Universitat Oberta de Catalunya

uoc.edu

Arquitectura Big Data de ingesta en Real Time

Ferran Fernández Garrido
Master en inteligencia de negocio y Big Data
Itinerario Sistemas de información

David Cabanillas Barbacil
Josep Curto Díaz

07/07/2017

Universitat Oberta de Catalunya uoc.edu

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Arquitectura Big Data de ingesta en Real Time*
Nombre del autor:	*Ferran Fernández Garrido*
Nombre del consultor/a:	*David Cabanillas Barbacil*
Nombre del PRA:	*Josep Curto Díaz*
Fecha de entrega (mm/aaaa):	07/2017
Titulación::	*Master en inteligencia de negocio y Big Data*
Área del Trabajo Final:	*Sistemas de información*
Idioma del trabajo:	*Castellano*
Palabras clave	*Arquitectura, ingesta, real time*

Resumen del Trabajo (máximo 250 palabras):

El objetivo de este proyecto, es desarrollar y desplegar un sistema de arquitectura Big Data de ingesta en real time.

Tomaremos como referencia las tecnologías usadas por las empresas más punteras en la actualidad en el uso de los datos: Netflix, Facebook, Amazon,…

Por ejemplo Netflix hace uso de Kafka para su sistema de recomendación de series y películas en real time.

Para ello, haremos uso de las últimas tecnologías Big Data y nos plantearemos un problema cuotidiano a resolver, que podrá ser extrapolable a muchos otros ámbitos de ingesta de datos.

Los resultados que se esperan obtener con este proyecto, son por una parte la creación de una arquitectura virtualizada totalmente funcional mediante un prototipo a pequeña escala. La cual será probada mediante el uso de diversos juegos de pruebas. Así como de todo el aprendizaje de las tecnologías implicadas.

Abstract (in English, 250 words or less):

The aim of this project, is to develop and deploy a new Big Data architecture system to ingest data in real time.

We will take as a reference the most common technologies spread across the main technological companies nowadays such as: Netflix, Facebook, Amazon,…

As an example, Netflix uses Kafka as a main technology for TV shows and movies recommendations.

In order to develop our target, we will use the most trendy technologies that are commonly used on Big Data. We will use it to solve an easy problem that could be extrapolated to many other use cases in data ingestion environment.

The expected results for this project are, on the one hand the creation of a new virtual ingestion architecture which is going to be completely functional at scale. Although the architecture will be tested using different test set. In addition, another result of the project will be all the know-how Big data technology that is going to be acquired during the development of the project.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

La idea del proyecto, es suponer que una empresa de alquiler de coches nos ha solicitado la creación de una infraestructura de monitorización de sus vehículos en tiempo real.

Para un objetivo de estas características, es necesario el uso de tecnologías Big Data. Esto es debido a que implica lo que en el mundo del Big Data se conoce como las 3 V (Velocidad, variedad y volumen).Debemos ser capaces de procesar y mostrar al usuario una gran cantidad de datos en tiempo real y ser capaces de procesarlos de forma distribuida, de forma en que para el usuario resulte transparente. Debemos tener en mente que el envío de trazas por parte de los emisores (los coches) será constante y en gran volumen debido a la cantidad de coches. Es por ello que un sistema de ingesta tradicional no cumpliría con los requisitos.

Los objetivos a cubrir por tanto pasarían por la creación de la arquitectura de ingesta, así como del tratamiento intermedio del dato. Quedarían por tanto exentos los puntos más propios de la explotación final del dato, así como los aspectos económicos del despliegue.
Deberemos distinguir también que nuestra arquitectura estará orientada al procesado en real time y no al batch. Este último sistema supone la carga de datos dentro de un único flujo que no se ejecuta en tiempo real, de forma habitual suele sufrir decalajes de 1 día.

Por tanto, podríamos decir que es un proyecto propio de arquitectura. La aportación principal al proyecto, es el gran mix de tecnologías Big Data implicadas en el proceso de creación y que se detallarán en puntos siguientes.

Siguiendo el ejemplo de otras grandes empresas orientadas al dato como Netflix, Linkedin, Facebook, etc. Se intentará replicar un modelo de arquitectura similar, que consiga cubrir las necesidades básicas que nuestro cliente se plantea.

Por tanto el objetivo final, seria disponer de un modelo de arquitectura de ingesta de datos masivos, que permitiera ser extrapolable a muchos otros ámbitos no solo al que ocupa este proyecto y que no tiene más fin más allá que ejemplificar.

A continuación se muestra a nivel de aplicativo, un esquema de nuestra solución:

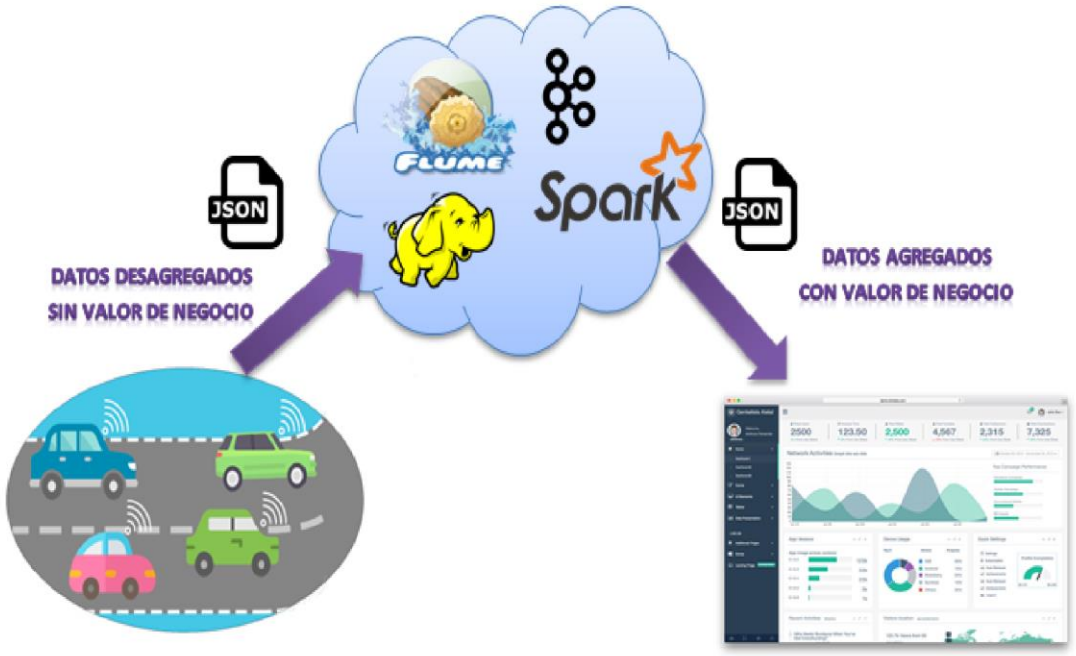


Ilustración 1: Representación de la solución a alto nivel

1.2 Objetivos del Trabajo

Los objetivos del trabajo quedarían separados en tres bloques fundamentales, los cuales los dos últimos son de especial relevancia, pero no podrían funcionar sin el primero.

En primer lugar se debe poder desarrollar un sistema web service en una Raspberry. Basado en tecnología PHP. Dicho sistema deberá simular el envío de miles de trazas (correctas, incompletas, erróneas…). Que contendrán información de valor sobre el coche sobre el que estén reportando, nivel de gasolina, batería, frenos, velocidad, …

La estructura de traza será presentada en siguientes capítulos y será mostrada con más detalle, sin embargo se debe tener en cuenta que esta información no será de carácter estructurado, por lo que el sistema debe ser capaz de interpretar la falta de datos considerados fundamentales.

Seguidamente, deberemos desplegar sobre una máquina virtual y Docker un sistema de arquitectura Big Data detallado a alto nivel más adelante. Este supondrá un reto mayor que el anterior, puesto que no solo nos quedaremos en la superficie de la implantación. Dentro de algunas tecnologías concretas como Flume o Spark, crearemos componentes específicos que ayudarán en nuestro objetivo de ingesta.

Finalmente, se deberán hacer pruebas de rendimiento sobre el sistema completo, de esta forma podremos saber cómo se comporta en unas condiciones u otras.
Deberíamos ser capaces por ejemplo de plantear la casuística de que pasaría si un nodo de datos quedara inservible entre otras.
El usuario final bajo el ejemplo planteado debería ser capaz de ver una interfaz web similar a la siguiente:

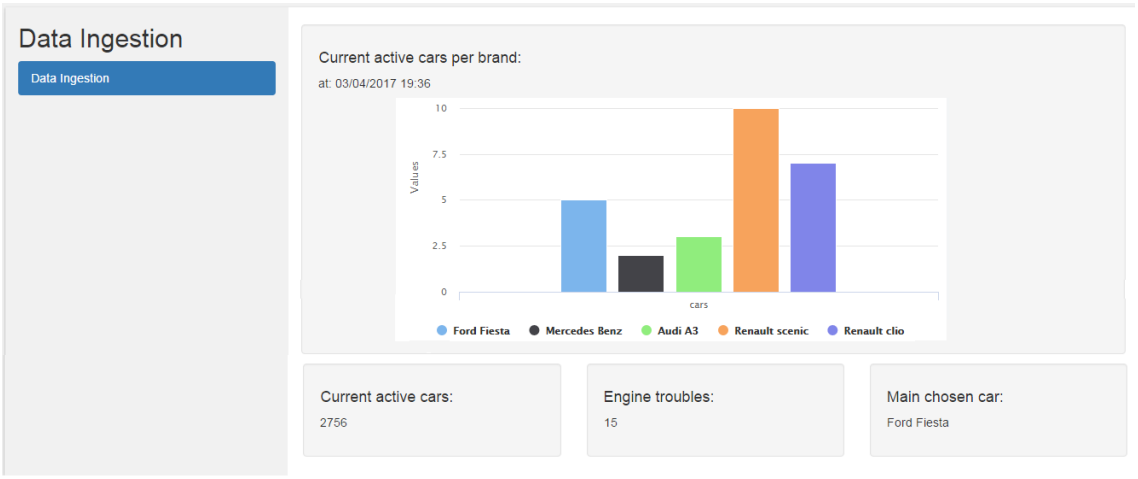


Ilustración 2: Ejemplo de Dashboard resultante.

Donde se detallarían algunos de los indicadores en real time.

1.3 Enfoque y método seguido

Para este proyecto, evitaremos el uso de distribuciones de software ya creadas para fines similares, como por ejemplo: Cloudera, Hortonworks, MapR, etc.
Desplegaremos todos los elementos del proyecto de forma individual y a la carta, eso supondrá un reto mayor puesto que deberemos lidiar con las dificultades que eso supone.
Lógicamente esta estrategia es la mejor a seguir puesto que podrá dar una mayor profundización en la arquitectura propiamente que usando métodos “pre cocinados”.
Por tanto estaríamos realizando un producto a la carta donde no únicamente desplegaremos arquitectura, sino que aportaremos modificaciones sobre dicha arquitectura para que cumpla el fin que nos hemos propuesto en el apartado 1.1.
Para tener una mejor idea del sistema que se va a montar, a continuación se muestra un esquema de muy alto nivel, donde se resaltan las tecnologías usadas y como estarían dispuestas en el sistema final:

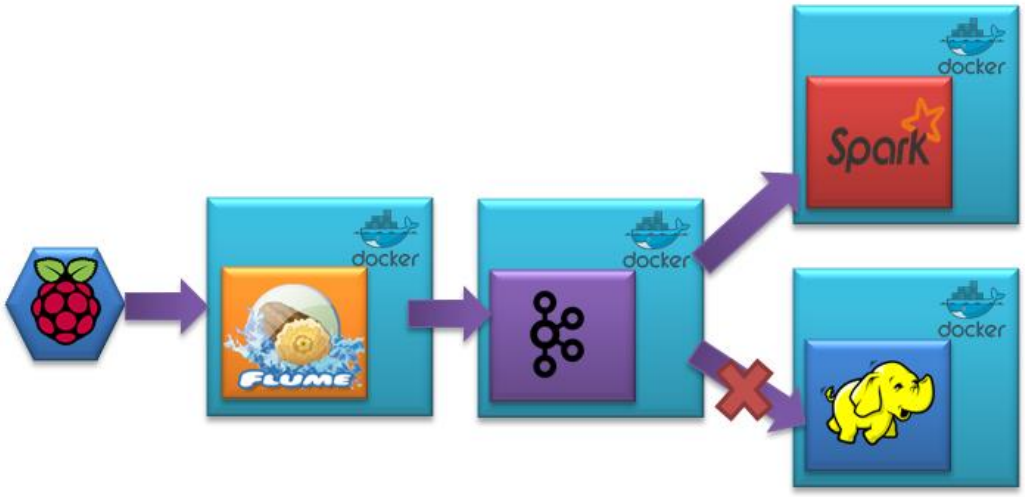


Ilustración 3: Representación de la arquitectura a alto nivel.

Como ya se ha comentado, en el desarrollo del proyecto se utilizarán tecnologías Big Data que permitirán cumplir nuestro
objetivo de ingestar el sistema de forma eficiente. Es decir enfocadas en la ingesta de datos en tiempo real

1.4 Planificación del Trabajo

Puesto que para montar una arquitectura de estas características, se debería disponer de un cluster y no se dispone de dichos recursos, se deberá emular en cierta manera dicho comportamiento. Para ello haremos como ya se ha comentado, se hará uso de una máquina virtual con Docker instalado, de esta forma podremos virtualizar el paralelismo que supondría trabajar con un conjunto de máquinas.

A continuación, se detalla mediante el uso de tablas las principales fases del proyecto. Dentro del diagrama de Gantt adjunto en los anexos se obtiene una mayor granularidad de las fases del proyecto.

Título de la tarea: Documentación	
Fecha inicio: 20/03/2017	Fecha fin: 06/07/2017
Descripción general	
Esta tarea engloba la creación de todos los documentos, desde la creación de las plantillas para el proyecto, así como la posible documentación suplementaria asociada al proyecto.	

Título de la tarea: Implantación de la arquitectura de Web Service	
Fecha inicio:23/03/2017	Fecha fin:07/03/2017
Descripción general	
Esta tarea describe la creación de la instalación de todos los componentes	

necesarios para el inicio del desarrollo de nuestro servicio web. De esta forma podremos pasar al desarrollo de nuestro servicio de simulación de datos sobre los coches de la empresa.

Título de la tarea: Fase de desarrollo Web Service	
Fecha inicio:02/03/2017	Fecha fin:12/04/2017
Descripción general	

Esta tarea constará de dos hitos fundamentales:

- Creación de un formato de traza consistente en formato JSON
- Creación de los diferentes juegos de pruebas
- Desarrollo del propio programa de generación de trazas.

Título de la tarea: Implantación de la arquitectura Big Data	
Fecha inicio:12/04/2017	Fecha fin:31/05/2017
Descripción general	

Esta fase debe contar con un mínimo de las siguientes fases:

- Creación de la máquina virtual CentOS con Docker
- Implantación de Hadoop (HDFS + YARN) en multinodo (un mínimo de un namenode y 2 datanodes).
- Implantación de una máquina Flume con configuración de interceptor + Kafka Channel
- Implantación de una máquina con Kafka.
- Implantación de una máquina con Flume para collector + Kafka Channel
- Implantación de una máquina con Apache Spark

Al final de esta fase se debe disponer de un mínimo de 7 contenedores Docker funcionales capaces de interactuar unos con otros del modo esperado.

Título de la tarea: Desarrollo de soluciones custom Big Data	
Fecha inicio:22/05/2017	Fecha fin:22/06/2017
Descripción general	

Esta fase debe contar con las siguientes fases:

- Desarrollo de un custom Interceptor de Flume para el descarte de trazas erróneas.
- Desarrollo de una solución propia de parseo para el modelo de trazas con Spark Streaming.
- Pruebas de rendimiento del sistema global

Extras:

- Creación de un portal simple propio de muestra de datos en una máquina externa.

1.5 Breve sumario de productos obtenidos

Los productos propios obtenidos en este proyecto y que por tanto serian de desarrollo propio serían los siguientes:

Web Service: Este web service sería un servicio de test generado desde cero. Capaz de enviar trazas hacia una fuente seleccionable.

Máquina virtual (Con Docker instalado): Dentro de esta máquina virtual se podrán encontrar todas las imágenes creadas y operativas para el proyecto.

Interceptor Validador Flume: Se compondrá de un programa desarrollado en java que validará la metadata de las trazas enviadas y que filtrará en consecuencia.

Parseador en real time Spark Streaming: Será un parseador en java que se encargará de la interpretación final de las trazas y mostrar los resultados en una interfaz web. Dicha interfaz web seria desarrollada en PHP.

1.6 Breve descripción de los otros capítulos de la memoria

2. Diseño de la arquitectura.

Durante este capítulo, se explicará la tipología de arquitectura que se usará en el proyecto.
Se verán los pros y los contras de su uso y se realizará una comparativa con otras tecnologías que podrían cumplir los mismos requisitos.

3. Instalación y configuración de Componentes

Este capítulo, será de los más extensos de la memoria. Contará con apartado por apartado, la instalación de cada uno de los componentes dentro de nuestra arquitectura.
Se hará uso de material visual siempre que se pueda o sea necesario para apoyar las explicaciones y en caso de grandes volúmenes de código o configuraciones serán desviadas a los anexos.

4. Diseño de trazas y pruebas de rendimiento

En este apartado, se explicarán el formato de trazas usado, sus componentes y su interpretación.
También se ejecutaran distintas pruebas para medir el rendimiento general del sistema con estas trazas. De esta forma se conocerá si el rendimiento global cae en caso de mostrarse algún nodo del cluster inoperativo o bien podemos seguir manteniendo el mismo nivel de rendimiento.

5. Muestra de resultados

Una vez ejecutadas todas las pruebas en el capítulo anterior, se analizarán y se verá la versión más óptima.
En ese punto, se mostrarán los resultados mediante pantallas de visualización.

6. Conclusiones

Contendrá una reflexión final de todos los aspectos del trabajo por tal de concluir el proyecto.

2. Diseño de la arquitectura

En primer lugar, deberemos justificar de algún modo la elección de esta arquitectura y no otra. Este capítulo se encuentra dividido por tanto en dos subcapítulos donde se muestra cual será la arquitectura y sus beneficios. Consta también de otro capítulo donde se detallan problemáticas y alternativas.

2.1 Arquitectura actual

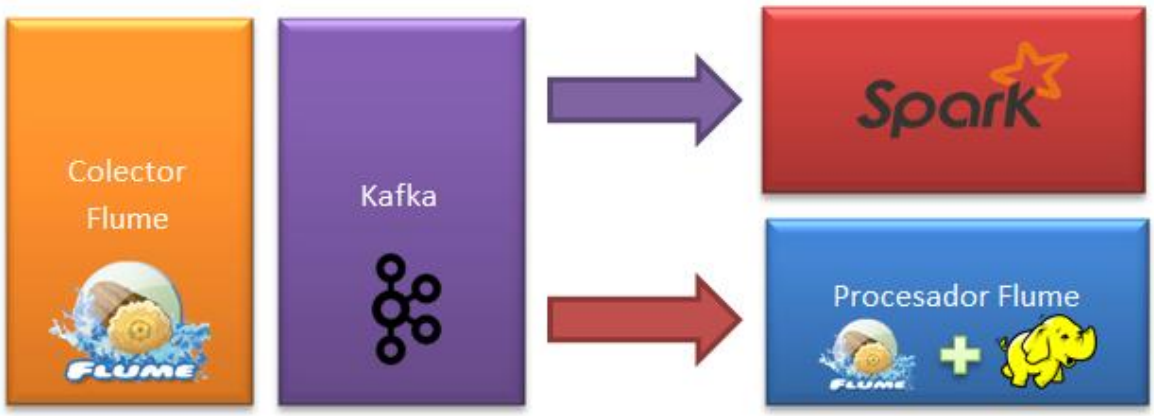


Ilustración 4: Distribución de la arquitectura de alto nivel.

Como se puede ver en la figura anterior, la idea es usar un colector Flume, un canal Kafka y Spark o un procesador Flume dependiendo del caso.

Estos conceptos pueden parecer difíciles a priori, por eso vamos a definir cada uno de los componentes de forma individualizada para mejorar su comprensión.

Empezaremos por el colector Flume, como su propio nombre indica, su deber es recolectar las trazas que llegan y enviarlas hacia Kafka. Lógicamente esto no puede hacerse sin un criterio, primero debemos identificar en este procesador que trazas cumplen con nuestros estándares y cuáles no.
Al final, debemos comprender que Flume es una pipeline persistente, que nos ayudará en al acceso a nuestro sistema.

Seguidamente nos encontraremos con Kafka. Kafka catalogará dentro de topics las trazas válidas y las invalidas. Llegados a este punto es lógico que surja una duda bastante clara ¿Si el sistema es real time, para que usar Kafka? ¿No podríamos usar solo Flume y olvidarnos de la persistencia? ¿De qué sirve esto?
Este planteamiento es lógico, sin embargo debemos pensar siempre a un nivel escalable. En el caso que el cliente quiera montar un sistema de ingesta Batch en un futuro, tener esta persistencia nos permitirá una adaptación mucho más rápida. Es necesario pensar que estos montajes suelen resultar costosos, montar un sistema exclusivo para el real time y otro para el batch puede ser útil pero deberemos adecuarnos a los recursos disponibles en cada momento.

Siguiendo con Kafka, tendríamos un primer sistema de persistencia distribuida. Nuestro siguiente paso por tanto seria doble (puesto que hay que recordar tenemos dos topics, trazas validas e invalidas).
Para las trazas validas Spark las consumirá de forma directa sin intermediarios y serán procesadas y enviadas para mostrar.
En el caso de las invalidas, lo más lógico es montar un sistema de colector Flume donde las trazas se almacenarían en HDFS. A priori se podría pensar ¿Y esto por qué? ¿Una vez más, si es real time, que supone saber esto?

La respuesta a esto no es más que la mejora del sistema general. En esta clase de proyectos existe la figura de un Data Governance, ante la aparición de estas trazas invalidas, se deberá estudiar si son resultado de un mal procesado por nuestro sistema o bien de un envío incorrecto en origen.

Esto es muy importante, puesto que incluso con los datos inválidos le estamos proporcionando al cliente información de valor para mejorar sus sistemas.

2.1 Problemas y alternativas

En el capítulo anterior, hemos visto las bondades del sistema, sin embargo no debemos pensar que está exento de problemáticas, así como tampoco debemos pensar que es una solución única e inamovible.

El principal problema del sistema, se ha podido entrever en el capítulo anterior. El sistema no está al cien por cien orientado al real time, tiene un sistema intermedio de persistencia (Kafka). Igual que en el capítulo anterior mostrábamos que tener este sistema intermedio nos podría servir en implementaciones futuras, debemos pensar que añade delay al conjunto del sistema.

Además, no únicamente añade un delay en el sistema, debemos pensar también que la inclusión de una variedad mayor de sistemas implica la necesidad de reparar una variedad mayor de casuísticas de error en el futuro.

Alternativas

De igual forma, no se debe pensar que la implantación plantada necesariamente sigue un modelo rígido y que por tanto no puede existir sistemas de ingesta en real time distintos al planteado.

Algunas alternativas planteadas son las siguientes:

- Un sistema donde se haya eliminado Kafka. El flujo por tanto seria el siguiente: Flume -> Spark
- Eliminar Kafka y Spark de la ecuación. Podríamos usar un flujo similar al siguiente: Flume -> HBase. HBase es una base de datos NoSQL que dispone de versionado. Poniendo el versionado a 1 dispondríamos de un

sistema en actualización en real time (sin embargo deberíamos usar polling continuo para recuperar los datos).
- Flume y Storm. De la misma forma que actúa Spark, puede ser remplazado por Storm para cumplir la tarea.
- Flume y Flink. En cierta manera similar a Storm o Spark (aunque cada uno tienen sus particularidades).
- Logstash y Spark: Podríamos usar el stack de ELK para el procesado previo y evitar el uso de Flume. Incluso para evitar el sobrecoste de peso sobre la JVM que supone Logstash podríamos remplazarlo por Beats.

Como se puede ver las alternativas son más que variadas e incluso podrían llegar a surgir más.

3. Instalación y Configuración de componentes

3.1 Instalación de la VM con Docker

Lo primero, nos hemos descargado la imagen más actual de 64 bits de CentOS 7 desde su página web oficial. La instalación como tal del sistema no tiene misterio alguno gracias a la user interface, sin embargo sí que se debe tener en cuenta la configuración utilizada para la VM:

Configuración	Definida	Necesidad
Memoria	3GB	Requerimientos Básicos
Procesador	2	Requerimientos Básicos
Disco	40GB	Requerimientos Básicos
Adaptador de red	Bridged	Muy importante, esto nos permitirá tratar a la VM como otro elemento dentro de nuestra red local.
Controlador USB	Si	En algunos momentos puede ser necesario transferir archivos. También es importante habilitar la opción de copia entre Host y VM
Tarjeta de audio	--	Indiferente
Impresora	--	Indiferente
Pantalla	--	Indiferente

Una vez terminada la instalación tenemos una vista similar a esta:

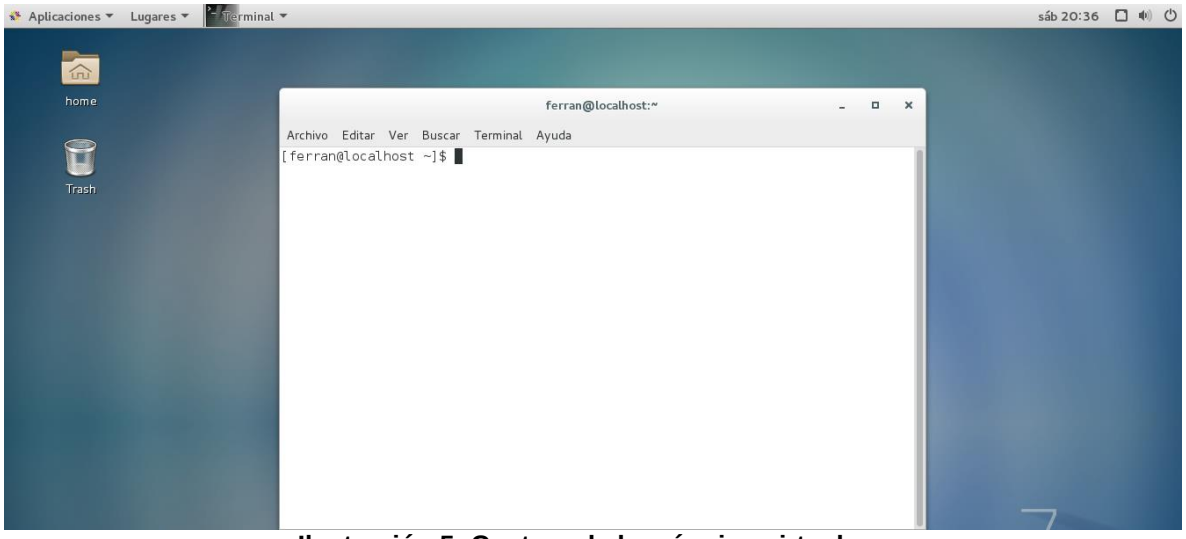


Ilustración 5: Captura de la máquina virtual

El nombre de la máquina es TFM_UOC y los usuarios definidos por defecto en la máquina son:

- root
- ferran

La password para ambos casos es: testVMuoc3495

Previo a la instalación de Docker, será necesario realizar una serie de pasos en cierta manera protocolarios que nos facilitarán mucho la vida.
En primera instancia debemos poner una IP fija a nuestra máquina fuera del rango DHCP de nuestro router para evitar cambios imprevistos de IP.

Para ello realizamos los siguientes pasos:

Ejecutamos la siguiente sentencia: **vi /etc/sysconfig/network-scripts/ifcfg-enp0s3**

Ahora pasaremos a editar el fichero con los siguientes parámetros:

```
HWADDR=xxxxxxxxxxx
TYPE=Ethernet
BOOTPROTO=static
DEFROUTE=yes
PEERDNS=yes
PEERROUTES=yes
IPV4_FAILURE_FATAL=no
IPADDR=192.168.0.211
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_PEERDNS=yes
```

```
IPV6_PEERROUTES=yes
IPV6_FAILURE_FATAL=no
NAME=enp0s3
UUID=xxxxxxxx
ONBOOT=yes
```

Los campos resaltados en amarillo nos asegurarán una IP estática dentro de nuestra red local. Como se puede ver los campos sensibles se han marcado con “x”.
Una vez terminada la configuración y reiniciada la máquina, deberíamos poder hacer un *ipconfig* y comprobar que nuestra IP es la misma que la que hemos marcado en el fichero de configuración.

Una vez hecho esto, podemos pasar a instalar Docker, a continuación se detallan los comandos necesarios:

Comando	Resultado
`sudo yum install -y yum-utils`	Instala los utils de yum por si no vienen por defecto

`sudo yum-config-manager \`
`    --add-repo \`
 	
`https://download.docker.com/linux/centos/docker-ce.repo`
	Añadimos el repositirio de docker community edition a nuestro repositorio local

`yum list docker-ce.x86_64  --showduplicates |sort -r`	Este commando es útil para determinar en que que versión de Docker estable se encuentran.
`sudo yum install docker-ce.x86_64`	Comando que instala Docker en la máquina.
`sudo systemctl start docker`	Una vez instalado, arrancamos Docker

Una vez hecho esto, podemos probar a abrir otro terminar y escribir la palabra **docker,** el sistema ahora la reconocerá y se podrán empezar a crear imágenes o bien instalarlas.

Hay que destacar, que todas imágenes de Frameworks Big Data que se verán durante este capítulo, son auto contenidas y exportables. Esto quiere decir que pueden ser utilizadas en cualquier otro proyecto, con una independencia total unas de otras.

3.2 Instalación de componentes para Raspberry

Para la realización de este proyecto, contamos con una Raspberry PI 3 Modelo B.
Esta Raspberry, hará las veces de emisor de trazas por dos motivos principales:

- Evitar sobrecargas sobre la ya bastante explotada VM
- Comprobar la viabilidad de los contenedores Docker para capturar eventos externos.

Para simular nuestro foco emisor, se ha creado un script que emula el envío de múltiples peticiones POST simultaneas hacia nuestro procesador Flume. Para una versión más detallada del script, referirse a los anexos sección **9.3 Scripts**.

A continuación expondremos los componentes necesarios para desplegar el script en el entorno de Raspbian. El sistema operativo que tiene la Raspberry. La instalación propia del sistema operativo queda fuera de esta explicación puesto que no aporta elementos de valor al proyecto.

El primer paso es cambiar la IP de la Raspberry para que sea de tipo estático, de una forma similar que hemos hecho en la máquina virtual.

```
sudo nano -w /etc/network/interfaces
```

- Editamos el script

```
auto eth0

  iface lo inet loopback

  iface eth0 inet static
  address 192.168.1.201
  netmask 255.255.255.0
  gateway 192.168.1.1
```

- Guardamos y reiniciamos

Si hemos seguido los pasos anteriores de forma correcta, ahora dispondremos de una IP estática en nuestra Raspberry.
En este caso la IP es: **192.168.1.201**

El primer paso que deberíamos realizar para comprobar que tenemos todo bien configurado, es realizar un ping desde la Raspberry a nuestra VM igual que se muestra en la imagen:

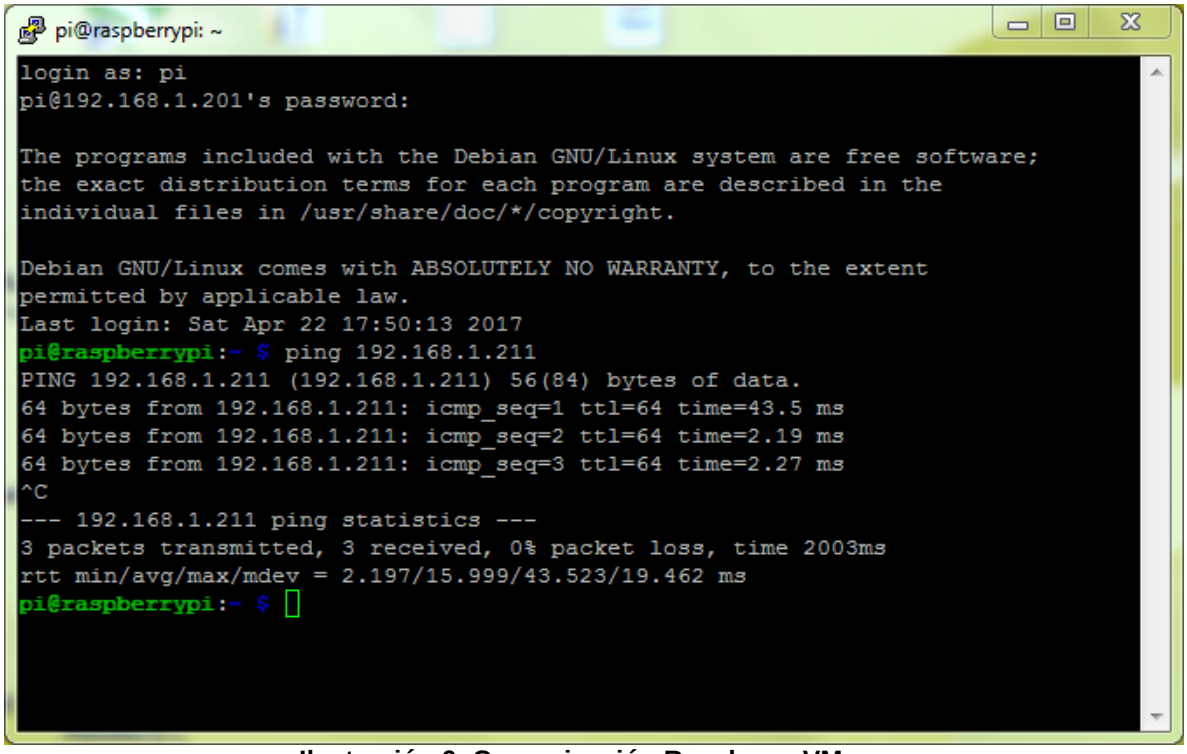


Ilustración 6: Comunicación Raspberry VM

El script realizado para el envío de trazas está hecho en PHP, por lo que será necesario instalar PHP en nuestra Raspberry de cara a poder ejecutar el script. Para ello ejecutaremos la siguiente sentencia:

```
sudo apt-get install libapache2-mod-php5 php5 php-pear php5-xcache
php5-mysql php5-curl php5-gd
```

En la sentencia anterior realmente estamos instalando un número de recursos mayor que los necesarios para el proyecto. Sin embargo de cara a evitar problemas de dependencias, es preferible hacer una instalación lo más completa posible.

Para poder comprobar que tenemos bien instalado PHP en nuestra Raspberry, deberemos poder realizar la misma sentencia que en la imagen y ver que el resultado es el mismo:

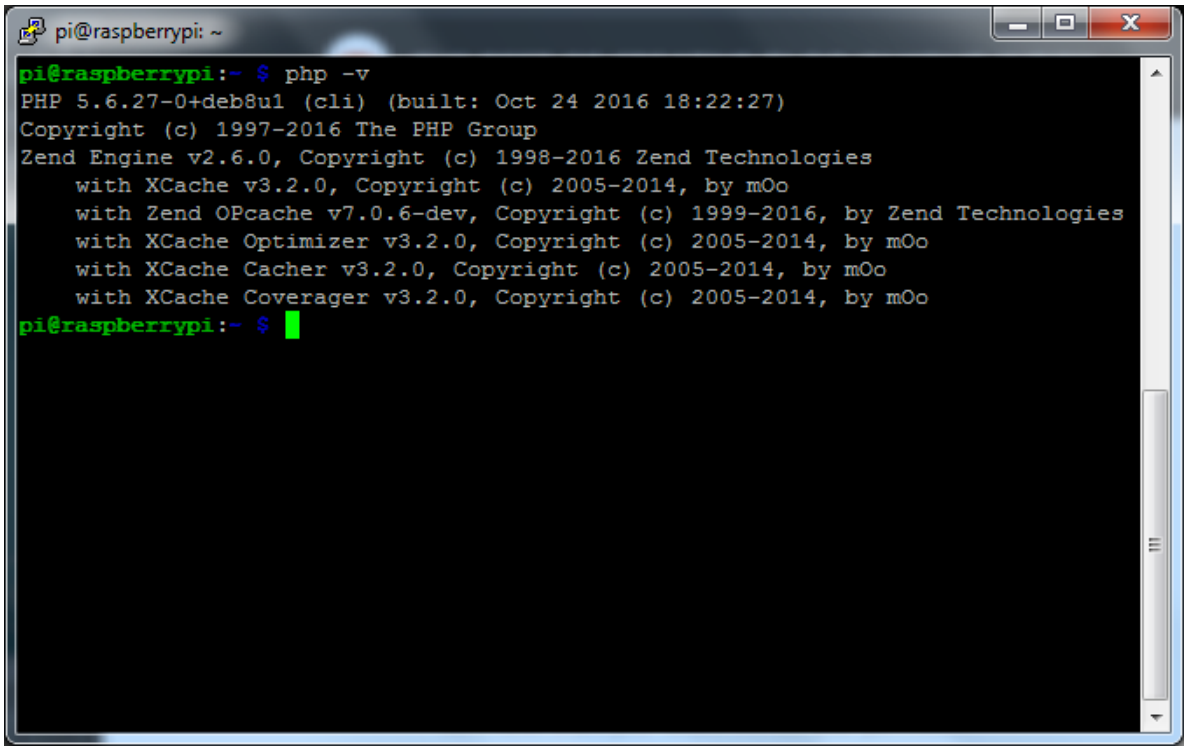


Ilustración 7: Comprobación instalación PHP

Una vez instalados los componentes, será necesario transferir el script por FTP a un directorio de nuestra Raspberry para su ejecución.

Una vez ejecutemos el scripts de envío, las trazas enviadas al colector Flume seguirán la siguiente composición:


```
[
  {
    "headers": {
      "timestamp":"30-05-2017 10:54:21",
      "host":"gtu0091013"
    },
    "body":"
    {\"id\":\"gtu0091013\",\"model\":\"Ford\",\"sta
    tus\":\"running\",\"speed\":3,\"coordinates\":
    {\"long\":\"41.476550\",\"lat\":\"2.251875\"},\
    "tech_features\":
    {\"engine_status\":\"ok\",\"gasoline_level\":49
    ,\"battery_status\":\"ok\",\"oil_status\":\"ok\
    ",\"breaks_status\":\"ok\",\"preasure_level\":
    {\"upper_left\":\"ok\",\"upper_right\":\"ko\",\
    "down_left\":\"ok\",\"down_right\":\"ok\"}}}"
  }
]
```


Ilustración 8: Ejemplificación de traza Flume desde Raspberry

La clave del contenido viajará paquetizado en un JSON dentro del propio body. El trabajo de Flume, no será entender que contiene ese traza, simplemente

deberá hacer la comprobación que el timestamp de la traza no diste más de 10 minutos de la hora del servidor Flume.
Además deberá comprobar que el campo de host desde el cual se envía esa información esta completado, no se deberían aceptar trazas de orígenes no documentados.

3.3 Setup de la imagen Docker Hadoop

En este apartado explicaremos el despliegue de un cluster Hadoop en Docker. La topología del cluster se muestra en la siguiente imagen:

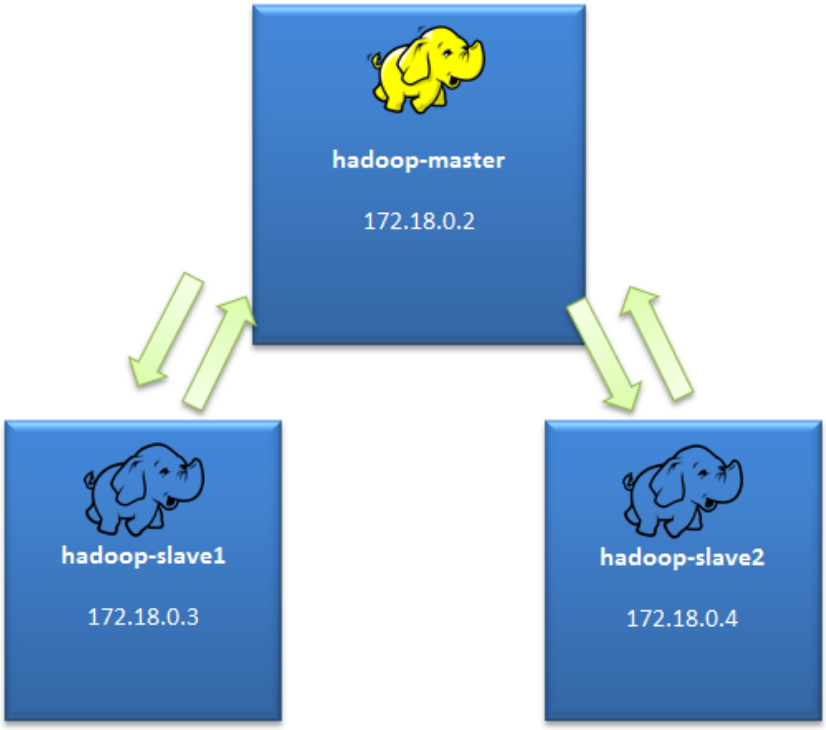


Ilustración 9: Descripción de Cluster Hadoop

Por tal de simplificar, los scripts que crean la imagen están disponibles en la sección **9.3.2 Scripts Hadoop en Docker**

Lo primero que vamos a hacer, es crear una red propia para nuestros elementos Big Data. De tal forma en que nosotros la podamos controlar y modelar con más facilidad. Para ello ejecutaremos la siguiente sentencia Docker:

```
docker network create --driver=bridge bigdata
```

La red como se puede ver, es de tipo Bridge, esto hará que los contenedores Docker puedan comunicarse con la VM y esta a su vez al estar en modo Bridge con el Host nos permitirá comunicarnos con los contenedores como si se trataran de una única máquina.

Seguidamente, debemos navegar hasta la carpeta que contiene el Dockerfile y ejecutar el siguiente comando:

```
docker build -t=myhadoop .
```

Puesto que el despliegue de todos los elementos no es inmediato, se muestra detallado en los scripts del anexo apartado 9.3.

Seguidamente a la ejecución del script **start-container.sh** debemos ejecutar el siguiente comando ya dentro del contenedor **hadoop-master** donde nos encontramos:

```
./start-hadoop.sh
```

Esto puede parecer poco eficaz, puesto que esta sentencia se podría pensar que se puede añadir en el propio script del Dockerfile.
Esta idea es errónea, puesto que el Dockerfile no permite iniciar servicios más allá del tiempo de construcción. Esto quiere decir que cualquier servicio que queramos inicializar con Docker, deberá hacerse a través de un entrypoint posterior a la creación de su imagen y no durante.

Para comprobar que la instalación ha sido satisfactoria deberíamos ser capaces de acceder a las siguientes URLs:

http://192.168.1.211:50070/dfshealth.html#tab-datanode

http://192.168.1.211:8088/cluster

A continuación podemos ver las imágenes de las dos páginas anteriores.

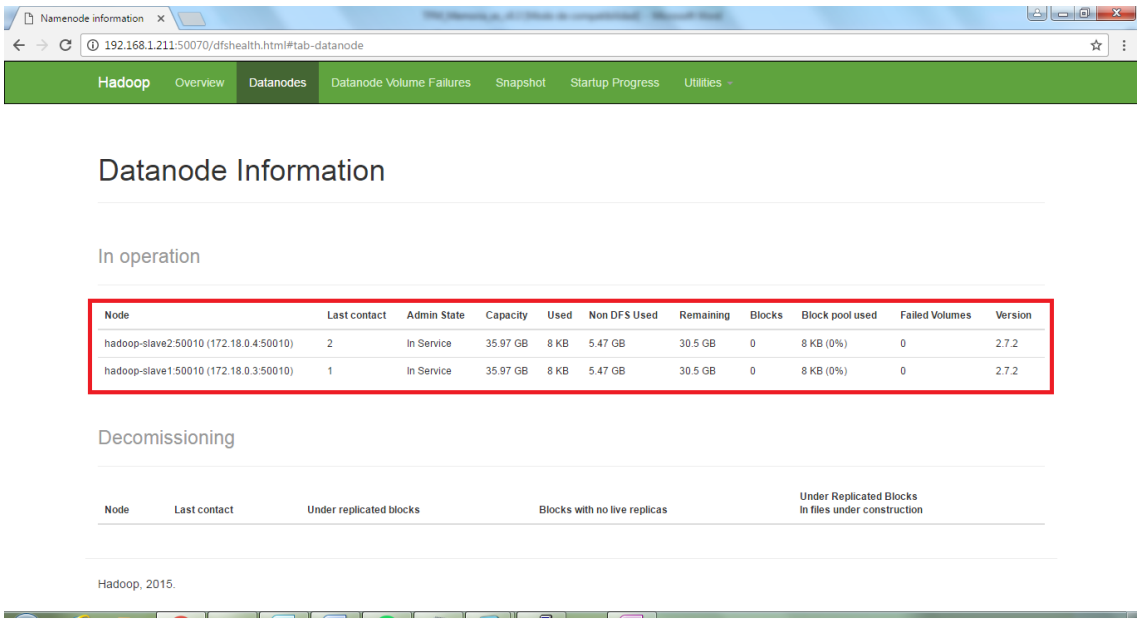


Ilustración 10: Página principal Hadoop

En esta primera página en concreto, debemos ser capaces de ver nuestros dos data nodes que mostrábamos en la imagen de la topología.

Además de información de la capacidad, se muestra la versión de Hadoop que estamos usando, en este caso la **2.7.2**

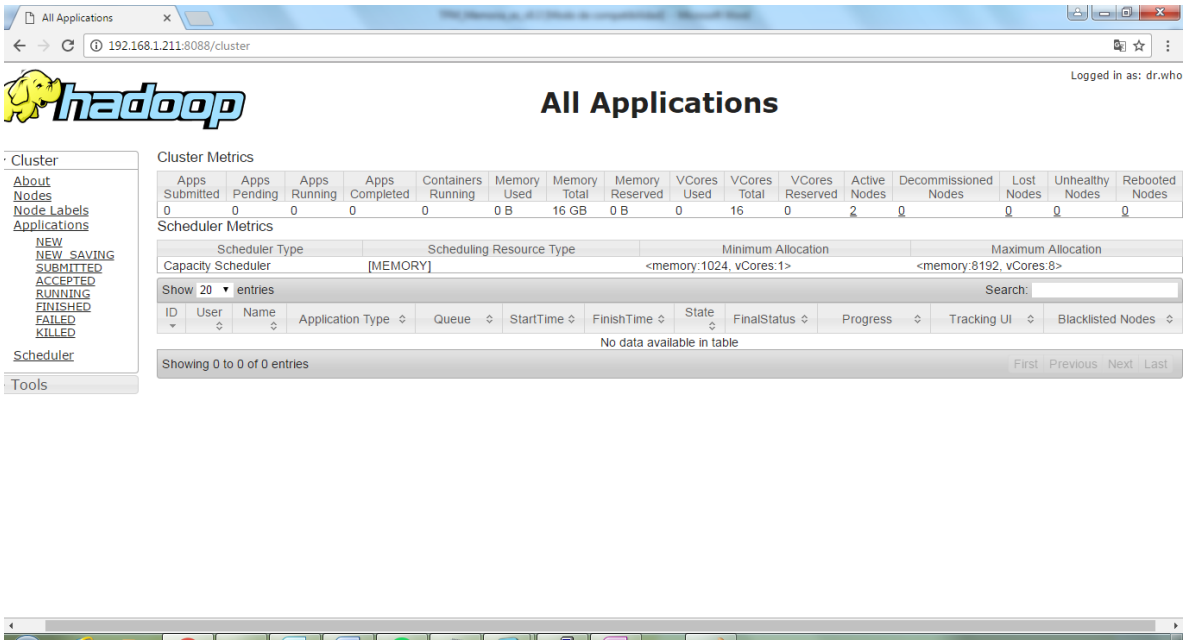


Ilustración 11: Página de métricas y aplicaciones del cluster

En esta captura podemos ver las métricas de nuestro cluster y las aplicaciones que han sido ejecutadas en nuestros nodos.
Esto nos será de gran utilidad en el momento que lancemos aplicaciones Spark y podamos ver si su ejecución ha sido correcta.

3.4 Setup de la imagen Docker Flume

Este capítulo, estará separado en dos secciones. Por una parte el despliegue de la imagen “colector” de Flume y por otro lado la imagen “processor”.

Por tal de ser más entendible, a continuación se mostrará la topología del sistema conjunto con Kafka aunque no se entrará en grandes detalles sobre esta puesto que se verá en el siguiente capítulo:

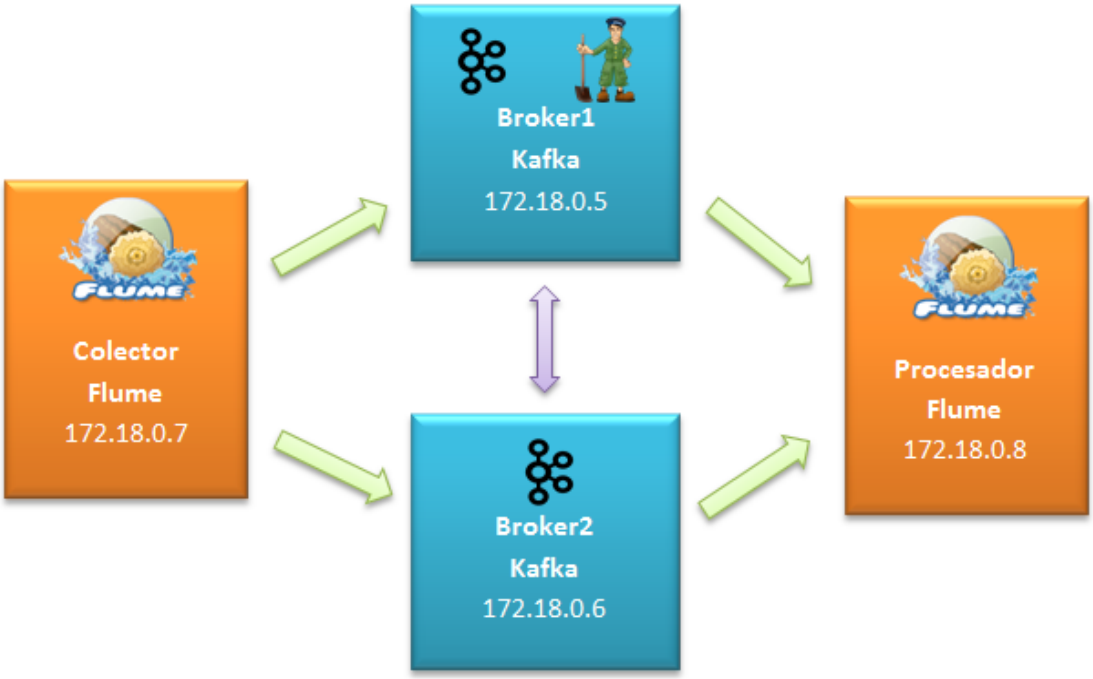


Ilustración 12: Descripción arquitectura Flume + Kafka

3.4.1 Setup de la imagen Colector Flume

Como ya se ha comentado, Flume está separado en dos partes y en este subcapítulo trataremos la instalación de la primera, la sección del colector. Aunque ambas instalaciones son muy similares, es interesante ver sus configuraciones por separado puesto que presentan ciertas particularidades.

Para instalar el colector, deberemos seguir las siguientes instrucciones:

Debemos situarnos en la carpeta donde está el Dockerfile

```
docker build -t=myflumecolector .
```

La imagen de Docker en este caso ya está autocontenida, es decir dispone de un entrypoint con los parámetros de configuración necesarios para poder disponer lanzarla simplemente ejecutando el siguiente comando:

```
docker run -p 44444:44444 --net bigdata myflumecolector
```

Una vez lanzado el comando, el agente empezará a funcionar de forma automática escuchando en el puerto **44444**. Como podemos ver le especificamos que trabajamos en la misma subnet que nuestro cluster Hadoop.

Es necesario recordar, que debido a que estamos usando un Interceptor de desarrollo propio, debemos o bien añadir el jar (y sus dependencias) al path de Flume o bien modificar la configuración de la llamada a flume-ng para añadir dichos plugins.

La tarea de nuestro interceptor, será como el propio nombre indica interceptar nuestros mensajes para un fin en concreto.
La configuración más habitual de Flume, es usar este tipo de herramientas como multiplexores de eventos. Esto quiere decir que no es la tarea del interceptor editar la traza entera para evaluarla, lo más lógico es asignarle unos valores a sus cabeceras para poder distinguir el canal al que serán enviadas.

En caso de que hubiésemos querido actuar sobre la traza de forma previa a nuestro interceptor, deberíamos haber desarrollado un converter. Dicho elemento nos permitiría manipular el body y posteriormente evaluarlo en el interceptor.
Este no es el objetivo en este caso, puesto que será trabajo de Spark generar estos agregados. La razón de esto, es porque Spark actúa de forma distribuida y por tanto actúa de forma más eficiente que un converter de Flume.

El script de Dockerfile y las clases java, se encuentran disponibles en la sección 9.3.3 Scripts Flume Colector.

Siguiendo el archivo de configuración, se puede ver que tenemos nuestro Source HTTP y nuestro interceptor.
El trabajo del interceptor, será decidir (multiplexar) los eventos hacia un canal Kafka u otro. Como se puede ver en la configuración, cada canal apunta a un topic distinto, donde uno de ellos es para las trazas consideradas validas por el interceptor, y otro el de las invalidas.

Finalmente debemos recordar que la Sink no está definida en este archivo de configuración puesto que no está desplegada en este agente.
Suele considerarse una mala práctica añadir la Sink en el mismo archivo de configuración del resto de elementos puesto que puede dar fallos de configuración para algunos casos.

3.4.2 Setup de la imagen Procesador Flume

La forma de desplegar la imagen del procesador, es casi idéntica a la que hemos realizado en el apartado de colector.

Para ello debemos seguir los siguientes pasos:

Debemos situarnos en la carpeta donde está el Dockerfile

```
docker build -t=myflumeprocesador .
```

La imagen de Docker, una vez más está autocontenida, es decir dispone de un entrypoint con los parámetros de configuración necesarios para poder disponer lanzarla simplemente ejecutando el siguiente comando:

```
docker run --net bigdata myflumeprocesador
```

Para este caso concreto no debemos programar nada de forma personalizada, todo lo necesario lo trae Flume ya de por si, solo es necesario configurar de forma correcta el canal Kafka y las rutas a HDFS.

Es necesario recordar, que nuestro procesador apuntará al topic de invalids. No debemos equivocarnos al definir esto puesto que consumiría también las trazas válidas.
En caso que esto pasara, no hay problema puesto que Spark y el procesador Flume pertenecen a dos consumer groups distintos y por tanto podría hacerse sin problema, pero se perdería la razón del topic de invalids.

El Dockerfile, igual que en el caso anterior, se encuentra disponible en el apartado 9.3.4 Scripts Flume Processor.

Como se puede ver en el archivo de configuración, el procesador Flume únicamente ataca sobre el topic de trazas invalidas.
Como ya se ha comentado con anterioridad, las trazas válidas serán procesadas por Spark y las inválidas serán guardadas en HDFS para su posterior estudio.

3.5 Setup de la imagen Docker Kafka

La topología de nuestra imagen Kafka, será como la de la imagen mostrada en el apartado anterior.
Dispondremos de dos Brokers. Uno de ellos actuará como Broker y Zookeeper mientras que el otro es únicamente una instancia Broker.

Realmente la instalación de Kafka en Docker requiere un procedimiento algo manual, puesto que hay que recordar que Docker no persiste servicios fuera del tiempo de construcción, para que pueda hacer eso se debe definir un script como entrypoint.

Es por eso que para poder ejecutar las dos instancias de Kafka, deberemos seguir los siguientes pasos.

Como viene siendo habitual, deberemos navegar hasta la carpeta donde se encuentre el Dockerfile de Kafka y ejecutar los siguientes comandos:

```
docker build –t=mykafka .
docker run -itd --net bigdata mykafka
```

Seguidamente debemos ejecutar el script dos veces con los siguientes parámetros:

```
./start-kafka.sh “BRZ” o “BR”
```

Los parámetros a seguir deben ser BRZ y después BR. Esto es debido a que el primero despertará una instancia de Zookeeper y un Broker y el segundo un solo Broker.

Una vez abierta la primera consola BRZ, debemos ejecutar los siguientes comandos:

```
$KAFKA_HOME/bin/zookeeper-server-start.sh -daemon
$KAFKA_HOME/config/zookeeper.properties
$KAFKA_HOME/bin/kafka-server-start.sh -daemon
/opt/kafka/broker1/server.properties
```

Seguidamente abrimos la consola BR como se ha especificado y ejecutamos el siguiente comando:

```
$KAFKA_HOME/bin/kafka-server-start.sh -daemon
/opt/kafka/broker2/server2.properties
./kafka-topics.sh --create --zookeeper 172.18.0.5:2181 --replication-
factor 1 --partitions 2 --topic valid
./kafka-topics.sh --create --zookeeper 172.18.0.5:2181 --replication-
factor 1 --partitions 2 --topic invalid
```

Una vez desplegado Kafka, lo usaremos de forma nativa. Aunque se pueden hacer desarrollos usando su API de alto nivel, en este proyecto, únicamente haremos uso de su arquitectura propia de persistencia. Los scripts están disponibles en 9.3.5 Scripts Kafka

3.6 Setup de la imagen Docker Spark

La imagen de Spark es relativamente simple de desplegar y no supone mucha problemática.

Hay que tener en cuenta eso sí, una serie de aspectos importantes. Como vamos a usar Spark de forma distribuida a través de YARN, deberemos copiar los archivos de core-site.xml y yarn-site.xml idénticos a los que tenemos en nuestro cluster Hadoop.

Una vez realizado este paso, debemos recordar también que nuestro proceso Spark es propio. Esto quiere decir que es un programa en Java compilado y por tanto deberemos copiarlo a un path de nuestro directorio de Spark y añadirlo como dependencia en nuestro Spark submit o bien, podemos simplemente añadir el jar en el path de jars de Spark.

Hay que recordar también, que de forma habitual los programas generados tienen dependencias de otras librerías. Estas librerías las deberemos importar para poder enviar las dependencias a los workers y ejecutar el programa de forma satisfactoria.
Como hasta ahora, deberemos navegar hasta la carpeta donde se encuentre el Dockerfile de Spark y ejecutar los siguientes comandos:

```
docker build -t=myspark .
```

Seguidamente ejecutamos los siguientes comandos:

```
docker run -itd --net bigdata --name spark-master --hostname spark-
master myspark
```

Llegados a este punto, deberemos navegar hasta la carpeta **/usr/local/spark/bin** y ejecutar Spark o bien en modo distribuido como el primer comando o bien en modo Standalone como en el segundo:

```
./spark-submit --class com.spark.custom.SparkNormalization --master
yarn --deploy-mode cluster\
 --executor-memory 1G --num-executors 2\
 /usr/local/spark/jars/spark-custom-streaming-0.0.1-SNAPSHOT.jar
"172.18.0.5:9092,172.18.0.6:9092","ws://192.168.1.201:9300"
```

```
./spark-submit --class com.spark.custom.SparkNormalization \
 /usr/local/spark/jars/spark-custom-streaming-0.0.1-SNAPSHOT.jar
"172.18.0.5:9092,172.18.0.6:9092","ws://192.168.1.201:9300"
```

Las diferencias entre uno u otro modo se aprecian en los siguientes esquemas ilustrativos de funcionamiento:
En el primer caso, actúa de forma distribuida usando YARN:

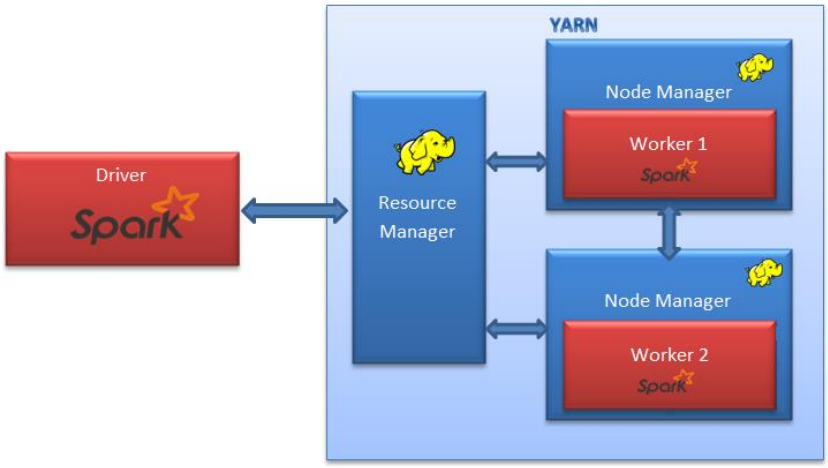


Ilustración 13: Spark submit mediante YARN

En el segundo caso, standalone, Spark no distribuye entre diversos Workers en varias máquinas sino que únicamente en una única máquina. Es necesario recordar que aunque no existan diversos Workers, dentro de esos Workers sí que tienen diversos Executors, por tanto el procesado en paralelo se mantiene:

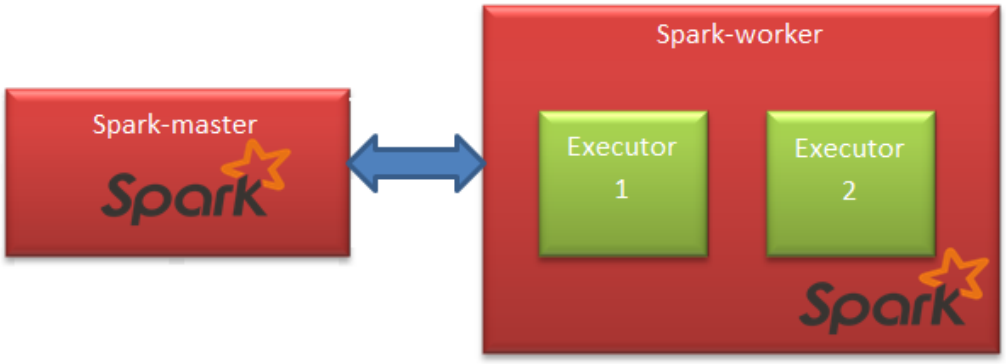


Ilustración 14: Spark submit mediante standalone

Todos los scripts referentes a Spark, se encuentran disponibles en 9.3.6 Scripts Spark

4. Diseño de trazas y pruebas de rendimiento

Las trazas que procesa en su punto final Spark, después de pasar por Flume y Kafka, son justamente el body que se mostraba en la ilustración 8 del apartado 3.2.

A continuación se muestra una imagen más completa donde se detallan cada uno de los puntos:

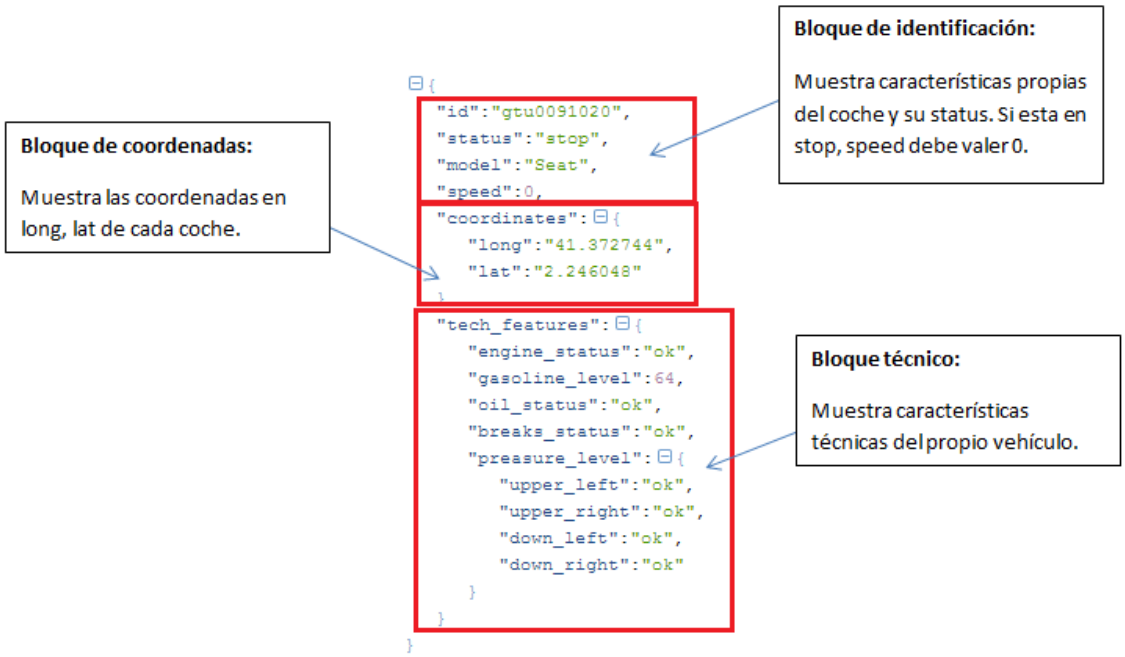


Ilustración 15: Descripción técnica de la traza

Una vez vista la estructuración de la traza final JSON, Spark deberá reconocerla y agregarla en su ventana temporal de Spark Streaming por tal de sacar información de valor.
Este punto se verá en mayor detalle en el siguiente apartado 5. Muestra de resultados, donde se detallarán las pantallas de visualización y los agregados realizados por Spark para poder llegar a ellas.

En este capítulo por tanto nos centraremos en un apartado algo más técnico y no de visualización. Una vez sabido la estructuración que tendrá Spark, ha llegado la hora de probar el rendimiento del sistema global. Para ello se ha preparado una batería de pruebas donde se estudiará el rendimiento del sistema global bajo una seria de condiciones.

Antes de detallar las pruebas, hay que recordar que el foco emisor, la Raspberry en el caso de este proyecto, no ha cambiado ni la cantidad ni el tamaño de las trazas en ninguna de las pruebas.

De esta forma se garantizan condiciones reales de entrada y por tanto se puede llegar a apreciar el funcionamiento real en mejor o peor medida de las tecnologías Big Data empleadas.

4.1 Nodo HDFS caído y el resto del sistema activo

En este apartado, simularemos que uno de los datanodes de HDFS, está caído y por tanto no es capaz de guardar elementos. Puesto que la SINK de Flume de trazas invalidas escribe sobre HDFS, estudiaremos si existe o no una bajada notable de rendimiento bajo estas premisas.

Para poder simular que un nodo de HDFS está caído, procederemos a ejecutar el siguiente comando en la Shell:

```
docker stop hadoop-slave2
```

Tras esto, iniciamos el sistema ejecutando el siguiente comando desde la Shell de la Raspberry:

```
php server.php (SHELL 1)
php mainSend.php (SHELL 2)
```

No es necesario afectar al resto del sistema para poder realizar este test, simplemente el sistema se adapta de forma automática a la nueva condición.

Una vez iniciado el test, debemos repasar los logs del procesador de Flume, en caso de existir algún tipo de problema en la inserción estaría allí.

Parece que los tiempos de inserción en HDFS no han cambiado con respecto al modelo funcional total. Esto quiere decir que el sistema está preparado para aguantar el flujo normal incluso sin uno de sus datanodes activo.

En cierta forma se podía ya intuir que el resultado iba a ser así, realmente la cadencia de escritura sobre HDFS debido a trazas invalidas es realmente poca. Esto es debido a que como premisa, se ha supuesto que el número de trazas validas, sería superior a las invalidas y que por tanto la cantidad de estas últimas se vería ampliamente superada.

4.2 Broker de Kafka caído y el resto del sistema activo

En este apartado, se simulará la caída del bróker 2 de Kafka, haciendo así que Kafka trabaje solo usando un bróker.
Así se podrá medir el rendimiento del sistema bajo estas condiciones, a diferencia del caso anterior, en Kafka entran el 100% de todas las trazas enviadas por la fuente emisora, por lo que cualquier bajada de rendimiento por parte de Kafka afectaría de forma notable al conjunto del sistema.

Se ha realizado un experimento de estrés, enviando mensajes al sistema con una cadencia de 0,2 segundos en el sistema completo y en el sistema sin el bróker 2.

Los resultados que se esperaban de este experimento, era poder comprobar que existía una clara desviación en el hecho de usar dos nodos o uno.
En la siguiente imagen se puede ver un gráfico con el número de mensajes enviados y el tiempo de respuesta de Kafka basado en los logs resultantes.

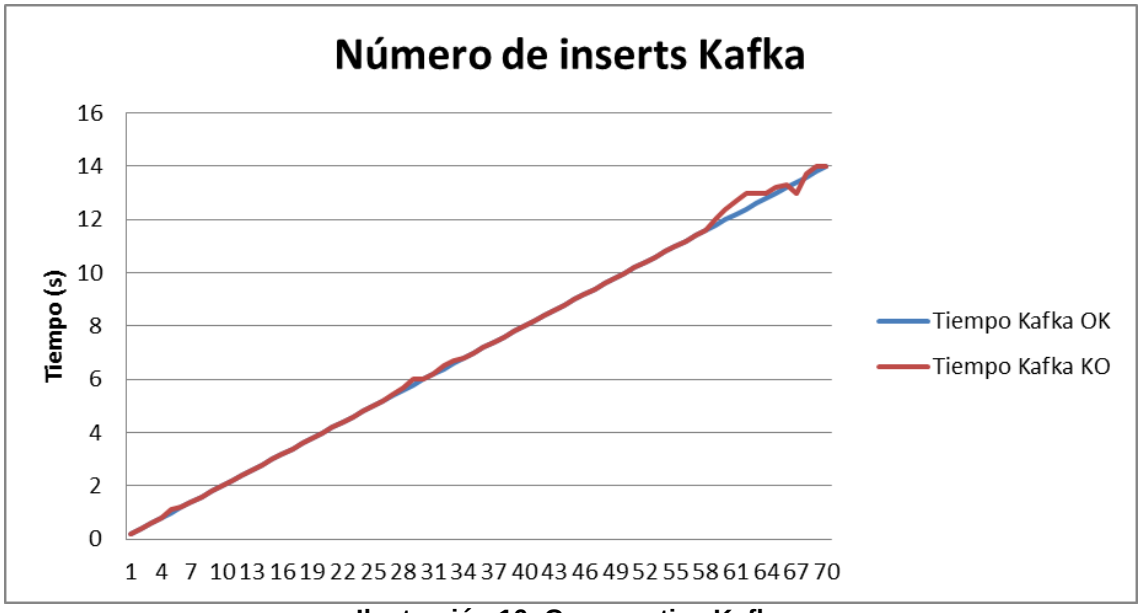


Ilustración 16: Comparativa Kafka

El resultado es una clara sorpresa puesto que no se cumple la hipótesis que se había planteado desde el inicio.
En el gráfico se puede apreciar una ligera diferencia de los datos cuando el bróker está o no operativo, sin embargo dicha diferencia es muy poco sustancial como para poder determinar con seguridad que el sistema pierde un rendimiento claro.

Las posibles explicaciones de este fenómeno, podrían ser diversas aunque la que se plantea como más clara, es el hecho de que la fuente de datos no envía la suficiente cantidad de trazas como para desbordar el sistema en ninguno de los casos.

Kafka está diseñado para aguantar (incluso en las condiciones de Docker + VM) una gran cantidad de carga de datos de entrada. Nuestra fuente no es capaz de hacer un daño real a Kafka como para poder ver una diferenciación real de rendimiento.

Es de suponer que en sistemas productivos la pérdida de un bróker contando con únicamente dos nodos tendría una afectación sobre el resto de sistemas.

4.3 Nodo HDFS y Broker Kafka caído con el resto del sistema activo

Por tal de intentar causar un crash al sistema ahora tiraremos un nodo de HDFS y uno de Kafka.

Una vez más ejecutaremos los comandos de stop de docker:

```
docker stop hadoop-slave1
docker stop kafka-broker2
```

Después de unos segundos ya tendremos a nuestro sistema funcionando a medio gas.

Igual que en el resto de experimentos enviaremos mensajes con una cadencia de envío de 0,2s. A la espera de ver una ralentización real del sistema completo.

Lamentablemente los resultados se asemejan al apartado anterior, el hecho de haber tumbado la mitad de nuestro sistema no ha tenido una afectación real puesto que nuestra fuente no hace un envío lo suficientemente contundente como para afectar a los contenedores.

Podemos por tanto asumir que hemos tomado unos criterios de diseño iniciales algo sobredimensionados. Es posible por tanto con la fuente actual trabajar con todos los componentes hasta Kafka sin que el sistema se resienta.

4.4 Spark submit en modo YARN con todo el sistema activo

A continuación el siguiente experimento lo realizaremos lanzando nuestro motor de agregación, en este caso usando Spark.
Lo lanzaremos en modo YARN y por tanto trabajará en cluster repartiendo los workers y lanzándose de forma distribuida.

El resultado de este experimento es por primera vez una bajada de rendimiento espectacular del sistema. Hasta ahora habiams desactivado tecnologías que usaban disco para trabajar y la VM podía cargar con ellas sin problema y su falta no le suponía tampoco problemática.
El hecho de usar Spark que se carga en gran parte en memoria ha hecho que la VM casi se pare en el momento de la ingesta. Se ha intentado modificar los parámetros de asignación de memoria de Spark para el driver y los workers de cara a ser más austeros pero no se ha podido hacer funcionar de forma correcta.

Durante este experimento Spark no ha sido capaz de enviar ninguna traza al web socket, ha sido tan espectacular su impacto, que ha afectado a otras tecnologías como Kafka.

Kafka presentaba un error en el que decía que debido a la falta de memoria del sistema, no era capaz de escribir la información en los tópics haciendo por tanto que Flume fallara.

Ejecutar por tanto Spark en modo YARN ha sido devastador para el sistema global, no permitiendo por tanto realizar una prueba bajo estas condiciones. En un sistema productivo real la ejecución en YARN sería la forma más eficiente de proceder.

4.5 Spark submit en modo standalone con todo el sistema activo

Por último, el experimento final será ejecutar Spark en modo standalone. Puesto que en modo YARN no ha sido posible, se espera que en modo standalone sí que lo sea y que por tanto se puedan llegar a ejecutar la generación de los agregados.

Una vez restaurado todo el sistema, después del error debido a la falta de memoria del experimento anterior, podemos proceder a la ejecución del proceso Spark.

El resultado de esta ejecución fue 100% satisfactoria generando el resultado esperado. El resultado es planteado en más detalle en el video de Demo de la presentación adjunta.

El rendimiento del sistema no ha sufrido en absoluto, en algunos puntos tal vez sufría una pequeña ralentización debido en gran medida al rendimiento de la VM pero no era realmente apreciable

5. Muestra de resultados

En este apartado, se verá la parte más gráfica del proyecto. En ella se explicará la pantalla de visualización diseñada para el proyecto y que genera Spark mediante diversos agregados de la traza.

El resultado general de la pantalla de visualización se puede ver en la siguiente imagen general:

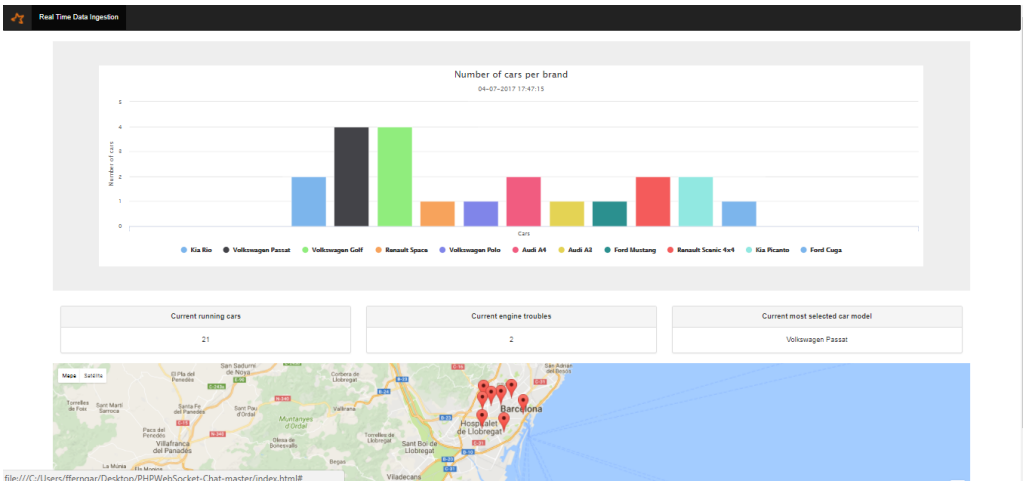


Ilustración 17: Dashboard Spark

La pantalla de visualización se compone de los siguientes apartados:

Gráfico de agregados de coches por marca:

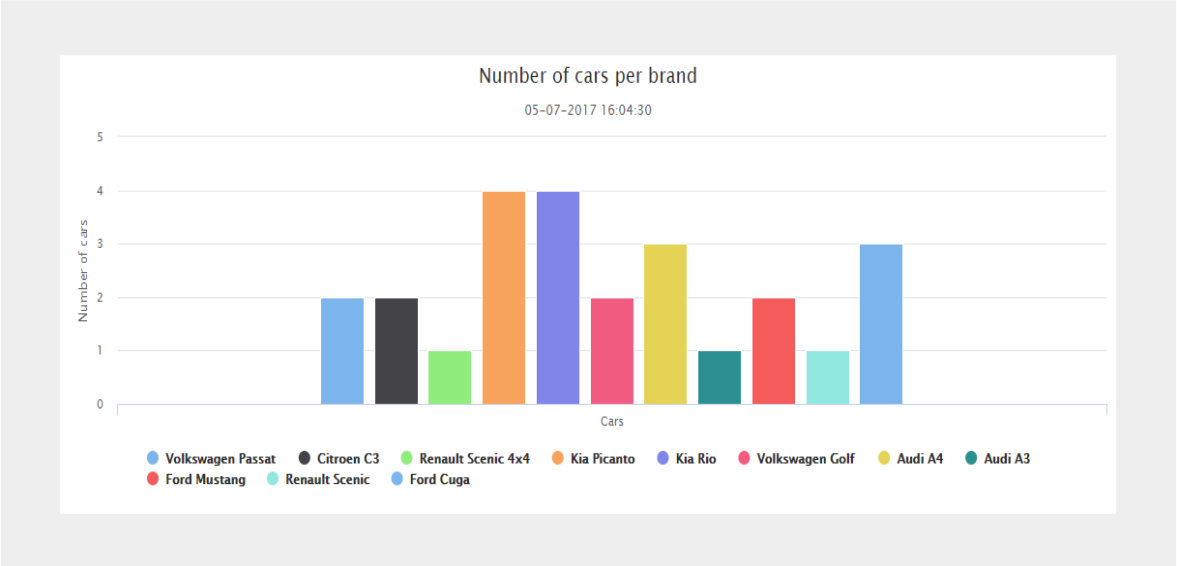


Ilustración 18: Gráfico de coches por marca

En este gráfico generado con Highcharts, se puede ver en tiempo real, la cantidad de coches que se encuentran en circulación (estado “running” en las trazas) en ese momento.

Seguidamente si nos desplazamos abajo, tenemos diversas cajas que muestran información agregada de la cantidad de coches, el más usado y su estado del motor.

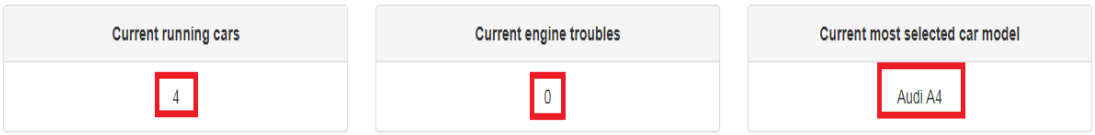


Ilustración 19: Agregados Box

Realmente gracias al diseño de trazas podrían existir muchos más agregados a realizar, sin embargo para este caso particular se ha decido usar estos.
Finalmente gracias a la información geográfica que tienen las trazas, ha sido posible posicionar los diferentes coches en un mapa generado mediante la API de Google Maps.

Ilustración 20: Mapa Dashboard

6. Conclusiones

Durante el desarrollo he podido trabajar con diversas tecnologías Big Data bastante punteras a día de hoy. Además he podido probar la tecnología de virtualización de Docker que ofrece mayor flexibilidad y rendimiento que las típicas máquinas virtuales, esto ha permitido poder destruir y crear contenedores de forma muy rápida para poder dimensionar el sistema.

También he aprendido a trabajar mediante el uso de los contenedores Docker y con un sistema distribuido. El proyecto ha supuesto un verdadero reto en la dificultad de hacer entender a todas las tecnologías unas con otras.

Por suerte creo que se han conseguido todos los objetivos del proyecto que se plantearon en su inicio. Se ha sido capaz de crear un sistema de ingesta en tiempo real y además hacerlo de forma eficiente y mostrar los resultados de forma atractiva. Además mediante el uso de Docker los módulos pueden ser reutilizables para futuros proyectos así como escalables a nivel de hardware.

La única problemática que no se ha podido realizar de forma satisfactoria, ha sido la prueba usando YARN debido a las condiciones de memoria del equipo. Este ha sido el único objetivo que marcaría como pendiente en el proyecto y que podría quedar como punto para el futuro a nivel estructural.

Los puntos de mejora a nivel técnico de uso de nuevas tecnologías para el futuro podrian ser:

El uso de Apache Storm o Apache Flink como substitución de Spark Streaming como punto final del sistema de ingesta.

Incluso se podría crear agregados en tiempo real mediante el uso de HBase y coprocessors.

7. Glosario

- **Batch**: Programa que no es ejecutado en tiempo real
- **Real Time**: Ejecución en tiempo real
- **Cluster**: Conjunto de servidores que se comportan como una única máquina.
- **VM**: Virtual Machine (Máquina virtual)
- **HDFS**: Hadoop Distributed File System
- **YARN**: Yet Another Resource Negociator
- **Web Service**: Es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.
- **Streaming**: Retransmisión continua sin interrupción
- Data Governance: Tiene la función de establecer estándares, políticas y procesos que determinen el uso, desarrollo y gestión de los datos a nivel corporativo.
- **IP**: Internet Protocol; El número que identifica a cada dispositivo dentro de una red.
- **DHCP**: Dynamic Host Configuration Protocol; Es un servidor que usa protocolo de red de tipo cliente/servidor en el que generalmente un servidor posee una lista de direcciones IP dinámicas y las va asignando a los clientes.
- **ELK**: Acrónimo que se refiere al stack de Elasticsearch
- **JSON**: JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos
- **HTTP**: Hypertext Transfer Protocol
- **Sink**: Componente propio de Apache Flume, vendría a significar (sumidero) y supone el punto final de procesado de datos de un agente.
- **API:** Application Programming Interface, es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
- **Path:** Ruta a un archivo o carpeta dentro de nuestro sistema de ficheros.
- **FTP:** File Transfer Protocol; Protocolo de transferencia de ficheros.

8. Bibliografía

Libro: Hadoop The Definitive guide; Autor: Tom White; Año publicación: 2009

Libro: Spark Coockbook; Autor: Rishi Yadav; Año publicación: 2015

Libro: Introducción a Apache Spark; Autores: Mario Macías, Mauro Gómez, Rubèn Tous, Jordi Torres; Año publicación: 2015

Web: https://www.docker.com/

Web: https://docs.docker.com/engine/installation/linux/centos/#install-using-the-repository

Web:https://docs.docker.com/engine/reference/commandline/docker/#child-commands

Web: https://lintut.com/how-to-configure-static-ip-address-on-centos-7/

Web:https://www.digitalocean.com/community/questions/how-to-enable-ip-masquerading-forwarding-on-centos-7

Web:http://stackoverflow.com/questions/37920923/how-to-check-whether-kafka-server-is-running

Web: https://kafka.apache.org/intro

Web: https://es.wikipedia.org/wiki/Apache_Kafka

Web: https://es.wikipedia.org/wiki/Apache_Flume

Web: https://en.wikipedia.org/wiki/Apache_Spark

Web: https://flume.apache.org/FlumeUserGuide.html

Web: https://spark.apache.org/docs/latest/streaming-programming-guide.html

Web:https://www.gitbook.com/book/jaceklaskowski/mastering-apache-spark/details

Web: https://es.wikipedia.org/wiki/Docker_(software)

Web:https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/FileSystemShell.html#mkdir

Web:https://www.ashishpaliwal.com/blog/2013/06/flume-cookbook-implementing-custom-interceptors/

Web:https://www.cloudera.com/documentation/other/tutorial/CDH5/topics/ht_flume_to_hdfs.html

Web:http://sosedoff.com/2013/12/17/cleanup-docker-containers-and-images.html

Web: https://askubuntu.com/questions/505506/how-to-get-bash-or-ssh-into-a-running-container-in-background-mode

Web: http://theckang.com/2015/remote-spark-jobs-on-yarn/

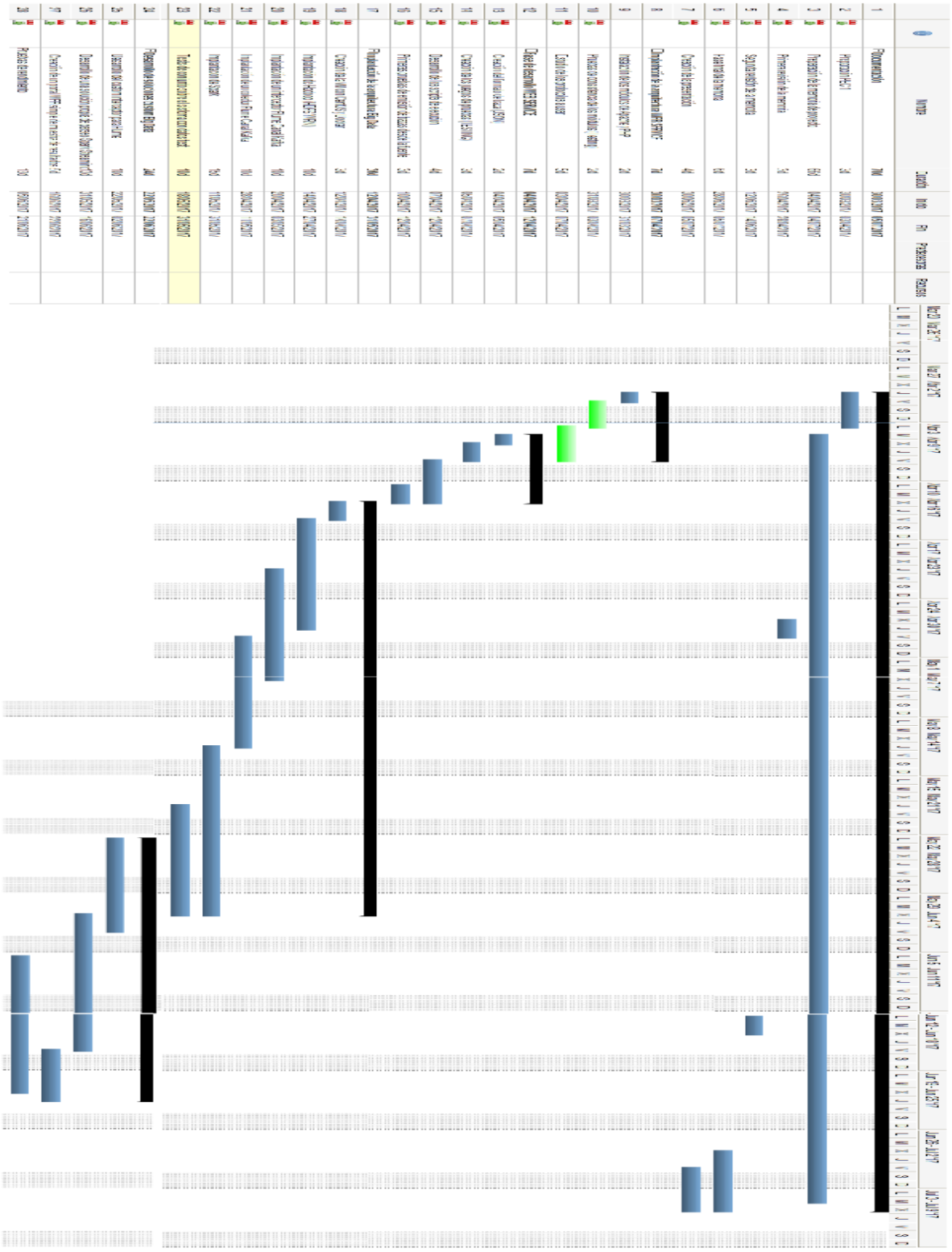
Web: https://flink.apache.org/

Web: https://www.elastic.co/webinars/introduction-elk-stack

Web: http://hbase.apache.org/book.html#arch.overview

9 Anexos

9.1 Diagrama de Gantt:



9.2 Tecnologías:

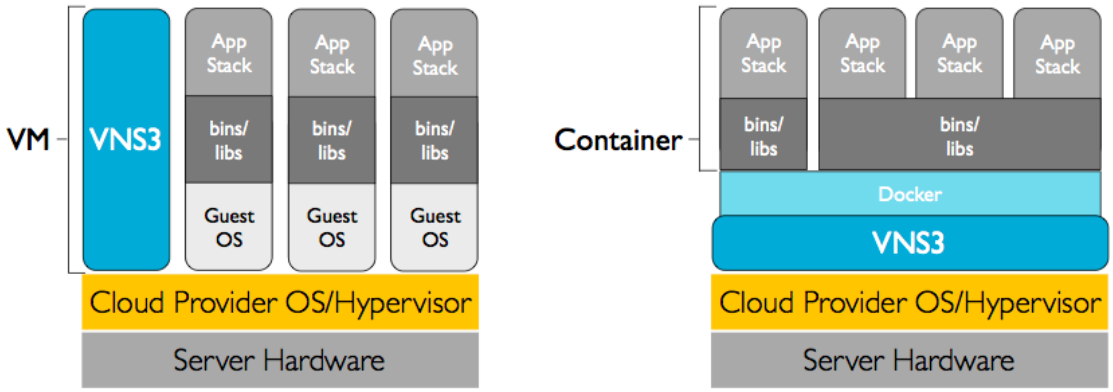
9.2.1 Docker

Docker es una plataforma para desarrollar, empaquetar y ejecutar aplicaciones empleando tecnologías de virtualización de contenedores. Está escrito en Go y se aprovecha de características importantes del kernel de Linux.

¿Qué son los contenedores?

- Entornos de ejecución autocontenidos con su propia CPU, memoria, I/O, FileSystem e interfaces de red que dan como resultado procesos aislados e independientes del sistema operativo anfitrión.
- Comparten el kernel del sistema operativo anfitrión.
- Aíslan el software para que siempre se ejecuten en el mismo entorno. Ya no hay diferencias entre entornos.
- Similares a las máquinas virtuales pero sin el overhead del sistema operativo invitado.
- Son más ligeros y arrancan más rápido. Se puede de disponer de múltiples contenedores por máquina.

Durante el desarrollo de este proyecto, se ha usado Docker para paquetizar todas las tecnologías Big Data de cara a permitir emular disponer de un cluster de máquinas. En caso de no haber podido disponer de Docker, para simular la misma situación que se tiene en la topología del proyecto, habría sido necesarias varias máquinas virtuales, las cuales suponen un overhead muy importante para el sistema host el cual probablemente no habría aguantado apenas más de una o dos máquinas virtuales de poca potencia funcionando de forma simultánea. Se puede ver clara una comparativa en la siguiente imagen:



9.2.2 Hadoop (HDFS y YARN)

Hadoop fue creado por Doug Cutting, el creador de Apache Lucene y su curioso nombre, viene dado debido a que el hijo de Doug Cutting tenía un peluche de un elefante amarillo el cual tenía el nombre de Hadoop.

Hadoop está a su vez basado en un paper del año 2003 publicado por Google, donde describía el funcionamiento de su GFS (Google Distributed Filesystem). Poco después, en el año 2004 Google publicó otro paper donde se explicaba el funcionamiento de MapReduce. Gracias a estos dos papers y a la aportación de Yahoo que creó un equipo de desarrollo dedicado a ello, se creó el proyecto de Hadoop.
Hay que recordar, que Hadoop tiene prioridad sobre tres grandes áreas:

- Fiabilidad y tolerancia a fallos. Esto lo hace mediante el uso de replicación entre nodos.
- Económico: No requiere de una gran inversión inicial para funcionar porque está adaptado a diversas tipologías de hardware.
- Escalabilidad: Esta última es clave, la idea de ir añadiendo nuevos nodos a la red de forma simple y que suponga un aumento de capacidad casi instantáneo es una particularidad increíble,

Dentro de este proyecto, se han utilizado dos de los servicios los cuales ofrece Hadoop, que son HDFS y YARN. Sin embargo el Stack de Hadoop es mucho más extenso como se aprecia en la imagen:

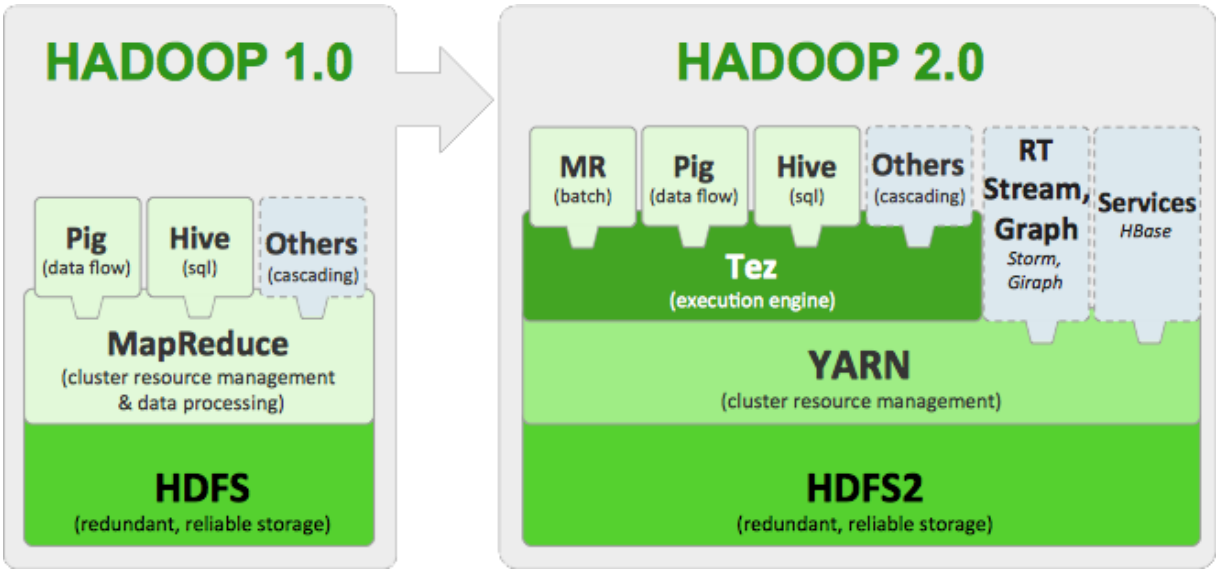


Ilustración 21: Ecosistema Hadoop

En este proyecto se ha usado la imagen de Hadoop 2.7.2 y por tanto se han usado las tecnologías:

HDFS (Hadoop Distributed File System): Es el servicio de almacenamiento de datos de forma distribuida entre los nodos del cluster.

YARN (Yet Another Resource Negociator): Se ocupa de la gestión de recursos dentro del cluster.

9.2.3 Apache Flume

Apache Flume es una herramienta que trabaja en streaming y su principal característica es ser una pipeline persistente.
Es decir, su funcionalidad principal es recoger datos del punto “A” y transportarlos al punto “B”. Es posible que entre medias los datos puedan sufrir agregados de algún tipo, sin embargo este tipo de funcionalidades se le suelen dejar a Spark Streaming.

Flume funciona por agentes, un agente es el encargado de ejecutar todas las funcionalidades y para ello dispone de una serie de herramientas que son:

Source (Fuente): Técnicamente la source o fuente no formaría parte del propio agente pero se le debe indicar cuál es, por ejemplo un nodo HDFS, un Elastic, Otro agente Flume, etc.

Channel (Canal): De canales hay muchos y variados, en este proyecto se a optado por un canal persistente Kafka,sin embargo, se podría haber creado un canal in memory.
Todo tiene su parte buena y su parte mala, un canal in memory puede llegar a sufrir pérdidas mientras que un canal Kafka no (lo cual no lo hace perfecto).

Sink (Sumidero): Es el punto final donde llegarán los datos, por ejemplo HDFS, Hive, HBase…

Además de estos componentes, existen otros dos no tan conocidos aunque importantes de cara a hacer agregaciones con los datos de entrada:

Interceptor (Interceptor): Su trabajo por lo general es validar las trazas y multiplexar en diversos canales en función de un criterio establecido.

Converter: Se ejecuta previo al interceptor, su función es modificar la traza de una u otra forma por la razón que se quiera.

Desplegar un agente Flume, es relativamente simple, sin embargo hay que recordar que la configuración debe estar correcta para que todo funcione.

Además es necesario recordar que Flume no funciona distribuido en un cluster como podría funcionar Spark. La topología que tome Flume es responsabilidad de cada arquitecto y por tanto por esa misma razón se debe tener presente que Flume debe intentar evitar agregaciones que supongan un gran consumo de recursos, sobre todo si el volumen de eventos a procesar es muy grande.

9.2.4 Apache Kafka

Apache Kafka es una distribución bajo la Apache Software Fundation, su principal característica es crear una plataforma de alto rendimiento y baja latencia.
Su principal objetivo es poder lidiar con fuentes en tiempo real y su símil con las tecnologías ya existentes podría ser una cola de mensajes como las típicas colas JMS.

Kafka sigue el patrón de publicación-subscriptor mediante el uso de los topics. Dichos elementos son escalables a nivel de replicación y distribución dentro del cluster de Kafka.

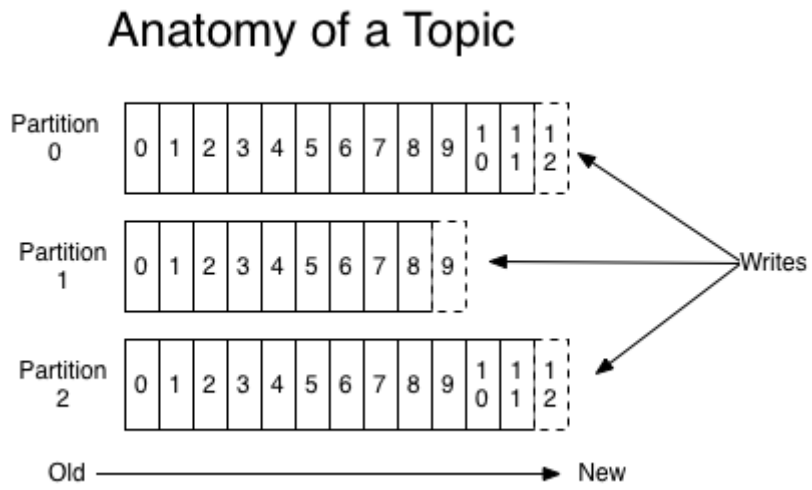


Ilustración 22: Cola Kafka

Como se aprecia en la imagen un topic es la unidad mínima de la que se compone Kafka y como ya se ha comentado está particionado y replicado. Esto ayuda a poder tener alta disponibilidad además de ganar velocidad en la escritura.

En cuanto a la lectura, cada lector lee únicamente de una partición y tiene la responsabilidad de guardar el offset de la lectura. Es decir para ganar el velocidad, Kafka no posee un registro de hasta donde ha leído un lector. Es responsabilidad del lector que programe el consumidor hacerlo para que sea capaz de guardar dicha posición de cara a poder continuar por ese punto si ocurre un fallo inesperado del sistema.

Dentro del proyecto, Kafka se usa como elemento de arquitectura pasiva, es decir no se han desarrollado elementos concretos que den una funcionalidad extra a la arquitectura ya existente. Por tanto se confía en el emisor y escritor Flume de cara a la escritura y el consumo de los mensajes de nuestro cluster Kafka.

9.2.5 Spark Streaming

Spark es una distribución bajo la Apache Software Fundation que provee un framework open source de computación distribuida para el tratamiento de datos.
Spark Streaming por tanto, es una de las herramientas de las que se compone dicho framework y que está preparado para el tratamiento de datos en tiempo real.

Su nombre puede llegar a ser engañoso puesto que el tratamiento que hace Spark de los datos no es en verdadero tiempo real. Utiliza una ventana temporal denominada “micro-batch” la cual como mucho puede llegar a ser de 0,5s fina en el margen temporal.

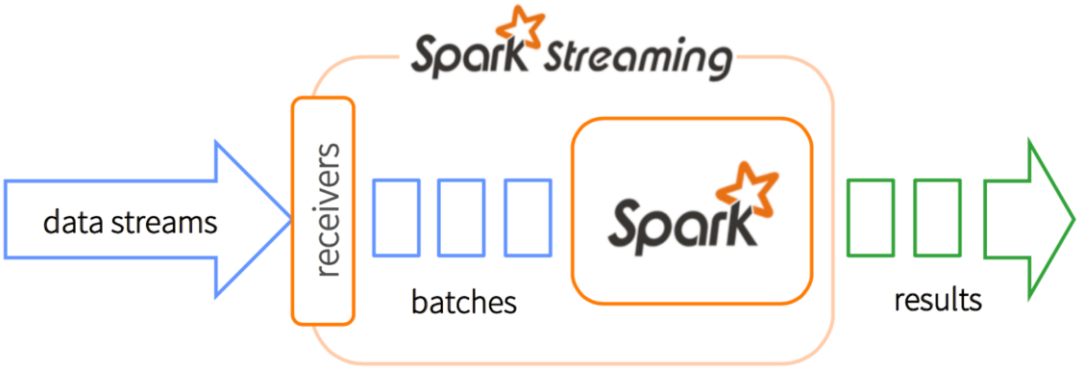


Ilustración 23: Funcionamiento Spark Streaming

Durante este proyecto se ha usado Spark Streaming para generar agregados temporales con los datos provenientes de Kafka.
Spark Streaming posee integración con Kafka desde sus primeras versiones y desde ese momento han ido mejorando y perfeccionando su integración para facilitar su uso por parte de los desarrolladores.

9.3 Scripts

9.3.1 Script de fuente PHP y Web

A continuación mostramos el script que se usa como libreria para generar las trazas.

```php
<?php
/*
@Author Ferran Fernández Garrido
Libreria de generación de trazas JSON random


*/

//Función que genera numeros random de forma custom
function customRand($prob){
$random = array();
        foreach($prob as $key => $value) {
                for($i = 0; $i < $value; $i++) {
                        $random[] = $key;
                }
        }
    shuffle($random);
return $random[0];
}

//Función de generación de cabeceras aleatorias con porcentaje de error
function rHeaders($nc,$id){
    if($nc == 1){
       $hd = array("timestamp"=>date('d-m-Y H:i:s'),"host"=>$id);
    }else{
       $hd = array("timestamp"=>date('d-m-Y H:i:s', strtotime('-20 minutes')),"host"=>"");
    }
    return $hd;
}

function carModels(){
        $crs = array("Ford Mustang","Ford Cuga","Audi A3","Audi A4","Renault Scenic","Renault S
Space","Citroen C3","Kia Picanto","Kia Rio","Volkswagen Golf","Volkswagen Polo","Volkswagen Pas
        return $crs[array_rand($crs)];
}

function doTrace(){
        $id = "gtu009".rand (1000,1020);
        $run = rand(0,1);
        if($run == 0){
                $runstatus = "running";
                $vel = rand(1,120);
        }else{
                $vel = 0;
                $runstatus =  "stop";
        }
        $long = "41.".rand(338389,488865);
        $lat = "2.".rand(121079,253452);


        $probabilitiesEng = array(1 => 95,2 => 5);
        $probabilitiesOil = array(1 => 90,2 => 10);
        $probabilitiesBattery = array(1 => 90,2 => 10);
```

```
        $probabilitiesBreaks = array(1 => 85,2 => 15);
        $probabilitiesLUW = array(1 => 70,2 => 30);
        $probabilitiesRUW = array(1 => 70,2 => 30);
        $probabilitiesLDW = array(1 => 70,2 => 30);
        $probabilitiesRDW = array(1 => 70,2 => 30);

        $prEng = customRand($probabilitiesEng);
        $prOil = customRand($probabilitiesOil);
        $prBat = customRand($probabilitiesBattery);
        $prBrk = customRand($probabilitiesBreaks);
        $prLUW = customRand($probabilitiesLUW);
        $prRUW = customRand($probabilitiesRUW);
        $prLDW = customRand($probabilitiesLDW);
        $prRDW = customRand($probabilitiesRDW);

        if($prEng == 1){
           $engine = "ok";
        }else{
           $engine = "ko";
        }

        if($prOil == 1){
           $oil = "ok";
        }else{
           $oil = "ko";
        }
        if($prBat == 1){
           $batt = "ok";
        }else{
           $batt = "ko";
        }
        if($prBrk == 1){
           $break = "ok";
        }else{
           $break = "ko";
        }
        if($prLUW == 1){
           $luw = "ok";
        }else{
           $luw = "ko";
        }
        if($prRUW == 1){
           $ruw = "ok";
        }else{
           $ruw = "ko";
        }
        if($prLDW == 1){
           $ldw = "ok";
        }else{
           $ldw = "ko";
        }
        if($prRDW == 1){
           $rdw = "ok";
        }else{
           $rdw = "ko";
        }

        $gas = rand(0,100);
        if($gas == 0){
           $vel = 0;
           $runstatus =  "stop";
```

```
        }


        $press = array("upper_left"=>$luw, "upper_right"=>$ruw,"down_left"=>$ldw,"down_right"=>
        $tech =
array("engine_status"=>$engine,"gasoline_level"=>$gas,"battery_status"=>$batt,"oil_status"=>$oi
        "preasure_level"=>$press);
        $arcor = array("long"=>$long,"lat"=>$lat);

        $arrJs =
array("id"=>$id,"model"=>carModels(),"status"=>$runstatus,"speed"=>$vel,"coordinates"=>$arcor,"
        $data_string =  json_encode($arrJs);

                $probabilitiesHead = array(1 => 98,2 => 2);

        $prH = customRand($probabilitiesHead);
        $headers = rHeaders($prH,$id);

        //retorno de la traza JSON en formato Flume
        return "[".json_encode(array("headers"=>$headers,"body"=>$data_string))."]";

}
```

Aquí se puede ver el script main para el envio de las trazas:

```
<?php

//importamos la libreria de generación de trazas
include 'libtrace.php';

$ch = curl_init('http://192.168.1.211:44444/');
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "POST");

$i = 0;
while(true){
        $data_string = doTrace();
        echo $data_string;
        curl_setopt($ch, CURLOPT_POSTFIELDS, $data_string);
        curl_setopt($ch, CURLOPT_HTTPHEADER, array(
                'Content-Type: application/json',
                'Content-Length: ' . strlen($data_string)));
        curl_exec($ch);

        $i++;
        usleep(250000);
echo $i;

}
```

Aquí se expone el código del servidor web socket que se encuentra a la escucha de las trazas procesadas de Spark:

```
<?php

/*
    Based on PHP WebSocket Server 0.2
```

```
     - http://code.google.com/p/php-websocket-server/
     - http://code.google.com/p/php-websocket-server/wiki/Scripting

    WebSocket Protocol 07
     - http://tools.ietf.org/html/draft-ietf-hybi-
thewebsocketprotocol-07
     - Supported by Firefox 6 (30/08/2011)

    Whilst a big effort is made to follow the protocol documentation,
the current script version may unknowingly differ.
    Please report any bugs you may find, all feedback and questions
are welcome!
*/


class PHPWebSocket
{
    // maximum amount of clients that can be connected at one time
    const WS_MAX_CLIENTS = 100;

    // maximum amount of clients that can be connected at one time on
the same IP v4 address
    const WS_MAX_CLIENTS_PER_IP = 15;

    // amount of seconds a client has to send data to the server,
before a ping request is sent to the client,
    // if the client has not completed the opening handshake, the ping
request is skipped and the client connection is closed
    const WS_TIMEOUT_RECV = 10;

    // amount of seconds a client has to reply to a ping request,
before the client connection is closed
    const WS_TIMEOUT_PONG = 5;

    // the maximum length, in bytes, of a frame's payload data (a
message consists of 1 or more frames), this is also internally limited
to 2,147,479,538
    const WS_MAX_FRAME_PAYLOAD_RECV = 100000;

    // the maximum length, in bytes, of a message's payload data, this
is also internally limited to 2,147,483,647
    const WS_MAX_MESSAGE_PAYLOAD_RECV = 500000;




    // internal
    const WS_FIN =  128;
    const WS_MASK = 128;

    const WS_OPCODE_CONTINUATION = 0;
    const WS_OPCODE_TEXT =         1;
    const WS_OPCODE_BINARY =       2;
    const WS_OPCODE_CLOSE =        8;
    const WS_OPCODE_PING =         9;
    const WS_OPCODE_PONG =         10;

    const WS_PAYLOAD_LENGTH_16 = 126;
    const WS_PAYLOAD_LENGTH_63 = 127;

    const WS_READY_STATE_CONNECTING = 0;
```

```
    const WS_READY_STATE_OPEN =       1;
    const WS_READY_STATE_CLOSING =    2;
    const WS_READY_STATE_CLOSED =     3;

    const WS_STATUS_NORMAL_CLOSE =              1000;
    const WS_STATUS_GONE_AWAY =                 1001;
    const WS_STATUS_PROTOCOL_ERROR =            1002;
    const WS_STATUS_UNSUPPORTED_MESSAGE_TYPE = 1003;
    const WS_STATUS_MESSAGE_TOO_BIG =           1004;

    const WS_STATUS_TIMEOUT = 3000;

    // global vars
    public $wsClients       = array();
    public $wsRead          = array();
    public $wsClientCount   = 0;
    public $wsClientIPCount = array();
    public $wsOnEvents      = array();

    /*
        $this->wsClients[ integer ClientID ] = array(
            0 => resource  Socket,                              //
client socket
            1 => string    MessageBuffer,                       // a
blank string when there's no incoming frames
            2 => integer   ReadyState,                          //
between 0 and 3
            3 => integer   LastRecvTime,                        // set
to time() when the client is added
            4 => int/false PingSentTime,                        // false
when the server is not waiting for a pong
            5 => int/false CloseStatus,                         // close
status that wsOnClose() will be called with
            6 => integer   IPv4,                                //
client's IP stored as a signed long, retrieved from ip2long()
            7 => int/false FramePayloadDataLength,              //
length of a frame's payload data, reset to false when all frame data
has been read (cannot reset to 0, to allow reading of mask key)
            8 => integer   FrameBytesRead,                      //
amount of bytes read for a frame, reset to 0 when all frame data has
been read
            9 => string    FrameBuffer,                         //
joined onto end as a frame's data comes in, reset to blank string when
all frame data has been read
            10 => integer  MessageOpcode,                       //
stored by the first frame for fragmented messages, default value is 0
            11 => integer  MessageBufferLength                  // the
payload data length of MessageBuffer
        )

        $wsRead[ integer ClientID ] = resource Socket           // this
one-dimensional array is used for socket_select()
                                                                //
$wsRead[ 0 ] is the socket listening for incoming client connections

        $wsClientCount = integer ClientCount                    //
amount of clients currently connected

        $wsClientIPCount[ integer IP ] = integer ClientCount  //
amount of clients connected per IP v4 address
    */
```

```
    // server state functions
    function wsStartServer($host, $port) {
        if (isset($this->wsRead[0])) return false;

        if (!$this->wsRead[0] = socket_create(AF_INET, SOCK_STREAM,
SOL_TCP)) {
            return false;
        }
        if (!socket_set_option($this->wsRead[0], SOL_SOCKET,
SO_REUSEADDR, 1)) {
            socket_close($this->wsRead[0]);
            return false;
        }
        if (!socket_bind($this->wsRead[0], $host, $port)) {
            socket_close($this->wsRead[0]);
            return false;
        }
        if (!socket_listen($this->wsRead[0], 10)) {
            socket_close($this->wsRead[0]);
            return false;
        }

        $write = array();
        $except = array();

        $nextPingCheck = time() + 1;
        while (isset($this->wsRead[0])) {
            $changed = $this->wsRead;
            $result = socket_select($changed, $write, $except, 1);

            if ($result === false) {
                socket_close($this->wsRead[0]);
                return false;
            }
            elseif ($result > 0) {
                foreach ($changed as $clientID => $socket) {
                    if ($clientID != 0) {
                        // client socket changed
                        $buffer = '';
                        $bytes = @socket_recv($socket, $buffer, 4096,
0);

                        if ($bytes === false) {
                            // error on recv, remove client socket
(will check to send close frame)
                            $this->wsSendClientClose($clientID,
self::WS_STATUS_PROTOCOL_ERROR);
                        }
                        elseif ($bytes > 0) {
                            // process handshake or frame(s)
                            if (!$this->wsProcessClient($clientID,
$buffer, $bytes)) {
                                $this->wsSendClientClose($clientID,
self::WS_STATUS_PROTOCOL_ERROR);
                            }
                        }
                        else {
                            // 0 bytes received from client, meaning
the client closed the TCP connection
                            $this->wsRemoveClient($clientID);
```

```
                            }
                        }
                        else {
                            // listen socket changed
                            $client = socket_accept($this->wsRead[0]);
                            if ($client !== false) {
                                // fetch client IP as integer
                                $clientIP = '';
                                $result = socket_getpeername($client,
$clientIP);
                                $clientIP = ip2long($clientIP);

                                if ($result !== false && $this-
>wsClientCount < self::WS_MAX_CLIENTS && (!isset($this-
>wsClientIPCount[$clientIP]) || $this->wsClientIPCount[$clientIP] <
self::WS_MAX_CLIENTS_PER_IP)) {
                                    $this->wsAddClient($client,
$clientIP);
                                }
                                else {
                                    socket_close($client);
                                }
                            }
                        }
                    }
                }

                if (time() >= $nextPingCheck) {
                    $this->wsCheckIdleClients();
                    $nextPingCheck = time() + 1;
                }
            }

            return true; // returned when wsStopServer() is called
        }
        function wsStopServer() {
            // check if server is not running
            if (!isset($this->wsRead[0])) return false;

            // close all client connections
            foreach ($this->wsClients as $clientID => $client) {
                // if the client's opening handshake is complete, tell the
client the server is 'going away'
                if ($client[2] != self::WS_READY_STATE_CONNECTING) {
                    $this->wsSendClientClose($clientID,
self::WS_STATUS_GONE_AWAY);
                }
                socket_close($client[0]);
            }

            // close the socket which listens for incoming clients
            socket_close($this->wsRead[0]);

            // reset variables
            $this->wsRead          = array();
            $this->wsClients       = array();
            $this->wsClientCount   = 0;
            $this->wsClientIPCount = array();

            return true;
        }
```

```
    // client timeout functions
    function wsCheckIdleClients() {
        $time = time();
        foreach ($this->wsClients as $clientID => $client) {
            if ($client[2] != self::WS_READY_STATE_CLOSED) {
                // client ready state is not closed
                if ($client[4] !== false) {
                    // ping request has already been sent to client,
pending a pong reply
                    if ($time >= $client[4] + self::WS_TIMEOUT_PONG) {
                        // client didn't respond to the server's ping
request in self::WS_TIMEOUT_PONG seconds
                        $this->wsSendClientClose($clientID,
self::WS_STATUS_TIMEOUT);
                        $this->wsRemoveClient($clientID);
                    }
                }
                elseif ($time >= $client[3] + self::WS_TIMEOUT_RECV) {
                    // last data was received >= self::WS_TIMEOUT_RECV
seconds ago
                    if ($client[2] != self::WS_READY_STATE_CONNECTING)
{
                        // client ready state is open or closing
                        $this->wsClients[$clientID][4] = time();
                        $this->wsSendClientMessage($clientID,
self::WS_OPCODE_PING, '');
                    }
                    else {
                        // client ready state is connecting
                        $this->wsRemoveClient($clientID);
                    }
                }
            }
        }
    }

    // client existence functions
    function wsAddClient($socket, $clientIP) {
        // increase amount of clients connected
        $this->wsClientCount++;

        // increase amount of clients connected on this client's IP
        if (isset($this->wsClientIPCount[$clientIP])) {
            $this->wsClientIPCount[$clientIP]++;
        }
        else {
            $this->wsClientIPCount[$clientIP] = 1;
        }

        // fetch next client ID
        $clientID = $this->wsGetNextClientID();

        // store initial client data
        $this->wsClients[$clientID] = array($socket, '',
self::WS_READY_STATE_CONNECTING, time(), false, 0, $clientIP, false,
0, '', 0, 0);

        // store socket - used for socket_select()
        $this->wsRead[$clientID] = $socket;
    }
```

```
    function wsRemoveClient($clientID) {
        // fetch close status (which could be false), and call
wsOnClose
        $closeStatus = $this->wsClients[$clientID][5];
        if ( array_key_exists('close', $this->wsOnEvents) )
            foreach ( $this->wsOnEvents['close'] as $func )
                $func($clientID, $closeStatus);

        // close socket
        $socket = $this->wsClients[$clientID][0];
        socket_close($socket);

        // decrease amount of clients connected on this client's IP
        $clientIP = $this->wsClients[$clientID][6];
        if ($this->wsClientIPCount[$clientIP] > 1) {
            $this->wsClientIPCount[$clientIP]--;
        }
        else {
            unset($this->wsClientIPCount[$clientIP]);
        }

        // decrease amount of clients connected
        $this->wsClientCount--;

        // remove socket and client data from arrays
        unset($this->wsRead[$clientID], $this->wsClients[$clientID]);
    }

    // client data functions
    function wsGetNextClientID() {
        $i = 1; // starts at 1 because 0 is the listen socket
        while (isset($this->wsRead[$i])) $i++;
        return $i;
    }
    function wsGetClientSocket($clientID) {
        return $this->wsClients[$clientID][0];
    }

    // client read functions
    function wsProcessClient($clientID, &$buffer, $bufferLength) {
        if ($this->wsClients[$clientID][2] ==
self::WS_READY_STATE_OPEN) {
            // handshake completed
            $result = $this->wsBuildClientFrame($clientID, $buffer,
$bufferLength);
        }
        elseif ($this->wsClients[$clientID][2] ==
self::WS_READY_STATE_CONNECTING) {
            // handshake not completed
            $result = $this->wsProcessClientHandshake($clientID,
$buffer);
            if ($result) {
                $this->wsClients[$clientID][2] =
self::WS_READY_STATE_OPEN;

                if ( array_key_exists('open', $this->wsOnEvents) )
                    foreach ( $this->wsOnEvents['open'] as $func )
                        $func($clientID);
            }
        }
        else {
```

```
            // ready state is set to closed
            $result = false;
        }

        return $result;
    }
    function wsBuildClientFrame($clientID, &$buffer, $bufferLength) {
        // increase number of bytes read for the frame, and join
buffer onto end of the frame buffer
        $this->wsClients[$clientID][8] += $bufferLength;
        $this->wsClients[$clientID][9] .= $buffer;

        // check if the length of the frame's payload data has been
fetched, if not then attempt to fetch it from the frame buffer
        if ($this->wsClients[$clientID][7] !== false || $this-
>wsCheckSizeClientFrame($clientID) == true) {
            // work out the header length of the frame
            $headerLength = ($this->wsClients[$clientID][7] <= 125 ? 0
: ($this->wsClients[$clientID][7] <= 65535 ? 2 : 8)) + 6;

            // check if all bytes have been received for the frame
            $frameLength = $this->wsClients[$clientID][7] +
$headerLength;
            if ($this->wsClients[$clientID][8] >= $frameLength) {
                // check if too many bytes have been read for the
frame (they are part of the next frame)
                $nextFrameBytesLength = $this->wsClients[$clientID][8]
- $frameLength;
                if ($nextFrameBytesLength > 0) {
                    $this->wsClients[$clientID][8] -=
$nextFrameBytesLength;
                    $nextFrameBytes = substr($this-
>wsClients[$clientID][9], $frameLength);
                    $this->wsClients[$clientID][9] = substr($this-
>wsClients[$clientID][9], 0, $frameLength);
                }

                // process the frame
                $result = $this->wsProcessClientFrame($clientID);

                // check if the client wasn't removed, then reset
frame data
                if (isset($this->wsClients[$clientID])) {
                    $this->wsClients[$clientID][7] = false;
                    $this->wsClients[$clientID][8] = 0;
                    $this->wsClients[$clientID][9] = '';
                }

                // if there's no extra bytes for the next frame, or
processing the frame failed, return the result of processing the frame
                if ($nextFrameBytesLength <= 0 || !$result) return
$result;

                // build the next frame with the extra bytes
                return $this->wsBuildClientFrame($clientID,
$nextFrameBytes, $nextFrameBytesLength);
            }
        }

        return true;
    }
```

```
    function wsCheckSizeClientFrame($clientID) {
        // check if at least 2 bytes have been stored in the frame
buffer
        if ($this->wsClients[$clientID][8] > 1) {
            // fetch payload length in byte 2, max will be 127
            $payloadLength = ord(substr($this-
>wsClients[$clientID][9], 1, 1)) & 127;

            if ($payloadLength <= 125) {
                // actual payload length is <= 125
                $this->wsClients[$clientID][7] = $payloadLength;
            }
            elseif ($payloadLength == 126) {
                // actual payload length is <= 65,535
                if (substr($this->wsClients[$clientID][9], 3, 1) !==
false) {
                    // at least another 2 bytes are set
                    $payloadLengthExtended = substr($this-
>wsClients[$clientID][9], 2, 2);
                    $array = unpack('na', $payloadLengthExtended);
                    $this->wsClients[$clientID][7] = $array['a'];
                }
            }
            else {
                // actual payload length is > 65,535
                if (substr($this->wsClients[$clientID][9], 9, 1) !==
false) {
                    // at least another 8 bytes are set
                    $payloadLengthExtended = substr($this-
>wsClients[$clientID][9], 2, 8);

                    // check if the frame's payload data length
exceeds 2,147,483,647 (31 bits)
                    // the maximum integer in PHP is "usually" this
number. More info: http://php.net/manual/en/language.types.integer.php
                    $payloadLengthExtended32_1 =
substr($payloadLengthExtended, 0, 4);
                    $array = unpack('Na', $payloadLengthExtended32_1);
                    if ($array['a'] != 0 ||
ord(substr($payloadLengthExtended, 4, 1)) & 128) {
                        $this->wsSendClientClose($clientID,
self::WS_STATUS_MESSAGE_TOO_BIG);
                        return false;
                    }

                    // fetch length as 32 bit unsigned integer, not as
64 bit
                    $payloadLengthExtended32_2 =
substr($payloadLengthExtended, 4, 4);
                    $array = unpack('Na', $payloadLengthExtended32_2);

                    // check if the payload data length exceeds
2,147,479,538 (2,147,483,647 - 14 - 4095)
                    // 14 for header size, 4095 for last recv() next
frame bytes
                    if ($array['a'] > 2147479538) {
                        $this->wsSendClientClose($clientID,
self::WS_STATUS_MESSAGE_TOO_BIG);
                        return false;
                    }
```

```
                    // store frame payload data length
                    $this->wsClients[$clientID][7] = $array['a'];
                }
            }

            // check if the frame's payload data length has now been
stored
            if ($this->wsClients[$clientID][7] !== false) {

                // check if the frame's payload data length exceeds
self::WS_MAX_FRAME_PAYLOAD_RECV
                if ($this->wsClients[$clientID][7] >
self::WS_MAX_FRAME_PAYLOAD_RECV) {
                    $this->wsClients[$clientID][7] = false;
                    $this->wsSendClientClose($clientID,
self::WS_STATUS_MESSAGE_TOO_BIG);
                    return false;
                }

                // check if the message's payload data length exceeds
2,147,483,647 or self::WS_MAX_MESSAGE_PAYLOAD_RECV
                // doesn't apply for control frames, where the payload
data is not internally stored
                $controlFrame = (ord(substr($this-
>wsClients[$clientID][9], 0, 1)) & 8) == 8;
                if (!$controlFrame) {
                    $newMessagePayloadLength = $this-
>wsClients[$clientID][11] + $this->wsClients[$clientID][7];
                    if ($newMessagePayloadLength >
self::WS_MAX_MESSAGE_PAYLOAD_RECV || $newMessagePayloadLength >
2147483647) {
                        $this->wsSendClientClose($clientID,
self::WS_STATUS_MESSAGE_TOO_BIG);
                        return false;
                    }
                }

                return true;
            }
        }

        return false;
    }
    function wsProcessClientFrame($clientID) {
        // store the time that data was last received from the client
        $this->wsClients[$clientID][3] = time();

        // fetch frame buffer
        $buffer = &$this->wsClients[$clientID][9];

        // check at least 6 bytes are set (first 2 bytes and 4 bytes
for the mask key)
        if (substr($buffer, 5, 1) === false) return false;

        // fetch first 2 bytes of header
        $octet0 = ord(substr($buffer, 0, 1));
        $octet1 = ord(substr($buffer, 1, 1));

        $fin = $octet0 & self::WS_FIN;
        $opcode = $octet0 & 15;
```

```
        $mask = $octet1 & self::WS_MASK;
        if (!$mask) return false; // close socket, as no mask bit was
sent from the client

        // fetch byte position where the mask key starts
        $seek = $this->wsClients[$clientID][7] <= 125 ? 2 : ($this-
>wsClients[$clientID][7] <= 65535 ? 4 : 10);

        // read mask key
        $maskKey = substr($buffer, $seek, 4);

        $array = unpack('Na', $maskKey);
        $maskKey = $array['a'];
        $maskKey = array(
            $maskKey >> 24,
            ($maskKey >> 16) & 255,
            ($maskKey >> 8) & 255,
            $maskKey & 255
        );
        $seek += 4;

        // decode payload data
        if (substr($buffer, $seek, 1) !== false) {
            $data = str_split(substr($buffer, $seek));
            foreach ($data as $key => $byte) {
                $data[$key] = chr(ord($byte) ^ ($maskKey[$key % 4]));
            }
            $data = implode('', $data);
        }
        else {
            $data = '';
        }

        // check if this is not a continuation frame and if there is
already data in the message buffer
        if ($opcode != self::WS_OPCODE_CONTINUATION && $this-
>wsClients[$clientID][11] > 0) {
            // clear the message buffer
            $this->wsClients[$clientID][11] = 0;
            $this->wsClients[$clientID][1] = '';
        }

        // check if the frame is marked as the final frame in the
message
        if ($fin == self::WS_FIN) {
            // check if this is the first frame in the message
            if ($opcode != self::WS_OPCODE_CONTINUATION) {
                // process the message
                return $this->wsProcessClientMessage($clientID,
$opcode, $data, $this->wsClients[$clientID][7]);
            }
            else {
                // increase message payload data length
                $this->wsClients[$clientID][11] += $this-
>wsClients[$clientID][7];

                // push frame payload data onto message buffer
                $this->wsClients[$clientID][1] .= $data;

                // process the message
                $result = $this->wsProcessClientMessage($clientID,
```

```
$this->wsClients[$clientID][10], $this->wsClients[$clientID][1],
$this->wsClients[$clientID][11]);

                // check if the client wasn't removed, then reset
message buffer and message opcode
                if (isset($this->wsClients[$clientID])) {
                    $this->wsClients[$clientID][1] = '';
                    $this->wsClients[$clientID][10] = 0;
                    $this->wsClients[$clientID][11] = 0;
                }

                return $result;
            }
        }
        else {
            // check if the frame is a control frame, control frames
cannot be fragmented
            if ($opcode & 8) return false;

            // increase message payload data length
            $this->wsClients[$clientID][11] += $this-
>wsClients[$clientID][7];

            // push frame payload data onto message buffer
            $this->wsClients[$clientID][1] .= $data;

            // if this is the first frame in the message, store the
opcode
            if ($opcode != self::WS_OPCODE_CONTINUATION) {
                $this->wsClients[$clientID][10] = $opcode;
            }
        }

        return true;
    }
    function wsProcessClientMessage($clientID, $opcode, &$data,
$dataLength) {
        // check opcodes
        if ($opcode == self::WS_OPCODE_PING) {
            // received ping message
            return $this->wsSendClientMessage($clientID,
self::WS_OPCODE_PONG, $data);
        }
        elseif ($opcode == self::WS_OPCODE_PONG) {
            // received pong message (it's valid if the server did not
send a ping request for this pong message)
            if ($this->wsClients[$clientID][4] !== false) {
                $this->wsClients[$clientID][4] = false;
            }
        }
        elseif ($opcode == self::WS_OPCODE_CLOSE) {
            // received close message
            if (substr($data, 1, 1) !== false) {
                $array = unpack('na', substr($data, 0, 2));
                $status = $array['a'];
            }
            else {
                $status = false;
            }

            if ($this->wsClients[$clientID][2] ==
```

```
self::WS_READY_STATE_CLOSING) {
                // the server already sent a close frame to the
client, this is the client's close frame reply
                // (no need to send another close frame to the client)
                $this->wsClients[$clientID][2] =
self::WS_READY_STATE_CLOSED;
            }
            else {
                // the server has not already sent a close frame to
the client, send one now
                $this->wsSendClientClose($clientID,
self::WS_STATUS_NORMAL_CLOSE);
            }

            $this->wsRemoveClient($clientID);
        }
        elseif ($opcode == self::WS_OPCODE_TEXT || $opcode ==
self::WS_OPCODE_BINARY) {
            if ( array_key_exists('message', $this->wsOnEvents) )
                foreach ( $this->wsOnEvents['message'] as $func )
                    $func($clientID, $data, $dataLength, $opcode ==
self::WS_OPCODE_BINARY);
        }
        else {
            // unknown opcode
            return false;
        }

        return true;
    }
    function wsProcessClientHandshake($clientID, &$buffer) {
        // fetch headers and request line
        $sep = strpos($buffer, "\r\n\r\n");
        if (!$sep) return false;

        $headers = explode("\r\n", substr($buffer, 0, $sep));
        $headersCount = sizeof($headers); // includes request line
        if ($headersCount < 1) return false;

        // fetch request and check it has at least 3 parts (space
tokens)
        $request = &$headers[0];
        $requestParts = explode(' ', $request);
        $requestPartsSize = sizeof($requestParts);
        if ($requestPartsSize < 3) return false;

        // check request method is GET
        if (strtoupper($requestParts[0]) != 'GET') return false;

        // check request HTTP version is at least 1.1
        $httpPart = &$requestParts[$requestPartsSize - 1];
        $httpParts = explode('/', $httpPart);
        if (!isset($httpParts[1]) || (float) $httpParts[1] < 1.1)
return false;

        // store headers into a keyed array: array[headerKey] =
headerValue
        $headersKeyed = array();
        for ($i=1; $i<$headersCount; $i++) {
            $parts = explode(':', $headers[$i]);
            if (!isset($parts[1])) return false;
```

```
            $headersKeyed[trim($parts[0])] = trim($parts[1]);
        }

        // check Host header was received
        if (!isset($headersKeyed['Host'])) return false;

        // check Sec-WebSocket-Key header was received and decoded
value length is 16
        if (!isset($headersKeyed['Sec-WebSocket-Key'])) return false;
        $key = $headersKeyed['Sec-WebSocket-Key'];
        if (strlen(base64_decode($key)) != 16) return false;

        // check Sec-WebSocket-Version header was received and value
is 7
        if (!isset($headersKeyed['Sec-WebSocket-Version']) || (int)
$headersKeyed['Sec-WebSocket-Version'] < 7) return false; // should
really be != 7, but Firefox 7 beta users send 8

        // work out hash to use in Sec-WebSocket-Accept reply header
        $hash = base64_encode(sha1($key.'258EAFA5-E914-47DA-95CA-
C5AB0DC85B11', true));

        // build headers
        $headers = array(
            'HTTP/1.1 101 Switching Protocols',
            'Upgrade: websocket',
            'Connection: Upgrade',
            'Sec-WebSocket-Accept: '.$hash
        );
        $headers = implode("\r\n", $headers)."\r\n\r\n";

        // send headers back to client
        $socket = $this->wsClients[$clientID][0];

        $left = strlen($headers);
        do {
            $sent = @socket_send($socket, $headers, $left, 0);
            if ($sent === false) return false;

            $left -= $sent;
            if ($sent > 0) $headers = substr($headers, $sent);
        }
        while ($left > 0);

        return true;
    }

    // client write functions
    function wsSendClientMessage($clientID, $opcode, $message) {
        // check if client ready state is already closing or closed
        if ($this->wsClients[$clientID][2] ==
self::WS_READY_STATE_CLOSING || $this->wsClients[$clientID][2] ==
self::WS_READY_STATE_CLOSED) return true;

        // fetch message length
        $messageLength = strlen($message);

        // set max payload length per frame
        $bufferSize = 4096;
```

```
        // work out amount of frames to send, based on $bufferSize
        $frameCount = ceil($messageLength / $bufferSize);
        if ($frameCount == 0) $frameCount = 1;

        // set last frame variables
        $maxFrame = $frameCount - 1;
        $lastFrameBufferLength = ($messageLength % $bufferSize) != 0 ?
($messageLength % $bufferSize) : ($messageLength != 0 ? $bufferSize :
0);

        // loop around all frames to send
        for ($i=0; $i<$frameCount; $i++) {
            // fetch fin, opcode and buffer length for frame
            $fin = $i != $maxFrame ? 0 : self::WS_FIN;
            $opcode = $i != 0 ? self::WS_OPCODE_CONTINUATION :
$opcode;

            $bufferLength = $i != $maxFrame ? $bufferSize :
$lastFrameBufferLength;

            // set payload length variables for frame
            if ($bufferLength <= 125) {
                $payloadLength = $bufferLength;
                $payloadLengthExtended = '';
                $payloadLengthExtendedLength = 0;
            }
            elseif ($bufferLength <= 65535) {
                $payloadLength = self::WS_PAYLOAD_LENGTH_16;
                $payloadLengthExtended = pack('n', $bufferLength);
                $payloadLengthExtendedLength = 2;
            }
            else {
                $payloadLength = self::WS_PAYLOAD_LENGTH_63;
                $payloadLengthExtended = pack('xxxxN', $bufferLength);
// pack 32 bit int, should really be 64 bit int
                $payloadLengthExtendedLength = 8;
            }

            // set frame bytes
            $buffer = pack('n', (($fin | $opcode) << 8) |
$payloadLength) . $payloadLengthExtended . substr($message,
$i*$bufferSize, $bufferLength);

            // send frame
            $socket = $this->wsClients[$clientID][0];

            $left = 2 + $payloadLengthExtendedLength + $bufferLength;
            do {
                $sent = @socket_send($socket, $buffer, $left, 0);
                if ($sent === false) return false;

                $left -= $sent;
                if ($sent > 0) $buffer = substr($buffer, $sent);
            }
            while ($left > 0);
        }

        return true;
    }
    function wsSendClientClose($clientID, $status=false) {
        // check if client ready state is already closing or closed
```

```
        if ($this->wsClients[$clientID][2] ==
self::WS_READY_STATE_CLOSING || $this->wsClients[$clientID][2] ==
self::WS_READY_STATE_CLOSED) return true;

        // store close status
        $this->wsClients[$clientID][5] = $status;

        // send close frame to client
        $status = $status !== false ? pack('n', $status) : '';
        $this->wsSendClientMessage($clientID, self::WS_OPCODE_CLOSE,
$status);

        // set client ready state to closing
        $this->wsClients[$clientID][2] = self::WS_READY_STATE_CLOSING;
    }

    // client non-internal functions
    function wsClose($clientID) {
        return $this->wsSendClientClose($clientID,
self::WS_STATUS_NORMAL_CLOSE);
    }
    function wsSend($clientID, $message, $binary=false) {
        return $this->wsSendClientMessage($clientID, $binary ?
self::WS_OPCODE_BINARY : self::WS_OPCODE_TEXT, $message);
    }

    function log( $message )
    {
        echo date('Y-m-d H:i:s: ') . $message . "\n";
    }

    function bind( $type, $func )
    {
        if ( !isset($this->wsOnEvents[$type]) )
            $this->wsOnEvents[$type] = array();
        $this->wsOnEvents[$type][] = $func;
    }

    function unbind( $type='' )
    {
        if ( $type ) unset($this->wsOnEvents[$type]);
        else $this->wsOnEvents = array();
    }
}
?>



<?php
// prevent the server from timing out
set_time_limit(0);

// include the web sockets server script (the server is started at the
far bottom of this file)
require 'class.PHPWebSocket.php';

// when a client sends data to the server
function wsOnMessage($clientID, $message, $messageLength, $binary) {
    global $Server;
    $ip = long2ip( $Server->wsClients[$clientID][6] );
```

```
    // check if message length is 0
    if ($messageLength == 0) {
        $Server->wsClose($clientID);
        return;
    }

    //The speaker is the only person in the room. Don't let them feel
lonely.

        foreach ( $Server->wsClients as $id => $client )
                $Server->wsSend($id,$message);
}

// when a client connects
function wsOnOpen($clientID)
{
    global $Server;
    $ip = long2ip( $Server->wsClients[$clientID][6] );

    $Server->log( "$ip ($clientID) has connected." );

    //Send a join notice to everyone but the person who joined
    foreach ( $Server->wsClients as $id => $client )
        if ( $id != $clientID ){}
            //$Server->wsSend($id, "Visitor $clientID ($ip) has joined
the room.");
}

// when a client closes or lost connection
function wsOnClose($clientID, $status) {
    global $Server;
    $ip = long2ip( $Server->wsClients[$clientID][6] );

    $Server->log( "$ip ($clientID) has disconnected." );

    //Send a user left notice to everyone in the room
    foreach ( $Server->wsClients as $id => $client ){}
        //$Server->wsSend($id, "Visitor $clientID ($ip) has left the
room.");
}

// start the server
$Server = new PHPWebSocket();
$Server->bind('message', 'wsOnMessage');
$Server->bind('open', 'wsOnOpen');
$Server->bind('close', 'wsOnClose');
// for other computers to connect, you will probably need to change
this to your LAN IP or external IP,
// alternatively use:
gethostbyaddr(gethostbyname($_SERVER['SERVER_NAME']))
$Server->wsStartServer('192.168.1.201', 9300);

?>
```

Libreria JS para la recepción de las trazas en la web:

```
var FancyWebSocket = function(url)
{
    var callbacks = {};
    var ws_url = url;
```

```
    var conn;

    this.bind = function(event_name, callback){
        callbacks[event_name] = callbacks[event_name] || [];
        callbacks[event_name].push(callback);
        return this;// chainable
    };

    this.send = function(event_name, event_data){
        this.conn.send( event_data );
        return this;
    };

    this.connect = function() {
        if ( typeof(MozWebSocket) == 'function' )
            this.conn = new MozWebSocket(url);
        else
            this.conn = new WebSocket(url);

        // dispatch to the right handlers
        this.conn.onmessage = function(evt){
            dispatch('message', evt.data);
        };

        this.conn.onclose = function(){dispatch('close',null)}
        this.conn.onopen = function(){dispatch('open',null)}
    };

    this.disconnect = function() {
        this.conn.close();
    };

    var dispatch = function(event_name, message){
        var chain = callbacks[event_name];
        if(typeof chain == 'undefined') return; // no callbacks for
this event
        for(var i = 0; i < chain.length; i++){
            chain[i]( message )
        }
    }
};
```

Web en código HTML

```
<!doctype html>
<html>
<head>
    <meta charset='UTF-8' />
    <style>
        input, textarea {border:1px solid #CCC;margin:0px;padding:0px}

        #body {max-width:800px;margin:auto}
        #log {width:100%;height:400px}
        #message {width:100%;line-height:20px}
    </style>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
  <script src="https://code.highcharts.com/stock/highstock.js"></script>
  <script src="fancywebsocket.js"></script>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.mi
```

```
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
  <script>


      function initMap() {
        var myLatLng = {lat: 41.489735, lng: 2.1756511};
        var map = new google.maps.Map(document.getElementById('map'), {
          zoom: 10,
          center: myLatLng
        });

        google.maps.event.addListener(map, 'click', function(event) {
          addMarker(event.latLng, map);
        });
      }

      function addMarker(location, map) {
        // Add the marker at the clicked location, and add the next-available label
        // from the array of alphabetical characters.
        var marker = new google.maps.Marker({
          position: location,
          label: '',
          map: map
        });
      }
    </script>
    <script>
        var Server;
        var Graph;
        var LatLong = [];

        //Helper functions
        /*function log( text ) {
            $log = $('#log');
            //Add text to log
            $log.append(($log.val()?"\n":'')+text);
            //Autoscroll
            $log[0].scrollTop = $log[0].scrollHeight - $log[0].clientHeight;
        }*/

        function findProp(obj, prop, defval){
            if (typeof defval == 'undefined') defval = null;
            prop = prop.split('.');
                for (var i = 0; i < prop.length; i++) {
                if(typeof obj[prop[i]] == 'undefined')
                    return defval;
                obj = obj[prop[i]];
            }
            return obj;
        }

        function parseResult(txt, chart){
            while(chart.series.length > 0){
                chart.series[0].remove(true);
            }
           //dummy = '{"timestamp":"dd/mm/YYYY HH:mm:ss", "graph":[{"name": "Ford", "data":[22]
"data":[12]}]}';
           obj = JSON.parse(txt);
           console.log(txt);

           //$("#acars").html(txt);
```

```
        $("#acars").html(obj.running_cars);
        $("#trendCars").html(obj.trend_car);
        $("#enginacars").html(obj.engine_t_cars);
        chart.setTitle(null, { text: obj.timestamp});

        console.log(obj.coordinates);
        for (var i = 0; i < obj.graph.length; i++) {
         chart.addSeries({
          name: obj.graph[i].name,
          data: [obj.graph[i].data]
          });
        }
        for (var i = 0; i < obj.coordinates.length; i++) {
            console.log(obj.coordinates[i].longitud);
            var lt = Number(obj.coordinates[i].lat);console.log(lt)
            var longt = Number(obj.coordinates[i].longitud);
             var myLatLng = new google.maps.LatLng(longt,lt);
             console.log(myLatLng);
             var marker = new google.maps.Marker({
                 position: myLatLng,
                 map: map,
                 title: 'Hello World!'
             });
             marker.setMap(map);
        }


    }

    function send( text ) {
        Server.send( 'message', text );
    }

    $(document).ready(function() {
        $('#message').hide()
        Graph = Highcharts.chart('container', {
        chart: {
            type: 'column'
        },
        title: {
            text: 'Number of cars per brand'
        },
        subtitle: {
            text: 'Last Update:'
        },
        xAxis: {
            categories: [
                'Cars'
            ],
            crosshair: true
        },
        yAxis: {
            min: 0,
            title: {
                text: 'Number of cars'
            }
        },
        credits: {
            enabled: false
        },
```

```
            series: [{
            }]
            });

        });

        Server = new FancyWebSocket('ws://192.168.1.201:9300');

            $('#message').keypress(function(e) {
                if ( e.keyCode == 13 && this.value ) {
                    //log( 'You: ' + this.value );
                    send( this.value );

                    $(this).val('');
                }
            });

            //Let the user know we're connected
            Server.bind('open', function() {
                //log( "Connected." );
            });

            //OH NOES! Disconnection occurred.
            Server.bind('close', function( data ) {
                //log( "Disconnected." );
            });

            //Log any messages sent from server
            Server.bind('message', function( payload ) {
                //log( payload );
                parseResult(payload, Graph);
                console.log(payload);
            });

            Server.connect();
    </script>
</head>

<body>
    <!--<div id='body'>
        <textarea id='log' name='log' readonly='readonly'></textarea><br/>
        <input type='text' id='message' name='message' />
    </div>-->
    <input type='text' id='message' name='message' />
    <nav class="navbar navbar-inverse">
        <div class="container-fluid">
            <div class="navbar-header">
                <button type="button" class="collapsed navbar-toggle" data-toggle="collapse" da
example-navbar-collapse-9" aria-expanded="false"> <span class="sr-only">Toggle navigation</span
bar"></span> <span class="icon-bar"></span> <span class="icon-bar"></span> </button> <a href="#
brand"><img src="sprk.png" width="27px" /></a> </div> <div class="collapse navbar-collapse" id=
collapse-9"> <ul class="nav navbar-nav"> <li class="active"><a href="#">Real Time Data Ingestio
</div> </div> </nav>
    <center>
      <div class="row" style="width:90%;">
        <div class="jumbotron">
          <div id="container" style="width:90%;"></div>
        </div>
      </div>
    </center>
```

```
    <center><div class="row" style="width:90%;">
        <div class="col-sm-4">
            <div class="panel panel-default">
              <div class="panel-heading"><b>Current running cars</b></div>
              <div id="acars" class="panel-body"></div>
            </div>
        </div>
        <div class="col-sm-4">
            <div class="panel panel-default">
              <div class="panel-heading"><b>Current engine troubles</b></div>
              <div id="enginacars" class="panel-body"></div>
            </div>
        </div>
        <div class="col-sm-4">
            <div class="panel panel-default">
              <div class="panel-heading"><b>Current most selected car model</b></div>
              <div id="trendCars" class="panel-body"></div>
            </div>
        </div>
         <div class="col-sm-12" id="map" style="height: 350px";></div>

    <script async defer
    src="https://maps.googleapis.com/maps/api/js?key=xxxxxxxxx&callback=initMap">
    </script>
    </div></center>

</body>

</html>
```

9.3.2 Scripts Hadoop en Docker

A continuación mostramos el Script de Dockerfile de nuestro clúster Hadoop

Dockerfile

```
# Instalación de Hadoop (HDFS + YARN)

FROM ubuntu:14.04
MAINTAINER Ferran Fernandez "ffernandez.upc@gmail.com"

WORKDIR /root

RUN apt-get update && apt-get install -y openssh-server openjdk-7-jdk
wget

# Versión 2.7.2
RUN wget https://github.com/kiwenlau/compile-
hadoop/releases/download/2.7.2/hadoop-2.7.2.tar.gz && \
    tar -xzvf hadoop-2.7.2.tar.gz && \
    mv hadoop-2.7.2 /usr/local/hadoop && \
    rm hadoop-2.7.2.tar.gz

ENV JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
ENV HADOOP_HOME=/usr/local/hadoop
ENV PATH=$PATH:/usr/local/hadoop/bin:/usr/local/hadoop/sbin
```

```
# SSH sin clave
RUN ssh-keygen -t rsa -f ~/.ssh/id_rsa -P '' && \
    cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

RUN mkdir -p ~/hdfs/namenode && \
    mkdir -p ~/hdfs/datanode && \
    mkdir $HADOOP_HOME/logs

COPY config/* /tmp/

RUN mv /tmp/ssh_config ~/.ssh/config && \
    mv /tmp/hadoop-env.sh /usr/local/hadoop/etc/hadoop/hadoop-env.sh
&& \
    mv /tmp/hdfs-site.xml $HADOOP_HOME/etc/hadoop/hdfs-site.xml && \
    mv /tmp/core-site.xml $HADOOP_HOME/etc/hadoop/core-site.xml && \
    mv /tmp/mapred-site.xml $HADOOP_HOME/etc/hadoop/mapred-site.xml &&
\
    mv /tmp/yarn-site.xml $HADOOP_HOME/etc/hadoop/yarn-site.xml && \
    mv /tmp/slaves $HADOOP_HOME/etc/hadoop/slaves && \
    mv /tmp/start-hadoop.sh ~/start-hadoop.sh

RUN chmod +x ~/start-hadoop.sh && \
    chmod +x $HADOOP_HOME/sbin/start-dfs.sh && \
    chmod +x $HADOOP_HOME/sbin/start-yarn.sh

# HDFS
EXPOSE 22 8020 8042 8088 14000 50070 50470

# MapReduce
EXPOSE 10020 13562 19888

# Formato al nodename
RUN /usr/local/hadoop/bin/hdfs namenode -format

CMD [ "sh", "-c", "service ssh start; bash"]
```

Los siguientes scripts se encuentran dentro de la carpeta /config dentro del directorio Hadoop de Docker creado para el proyecto

core-site.xml

```
<?xml version="1.0"?>
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://hadoop-master:9000/</value>
    </property>
</configuration>
```

hadoop-env.sh

```
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
```

```
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# Set Hadoop-specific environment variables here.

# The only required environment variable is JAVA_HOME.  All others are
# optional.  When running a distributed configuration it is best to
# set JAVA_HOME in this file, so that it is correctly defined on
# remote nodes.

# The java implementation to use.
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64

# The jsvc implementation to use. Jsvc is required to run secure
datanodes
# that bind to privileged ports to provide authentication of data
transfer
# protocol.  Jsvc is not required if SASL is configured for
authentication of
# data transfer protocol using non-privileged ports.
#export JSVC_HOME=${JSVC_HOME}

export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-"/etc/hadoop"}

# Extra Java CLASSPATH elements.  Automatically insert capacity-
scheduler.
for f in $HADOOP_HOME/contrib/capacity-scheduler/*.jar; do
  if [ "$HADOOP_CLASSPATH" ]; then
    export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$f
  else
    export HADOOP_CLASSPATH=$f
  fi
done

# The maximum amount of heap to use, in MB. Default is 1000.
#export HADOOP_HEAPSIZE=
#export HADOOP_NAMENODE_INIT_HEAPSIZE=""

# Extra Java runtime options.  Empty by default.
export HADOOP_OPTS="$HADOOP_OPTS -Djava.net.preferIPv4Stack=true"

# Command specific options appended to HADOOP_OPTS when specified
export HADOOP_NAMENODE_OPTS="-
Dhadoop.security.logger=${HADOOP_SECURITY_LOGGER:-INFO,RFAS} -
Dhdfs.audit.logger=${HDFS_AUDIT_LOGGER:-INFO,NullAppender}
$HADOOP_NAMENODE_OPTS"
export HADOOP_DATANODE_OPTS="-Dhadoop.security.logger=ERROR,RFAS
$HADOOP_DATANODE_OPTS"

export HADOOP_SECONDARYNAMENODE_OPTS="-
```

```
Dhadoop.security.logger=${HADOOP_SECURITY_LOGGER:-INFO,RFAS} -
Dhdfs.audit.logger=${HDFS_AUDIT_LOGGER:-INFO,NullAppender}
$HADOOP_SECONDARYNAMENODE_OPTS"

export HADOOP_NFS3_OPTS="$HADOOP_NFS3_OPTS"
export HADOOP_PORTMAP_OPTS="-Xmx512m $HADOOP_PORTMAP_OPTS"

# The following applies to multiple commands (fs, dfs, fsck, distcp
etc)
export HADOOP_CLIENT_OPTS="-Xmx512m $HADOOP_CLIENT_OPTS"
#HADOOP_JAVA_PLATFORM_OPTS="-XX:-UsePerfData
$HADOOP_JAVA_PLATFORM_OPTS"

# On secure datanodes, user to run the datanode as after dropping
privileges.
# This **MUST** be uncommented to enable secure HDFS if using
privileged ports
# to provide authentication of data transfer protocol.  This **MUST
NOT** be
# defined if SASL is configured for authentication of data transfer
protocol
# using non-privileged ports.
export HADOOP_SECURE_DN_USER=${HADOOP_SECURE_DN_USER}

# Where log files are stored.  $HADOOP_HOME/logs by default.
#export HADOOP_LOG_DIR=${HADOOP_LOG_DIR}/$USER

# Where log files are stored in the secure data environment.
export HADOOP_SECURE_DN_LOG_DIR=${HADOOP_LOG_DIR}/${HADOOP_HDFS_USER}

###
# HDFS Mover specific parameters
###
# Specify the JVM options to be used when starting the HDFS Mover.
# These options will be appended to the options specified as
HADOOP_OPTS
# and therefore may override any similar flags set in HADOOP_OPTS
#
# export HADOOP_MOVER_OPTS=""

###
# Advanced Users Only!
###

# The directory where pid files are stored. /tmp by default.
# NOTE: this should be set to a directory that can only be written to
by
#       the user that will run the hadoop daemons.  Otherwise there is
the
#       potential for a symlink attack.
export HADOOP_PID_DIR=${HADOOP_PID_DIR}
export HADOOP_SECURE_DN_PID_DIR=${HADOOP_PID_DIR}

# A string representing this instance of hadoop. $USER by default.
export HADOOP_IDENT_STRING=$USER
```

hdfs-site.xml

```
<?xml version="1.0"?>
<configuration>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:///root/hdfs/namenode</value>
        <description>NameNode directory for namespace and transaction
logs storage.</description>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:///root/hdfs/datanode</value>
        <description>DataNode directory</description>
    </property>
    <property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
</configuration>
```

mapred-site.xml

```
<?xml version="1.0"?>
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

slaves

```
hadoop-slave1
hadoop-slave2
```

ssh_config

```
Host localhost
  StrictHostKeyChecking no

Host 0.0.0.0
  StrictHostKeyChecking no

Host hadoop-*
   StrictHostKeyChecking no
   UserKnownHostsFile=/dev/null
```

yarn-site.xml

```
<?xml version="1.0"?>
<configuration>
```

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

start-hadoop.sh

```
#!/bin/bash

echo -e "\n"
$HADOOP_HOME/sbin/start-dfs.sh
echo -e "\n"
$HADOOP_HOME/sbin/start-yarn.sh
echo -e "\n"
```

9.3.3 Scripts Flume Colector

Script de Dockerfile:

```
FROM debian:jessie
MAINTAINER ffernandez3495

RUN apt-get update && apt-get install -q -y --no-install-recommends
wget

RUN mkdir /opt/java
RUN wget --no-check-certificate --header "Cookie:
oraclelicense=accept-securebackup-cookie" -qO- \
  http://download.oracle.com/otn-pub/java/jdk/8u101-b13/jdk-8u101-
linux-x64.tar.gz \
  | tar zxvf - -C /opt/java --strip 1

RUN mkdir /opt/flume
RUN wget -qO- http://archive.apache.org/dist/flume/1.7.0/apache-flume-
1.7.0-bin.tar.gz \
  | tar zxvf - -C /opt/flume --strip 1

COPY flume-colector.conf /opt/flume/conf/
COPY flume-custom-interceptor-tfm-0.0.1-SNAPSHOT.jar /opt/flume/lib/
COPY lombok-1.16.10.jar /opt/flume/lib

ENV JAVA_HOME /opt/java
ENV PATH /opt/flume/bin:/opt/java/bin:$PATH
```

```
EXPOSE 44444

ENTRYPOINT [ "flume-ng", "agent","-c", "/opt/flume/conf", "-f",
"/opt/flume/conf/flume-colector.conf", "-n", "agentec","-
Dflume.root.logger=INFO,console" ]
```

Programa Java de Interceptor Flume:

```
package com.flume.custom.interceptor;

import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Map;

import org.apache.flume.Context;
import org.apache.flume.Event;
import org.apache.flume.interceptor.Interceptor;

/**
 * @Author: Ferran Fernández Garrido
 *
 */
public class CarsInterceptor implements Interceptor {

    public static final long MIN10 = 10 * 60 * 1000;

    public Event intercept(Event event) {
        Map<String, String> headers = event.getHeaders();
        try {
            if (validateHeader(headers)) {
                headers.put("valid", "true");
            } else {
                DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-
dd");
                Date date = new Date();
                headers.put("valid", "false");
                headers.put("date", dateFormat.format(date));
            }
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return event;
    }

    public List<Event> intercept(List<Event> events) {
        List<Event> eventsParsed = new ArrayList<Event>();
        for (Event event : events) {
            if (intercept(event) != null) {
                eventsParsed.add(event);
            }
        }
        return eventsParsed;
    }
```

```
    public boolean validateHeader(Map<String, String> headers) throws
ParseException {
        // check if this trace has been generated in last hour and
check if the
        // host is an informed field.
        String ts = headers.get("timestamp");
        DateFormat format = new SimpleDateFormat("dd-MM-yyyy
HH:mm:ss");
        Date date = format.parse(ts);
        if ((date.getTime() > System.currentTimeMillis() - MIN10) &&
(!"".equals(headers.get("host")))) {
            return true;
        }
        return false;
    }

    public static class Builder implements Interceptor.Builder {
        public void configure(Context context) {

        }

        public Interceptor build() {
            return new CarsInterceptor();
        }
    }

    public void close() {
        // Default close method
    }

    public void initialize() {
        // Default initialize method
    }

}
```

Script de configuración Flume:

```
agentec.sources = source1
agentec.channels = channel1 channel2

agentec.sources.source1.type = http
agentec.sources.source1.handler =
org.apache.flume.source.http.JSONHandler
agentec.sources.source1.bind = 172.18.0.7
agentec.sources.source1.port = 44444
#agentec.sources.source1.max-line-length = 512000
agentec.sources.source1.interceptors = interceptor1
agentec.sources.source1.interceptors.interceptor1.type =
com.flume.custom.interceptor.CarsInterceptor$Builder
agentec.sources.source1.selector.type = multiplexing
agentec.sources.source1.selector.header = valid
agentec.sources.source1.selector.mapping.true = channel1
agentec.sources.source1.selector.mapping.false = channel2

agentec.sources.source1.channels = channel1 channel2

agentec.channels.channel1.type =
```

```
org.apache.flume.channel.kafka.KafkaChannel
agente.channels.channel1.kafka.bootstrap.servers = 172.18.0.5:9092,
172.18.0.6:9092
agentec.channels.channel1.kafka.topic = valid
#agentec.channels.channel1.kafka.consumer.group.id = flume-consumer
agentec.channels.channel1.kafka.parseAsFlumeEvent = true

agentec.channels.channel2.type =
org.apache.flume.channel.kafka.KafkaChannel
agentec.channels.channel2.kafka.bootstrap.servers = 172.18.0.5:9092,
172.18.0.6:9092
agentec.channels.channel2.kafka.topic = invalid
#agente.channels.channel2.kafka.consumer.group.id = flume-consumer
agentec.channels.channel2.kafka.parseAsFlumeEvent = true
```

9.3.4 Scripts Flume Processor

Script de Dockerfile:

```
FROM debian:jessie
MAINTAINER ffernandez3495

RUN apt-get update && apt-get install -q -y --no-install-recommends
wget

RUN mkdir /opt/java
RUN wget --no-check-certificate --header "Cookie:
oraclelicense=accept-securebackup-cookie" -qO- \
  http://download.oracle.com/otn-pub/java/jdk/8u101-b13/jdk-8u101-
linux-x64.tar.gz \
  | tar zxvf - -C /opt/java --strip 1

RUN mkdir /opt/flume
RUN wget -qO- http://archive.apache.org/dist/flume/1.7.0/apache-flume-
1.7.0-bin.tar.gz \
  | tar zxvf - -C /opt/flume --strip 1

COPY flume-processor.conf /opt/flume/conf/

ENV JAVA_HOME /opt/java
ENV PATH /opt/flume/bin:/opt/java/bin:$PATH

EXPOSE 44444

ENTRYPOINT [ "flume-ng", "agent","-c", "/opt/flume/conf", "-f",
"/opt/flume/conf/flume-processor.conf", "-n", "agentep","-
Dflume.root.logger=INFO,console" ]
```

Script de configuración del Processor:

```
agentep.channels = channel1
agentep.sinks = sink1

agentep.channels.channel1.type =
org.apache.flume.channel.kafka.KafkaChannel
agentep.channels.channel1.kafka.bootstrap.servers = 172.18.0.5:9092,
172.18.0.6:9092
agentep.channels.channel1.kafka.topic = invalid
agentep.channels.channel1.kafka.consumer.group.id = flume-procesor
agentep.channels.channel1.kafka.parseAsFlumeEvent = true

agentep.sinks.sink1.channel = channel1
agentep.sinks.sink1.type = hdfs
agentep.sinks.sink1.hdfs.path = hdfs://hadoop-
master:9000/flume/data/cars/invalids/dt=%{date}
agentep.sinks.sink1.hdfs.filePrefix = cars_log_invalid
agentep.sinks.sink1.hdfs.rollInterval = 600
agentep.sinks.sink1.hdfs.rollSize = 0
agentep.sinks.sink1.hdfs.rollCount = 300000
agentep.sinks.sink1.hdfs.idleTimeout = 0
agentep.sinks.sink1.hdfs.batchSize = 1
agentep.sinks.sink1.hdfs.round = false
agentep.sinks.sink1.hdfs.callTimeout = 120000
agentep.sinks.sink1.hdfs.threadsPoolSize = 1
agentep.sinks.sink1.hdfs.roundValue = 0
agentep.sinks.sink1.hdfs.roundUnit = minute
agentep.sinks.sink1.hdfs.useLocalTimeStamp = true
agentep.sinks.sink1.hdfs.fileType = DataStream
agentep.sinks.sink1.hdfs.maxOpenFiles = 50000
```

9.3.5 Scripts Kafka

Dockerfile de Kafka:

```
WORKDIR /root

RUN apt-get update && apt-get install -y openssh-server openjdk-7-jdk
wget

# Versión 2.7.2
RUN wget http://apache.rediris.es/kafka/0.10.2.0/kafka_2.10-
0.10.2.0.tgz && \
    tar -xzvf kafka_2.10-0.10.2.0.tgz && \
    mv kafka_2.10-0.10.2.0 /usr/local/kafka && \
    rm kafka_2.10-0.10.2.0.tgz

ENV JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
ENV KAFKA_HOME=/usr/local/kafka
ENV PATH=$PATH:$KAFKA_HOME/bin

# SSH sin clave
RUN ssh-keygen -t rsa -f ~/.ssh/id_rsa -P '' && \
    cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

RUN mkdir /opt/kafka
RUN mkdir /opt/kafka/broker1
```

```
RUN mkdir /opt/kafka/broker2

COPY config/broker1/server.properties /opt/kafka/broker1
COPY config/broker2/server.properties /opt/kafka/broker2

#RUN $KAFKA_HOME/bin/zookeeper-server-start.sh -daemon
$KAFKA_HOME/config/zookeeper.properties
#RUN $KAFKA_HOME/bin/kafka-server-start.sh -daemon
/opt/kafka/server.properties

EXPOSE 22 2181 9092

CMD [ "sh", "-c", "service ssh start; bash"]
```

Los server.properties del broker0 a su vez master Kafka:

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# see kafka.server.KafkaConfig for additional details and defaults

############################## Server Basics ##############################

# The id of the broker. This must be set to a unique integer for each broker.
broker.id=0

# Switch to enable topic deletion or not, default value is false
#delete.topic.enable=true

############################# Socket Server Settings #############################

# The address the socket server listens on. It will get the value returned from
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#     listeners = listener_name://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://172.18.0.5:9092

# Hostname and port the broker will advertise to producers and consumers. If not set,
# it uses the value for "listeners" if configured.  Otherwise, it will use the value
# returned from java.net.InetAddress.getCanonicalHostName().
advertised.listeners=PLAINTEXT://172.18.0.5:9092

# Maps listener names to security protocols, the default is for them to be the same. See the config
documentation for more details
#listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:SASL_PLAINTEXT,SASL_SSL:
```

```
# The number of threads handling network requests
num.network.threads=3

# The number of threads doing disk I/O
num.io.threads=8

# The send buffer (SO_SNDBUF) used by the socket server
socket.send.buffer.bytes=102400

# The receive buffer (SO_RCVBUF) used by the socket server
socket.receive.buffer.bytes=102400

# The maximum size of a request that the socket server will accept (protection against OOM)
socket.request.max.bytes=104857600


############################# Log Basics #############################

# A comma seperated list of directories under which to store log files
log.dirs=/tmp/kafka-logs

# The default number of log partitions per topic. More partitions allow greater
# parallelism for consumption, but this will also result in more files across
# the brokers.
num.partitions=1

# The number of threads per data directory to be used for log recovery at startup and flushing at sh
# This value is recommended to be increased for installations with data dirs located in RAID array.
num.recovery.threads.per.data.dir=1

############################# Log Flush Policy #############################

# Messages are immediately written to the filesystem but by default we only fsync() to sync
# the OS cache lazily. The following configurations control the flush of data to disk.
# There are a few important trade-offs here:
#    1. Durability: Unflushed data may be lost if you are not using replication.
#    2. Latency: Very large flush intervals may lead to latency spikes when the flush does occur as
will be a lot of data to flush.
#    3. Throughput: The flush is generally the most expensive operation, and a small flush interval
lead to exceessive seeks.
# The settings below allow one to configure the flush policy to flush data after a period of time o
# every N messages (or both). This can be done globally and overridden on a per-topic basis.

# The number of messages to accept before forcing a flush of data to disk
#log.flush.interval.messages=10000

# The maximum amount of time a message can sit in a log before we force a flush
#log.flush.interval.ms=1000

############################# Log Retention Policy #############################

# The following configurations control the disposal of log segments. The policy can
# be set to delete segments after a period of time, or after a given size has accumulated.
# A segment will be deleted whenever *either* of these criteria are met. Deletion always happens
# from the end of the log.

# The minimum age of a log file to be eligible for deletion due to age
log.retention.hours=168

# A size-based retention policy for logs. Segments are pruned from the log as long as the remaining
```

```
# segments don't drop below log.retention.bytes. Functions independently of log.retention.hours.
#log.retention.bytes=1073741824

# The maximum size of a log segment file. When this size is reached a new log segment will be creat
log.segment.bytes=1073741824

# The interval at which log segments are checked to see if they can be deleted according
# to the retention policies
log.retention.check.interval.ms=300000

############################# Zookeeper #############################

# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# root directory for all kafka znodes.
zookeeper.connect=172.18.0.5:2181

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=6000
```

Los server.properties del broker2 de Kafka:

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# see kafka.server.KafkaConfig for additional details and defaults

############################# Server Basics #############################

# The id of the broker. This must be set to a unique integer for each broker.
broker.id=1

# Switch to enable topic deletion or not, default value is false
#delete.topic.enable=true

############################# Socket Server Settings #############################

# The address the socket server listens on. It will get the value returned from
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#     listeners = listener_name://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://172.18.0.6:9092
```

```
# Hostname and port the broker will advertise to producers and consumers. If not set,
# it uses the value for "listeners" if configured.  Otherwise, it will use the value
# returned from java.net.InetAddress.getCanonicalHostName().
advertised.listeners=PLAINTEXT://172.18.0.6:9092

# Maps listener names to security protocols, the default is for them to be the same. See the config
documentation for more details
#listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:SASL_PLAINTEXT,SASL_SSL:

# The number of threads handling network requests
num.network.threads=3

# The number of threads doing disk I/O
num.io.threads=8

# The send buffer (SO_SNDBUF) used by the socket server
socket.send.buffer.bytes=102400

# The receive buffer (SO_RCVBUF) used by the socket server
socket.receive.buffer.bytes=102400

# The maximum size of a request that the socket server will accept (protection against OOM)
socket.request.max.bytes=104857600


############################# Log Basics #############################

# A comma seperated list of directories under which to store log files
log.dirs=/tmp/kafka-logs

# The default number of log partitions per topic. More partitions allow greater
# parallelism for consumption, but this will also result in more files across
# the brokers.
num.partitions=1

# The number of threads per data directory to be used for log recovery at startup and flushing at sh
# This value is recommended to be increased for installations with data dirs located in RAID array.
num.recovery.threads.per.data.dir=1

############################# Log Flush Policy #############################

# Messages are immediately written to the filesystem but by default we only fsync() to sync
# the OS cache lazily. The following configurations control the flush of data to disk.
# There are a few important trade-offs here:
#    1. Durability: Unflushed data may be lost if you are not using replication.
#    2. Latency: Very large flush intervals may lead to latency spikes when the flush does occur as
will be a lot of data to flush.
#    3. Throughput: The flush is generally the most expensive operation, and a small flush interval
lead to exceessive seeks.
# The settings below allow one to configure the flush policy to flush data after a period of time o
# every N messages (or both). This can be done globally and overridden on a per-topic basis.

# The number of messages to accept before forcing a flush of data to disk
#log.flush.interval.messages=10000

# The maximum amount of time a message can sit in a log before we force a flush
#log.flush.interval.ms=1000

############################# Log Retention Policy #############################
```

```
# The following configurations control the disposal of log segments. The policy can
# be set to delete segments after a period of time, or after a given size has accumulated.
# A segment will be deleted whenever *either* of these criteria are met. Deletion always happens
# from the end of the log.

# The minimum age of a log file to be eligible for deletion due to age
log.retention.hours=168

# A size-based retention policy for logs. Segments are pruned from the log as long as the remaining
# segments don't drop below log.retention.bytes. Functions independently of log.retention.hours.
#log.retention.bytes=1073741824

# The maximum size of a log segment file. When this size is reached a new log segment will be creat
log.segment.bytes=1073741824

# The interval at which log segments are checked to see if they can be deleted according
# to the retention policies
log.retention.check.interval.ms=300000

############################# Zookeeper #############################

# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# root directory for all kafka znodes.
zookeeper.connect=172.18.0.5:2181

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=6000
```

El start container de Kafka:

```
#!/bin/bash

# Iniciando Zookeeper y el broker1 de Kafka
sudo docker rm -f kafka-master &> /dev/null

echo "Start Kafka Master..."
sudo docker run -itd \
                --net=bigdata \
                -p 2181:2181 \
                -p 9092:9092 \
                --name kafka-master \
                --hostname kafka-master \
                mykafka &> /dev/null

# Iniciando el broker2 de Kafka con instancia al zookeeper anterior

sudo docker rm -f kafka-broker2 &> /dev/null

echo "Start Kafka Broker 2..."
sudo docker run -itd \
                --net=bigdata \
                --name kafka-broker2 \
                --hostname kafka-broker2 \
                mykafka &> /dev/null
```

```
# Iniciamos el master de Kafka y despúes deberemos configurar el
Broker 2

sudo docker exec -it kafka-master bash

#NOTA:

#Kafka no puede arrancar directo desde Docker debido a que el servicio
solo se inicia en tiempo de build pero no en tiempo
#RUN, por lo que es necesario ejecutar los siguientes comandos:

#Para el kafka-master
#$KAFKA_HOME/bin/zookeeper-server-start.sh -daemon
$KAFKA_HOME/config/zookeeper.properties
#$KAFKA_HOME/bin/kafka-server-start.sh -daemon
/opt/kafka/broker1/server.properties

#Para el kafka-broker2
#$KAFKA_HOME/bin/zookeeper-server-start.sh -daemon
$KAFKA_HOME/config/zookeeper.properties
#$KAFKA_HOME/bin/kafka-server-start.sh -daemon
/opt/kafka/broker2/server.properties
```

9.3.6 Scripts Spark

Dockerfile de Spark:

```
FROM ubuntu:14.04

WORKDIR /root

RUN apt-get update && apt-get install -y openssh-server openjdk-7-jdk
wget

# Versión 2.7.2
RUN wget http://d3kbcqa49mib13.cloudfront.net/spark-2.1.0-bin-
hadoop2.7.tgz && \
    tar -xzvf spark-2.1.0-bin-hadoop2.7.tgz && \
    mv spark-2.1.0-bin-hadoop2.7 /usr/local/spark && \
    rm spark-2.1.0-bin-hadoop2.7.tgz

ENV JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
ENV SPARK_HOME=/usr/local/spark
ENV PATH=$PATH:$SPARK_HOME/bin

RUN mkdir /opt/spark
RUN mkdir /opt/spark/core
RUN mkdir /opt/spark/yarn

COPY core-site.xml /opt/spark/core
COPY yarn-site.xml /opt/spark/yarn
COPY spark-custom-streaming-0.0.1-SNAPSHOT.jar /usr/local/spark/jars/
COPY spark-streaming-kafka-0-10_2.11-2.0.0.jar /usr/local/spark/jars/
COPY kafka-clients-0.10.0.1.jar /usr/local/spark/jars/
```

```
# SSH sin clave
RUN ssh-keygen -t rsa -f ~/.ssh/id_rsa -P '' && \
    cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

ENV HADOOP_CONF_DIR=/opt/spark/core
ENV YARN_CONF_DIR=/opt/spark/yarn

EXPOSE 22 7077 8080 8081

CMD [ "sh", "-c", "service ssh start; bash"]
```

Los ficheros de core-site.xml y yarn-site.xml son los mismos que en la sección de Hadoop.

Programa Java Spark:

Modelo:

```
package com.spark.model;

import java.io.Serializable;

import com.fasterxml.jackson.annotation.JsonProperty;

import lombok.Data;

@Data
public class CarStatusModel implements Serializable {
    private static final long serialVersionUID = 3106558264531850691;

    private String id;
    private String model;
    private String status;
    private Double speed;
    private String timestamp;
    private CarCoordinates coordinates;

    @JsonProperty("tech_features")
    private CarTechFeatures techFeatures;

}
```

```
@Data
public class CarCoordinates implements Serializable {
    private static final long serialVersionUID = 1739646169560595720L;
    @JsonProperty("long")
    private String longitud;
    private String lat;
}
```

```
package com.spark.model;

import java.io.Serializable;

import com.fasterxml.jackson.annotation.JsonProperty;

import lombok.Data;
```

```
@Data
public class CarTechFeatures implements Serializable {
    private static final long serialVersionUID = 8586327609401184722L;

    @JsonProperty("engine_status")
    private String engineStatus;

    @JsonProperty("gasoline_level")
    private Double gasolineLevel;

    @JsonProperty("oil_status")
    private String oilStatus;

    @JsonProperty("breaks_status")
    private String breaksStatus;

    @JsonProperty("battery_status")
    private String batteryStatus;

    @JsonProperty("preasure_level")
    private CarTechPreasure preasureLevel;

}
```

```
package com.spark.model;

import java.io.Serializable;

import com.fasterxml.jackson.annotation.JsonProperty;

import lombok.Data;

@Data
public class CarTechPreasure implements Serializable {
    private static final long serialVersionUID = -
3479410287502302179L;

    @JsonProperty("upper_left")
    private String upperLeft;

    @JsonProperty("upper_right")
    private String upperRight;

    @JsonProperty("down_left")
    private String downLeft;

    @JsonProperty("down_right")
    private String downRight;

}
```

Logica:

```
package com.spark.logic;

import java.io.IOException;
import java.io.Serializable;
```

```
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Collections;
import java.util.Comparator;
import java.util.Date;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;

import com.fasterxml.jackson.core.JsonParseException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.spark.model.CarStatusModel;

public class TraceParser implements Serializable {

    private CarStatusModel csm;

    private static final long serialVersionUID = 3593001528897234858L;

    public TraceParser(String trace) throws JsonParseException,
JsonMappingException, IOException {
        byte[] rs = trace.replace("\n", "").replace("\r",
"").getBytes();
        String value = new String(rs, "UTF-8");
        ObjectMapper mapper = new ObjectMapper();
        this.csm =
mapper.readValue(value.split("\\?")[1].replaceAll("[\\x00-
\\x09\\x11\\x12\\x14-\\x1F\\x7F]", ""), CarStatusModel.class);
    }

    public String carModel() throws JsonParseException,
JsonMappingException, IOException {
        return csm.getModel();
    }

    public Boolean hasEngineTroubles() {
        return "ko".equals(csm.getTechFeatures().getEngineStatus()) ?
true : false;
    }

    public Boolean isRunning() {
        return "running".equals(csm.getStatus()) ? true : false;
    }

    public String getLatLong() {
        return csm.getCoordinates().getLongitud() + "|" +
csm.getCoordinates().getLat();
    }

    public String getTime() {
        DateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy
HH:mm:ss");
        Date date = new Date();
        return dateFormat.format(date);
    }

    public String getRunningCars(List<String> models) {
```

```
        return "\"running_cars\":" + models.size();
    }

    public String getTroubleCars(Integer ent) {
        return "\"engine_t_cars\":" + ent;
    }

    public String formatTrace(List<String> models, Integer engineT,
List<String> location) {
        return "{\"timestamp\":\"" + getTime() + "\"," +
buildGraph(models) + "," + getRunningCars(models) + "," +
getTroubleCars(engineT) + "," + buildLatLong(location) + "}";
    }

    public String buildGraph(List<String> models) {
        Set<String> unique = new HashSet<String>(models);
        Map<String, Integer> mapModel = new HashMap<String,
Integer>();
        StringBuilder sb = new StringBuilder("\"graph\":[");
        int i = 0;
        for (String key : unique) {
            if (i == unique.size() - 1) {
                sb.append("{\"name\": \"").append(key).append("\",");
                sb.append("\"data\":
[").append(Collections.frequency(models, key)).append("]}");
            } else {
                sb.append("{\"name\": \"").append(key).append("\",");
                sb.append("\"data\":
[").append(Collections.frequency(models,
key)).append("]}").append(",");
            }
            i++;
            mapModel.put(key, Collections.frequency(models, key));
        }
        sb.append("],");
        sb.append(highDemandModel(mapModel));
        return sb.toString();
    }

    public String buildLatLong(List<String> ltn) {
        StringBuilder sb = new StringBuilder("\"coordinates\":[");
        Set<String> unique = new HashSet<String>(ltn);
        int i = 0;
        for (String key : unique) {
            String longi = key.split("\\|")[0];
            String lat = key.split("\\|")[1];
            if (i == unique.size() - 1) {
                sb.append("{\"longitud\":
\"").append(longi).append("\",");
                sb.append("\"lat\": \"").append(lat).append("\"}");
            } else {
                sb.append("{\"longitud\":
\"").append(longi).append("\",");
                sb.append("\"lat\": \"").append(lat).append("\"},");
            }
            i++;
        }
        sb.append("]");
        return sb.toString();
    }
```

```
    public String highDemandModel(Map<String, Integer> m) {
        if (m.size() != 0) {
            ValueComparator bvc = new ValueComparator(m);
            TreeMap<String, Integer> sorted_map = new TreeMap<String,
Integer>(bvc);
            sorted_map.putAll(m);
            return "\"trend_car\":" + "\"" +
sorted_map.firstEntry().getKey() + "\"";
        } else {
            return "\"trend_car\":\"NONE\"";
        }
    }
}

class ValueComparator implements Comparator<String> {
    Map<String, Integer> base;

    public ValueComparator(Map<String, Integer> base) {
        this.base = base;
    }

    // Note: this comparator imposes orderings that are inconsistent
with
    // equals.
    @Override
    public int compare(String a, String b) {
        if (base.get(a) >= base.get(b)) {
            return -1;
        } else {
            return 1;
        } // returning 0 would merge keys
    }
}
```

Main Spark:

```
package com.spark.custom;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.function.VoidFunction;
import org.apache.spark.streaming.Durations;
import org.apache.spark.streaming.api.java.JavaInputDStream;
import org.apache.spark.streaming.api.java.JavaStreamingContext;
import org.apache.spark.streaming.kafka010.ConsumerStrategies;
import org.apache.spark.streaming.kafka010.HasOffsetRanges;
import org.apache.spark.streaming.kafka010.KafkaUtils;
import org.apache.spark.streaming.kafka010.LocationStrategies;
```

```
import org.apache.spark.streaming.kafka010.OffsetRange;

import com.neovisionaries.ws.client.WebSocket;
import com.neovisionaries.ws.client.WebSocketFactory;
import com.spark.logic.TraceParser;

public class SparkNormalization {

    public static void main(String args[]) throws
InterruptedException, IOException {
        Map<String, Object> kafkaParams = new HashMap<String,
Object>();
        kafkaParams.put("bootstrap.servers",
"172.18.0.5:9092,172.18.0.6:9092");
        kafkaParams.put("key.deserializer", StringDeserializer.class);
        kafkaParams.put("value.deserializer",
StringDeserializer.class);
        kafkaParams.put("group.id", "spark-grouper-7289");
        kafkaParams.put("auto.offset.reset", "latest");
        kafkaParams.put("enable.auto.commit", false);

        SparkConf sparkConf = new
SparkConf().setAppName("JavaDirectKafkaWordCount");
        JavaStreamingContext jssc = new
JavaStreamingContext(sparkConf, Durations.seconds(15));

        Collection<String> topics = Arrays.asList("valid");

        final JavaInputDStream<ConsumerRecord<String, String>> stream
= KafkaUtils.createDirectStream(jssc,
LocationStrategies.PreferConsistent(),
                ConsumerStrategies.<String, String> Subscribe(topics,
kafkaParams));

        stream.foreachRDD(new
VoidFunction<JavaRDD<ConsumerRecord<String, String>>>() {
            private static final long serialVersionUID =
1596283014896065779L;

            @Override
            public void call(JavaRDD<ConsumerRecord<String, String>>
rdd) throws Exception {
                final OffsetRange[] offsetRanges = ((HasOffsetRanges)
rdd.rdd()).offsetRanges();
                rdd.foreachPartition(new
VoidFunction<Iterator<ConsumerRecord<String, String>>>() {
                    private static final long serialVersionUID =
7732340710914542410L;

                    @Override
                    public void call(Iterator<ConsumerRecord<String,
String>> consRec) throws Exception {
                        WebSocket websocket = new
WebSocketFactory().createSocket("ws://192.168.1.201:9300").connect();
                        TraceParser tp = null;
                        String trace = "";
                        ArrayList<String> models = new
ArrayList<String>();
                        ArrayList<String> location = new
ArrayList<String>();
                        int engineT = 0;
```

```
                        while (consRec.hasNext()) {
                            ConsumerRecord<String, String> msg =
consRec.next();
                            tp = new TraceParser(msg.value());
                            if (tp.isRunning()) {
                                models.add(tp.carModel());
                                location.add(tp.getLatLong());
                            }
                            if (tp.hasEngineTroubles()) {
                                engineT++;
                            }
                        }

                        if (tp != null) {
                            trace = tp.formatTrace(models, engineT,
location);

                            websocket.sendBinary(trace.getBytes());
                            websocket.sendClose();
                        }

                    }
                });
            }

        });
        jssc.start();
        jssc.awaitTermination();
    }

}
```

9.4 Problemáticas técnicas encontradas

9.4.1 Despliegue de Kafka en Docker

Se presentó una problemática algo difícil de detectar en el despliegue de Kafka en Docker.

Los brokers de Kafka deben tener definidos en su server.properties la dirección IP del servidor que aloja el bróker. Eso quiere decir que la resolución por nombre es esa versión concreta de Kafka no era posible.

El problema es que el contenedor Docker no tiene IP hasta el momento en que se despliega y por tanto no se puede saber a priori que IP tendrá. La solución paso por crear una red interna docker “bigdata”.

Dentro de esa red sabemos exactamente la disposición de las direcciones IP que tendrá y por tanto, podemos asignar una IP de forma previa a su despliegue.

Esta problemática es difícil de detectar puesto que los logs de Kafka no dejan ver de forma muy explícita que no son capaces de ejecutarse por esa razón. Después de sondear los logs durante un buen rato se llegó a la conclusión que

el problema podía venir derivado de la conexión y por tanto se tenían pistas de por dónde tirar.

9.4.2 Spark Streaming y Kafka con la nueva API

Debido a la versión de Kafka que se ha usado en este proyecto, 2.10-0.10.2.0, la versión de Spark Streaming que implementa el conector para esta versión aún se encuentra en fase experimental.

Realmente la dificultad fue en primera instancia puesto que es fácil no darse cuenta que las versiones a priori no cuadran. Esto tenia diversas implicaciones debido al serializador de Kafka.

Una vez detectado el error fue necesario descargar las dependencias desde el maven repository de cara a que el programa Spark fuera capaz de traerlas en el momento de su ejecución