sea como *manager* modificando el *token* por *master*. Igualmente Docker nos provee mecanismos para hacerlo a *posteriori* con `docker node promote`, si no lo indicamos se unirá como *Worker*. Finalmente, podemos ver el estado de nuestro cluster así como la configuración de sus roles ejecutando `docker node ls`.

Ahora pasaremos a crear una red *Overlay*, como hicimos en la anterior versión, y para ver el estado en directo de los nodos del cluster haremos uso de un visualizador provisto por la comunidad de usuarios de Docker [18] `turaaa/swarmvisualizer:latest`.

```
docker run --name visualizer -it -d -p 5000:5000 -e HOST=10.209.31.36 -e PORT=5000 -v
/var/run/docker.sock:/var/run/docker.sock manomarks/visualizer

docker network create -d overlay --subnet=192.168.100.0/24 --ip-
range=192.168.100.0/24  mynet
```

De esta forma el sistema ya estaría completo. Ahora solo quedaría ejecutar cada uno de los servicios *Spark*, para ello hemos provisto tres imágenes Docker:

- **Master**: Representa al nodo Master de Spark, que deberá ser ejecutado previamente al resto de servicios de a continuación. Este servicio se encarga de iniciar Spark_Master.sh
- **Worker**: Representa al nodo Worker de Spark. Este servicio se encarga de iniciar `Spark_Slave.sh spark://<SPARK_MASTER_IP>:7077`
- **Base**: Creado con vistas a instalar en él balanceadores de trabajos como *MESOS* u otras herramientas. Desde aquí serán lanzados los trabajos de *SPARK*, aunque pueden ser lanzados desde el propio *MASTER*, lo cual

es más rápido ya que podremos consultar mejor las herramientas web proporcionadas por *Spark*.

Para lanzar los servicios usaremos los siguientes comandos:

```
#lanzar service Master

docker service create --network mynet --name master -p 8080:8080 -p 7077:7077 -p
4040:4040 sersantin/ssm:latest


#lanzar service worker (2 Instancias [configurable --replicas valor])

docker service create --network mynet --name worker -p 8081:8081 --replicas 2
sersantin/ssw:latest
```

Es conveniente esperar a que el servicio `master` se encuentre en estado `Running` para evitar errores al unir los Workers. En cuanto se hayan desplegado los servicios, podremos ver que en <IP_MASTER>:5000 existen tantas cajas como instancias de los servicios existen, así como si comprobamos en <IP_MASTER>:8080 veremos que aparecen los Workers unidos al master.

Este modelo cuenta con la ventaja de que es "elástico", es decir, podremos añadir cuantos nodos worker deseemos haciendo uso del comando `docker service scale worker=15` y el sistema añadirá tantas instancias como fuese necesario para llegar a 15; por lo que si volvemos a introducir `docker service scale worker=1,` el sistema eliminará tantas instancias como fuese necesario para llegar a 1.

Para ejecutar un trabajo *Spark* tendremos que acceder al contenedor `master` usando `docker service tasks master`, y luego `docker ps` en el nodo en el que esté la tarea. Una vez conocido el ID del container, en la máquina en la que se encuentre, ejecutaremos `docker exec -ti <ID_CONTAINER> bash`, para poder acceder al contenedor. Luego solo necesitaremos enviar un trabajo de ejemplo.

```
MASTER=spark://<IP_MASTER>:7077 ./run-example SparkPi 1000
```

4.2.2 Usando *Containers as a Service*

Para hacer uso de este modelo únicamente tendremos que conocer la IP de cada uno de los nodos, y que todos se encuentren en una misma red. Por ejemplo, nosotros tendremos estas:

Equipo	IP
PC1 (*Master Spark*)	192.168.100.3
PC2	192.168.100.4
PC3	192.168.100.5

Por lo tanto, para simular el clúster, iniciaremos un contenedor en cada uno de los nodos, especificando la red *Host* y el rol dentro de Spark que realizará el nodo, de la siguiente forma.

```
//PC1 (Spark Manager)
docker run -ti -d --name master --network host -p 4040:4040 -p 8080:8080 -p 7077:7077
sersantin/scm
//PC2
docker run -ti -d --name worker0 --network host sersantin/scw
//PC3
docker run -ti -d --name worker1 --network host sersantin/scw
```

Tras arrancar cada contenedor estas imágenes están preparadas para hacer las conexiones entre Workers – Master de manera automática, ahora solo queda ejecutar una terminal en la máquina master y ejecutar nuestro trabajo Spark. Para ello ejecutaremos `docker exec -ti master bash`, o simplemente `docker attach master,` para poder ejecutar una consola dentro del contenedor master, luego solo tendremos que ejecutar el trabajo de Spark.

Capítulo 5.
Ejecución de la solución.

5.1 Pruebas de ejecución de la solución

Una vez creado nuestro clúster *Spark-Docker* 1.12, podemos ponerlo a prueba. Para ello, hemos optado por utilizar la versión basada en servicios para la muestra de estos resultados, debido a que consideramos que es más óptima y flexible. Para mostrar la ejecución utilizaremos el visualizador provisto por DovAmir [18].

Empezaremos ejecutando el contenedor *master*. Para ello haremos uso del comando: `docker service create --network mynet --name master -p 8080:8080 -p 7077:7077 -p 4040:4040 sersantin/ssm:latest`. Con ello crearemos un servicio con una única instancia (contenedor):

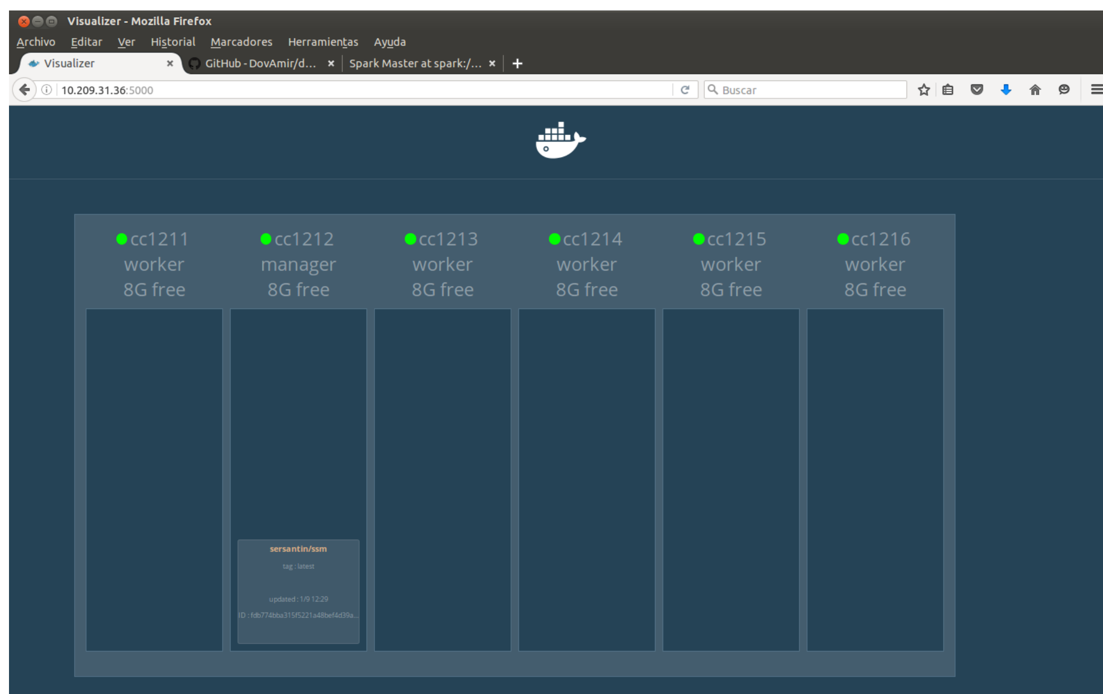


Figura 5.1. Visualizador con una instancia del Servicio Master

Ahora lanzaremos 5 instancias del servicio *worker*, `docker service create --network mynet --name worker -p 8081:8081 --replicas 2 sersantin/ssw:latest`, lo que nos generará:

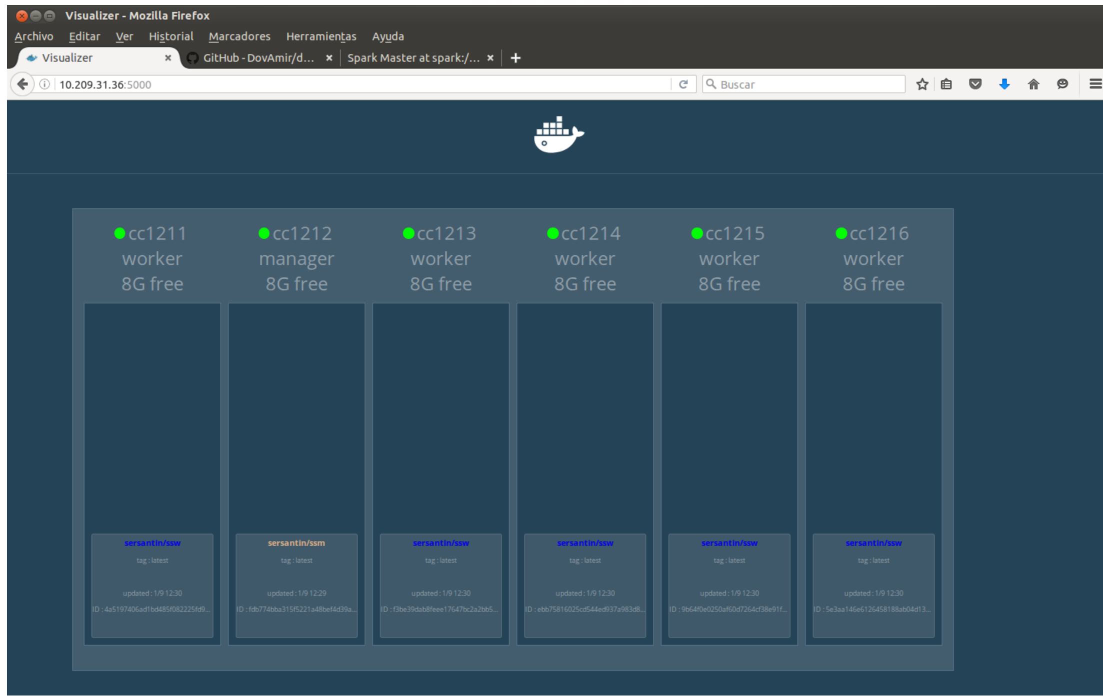


Figura 5.2 Visualizador con 5 Instancias del servicio Worker.

También podemos jugar con la escalabilidad de este sistema, y lo aumentaremos hasta las 11 instancias, `docker service scale worker=11`. Como vemos, el servicio es capaz de balancearse, quedando todos los nodos con el mismo número de contenedores.

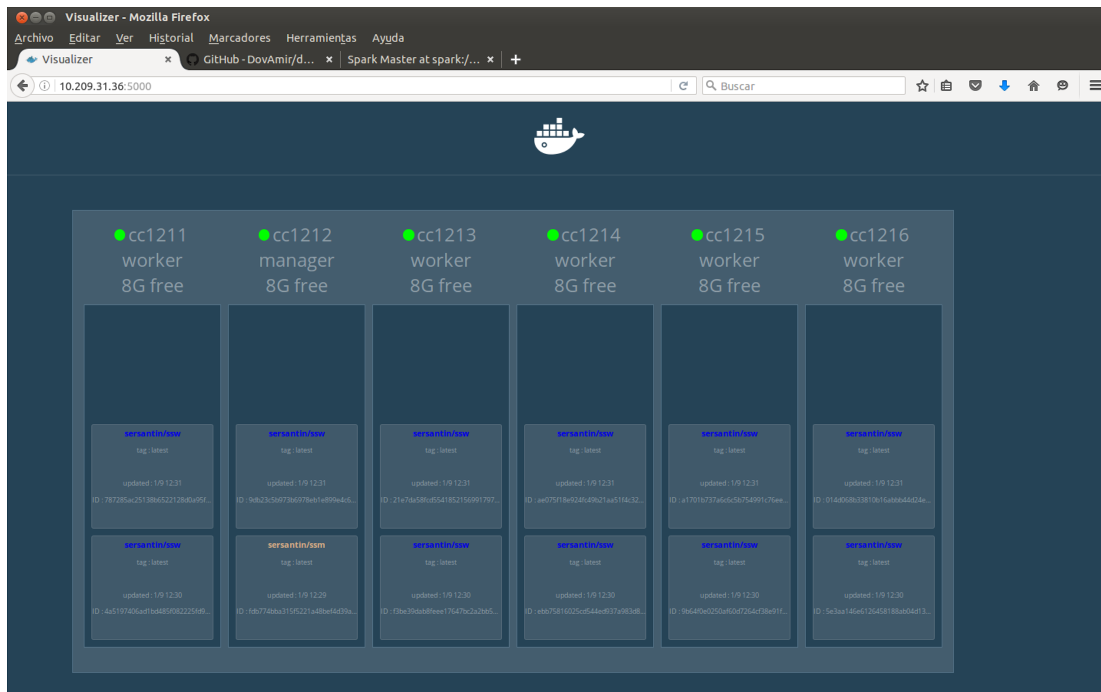


Figura 5.3 Visualizador con 11 Instancias del servicio Worker.

Ahora pensemos que queremos reducir el número de contenedores. Podemos hacerlo con el mismo comando, pero reduciendo el número de 11 instancias, `docker service scale worker=4`.

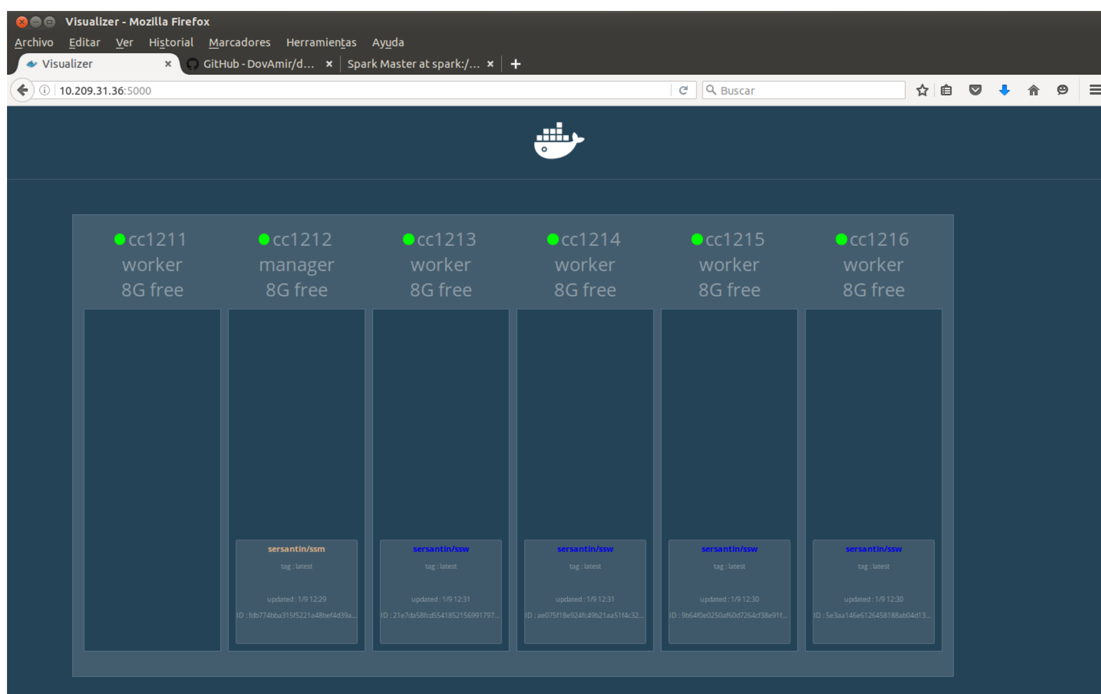


Figura 5.4 Visualizador con 4 Instancias del servicio Worker.

Ya hemos probado los resultados de Docker, por lo que ahora lanzaremos un trabajo en Spark para ver cómo es su funcionamiento. Para ello nos conectaremos al equipo donde se encuentre el servicio *master*, y haremos `docker exec -ti <Id_Container> bash`. Luego, ejecutaremos el script `run-example.sh` presente en la carpeta `/opt/spark/bin`, indicando que este debe de ejecutarse sobre el clúster y no sobre el nodo en solitario, usando `MASTER=spark://192.168.100.3:7077 ./run-example.sh SparkPi 1000`. También mostraremos las distintas, pantallas que ofrece Spark para visualizar la ejecución de nuestros *Jobs*.

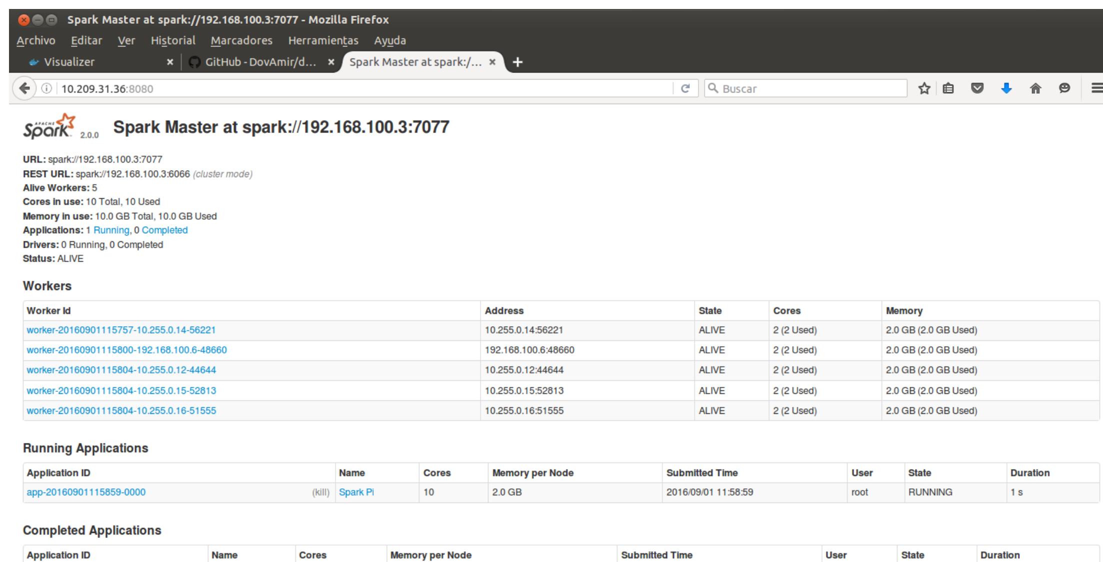


Figura 5.5 Pantalla inicio *Spark*, muestra los nodos (y sus estados) y las aplicaciones (activas e histórico).

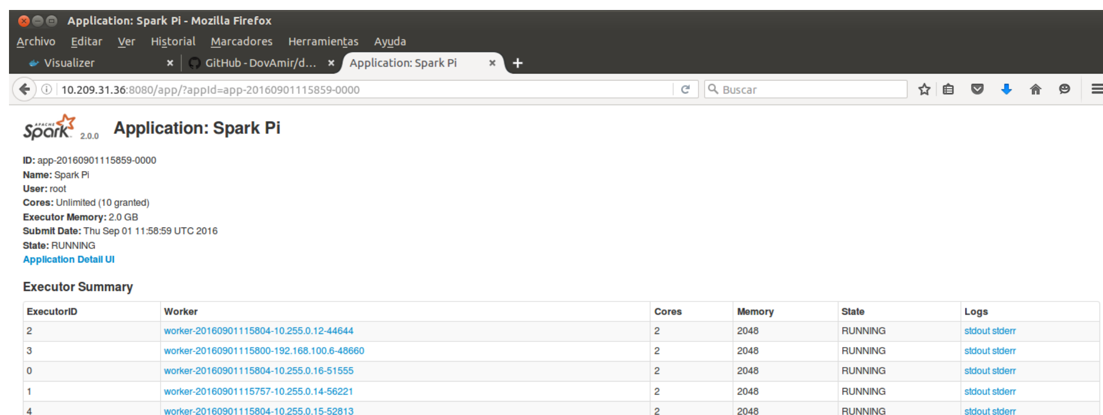


Figura 5.6 Pantalla Principal de la aplicación ejecutada en *Spark*

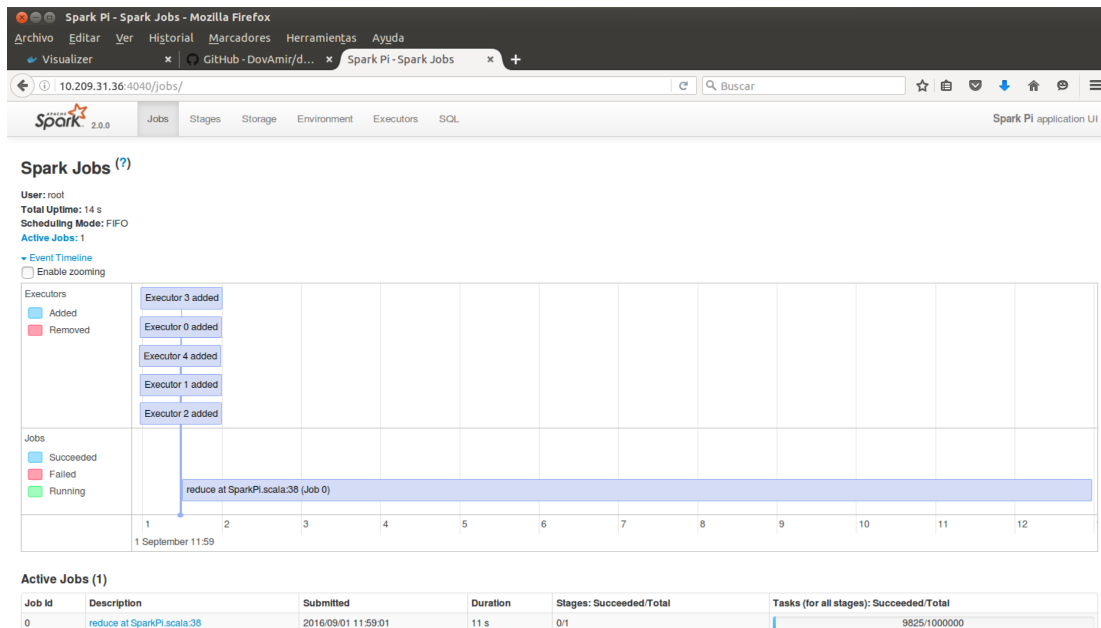


Figura 5.7 Información general de los *Jobs*

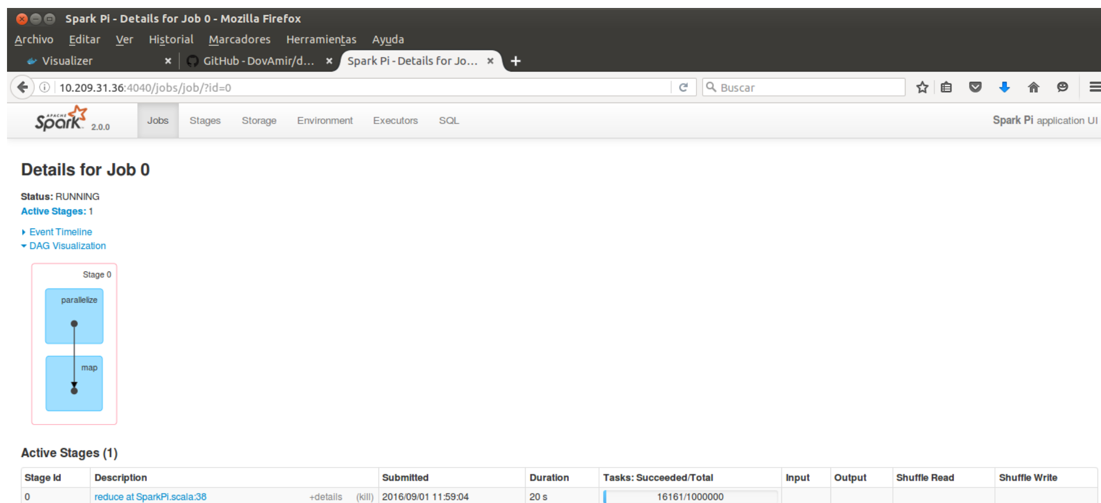


Figura 5.8 Información detallada de un Job en particular, y los escenarios activos.

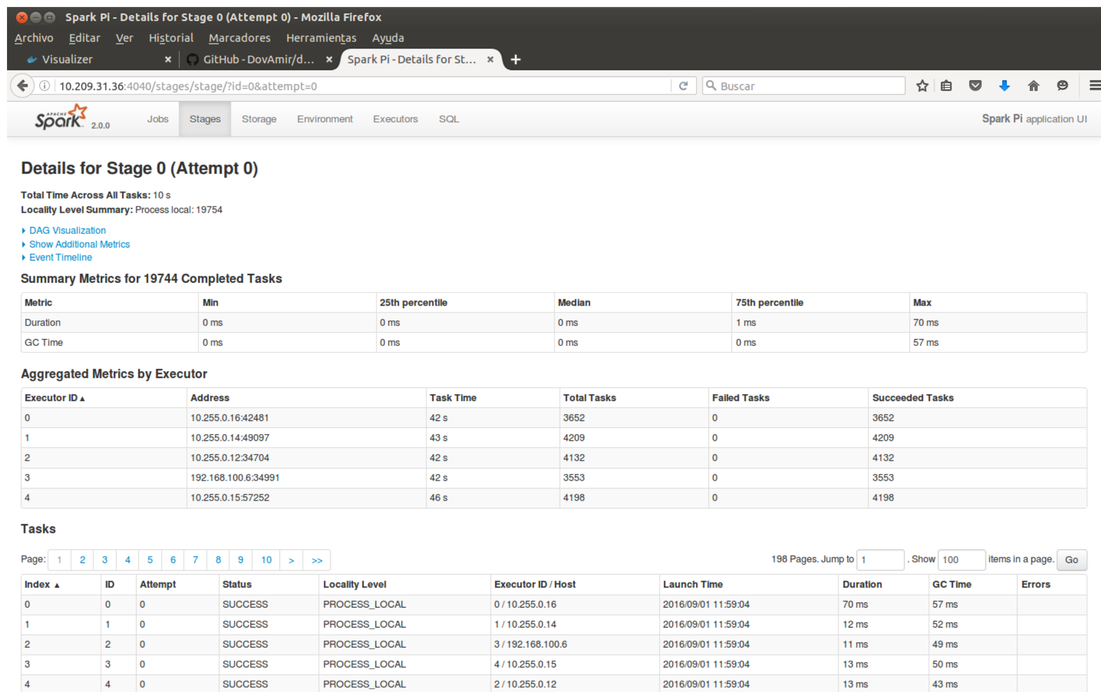


Figura 5.9 Detalles de un escenario específico.

Por último, mostraremos dos capturas desde el punto de vista de la consola Linux, donde aparece la ejecución de la aplicación *Spark*.

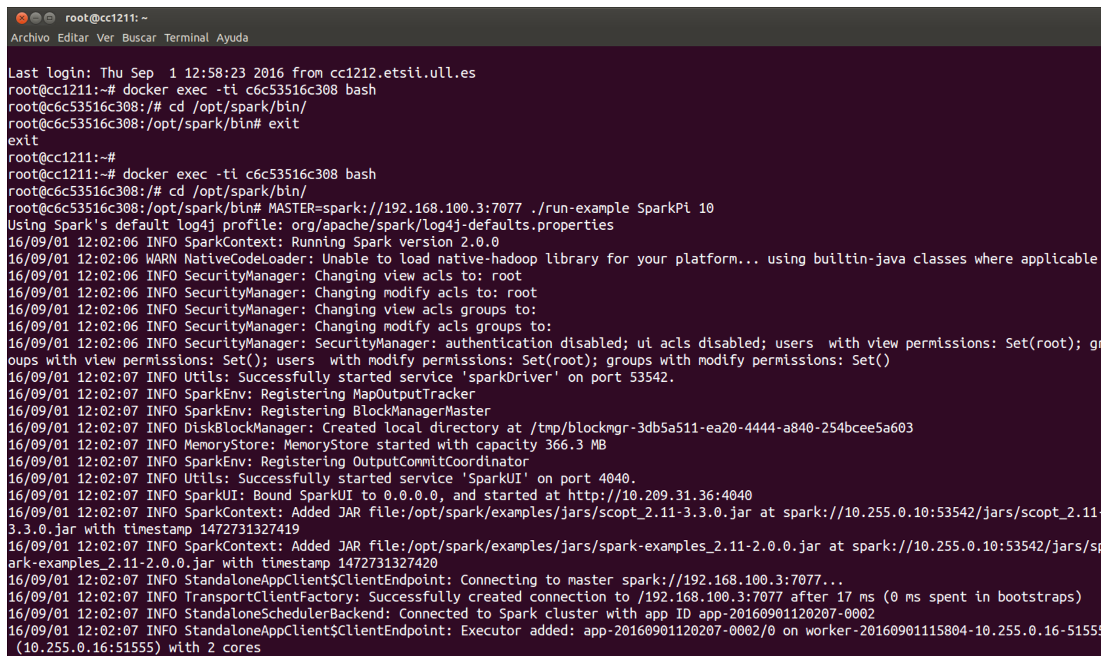


Figura 5.10 Detalles de la ejecución en Bash (I)

```
root@cc1211: ~
Archivo Editar Ver Buscar Terminal Ayuda
16/09/01 12:02:09 INFO TaskSetManager: Starting task 4.0 in stage 0.0 (TID 4, 10.255.0.16, partition 4, PROCESS_LOCAL, 5474 bytes)
16/09/01 12:02:09 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Launching task 4 on executor id: 0 hostname: 10.255.0.16.
16/09/01 12:02:09 INFO TaskSetManager: Finished task 2.0 in stage 0.0 (TID 2) in 133 ms on 10.255.0.16 (3/10)
16/09/01 12:02:09 INFO TaskSetManager: Starting task 5.0 in stage 0.0 (TID 5, 10.255.0.16, partition 5, PROCESS_LOCAL, 5474 bytes)
16/09/01 12:02:09 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Launching task 5 on executor id: 0 hostname: 10.255.0.16.
16/09/01 12:02:09 INFO TaskSetManager: Finished task 3.0 in stage 0.0 (TID 3) in 132 ms on 10.255.0.16 (4/10)
16/09/01 12:02:09 INFO TaskSetManager: Starting task 6.0 in stage 0.0 (TID 6, 10.255.0.16, partition 6, PROCESS_LOCAL, 5474 bytes)
16/09/01 12:02:09 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Launching task 6 on executor id: 0 hostname: 10.255.0.16.
16/09/01 12:02:09 INFO TaskSetManager: Finished task 4.0 in stage 0.0 (TID 4) in 14 ms on 10.255.0.16 (5/10)
16/09/01 12:02:09 INFO TaskSetManager: Starting task 7.0 in stage 0.0 (TID 7, 10.255.0.16, partition 7, PROCESS_LOCAL, 5474 bytes)
16/09/01 12:02:09 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Launching task 7 on executor id: 0 hostname: 10.255.0.16.
16/09/01 12:02:09 INFO TaskSetManager: Finished task 5.0 in stage 0.0 (TID 5) in 22 ms on 10.255.0.16 (6/10)
16/09/01 12:02:09 INFO TaskSetManager: Starting task 8.0 in stage 0.0 (TID 8, 10.255.0.16, partition 8, PROCESS_LOCAL, 5474 bytes)
16/09/01 12:02:09 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Launching task 8 on executor id: 0 hostname: 10.255.0.16.
16/09/01 12:02:09 INFO TaskSetManager: Finished task 6.0 in stage 0.0 (TID 6) in 17 ms on 10.255.0.16 (7/10)
16/09/01 12:02:09 INFO TaskSetManager: Starting task 9.0 in stage 0.0 (TID 9, 10.255.0.16, partition 9, PROCESS_LOCAL, 5474 bytes)
16/09/01 12:02:09 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Launching task 9 on executor id: 0 hostname: 10.255.0.16.
16/09/01 12:02:09 INFO TaskSetManager: Finished task 7.0 in stage 0.0 (TID 7) in 25 ms on 10.255.0.16 (8/10)
16/09/01 12:02:09 INFO TaskSetManager: Finished task 8.0 in stage 0.0 (TID 8) in 21 ms on 10.255.0.16 (9/10)
16/09/01 12:02:09 INFO TaskSetManager: Finished task 9.0 in stage 0.0 (TID 9) in 11 ms on 10.255.0.16 (10/10)
16/09/01 12:02:09 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
16/09/01 12:02:09 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 1.460 s
16/09/01 12:02:09 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 1.597367 s
Pi is roughly 3.1427591427591426
16/09/01 12:02:09 INFO SparkUI: Stopped Spark web UI at http://10.209.31.36:4040
16/09/01 12:02:09 INFO StandaloneSchedulerBackend: Shutting down all executors
16/09/01 12:02:09 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor to shut down
16/09/01 12:02:09 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
16/09/01 12:02:09 INFO MemoryStore: MemoryStore cleared
16/09/01 12:02:09 INFO BlockManager: BlockManager stopped
16/09/01 12:02:09 INFO BlockManagerMaster: BlockManagerMaster stopped
16/09/01 12:02:09 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
16/09/01 12:02:09 INFO SparkContext: Successfully stopped SparkContext
16/09/01 12:02:09 INFO ShutdownHookManager: Shutdown hook called
16/09/01 12:02:09 INFO ShutdownHookManager: Deleting directory /tmp/spark-6be222dc-f19d-40e7-a5c4-db902f72617b
root@c6c53516c308:/opt/spark/bin#
```

Figura 5.11 Detalles de la ejecución en Bash (II)

Capítulo 6. *Benchmarks* (Comparativas)

6.1 Spark 1.6 vs Spark 2.0

Para refutar una vez más la elección de esta nueva versión de *Spark*, cuyas novedades ya fueron explicadas con anterioridad en el apartado 1.2.3. y poder mostrar argumentos sólidos, hemos puesto a prueba nuestro clúster usando dos esquemas: el primero de ellos utilizando únicamente el nodo *Master* de *Spark* con 2 GB RAM y 2 *cores*; y por otro lado utilizamos 1 nodo *Spark Master*, junto a 17 nodos *Spark Workers*, todos ellos con 2GB RAM y 2 *cores* por unidad; y un pequeño *benchmark* propuesto por Databricks [17] que calcula la suma de mil millones (10^9) de números enteros.

Para hacer uso de este *benchmark* deberemos de lanzar nuestra `spark-shell`, sobre el clúster previamente creado:

```
> // Define a simple benchmark util function
  def benchmark(name: String)(f: => Unit) {
    val startTime = System.nanoTime
    f
    val endTime = System.nanoTime
    println(s"Time taken in $name: " + (endTime - startTime).toDouble / 1000000000 + " seconds")
  }

benchmark: (name: String)(f: => Unit)Unit
```

Figura 6.1 Definición de la función Benchmark.

```
> // This config turns off whole stage code generation, effectively changing the execution path to be similar to Spark 1.6.
  spark.conf.set("spark.sql.codegen.wholeStage", false)

  benchmark("Spark 1.6") {
    spark.range(1000L * 1000 * 1000).selectExpr("sum(id)").show()
  }

+------------------+
|           sum(id)|
+------------------+
|499999999500000000|
+------------------+

Time taken in Spark 1.6: 6.545222661 seconds
```

Figura 6.2 Ejecución en Spark 1.6.2.

```
> spark.conf.set("spark.sql.codegen.wholeStage", true)

  benchmark("Spark 2.0") {
    spark.range(1000L * 1000 * 1001).selectExpr("sum(id)").show()
  }

+------------------+
|           sum(id)|
+------------------+
|501000499499500000|
+------------------+

Time taken in Spark 2.0: 0.566650836 seconds
```

Figura 6.3 Ejecución en Spark 2.0.0.

En nuestra ejecución de esta pequeña prueba, hemos obtenido los siguientes resultados dando por claro vencedor a la versión 2.0.0 de *Spark*.

Esquema	**Tiempo (sec)**	
	Spark 1.6	**Spark 2.0**
Primer esquema (*Localhost*)	11,863 s	0,436 s
Segundo Esquema (1 *Master* + 17 *Workers*)	22,932 s	0,291 s

Tabla 6.1 Benchmark de las versiones de Spark

6.2 Comparativa entre tiempo de ejecución y número de nodos (contenedores)

Por otra parte, pusimos a prueba nuestro clúster usando el ejemplo aportado por Spark (*SparkPi*), desarrollado en lenguaje *Scala.* El cálculo de decimales del número pi (π), ha sido un problema a resolver por parte de toda la comunidad científica, por ello nos parecía una buena prueba para nuestro sistema.

El ejemplo, SparkPi, sigue un algoritmo donde se define un circulo de radio 1, dentro de un cuadrado ((0,0) a (1,1)), luego se lanzan n “dardos” sobre este círculo, luego se contabilizan los que han acertado en el círculo, con ello se aproximan los dígitos de pi (π) [16].

Las variables utilizadas en esta prueba, han sido el número de nodos o contenedores, que se incrementan hasta los 35 *Workers*, y fijando en cada uno de los contenedores la cantidad de memoria asignada 2 GB de RAM por nodo y proporcionándole 2 cores a cada uno de ellos. Todas estas pruebas se han realizado en un clúster con 18 nodos físicos, con 2Gb RAM e Intel i5 (4 *Cores*).

Nodos	**Cores por nodo**	**Total**	**RAM por nodo (GB)**	**Total (GB)**	**SparkPi (s)**
0	2	2	2	2	324,962
1	2	2	2	2	229,193
2	2	4	2	4	102,057
3	2	6	2	6	71,736
4	2	8	2	8	57,184
5	2	10	2	10	46,156
6	2	12	2	12	35,817
7	2	14	2	14	31,597
8	2	16	2	16	27,120
9	2	18	2	18	23,869
10	2	20	2	20	22,553
11	2	22	2	22	20,731
12	2	24	2	24	19,467
13	2	26	2	26	17,297
14	2	28	2	28	16,136
15	2	30	2	30	15,313
16	2	32	2	32	14,269
17	2	34	2	34	13,144
18	2	36	2	36	12,902
21	2	42	2	42	12,275
24	2	48	2	48	11,497
27	2	54	2	54	10,266
30	2	60	2	60	9,835
33	2	66	2	66	9,659
35	2	70	2	70	9,483

Tabla 6.2 Comparativa Clúster. Tabla de Tiempos

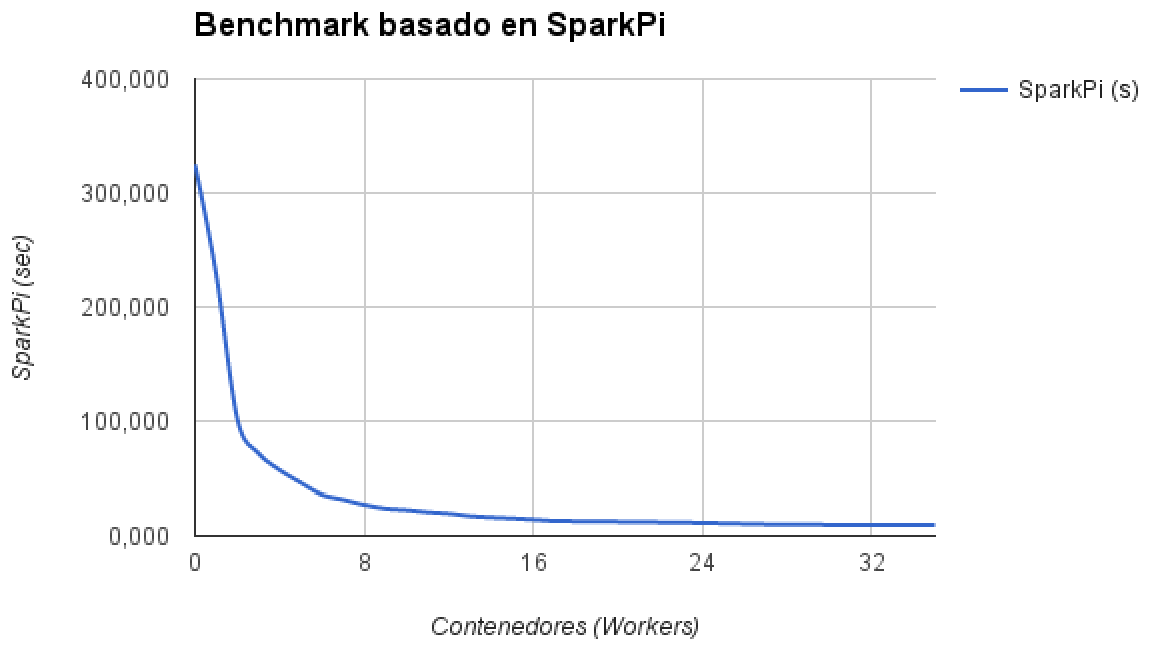


Figura 6.4 Comparativa Clúster. Gráfica

Capítulo 7. Conclusiones y líneas futuras

7.1 Conclusiones.

En conclusión, tras realizar un estudio de las tecnologías utilizadas, vemos que es un tema de actualidad y con un gran futuro dado que día a día se generan más datos que el día anterior. Por ello, cualquier sistema seguro y fiable de Big Data puede marcar diferencias en el mundo empresarial.

¿Cuál es el motivo para usar una herramienta como Docker? La respuesta es que, después de utilizarlo y ver su rendimiento, podríamos decir sin duda que es una alternativa consistente a la virtualización tradicional. Además, hemos observado como con las mejoras introducidas por Docker en su última versión, es bastante sencillo crear un clúster y la personalización de sistemas, por no comentar su velocidad de escalado y las infinitas posibilidades diferentes que nos provee.

Respecto a Spark, al comienzo del proyecto habían dudas de qué sistema de Big Data utilizar pero, tras lanzar la versión 2.0.0, ya se puede confirmar que Spark ha hecho bastante por ser el líder en cuanto al Big Data. Su velocidad de cómputo, que en esta versión se realiza prácticamente en tiempo real, así como los algoritmos internos utilizados para dividir los *jobs* o trabajos, supera con creces a otros sistemas.

7.2 Líneas Futuras.

A continuación se proponen algunas mejoras al proyecto que se pretenden realizar tras su presentación:

- Ampliar el *benchmark* para Spark teniendo como indicador el tiempo necesario para ejecutar un trabajo completo, fijando el número de iteraciones, y como variables el número de CPU y la cantidad de memoria RAM asignada a cada *worker*.

- Ampliar el alcance del proyecto para la creación de un clúster usando Docker sobre *Windows* o *Mac*.
- Crear un clúster más grande usando equipos de la ULL, también se propondrá para la TLP 2k17, usando equipos de los usuarios.
- Agregar *Jupyter* dockerizado a nuestro clúster.
- Optimizar el arranque del clúster desde el punto de vista del tiempo de arranque.
- Desplegar el proyecto en HPC Teide y estudiar resultados.

Capítulo 8. Summary and Conclusions

In conclusion, after conducting a study of the technologies used, we see that it is a topical issue and with a great future because every day more data than the previous day are generated. Therefore, any safe and reliable system of Big Data can make a difference in the business world.

What is the reason for using a tool like Docker? After using it and seeing its performance, we can say without doubt that is a consistent alternative to traditional virtualization. In addition, we have observed the improvements made by Docker in its latest version, and it is quite easy to create a cluster and customize the system, not to mention the speed of scaling and the infinite different possibilities that it gives us.

Concerning Spark, at the beginning of the project we had doubts about what kind of system we should use for Big Data, but after seeing the latest version of Spark, we chose it for many reasons: its computing speed, which in this version is done in near real time, as well as the internal algorithms used to split-join jobs, far surpasses other systems.

Capítulo 9. Presupuesto

Este proyecto está basado en herramientas de Software libre, por lo que no se contempla ningún gasto para software; tampoco se ha necesitado adquirir ningún equipo informático dado que han sido cedidos por el Centro de Cálculo, de la Escuela Superior de Ingeniería y Tecnología de la Universidad de La Laguna. Por tanto, solo se contempla el coste de los recursos humanos.

9.1 Presupuesto Detallado

Tipo	Concepto	Observaciones	Unidades	Horas	Coste/h	Coste/u	Total
Software	Apache Spark 2.0.0	Licencia Apache License 2.0	-	-	0	- €	- €
Software	Docker 1.12	Licencia Apache License 2.0	-	-	0	- €	- €
Hardware	Equipos	Cedidos	6	-	0	- €	- €
Recursos Humanos	Desarrollador Software	-	1	300	25,00 €	7.500,00 €	7.500,00 €
						Total	7.500,00 €

Tabla 9.1. Tabla Presupuesto detallado

Apéndice A. Bash

En este apéndice se mostrarán los scripts creados para Ubuntu, encargados de gestionar Docker. Cabe destacar que estos scripts podrían necesitar modificaciones para que sean funcionales en otro sistema diferente.

A.1. encender.sh

```
/*********************************************************************************
* NOMBRE encender.sh
* VERSION 1.0
*********************************************************************************
* AUTOR Sergio Martín Santana (Sergio.ms.91@gmail.com)
* FECHA agosto 2016
* DESCRIPCION
* Este Script automatiza el proceso de arranque del cluster.
*********************************************************************************/
# !/bin/bash
echo -e "\n" && echo "Wake Up Guys" && echo -e "\n"
bash "./wake"
echo -e "\n"

for x in `seq 70 -1 0`
    do
          echo "Waiting for the Hosts (Missing $x seconds)"
          sleep 1
    done
echo "Leaving the previously assigned cluster"
bash "./salir"
echo -e "\n"
bash "./v2link2swarm"
echo -e "\n"
bash "./ini-cluster"
```

A.2. salir.sh

```
/***********************************************************************************
* NOMBRE salir.sh
* VERSION 1.0
************************************************************************************
* AUTOR Sergio Martín Santana (Sergio.ms.91@gmail.com)
* FECHA agosto 2016
* DESCRIPCION
* Para evitar que los nodos se encuentren en otro clúster, podremos salir de
* cualquier clúster en el que estén los equipos del clúster, usando este script.
***********************************************************************************/
#!/bin/bash
p=1 # planta
s=2 # sala
for f in `seq 1 +1 3` #fila
    do
          for o in `seq 1 +1 6` #puesto
                do
                      echo -e "cc$p$s$f$o: "
                      ssh cc$p$s$f$o docker swarm leave --force
                done
    done
```

A.3. link2swarm.sh

```
/***********************************************************************************
* NOMBRE link2swarm.sh
* VERSION 1.0
************************************************************************************
* AUTOR Sergio Martín Santana (Sergio.ms.91@gmail.com)
* FECHA agosto 2016
* DESCRIPCION
* Este script crea el nodo manager del clúster y enlaza el resto de nodos con él,
* usando docker swarm. Al final de la ejecución, muestra el estado del clúster.
***********************************************************************************/

# !/bin/bash

Host=`ssh cc1212 hostname`
```

```
echo -e "Creating a master node in $Host"
echo "Adding the Master in in $Host"
ssh cc1212 docker swarm init --advertise-addr 10.209.31.36 --listen-addr 10.209.31.36:2377 #--$

p=1 # planta
s=2 # sala

for f in `seq 1 +1 1` #fila
    do
        for o in `seq 1 +1 6` #puesto
            do
                Host=`ssh cc$p$s$f$o hostname`
                Token=`docker swarm join-token -q worker`
                echo -e "Creating a worker node in $Host"
                Node=`echo -e "cc$p$s$f$o"`
                #if ["$Host" != "cc1212"]; then
                    ssh $Node docker swarm join 10.209.31.36:2377 --token $Token #--secret "DockerSparkTFG1516"
                    sleep 2
                #fi
            done
    done

echo "------------"
echo " Well Done!"
echo "------------"
docker node ls
```

A.4. ini-cluster.sh

```
/*******************************************************************************
* NOMBRE ini-cluster.sh
* VERSION 1.0
********************************************************************************
* AUTOR Sergio Martín Santana (Sergio.ms.91@gmail.com)
* FECHA agosto 2016
* DESCRIPCION
* Este script tiene la función de crear la red overlay del clúster, y arrancar el
* visualizador. Previamente los elimina, para asegurarnos su correcto funcionamiento.
*******************************************************************************/
```

```
# !/bin/bash


docker rm -f visualizer

docker run --name visualizer -it -d -p 5000:5000 -e HOST=10.209.31.36 -e PORT=5000 -v /var/run/docker.sock:/var/run/docker.sock turaaa/swarmvisualizer:latest

docker network rm mynet

docker network create -d overlay --subnet=192.168.100.0/24 --ip-range=192.168.100.0/24  mynet
```

A.6. restart.sh

```
/*********************************************************************************
* NOMBRE restart.sh
* VERSION 1.0
**********************************************************************************
* AUTOR Sergio Martín Santana (Sergio.ms.91@gmail.com)
* FECHA agosto 2016
* DESCRIPCION
* Reiniciamos el servicio docker en todo el clúster, recomendable salir previamente de
* él y después crearlo de nuevo.
*********************************************************************************/
# !/bin/bash


p=1 # planta
s=2 # sala


for f in `seq 1 +1 3` #fila
    do
          for o in `seq 1 +1 6` #puesto
                do
                      Node=`echo -e "cc$p$s$f$o"`
                      echo -e "cc$p$s$f$o"
                      ssh $Node restart docker


                done
    done
```

A.8. launchSpark.sh

```
/*********************************************************************************
* NOMBRE launcSpark.sh
* VERSION 1.0
*********************************************************************************
* AUTOR Sergio Martín Santana (Sergio.ms.91@gmail.com)
* FECHA agosto 2016
* DESCRIPCION
* Este script está encargado de lanzar el servicio Spark (Tasks as a Service).
*********************************************************************************/
#lanzar service master (también funciona como driver)
docker service create --network mynet --name master -p 8080:8080 -p 7077:7077
-p 4040:4040 sersantin/ssm:latest

#lanzar service worker
docker service create --network mynet --name worker -p 8081:8081 --replicas 5
sersantin/ssw:latest
```

A.9. verservice.sh

```
/*********************************************************************************
* NOMBRE verservice.sh
* VERSION 1.0
*********************************************************************************
* AUTOR Sergio Martín Santana (Sergio.ms.91@gmail.com)
* FECHA agosto 2016
* DESCRIPCION
* Visualiza el estado de los servicios de Spark. Para salir Ctrl+c
*********************************************************************************/
# !/bin/bash
while true
do
    clear
    docker service ls
    echo ""
    docker service ps master
```

```
    echo ""
    docker service ps worker
    sleep 1
done
```

A.10. rmservice.sh

```
/*********************************************************************************
* NOMBRE launcSpark.sh
* VERSION 1.0
*********************************************************************************
* AUTOR Sergio Martín Santana (Sergio.ms.91@gmail.com)
* FECHA agosto 2016
* DESCRIPCION
* Elimina los servicios de Spark.
*********************************************************************************/
# !/bin/bash
docker service rm master
docker service rm worker
```

A.11. runSpark.sh

```
/*********************************************************************************
* NOMBRE launcSpark.sh
* VERSION 1.0
*********************************************************************************
* AUTOR Sergio Martín Santana (Sergio.ms.91@gmail.com)
* FECHA agosto 2016
* DESCRIPCION
* Arranca el servicio, como contenedores (Containers as a Service).
*********************************************************************************/
# !/bin/bash

ssh cc1212 docker run -d -ti --net host --name master12 --hostname master12 -p
8080:8080
-p 4040:4040 -p 7077:7077 sersantin/tm:latest
sleep 2
```

```
ssh cc1211 docker run -d -ti --net host --name worker11 --hostname worker11 -p 8081:8081
sersantin/tw:latest
sleep 2
ssh cc1213 docker run -d -ti --net host --name worker13 --hostname worker13 -p 8081:8081
sersantin/tw:latest
sleep 2
ssh cc1214 docker run -d -ti --net host --name worker14 --hostname worker14 -p 8081:8081
sersantin/tw:latest
sleep 2
ssh cc1215 docker run -d -ti --net host --name worker15 --hostname worker15 -p 8081:8081
sersantin/tw:latest
sleep 2
ssh cc1216 docker run -d -ti --net host --name worker16 --hostname worker16 -p 8081:8081
sersantin/tw:latest
```

A.12. wake.sh

```
/*********************************************************************************
* NOMBRE wake.sh
* VERSION 1.0
**********************************************************************************
* AUTOR Sergio Martín Santana (Sergio.ms.91@gmail.com)
* FECHA agosto 2016
* DESCRIPCION
* Este Script enciende medianete WoL (WakeOnLan) los equipos del cluster.
*********************************************************************************/
# !/bin/bash
wakeonlan  b0:83:fe:90:5d:cc #cc1211
wakeonlan  b0:83:fe:90:5e:d2 #cc1212
wakeonlan  b0:83:fe:90:5f:fc #cc1213
wakeonlan  b0:83:fe:90:5c:13 #cc1214
wakeonlan  b0:83:fe:90:5a:3e #cc1215
wakeonlan  b0:83:fe:90:5d:c9 #cc1216
#wakeonlan <MAC>
```

Apéndice B. Docker

En este apartado se aportaran los Dockerfiles y Scripts necesarios para generar cada imagen Spark de Docker .

B.1 Servicios

B.1.1 Master

Dockerfile

```
FROM ubuntu:latest
MAINTAINER Sergio Martín Santana “sergio.ms.91@gmail.com”

RUN echo “Updating System” && apt-get update -y

RUN echo "Instaling Base Applications" && apt-get install -y wget && apt-get install -y iputils-ping && apt-get install -y default-jre && apt-get install -y ssh && apt-get install -y net-tools && apt-get install -y iproute2

RUN echo "Setting Up Environment Variables"
ENV SPARK_VER 2.0.0
ENV HADOOP_VER 2.7
ENV SPARK_DIST spark-$SPARK_VER-bin-hadoop$HADOOP_VER
ENV SPARK_URL "http://www.eu.apache.org/dist/spark/spark-$SPARK_VER/$SPARK_DIST.tgz"
ENV INSTALL_DIR /opt
ENV HOME /root
ENV BASHRC $HOME/.bashrc

RUN echo "Descarga/Instalación Spark" && wget -q -P /tmp -c $SPARK_URL && apt-get install tar && tar xzvf /tmp/$SPARK_DIST.tgz -C $INSTALL_DIR && mv /opt/spark-$SPARK_VER-bin-hadoop$HADOOP_VER/ /opt/spark && rm /tmp/$SPARK_DIST.tgz
```

```
RUN echo "Creating user for ssh service" && mkdir -p /localhome && useradd -d /localhome/udocker -m -s /bin/bash  udocker && echo "root:1234" | chpasswd && echo "udocker:1234" | chpasswd && mkdir -p /localhome/udocker/.ssh && chown -R udocker $INSTALL_DIR && chown -R udocker /localhome/udocker/ && echo "Creating SSH-KEY RSA" && su - udocker -c "cat /dev/zero | ssh-keygen -q -N ''" && su - udocker -c "ssh-keyscan -H localhost >> ~/.ssh/known_hosts" && su - udocker -c "cat /localhome/udocker/.ssh/id_rsa.pub >> /localhome/udocker/.ssh/authorized_keys"


RUN echo "Copying booting script"


COPY bootstrap.sh /etc/bootstrap.sh

RUN chown root.root /etc/bootstrap.sh && chmod 755 /etc/bootstrap.sh


COPY spark-env.sh /opt/spark/conf

RUN chown root.root /opt/spark/conf/spark-env.sh && chmod 755 /opt/spark/conf/spark-env.sh


EXPOSE 8080

EXPOSE 7077

EXPOSE 4040


ENTRYPOINT ["/etc/bootstrap.sh"]
```

Scripts

Spark.env

```
#!/usr/bin/env bash


export SPARK_MASTER_HOST=192.168.100.3

# Bind the master to a specific hostname or IP address, for example a public one.

export SPARK_PUBLIC_DNS=10.209.31.36

# The public DNS name of the Spark master and workers (default: none).(HOST IP)

export SPARK_WORKER_CORES=2

# Total number of cores to allow Spark applications to use on the machine (default: all available cores).

export SPARK_WORKER_MEMORY=2G

# Total amount of memory to allow Spark applications to use on the machine, e.g. 1000m, 2g (default: total memory minus 1 GB)

export SPARK_EXECUTOR_CORES=2

# The number of cores to use on each executor
```

```
export SPARK_EXECUTOR_MEMORY=2G
# Amount of memory to use per executor process
export SPARK_WORKER_DIR=/tmp/spark-worker
# Directory to run applications in, which will include both logs and scratch space
export SPARK_LOCAL_DIR=/tmp/spark-worker
# Directory to use for "scratch" space in Spark, including map output files and RDDs that get stored on disk.
export SPARK_LOG_DIR=/tmp/spark-logs
# Directory to use for logs.
```

Bootstrap.sh

```
#!/usr/bin/env bash
/usr/sbin/service ssh start           #Start SSH Service
/opt/spark/sbin/start-master.sh       #Start Spark Master
tail -f /dev/null                    #Sleep forever
```

B.1.2 Worker

Dockerfile

```
FROM ubuntu:latest
MAINTAINER Sergio Martín Santana “sergio.ms.91@gmail.com”


RUN echo “Updating System” && apt-get update -y


RUN echo "Instaling Base Applications" && apt-get install -y wget && apt-get install -y iputils-ping && apt-get install -y default-jre && apt-get install -y ssh && apt-get install -y net-tools && apt-get install -y iproute2


RUN echo "Setting Up Environment Variables"
ENV SPARK_VER 2.0.0
ENV HADOOP_VER 2.7
ENV SPARK_DIST spark-$SPARK_VER-bin-hadoop$HADOOP_VER
ENV SPARK_URL "http://www.eu.apache.org/dist/spark/spark-$SPARK_VER/$SPARK_DIST.tgz"
ENV INSTALL_DIR /opt
ENV HOME /root
ENV BASHRC $HOME/.bashrc
```

```
RUN echo "Descarga/Instalación Spark" && wget -q -P /tmp -c $SPARK_URL && apt-get install tar && tar xzvf /tmp/$SPARK_DIST.tgz -C $INSTALL_DIR && mv /opt/spark-$SPARK_VER-bin-hadoop$HADOOP_VER/ /opt/spark && rm /tmp/$SPARK_DIST.tgz

RUN echo "Creating user for ssh service" && mkdir -p /localhome && useradd -d /localhome/udocker -m -s /bin/bash  udocker && echo "root:1234" | chpasswd && echo "udocker:1234" | chpasswd && mkdir -p /localhome/udocker/.ssh && chown -R udocker $INSTALL_DIR && chown -R udocker /localhome/udocker/ && echo "Creating SSH-KEY RSA" && su - udocker -c "cat /dev/zero | ssh-keygen -q -N ''" && su - udocker -c "ssh-keyscan -H localhost >> ~/.ssh/known_hosts" && su - udocker -c "cat /localhome/udocker/.ssh/id_rsa.pub >> /localhome/udocker/.ssh/authorized_keys"


RUN echo "Copying booting script"


COPY bootstrap.sh /etc/bootstrap.sh

RUN chown root.root /etc/bootstrap.sh && chmod 755 /etc/bootstrap.sh


COPY spark-env.sh /opt/spark/conf

RUN chown root.root /opt/spark/conf/spark-env.sh && chmod 755 /opt/spark/conf/spark-env.sh


EXPOSE 8080

EXPOSE 7077

EXPOSE 4040


ENTRYPOINT ["/etc/bootstrap.sh"]
```

Scripts

Spark-env.sh

```
#!/usr/bin/env bash


export SPARK_WORKER_CORES=2

# Total number of cores to allow Spark applications to use on the machine (default: all available cores).

export SPARK_WORKER_MEMORY=2G

# Total amount of memory to allow Spark applications to use on the machine, e.g. 1000m, 2g (default: total memory minus 1 GB)

export SPARK_EXECUTOR_CORES=2

# The number of cores to use on each executor

export SPARK_EXECUTOR_MEMORY=2G            # Amount of memory to use per executor process

export SPARK_WORKER_DIR=/tmp/spark-worker    # Directory to run applications in, which will include both logs and scratch space
```

```
export SPARK_LOCAL_DIR=/tmp/spark-worker     # Directory to use for "scratch" space
in Spark, including map output files and RDDs that get stored on
                                             # disk.
export SPARK_LOG_DIR=/tmp/spark-logs         # Directory to usefor logs.
```

Bootstrap.sh

```
#!/usr/bin/env bash
# Start SSH Service
/usr/sbin/service ssh start
# Start a Worker node and connect it to the Master Node
/bin/bash -c "/opt/spark/sbin/start-slave.sh spark://192.168.100.3:7077"
# Sleep forever
tail -f /dev/null
```

B.2. Contenedores

B.2.1 Master

Dockerfile

```
FROM ubuntu:latest
MAINTAINER Sergio Martín Santana “sergio.ms.91@gmail.com”

RUN echo “Updating System” && apt-get update -y

RUN echo "Setting Up Environment Variables"
ENV SPARK_VER 1.6.2
ENV HADOOP_VER 2.6
ENV SPARK_DIST spark-$SPARK_VER-bin-hadoop$HADOOP_VER
ENV SPARK_URL "http://www.eu.apache.org/dist/spark/spark-$SPARK_VER/$SPARK_DIST.tgz"
ENV INSTALL_DIR /opt
ENV HOME /root
ENV BASHRC $HOME/.bashrc

RUN echo "Instaling Base Applications" && apt-get install -y \
      wget \
      iputils-ping \
      default-jre \
      ssh \
```

```
      net-tools \
      iproute2 \
      nano \
      tar \
&& rm -rf /var/lib/apt/lists/*


RUN echo "Descarga/Instalación Spark" && wget -q -P /tmp -c $SPARK_URL && tar xzvf /tmp/$SPARK_DIST.tgz -C $INSTALL_DIR && mv /opt/spark-$SPARK_VER-bin-hadoop$HADOOP_VER/ /opt/spark && rm /tmp/$SPARK_DIST.tgz


RUN echo "Creating user for ssh service" && mkdir -p /localhome && useradd -d /localhome/udocker -m -s /bin/bash  udocker && echo "root:1234" | chpasswd && echo "udocker:1234" | chpasswd && mkdir -p /localhome/udocker/.ssh && chown -R udocker $INSTALL_DIR && chown -R udocker /localhome/udocker/ && echo "Creating SSH-KEY RSA" && su - udocker -c "cat /dev/zero | ssh-keygen -q -N ''" && su - udocker -c "ssh-keyscan -H localhost >> ~/.ssh/known_hosts" && su - udocker -c "cat /localhome/udocker/.ssh/id_rsa.pub >> /localhome/udocker/.ssh/authorized_keys"
RUN echo "Copying booting script"
COPY spark-env.sh /opt/spark/conf
RUN chown root.root /opt/spark/conf/spark-env.sh && chmod 700 /opt/spark/conf/spark-env.sh


COPY bootstrap.sh /opt/spark
RUN chown root.root /opt/spark/bootstrap.sh && chmod 700 /opt/spark/bootstrap.sh


EXPOSE 8080
EXPOSE 7077
EXPOSE 4040


ENTRYPOINT ["/opt/spark/bootstrap.sh"]
```

Scripts

Bootstrap.sh

```
#!/usr/bin/env bash


/usr/sbin/service ssh start
/opt/spark/sbin/start-master.sh
`/bin/bash --login`
```

Spark-env.sh

```
#!/usr/bin/env bash

export SPARK_LOCAL_IP=`ip a ls dev eth0 | sed -n "s,.*inet  *\([^/]*\)/.*,\1,p"`
#export SPARK_MASTER_IP='master'
export SPARK_PUBLIC_DNS=`ip a ls dev eth0 | sed -n "s,.*inet  *\([^/]*\)/.*,\1,p"`
export SPARK_WORKER_INSTANCES=1
export SPARK_WORKER_CORES=2
export SPARK_WORKER_MEMORY=2G
export SPARK_DRIVER_MEMORY=1536M
export SPARK_WORKER_DIR=/tmp/spark-worker
export SPARK_LOCAL_DIRS=/tmp/spark-worker
export SPARK_LOG_DIR=/tmp/spark-logs
```

B.2.2 Worker

Dockerfile

```
FROM ubuntu:latest
MAINTAINER Sergio Martín Santana “sergio.ms.91@gmail.com”

RUN echo “Updating System” && apt-get update -y

RUN echo "Setting Up Environment Variables"
ENV SPARK_VER 1.6.2
ENV HADOOP_VER 2.6
ENV SPARK_DIST spark-$SPARK_VER-bin-hadoop$HADOOP_VER
ENV SPARK_URL "http://www.eu.apache.org/dist/spark/spark-$SPARK_VER/$SPARK_DIST.tgz"
ENV INSTALL_DIR /opt
ENV HOME /root
ENV BASHRC $HOME/.bashrc

RUN echo "Instaling Base Applications" && apt-get install -y \
	wget \
	iputils-ping \
	default-jre \
	ssh \
	net-tools \
	iproute2 \
```

```
      nano \
      tar \
&& rm -rf /var/lib/apt/lists/*


RUN echo "Descarga/Instalación Spark" && wget -q -P /tmp -c $SPARK_URL && tar xzvf /tmp/$SPARK_DIST.tgz -C $INSTALL_DIR && mv /opt/spark-$SPARK_VER-bin-hadoop$HADOOP_VER/ /opt/spark && rm /tmp/$SPARK_DIST.tgz


RUN echo "Creating user for ssh service" && mkdir -p /localhome && useradd -d /localhome/udocker -m -s /bin/bash  udocker && echo "root:1234" | chpasswd && echo "udocker:1234" | chpasswd && mkdir -p /localhome/udocker/.ssh && chown -R udocker $INSTALL_DIR && chown -R udocker /localhome/udocker/ && echo "Creating SSH-KEY RSA" && su - udocker -c "cat /dev/zero | ssh-keygen -q -N ''" && su - udocker -c "ssh-keyscan -H localhost >> ~/.ssh/known_hosts" && su - udocker -c "cat /localhome/udocker/.ssh/id_rsa.pub >> /localhome/udocker/.ssh/authorized_keys"



RUN echo "Copying booting script"


COPY spark-env.sh /opt/spark/conf
RUN chown root.root /opt/spark/conf/spark-env.sh && chmod 700 /opt/spark/conf/spark-env.sh


COPY bootstrap.sh /opt/spark
RUN chown root.root /opt/spark/bootstrap.sh && chmod 700 /opt/spark/bootstrap.sh


EXPOSE 8080
EXPOSE 7077
EXPOSE 4040


ENTRYPOINT ["/opt/spark/bootstrap.sh"]
```

Scripts

Bootstrap.sh

```
#!/usr/bin/env bash


/usr/sbin/service ssh start
/opt/spark/sbin/start-slave.sh spark://10.209.31.36:7077
`/bin/bash --login`
```

Spark-env.sh

```
#!/usr/bin/env bash
```

```
export SPARK_LOCAL_IP=`ip a ls dev eth0 | sed -n "s,.*inet  *\([^/]*\)/.*,\1,p"`
#export SPARK_MASTER_IP='master'
export SPARK_PUBLIC_DNS=`ip a ls dev eth0 | sed -n "s,.*inet  *\([^/]*\)/.*,\1,p"`
export SPARK_WORKER_INSTANCES=1
export SPARK_WORKER_CORES=2
export SPARK_WORKER_MEMORY=2G
export SPARK_DRIVER_MEMORY=1536M
export SPARK_WORKER_DIR=/tmp/spark-worker
export SPARK_LOCAL_DIRS=/tmp/spark-worker
export SPARK_LOG_DIR=/tmp/spark-logs
```

Apéndice C.
Spark

Tutorial, disponible en: spark.apache.org/docs/latest/quick-start.html

Bibliografía

[1] D. Laney. 3D data management: Controlling data volume, variety and velocity. 2001. Disponible en: blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf

[2] E. Dans. Big Data: una pequeña introducción. 2011. Disponible en: www.enriquedans.com/2011/10/big-data-una-pequena-introduccion.html

[3] M. Rouse. 3Vs (volume, variety and velocity). 2013. Disponible en: whatis.techtarget.com/definition/3Vs

[4] M. Pérez Apache Spark: qué es y cómo funciona. 2015. Disponible en: geekytheory.com/apache-spark-que-es-y-como-funciona/

[5] ASPgems. 7 razones por las que deberías usar Apache Spark. 2015 Disponible en: aspgems.com/blog/apache-spark/7-razones-por-las-que-deberias-usar-apache-spark

[6] R. Xin. Technical Preview of Apache Spark 2.0 Now on Databricks. 2016. Disponible en: databricks.com/blog/2016/05/11/apache-spark-2-0-technical-preview-easier-faster-and-smarter.html

[7] J. Soto. Docker – Conociendo un poco Más del comando Docker run. 2015 Disponible en: www.jsitech.com/linux/docker-conociendo-un-poco-mas-del-comando-docker-run/

[8] G. Vivancos. Docker y la era de los contenedores. 2015 Disponible en: blog.celingest.com/2015/02/17/docker-y-la-era-de-los-contenedores/

[9] Esaú A. Docker, Qué es y sus principales características. 2014. Disponible en: openwebinars.net/docker-que-es-sus-principales-caracteristicas/

[10] docs.docker.com

[11] M. Colebrook. DSbox. 2015. Disponible en:github.com/mcolebrook/DSbox

[12] Stratio. Spark. 2016. Disponible en: hub.docker.com/r/stratio/spark/

[13] J. Phillips. Official Consul Docker Image. 2016. Disponible en: www.hashicorp.com/blog/official-consul-docker-image.html

[14] docs.docker.com/swarm/install-manual/

[15] D. Laney. 2013. Batman on Big Data. Disponible en:
blogs.gartner.com/doug-laney/batman-on-big-data

[16] spark.apache.org/examples.html

[17] docs.cloud.databricks.com/docs/latest/sample_applications/04%20Apache
%20Spark%202.0%20Examples/03%20Performance%20Apache%20(Spark
%202.0%20vs%201.6).html

[18] DovAmir/docker-swarm-visualizer. Disponible en:
github.com/DovAmir/docker-swarm-visualizer