**UNIVERSIDAD AUTONOMA DE MADRID**

**ESCUELA POLITECNICA SUPERIOR**



**Máster en Big Data y Data Science: ciencia e ingeniería de datos**

**TRABAJO FIN DE MÁSTER**

Implementación de una arquitectura Lambda con Spark orientado a IoT

**Pablo Machío Rueda**

**Tutor: Paulo Villegas**

**Marzo 2023**

Implementación de una arquitectura Lambda con Spark orientado a IoT

**AUTOR: Pablo Machío**

**TUTOR: Paulo Villegas**

**Escuela Politécnica Superior**

**Universidad Autónoma de Madrid**

**Marzo de 2023**

**Resumen**

Este Trabajo Fin de Máster tiene como finalidad mostrar cómo se podría implementar una arquitectura Lamba orientada al análisis de un servicio IoT para la adquisición de datos de telemetría de vehículos movidos por combustibles fósiles.

Las principales funcionalidades de la arquitectura son:

* Creación del resumen de trayectos de los vehículos y su etiquetado de manera que se puedan comparar los resúmenes de los trayectos de una misma clase.
* Creación de eventos en diferido que puedan repercutir en el consumo del vehículo durante la realización de un trayecto.
* Creación de eventos en tiempo real que permitan una operación en inmediata de los operadores.

***Agradecimientos***

**INDICE DE CONTENIDOS**

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

Anexo ￼

**INDICE DE FIGURAS**

￼

**INDICE DE TABLAS**

# Introducción

## Motivación

Esta memoria de TFM tiene como finalidad mostrar cómo se podría implementar una arquitectura Lambda para Big Data dentro del *ecosistema Spark*, reutilizando en la medida de lo posible código, con la intención de ser lo más consistente y eficiente posible.

El contexto de aplicación queda determinado por el uso de *dispositivos de telemetría* instalados en vehículos de automoción como coches, camiones o motos. Estos dispositivos transmiten 1,4KB de información con una frecuencia de un minuto o menos, lo que supone, aproximadamente, unos 2 MB al día. Se espera que el número de dispositivos conectados sea de varios miles y por tanto la información diaria generada es de varios GigaBytes, así que podemos considerar que estamos trabajando en un entorno de *BigData*.

La arquitectura de análisis de datos Lambda es una de las más populares para entornos BigData. Para poder realizar el análisis de datos, la arquitectura, se divide en varias capas:

* *Capa Batch* que se encarga de generar vistas sobre los datos en reposos. Realiza el precálculo los datos para que no tener que crearlos al vuelo, algo imposible en BigData.
* *Capa RealTime* que se encarga de generar las vistas de los datos que van en entrando en tiempo real al sistema y que permite acceder accesible la información que aún no fue procesada la *capa Batch.*
* Capa de servicios que contiene las vistas generadas por las dos capas anteriores y por tanto nos permite analizar toda la información.

Spark proporciona herramientas para las *capas Batch y RealTime* de manera que podemos usar un mismo lenguaje para ambas capas, fomentando la mantenibilidad y la consistencia en la aplicación de lógicas de transformación y análasis de datos originados en distintas fuentes.

## Objetivos

El objetivo principal es realizar el análisis de los vehículos gracias al uso sensores instalados o conectados a la centralita del propio vehículo.

Los datos más importantes derivados de la información transmitida por los sensores son:

* Trayectos, que corresponde a la agregación de la distancia y consumo entre 2 coordenadas, una de inicio y otra de fin, y los momentos en los que se originaron.
  + Es interesante poder etiquetar los trayectos para poder realizar comparaciones entre los trayectos con una misma etiqueta y así saber en cuales se tardan más tiempo o se consume más combustible. La clasificación se puede automatizar con un clasificador de *Machine Leaning*
  + Los trayectos no requieren ser analizados en tiempo real y por tanto su creación se realiza en la *capa Batch*
* Eventos, que son el registro de sucesos en base a cambios de valores en los sensores de dispositivo asociado al vehículo.
  + Algunos eventos requieren ser atendidos inmediatamente, como puede ser una bajada brusca de nivel de combustible, y por tanto son generados en la *capa Realtime,* otros pueden ser atendidos posteriormente como complemento del análisis del trayecto y por tanto se pueden generar en la *capa Batch,* como pueden ser el pisar el pedal del acelerador más de lo necesario

## Organización de la memoría

* Estado del arte
* Diseño
* Desarrollo
* Integración, pruebas y resultados
* Concluiones y trabajos fúturos

# Estado del arte

## ¿

## Soluciones en tu área de aplicación

## Arquitecturas similares a la tuya

## Componentes empleados



Figura 2‑1: Logo EPS

# Diseño

## Modelos de datos

### Frame de información de los dispositivos IoT

Corresponde con un archivo JSON, donde se pueden econtrar distintos niveles de profundidad por atributo según la complejidad .Se compone de los siguientes campos:

* “id”. Identificador único del
* “version”. Versión del “modelo de datos” o “protocolo”. Indetificador del esquema usado para los datos. Cada versión puede tener más o menos campos.
* “timestamp”. Fecha en la que el dispositivo genero el frame de información.
* “server”.Información de registro la recepción del frame. Se espera que crezca
  + “timestamp”. Fecha en la que se recibio el frame y se establecio el “id” y “version”.
* “attributes”. Atributos del dispositivo
  + “tenantId”. Identificador del propietario del dispositivo.
  + “deviceId”. Idenficador del dispositivo
  + “manufacturer”. Nombre del fabricante del dispositivo
  + “model”. Modelo del dispositivo
  + “identifier”. Número del moden del dispositivo.
* “device”. Sensores del propio dispositivo
  + “battery”. Bateria del dispositivo
    - “voltage”. Lectura del voltage
    - “level”. Nivel debateria. De 0 a 100.
  + “mileage”. Odómetro mantenido por el propio dispositivo.
    - “distance”. Distancia calculada por del dispositivo en base al GPS
* “can”. Sensores del propio vehículo que se acceden mediante el puerto OBDII o Canbus
  + “vehicle”. Sensores del vehículo que no son ni del motor, ni de la bateria y ni relacionados con el consumo.
    - “mileage”. Odómetro mantenido por el vehículo
      * “distance”. Distancia total recorrida en metros.
    - “pedal”. Sensores sobre los pedales del vehiculo
      * “throttle”. Acelerador
        + “level”. Porcentaje pisado del accelerador
    - “cruise”. Control de crucero, para circular a una velocidad determinada
      * “status”. Activo o no
    - “handBrake”. Freno de mano
      * “status”. Activo o no
    - “doors”. Puertas del vehículo.
      * “indicator”. Notificación de puerta abierta
        + “status”. Activo o no
    - “lights”. Luces del vehículo
      * “hazard”. Luces de emergencia
        + “status”. Activo o no
      * “fog”. Luces de anti-niebla
        + “status”. Activo o no
    - “seatBelts”. Cinturones de seguridad
      * “indicator”. Notificación de puerta abierta
        + “status”. Activo o no
    - “beams”. Focos
      * “high”. Indicador de si lo focos están alto
        + “status”. Activo o no
    - “lock”. Seguro de las puertas
      * “central”. Cierre centralizado de los seguros
        + “status”. Activo o no
    - “gear”. Marchas
      * “reverse”. Marcha atrás
        + “status”. Activo o no
    - “airConditioning”. Aire acondicionado
      * “status”. Activo o no
  + “engine”. Sensores del motor.
    - “time”. Tiempo de actividad del motor
      * “duration”. Duración total del motor activo
  + “battery”. Sensores de la batería.
    - “charging”. Si se encuentra la batería recargándose.
      * “status”. Si o no
  + “fuel”. Sensores relacionados con el consumo de combustible.
    - “consumed”. Consumo de combustible.
      * “volume”. Volumen total consumido. En litros.
    - “level”. Nivel del tanque de combustible.
* “gnss”. Sensores relacionados con la geolocalización del dispositivo.
  + “type”: Que se usó para calcular.
    - GPS
    - GSM, triagulación de antenas GSM
  + “coordinate”
    - “lat”. Latitud. Grados
    - “lng”. Longitud. Grados
  + “altitude”. Altura en metros
  + “speed”. Velocidad en Km/h
  + “course”. Dirección en grados (0 a 360)
  + “address”. Direccón, si se aplicó geolocalización inversa.
  + “satellites”. Numero de satellites utilizados para el cálculo. Caso de GPS
* “ignition”. Sensor de la llave de ignición de vehículo.
  + “status”. Activa o no

### Trayectos

Los trayectos son agregaciones de consumo y distancia de un dispositivo asociado a un vehículo entre unas coordenadas de inicio y otras de fin. Estas agregaciones se realizan sobre un conjunto de frames consecutivos en el tiempo que mantienen el estado “ignition.status” activo de manera continua. Queda definido por los campos:

* “id”. UUID para identificar unívocamente cada trayecto. Se genera al crear el trayecto
* “device\_id”. Identificador del dispositivo
* “start\_timestamp”. Valor del campo “timestamp” inicial, que corresponde al primer frame con el campo “ignition.status” activo, el anterior estaban inactivos.
* “start\_location\_address”. Valor del campo “gnss.address” inicial, que corresponde al primer frame con el campo “ignition.status” activo. Los anteriores estaban inactivos.
* “start\_location\_latitude”. Valor del campo “gnss.coordinate.lat” inicial, que corresponde al primer frame con el campo “ignition.status” activo. Los anteriores estaban inactivos.
* “start\_location\_longitude”. Valor del campo “gnss.coordinate.longitude” inicial, que corresponde al primer frame con el campo “ignition.status” activo. Los anteriores estaban inactivos.
* “end\_timestamp”. Valor del campo “timestamp” final, que corresponde al último frame con el campo “ignition.status” activo, los posteriores estarán inactivos.
* “end\_location\_address”. Valor del campo “gnss.address” final, que corresponde al último frame con el campo “ignition.status” activo, los posteriores estarán inactivos.
* “end\_location\_latitude”. Valor del campo “gnss.coordinate.lat” final, que corresponde al último frame con el campo “ignition.status” activo, los posteriores estarán inactivos.
* “end\_location\_longitude”. Valor del campo “gnss.coordnate.lng” final, que corresponde al último frame con el campo “ignition.status” activo, los posteriores estarán inactivos.
* “distance”. Diferencia entre el último y el primer valor del campo “can.vehicle.mileage.distance”
* “consumption”. Diferencia entre el último y el primer valor del campo “can.fuel.consumed.volume”
* “label”. Etiqueta con la clase del trayecto. Si no se conoce se marcará como “unknown”

### Eventos

Son registros de valores de los sensores que se consideran importantes, ya sea porque se han superado un umbral o su evolución en el tiempo es anómala. Se componen de los siguientes campos:

* “id”. UUID para identificar unívocamente cada evento. Se genera al crear el evento.
* “created”. Corresponde con el campo “timestamp” del frame que disparó el evento.
* “type\_id”. Identifica el tipo de evento. En este caso tenemos:
  + 1 para eventos correspondientes a pisar el pedal del acelerador, campo “can.vehicle.pedals.throttle.level”,por encima del 20%.
  + 2 para eventos correspondientes a un descenso del 5% del nivel de combustible en menos de 5 minutos
* “location\_address”. Corresponde con el campo “gnss.address” del frame que disparó el evento.
* “location\_latitude”. Corresponde con el campo “gnss.coordinate.lat” del frame que disparó el evento.
* “location\_longitude”. Corresponde con el campo “gnss.coordinate.lng” del frame que disparó el evento.
* “value”. Valor descriptivo del evento.
  + Para eventos de type\_id = 1 indicamos el porcentaje de pedal
  + Para eventos de type\_id = 2 idicamos que el porcentaje del nivel combutible descendio más de 5% en 5 minutos.

## Arquitectura lambda

En la arquitectura Lamba se aúnan el procesamiento de datos en *reposo*, normalmente almacenados en un sistema de archivos distribuidos como puede ser HDFS o una base d a tos extensiva, y datos en *tiempo real o stream de datos.*

La *capa batch* se encarga de los *datos en reposo*. Esta capa realiza pre-cálculos sobre los datos en reposo ofreciendo una vista resumen o *batch view*, que se guardan en la *capa de servicio* para ser consultadas. Los usuarios consultan los *batch views* de forma que se ahorran el lanzar consultas al vuelo sobre todos los datos en reposo, con el consiguiente ahorro de tiempo y costes.

La generación de los *batch views* llevan una cantidad de tiempo considerable y cuando han terminado nuevos datos han entrado al sistema y no se han tenido en cuenta para su generación por eso se debe usar una *capa de tiempo real* que trabaja sobre los nuevos datos que van entrando en sistema para generar los *real-time views* los cálculos sobre estos datos para que estén disponibles para los usuarios.

En nuestro caso usaremos un archivo json plano para emular la fuente de información de la *capa batch* y Kafka para la de la *capa de realtime.* Como capa de servicio para guardar las vistas usaremos PostgreSQL.

# Desarrollo

## Capa batch

El *driver* se compone de 2 transformaciones*,* una para la creación y etiquetado de trayectos y otra para la creación del evento de aceleración excesiva. Ambos leen archivos JSON con los *frames de información* transmitidos por los dispositivos y almacenados en un sistema de archivos distribuido.

### Creación de trayectos

1. Detección del cambio de valor de “ignition.state” para saber dónde comienza y donde termina el grupo de frames que forma un trayecto.
   * Usamos ua función de ventana crea una columna con el valor 1 si hay un cambio de estado y 0 si no lo hay
2. Quitar los frames con “ignition.state” no activo ya que u trayecto se compone de frames con la ignición activa.
3. Agrupar en trayectos.
   * En base al punto 1, usamos una función de ventana para sumar la columna con los valores 1 y 0. La función suma los valores de la columna de forma que se crea un identificador incremental por cada agrupación. Todos los frames del mismo grupo tienen el mismo valor en esa columna
4. Establecer los valores iniciales del grupo de frames. Usando una función de ventana para saber el valor inicial dentro del grupo.
5. Establecer los valores finales del grupo de frames. Usando una función de ventana para saber el valor final dentro del grupo.
6. Determinar las diferencias de consumo y distancia entre los frames del grupo. Se usa una función de ventana para comparar el valor actual con el anterior dentro del grupo.
7. Agregar los los valores de grupo
   * Mínimo valor inicial coordenadas
   * Máximo valor final coordenadas
   * Suma de las diferencias de consumo y distancia para obtener el total del grupo.

### Etiquetado de trayectos

Para etiquetar los trayectos nos basamos en las coordenadas iniciales y finales del trayecto. Se ha probó inicialmente con un modelo de *regresión logística* que resulto ser suficiente para distinguir 6 tipos de trayectos distintos. Se considera que son trayectos del mismo tipo si las coordenadas no se desvian más de un 0.001 de la latitud o longitud ya que corresponde a una desviación de unos 111.1 metros ([*http://wiki.gis.com/wiki/index.php/Decimal\_degrees*](http://wiki.gis.com/wiki/index.php/Decimal_degrees)

).

Como no disponía de muchos ejemplos para entrenar el modelo he creado una pequeña función que genera nuevos trayectos a partir de otros con una modificación no mayor a 0.001 de la latitud y la longitud.

Para el modelo se ha seleccionado *Regresión Lineal* porque es uno de los modelos más sencillos y los resultados ofrecidos son más que satificactorios (entorno al 0.99 para metricas de recall, precision, f1 y accuracy).

Para el entrenamiento se usa un pipeline cuyos pasos son:

* Indexar las etiquetas para pasar de texto a número
* Generación del vector de features

Al pipleline le hemos aplicado CrossValidation con los parametros:

* RegParam: 0.1, 0.01, 0.001.
* MaxIter: 10, 20, 30.

Siendo los resultados muy similares para todas las combinaciones.

Finalmente hemos guardado el modelo para poder aplicarlo en el driver de creación de trayectos.

Cuando aplicamos el modelo, si no encontramos clases con una probabilidad superior a 0.9 etiquetamos el trayecto como desconocido.

### Creación de evento de exceso de aceleración

Es un evento muy sencillo, si el valor del campo “can.vehicle.pedals.throttle.level” es igual o superior a 20, se crea el evento usando los valores del frames.

## Capa realtime

El *driver* de la *capa de tiempo real* lee de Kafka y guarda los resultados en PostgreSQL

### Creación de evento de descenso brusco del nivel de combustible.

Por vehículo debemos comprobar que si el nivel de combustible, campo “can.fuel.level” ha bajado en los últimos 5 minutos, campo “timestamp”

Para esta comprobación se hace uso de agrupación dispositivo, función “groupByKey” y estados mantenidos por Spark mediante la función “flatMapGroupWithState”. Lo que hacemos es guardar los estados relativos a los frames correspondientes a los últimos 5 minutos y comprobamos respecto al actual si la diferencia es superior al 5 %. Si es superior se crea el evento. Los estados más antiguos a 5 minutos son descartados.

Para la gestión de los estados es necesario definir tres clases:

* Una que define el tipo de dato de entrada para la comprobación del cambio de estado
* Otra que define el esatdo
* Una última que define la respuesta a la comprobación.

Definimos una función que recibe como valores de entrada, el valor por el que se agrupan los estados, una secuencia de datos de entrada y una secuencia de salidas. Dentro de la función por cada dato de entrada se recupera o crea el listado de estado si no existe se comprueba si hay un decremento del nivel de combustible en los últimos 5 minuto y se devuelva el resutado, actualizando el estado.

Para aplicar la lógica se usan Datasets en lugar de Dataframes y hay que agrupar por el identificador del dispositivo y aplicar flatMapGroupsWithState para los frames del dispositivo se comparen contra el estado guardado

# Integración, pruebas y resultados

## Entorno

Los drivers de Spark se han escrito en Scala y se usando Java 11, SBT 1.8 y Scala 2.12

Para el entorno de ejecución se ha usado Docker-Compose, con los servicios:

* Cluster de spark
  + Se han usado 2 volumenes
    - Uno (apps) para compartir el jar con los drivers y poder enviarlos al cluster usando Spark-submit
    - El otro (data) con los archivos json que cotienene los frames y con el modelo de ML guardado
* PostgreSQL. Se ha creado un volumen para compartir el script de creación de base de datos.
* Kafka + Zookeeper. Se ha cerado un volumen con los mismos datos usados en el cluster de spark para que el productor de consola del Kafka los publique.

# Conclusiones y trabajo futuro

## Conclusiones

Spark ofrece un entorno ideal para el desarrollo de una arquitectura Lambda.

La abstracción que ofrece para múltiples fuentes y destinos, tanto de la capa Batch como de la capa RealTime, evitan que los desarrolladores empleen tiempo innecesario en su implementación y se focalicen en la lógica de transformaciones y acciones sobre los datos.

El desarrollo estás lógicas también se ve favorecido en el entorno de Spark porque en un mismo proyecto podemos tener distintas lógicas fomentando la reutilización de código, así como una mejor comprensión de estás

## Trabajo futuro

Aunque en el trabajo se ha tratado la arquitectura Lambda, el uso de Spark se ha centrado en las capas Batch y RealTime para generar las vistas que se consultan desde la capa de Servicio. En el trabajo no me he centrado en esta capa, aunque Spark ofrece un módulo de consulta SQL porque existen herramientas de BigData que permiten hacer consultan sobre los datos de forma más eficiente que Spark, como es el caso de Cassandra, una base datos no relacional distribuida que escala muy bien y que permite una eficiencia muy alta en las consultas siempre que estás tenga un claro casos de uso, no es para consultas generalistas sobre los datos. El siguiente paso sería usar Cassandra como capa de servicio para guardar las vistas y así cerrar la arquitectura.

# Referencias

1. En las referencias figurarán los autores (opcionalmebte los editors), el título del artículo, el nombre de la revista o libro, el volumen y número de la revista, las páginas del artículo, la fecha de edición,. A continuación se listan algunos ejemplos
2. K.N. Platanioitis, C.S. Regazzoni (eds.), “Special Issue in Visual-centric Surveillance Networks and Services”, IEEE Signal Processing Magazine, 22(2), Marzo 2005.
3. B.S. Manjunath, P. Salembier, T. Sikora (eds.), “Introduction to MPEG 7: Multimedia Content Description Language,”, John Wiley and Sons, 2002
4. G. R. Bradski, “Computer vision face tracking as a component of a perceptual user interface,” en Proc.IEEE Workshop on Applications of Computer Vision, Princeton, NJ, October 1998, pp. 214–219.
5. A. D. Bue, D. Comaniciu, V. Ramesh, and C. Regazzoni, “Smart cameras with real-time video object generation,” in Proc. IEEE Intl. Conf. on Image Processing, Rochester, NY, volume III, 2002, pp. 429–432.
6. P. Anandan. “A computacional cuadrowork and an algorithm for the measurement of visual motion”, International Journal of Computer Vision, 2(3):283-310, January, 1989.
7. W.J. Ruckelidge. “Efficient Computation of the minimum Hausdorff Distance for Visual Recognition”, Phd thesis, Cornell Universitym 1995. CS-TR1454
8. “Extensible Markup Language (XML) 1.0 (Second Edition)”, W3C Recommendation 6 October 2000 <http://www.w3.org/TR/REC-xml>
9. William H. Press, Saul A.Teukolsky, William T. Vetterling, Brian P. Flannery. “Numerical Recipes in C – The art of Scientific Computing 2nd Edition”. Cambridge University Press