**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 los datos de un servicio IoT que adquiere datos de telemetría de vehículos, siendo Spark la herramienta que vertebra el análisis en las distintas capas de la arquitectura

Primero se introducirá brevemente que IoT y la arquitectura más común empleada, seguido de la introducción el concepto de BigData y cómo encaja en la capa de análisis de la arquitectura IoT, y se razonará porqué se selecciona la arquitectura Lambda Unificada de BigData y cómo Spark facilita su implementación y se exponen unos casos de uso relacionados con la telemetría de vehículos para ilustrar está implementación.

A continuación, se presentarán tres casos de uso ilustrar la arquitectura Lambda aplicada al análisis de vehículos de combustión gracias a la información obtenida de la conexión al diagnóstico de a bordo del vehículo.

Después muestra el diseño de la arquitectura empleada y se comentan las herramientas utilizadas, así como su implementación final.

Por último, se realiza una evaluación de la arquitectura desarrollada teniendo en cuenta las facilidades ofrecidas

***Agradecimientos***

**INDICE DE CONTENIDOS**

Actualizar

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

￼

Anexo ￼

**INDICE DE FIGURAS**

Actualizar

￼

**INDICE DE TABLAS**

Actualizar

# Introducción

El Internet de las cosas o IoT por sus siglas en Ingles [1], permite crear una representación virtual de objetos físicos mediante:

* Sensores que miden el contexto de estos objetos físicos
* Actuadores que permiten la interacción con los mismos.

IoT vertebra la comunicación entre los objetos y hacia ellos, y para poder explotar su potencial, se base usa una arquitectura dividida en capas [1] [2], donde las primeras capas se ocupan de los dispositivos que miden el mundo físico, la comunicación entre ellos y el exterior, otras capas son dedicadas a recolección de los datos y su análisis, y finalmente unas capas de aplicación y negocio para obtener conocimiento de acuerdo al dominio deseado.

El gran volumen de datos que generan los sensores hace que el Big Data la tecnología seleccionada para la almacenar de datos y realizar el análisis de éstos [2]. Big Data está pensado para trabajar con volúmenes de datos grandes y complejos [3] que no podríamos gestionar con sistemas incrementales clásicos [4]. El Big Data se fundamenta en el uso de distintas tecnologías combinadas que se armonizan bajo una arquitectura. A lo largo de la vida del Big Data han surgido distintas arquitecturas [3], aunque las más extendidas son Kappa y Lambda. La elección de la arquitectura a emplear depende del caso de uso.

## Motivación

Esta memoria de TFM está motivada por la dificultad de elección de una arquitectura BigData ya que la decisión depende del uso que se le quiera dar. En este caso se va a aplicar una arquitectura Lambda orientada al análisis de datos de dispositivos conectados al OBDII de vehículos de combustión como coches o furgonetas.

## Objetivos

El objetivo principal es mostrar el diseño e implementación la arquitectura Lambda para Big Data usando el ecosistema *Spark* y tener así un repositorio único de código para el análisis de datos. El hecho de tener un repositorio único reduce la complejidad de esta arquitectura y facilita el poder aplicar operaciones de análisis, como el entreno de modelos de Machine Learning, que no estuvieran previamente previstas, algo que es más difícil de realizar en Kappa por estar orientada al tiempo real [5].

## Organización de la memoria

Para conseguir el objetivo propuesto primero estableceremos las bases sobre las que trabajaremos introduciendo como se hace actualmente el análisis de trayectos, así como la estimación del consumo de combustible en los vehículos de combustión. También se introducirá el concepto de IoT y su arquitectura más extendida. La introducción se completará con la idea de que es el BigData y las arquitecturas más extendidas. Y finalmente se expondrá el procesamiento en Batch y Streaming que forman parte de las arquitecturas Kappa y Lambda. Por último, en este punto, se introducen las tecnologías que se van a usar en la implementación.

Una vez expuesta base teórica se presentará la prueba de concepto (PoC) que servirá como ejemplo para ilustrar del diseño e implementación de la arquitectura Kappa. El PoC está orientado a la explotación de datos de telemetría de vehículos recolectados gracias a una infraestructura IoT y se divide en tres casos de uso.

Luego se establecerá el diseño de la arquitectura, sus componentes y las funciones de los componentes en la arquitectura, así como las tecnologías usadas en cada componente.

Después, se detalla la implementación de cada caso de uso. Por cada caso de uso se indica la capa implicada de la arquitectura, así como el origen y destino de los datos, la estructura de datos de datos de entrada y salida de cada transformación, y la lógica de la transformación y sus tests unitarios que ayudan a su compresión. Se hace referencia al repositorio que contiene un archivo docker-compose para levantar el entorno y una serie de scripts para ejecutar la prueba, todo ello detallado en el README.md del repositorio.

El último apartado contiene las conclusiones y se comentan posibles trabajos futuros.

# Estado del arte

[2] descripción de arquitecturas similares a la tuya (lambda)

\* IOT (centrarnos en la parte de analisis)

\* BIG DATA

\* ARQUITECTURAS BIGDATA

\* Procesamiento Batch y Streamin (Ver PDF de DiazSuarez)

\* Posbilidades de Clasificación ML

[3] explicación (breve) de los componentes que vayas a usar en tu trabajo¿Dónde buscarlo? Tendrías dos fuentes(a) información de productos y servicios (empresas, proveedores, etc)(b) artículos académicos sobre el tema (aquí lo más rápido es usar un buscador tipo Google Scholar <https://scholar.google.com/>)

\* Kafka

\* Spark

\* HDFS por encima

Para conseguir el objetivo propuesto primero estableceremos las bases sobre las que trabajaremos introduciendo como se hace actualmente el análisis de trayectos, así como la estimación del consumo de combustible en los vehículos de combustión. También se introducirá el concepto de IoT y su arquitectura más extendida. La introducción se completará con la idea de que es el BigData y las arquitecturas más extendidas. Y finalmente se expondrá el procesamiento en Batch y Streaming que forman parte de las arquitecturas Kappa y Lambda. Por último, en este punto, se introducen las tecnologías que se van a usar en la implementación.

## Análisis de trayectos Big Data

Se entiende por trayecto al recorrido realizado entre un punto de origen y uno de destino, en nuestro caso los puntos de origen y destino están ligados al estado del motor del vehículo, siendo el origen el punto en que el motor se enciende y el destino el punto en el que el motor se apaga, todo lo que pase fuera de esos dos estados no interesa para nuestro. Existen muchas maneras de filtrar dentro de grandes volúmenes los datos relacionados con trayectos con el fin de buscar patrones y comportamientos recurrentes [8]:

* Por origen y destino.
* Análisis espacial
* Agregacion de puntos sobre pequeñas celdas geográficas para un gran número de celdas.

Para la selección de modelos predictivos en datasets relacionados con trayectos es importante identificar las variables explicativas y los mecanismos físicos [8]. La selección de los modelos se base en aproximaciones entre modelos lineales o modelos basados en la regresión del kernel [8].

Para el PoC se usa regresión lineal para clasificar los trayectos con un origen y destino igual para poder realizar la comparación entre ellos.

## Estimaciones de consumo

Los modelos de estimación de combustible de décadas pasadas como los modelos americanos MOBILE y MOVES de la EPA, o el modelo europeo COPERT se basaban en las características de los vehículos como su motores o potencia, pero no eran muy exacto y se podían mejorar en base al posicionamiento GPS de los vehículos [6]. Combinando los modelos anteriores y el posicionamiento GPS se mejora la precisión de las predicciones [7]. Actualmente existen en el mercado dispositivos que se conectan directamente al puerto OBDII de los vehículos para obtener los valores de telemetría del vehículo, como el consumo instantáneo o el acumulado, y transmitir estos datos a un móvil mediante Bluetooth o directamente a un servidor mediante redes móviles. En definitiva, dispositivos IoT que capturan la telemetría del vehículo.

Para el PoC se hará uso de la información obtenida directamente del vehículo gracias a dispositivos conectados al puerto OBDII del vehículo.

## Internet de las cosas (IoT)

El internet de las cosas o IoT, es un tipo de red cuya finalidad es interconectar, mediante protocolos estandarizados, dispositivos capaces de obtener información (sensores), como pueden ser un localizador GPS, y dispositivos capaces de realizar una acción (actuadores), como puede ser una salida digital que activa el corte de motor de un vehículo, mediante Internet. Esta interconexión de dispositivos es la base para tareas más complejas como puede ser análisis de datos para distintos modelos de negocio, toma de decisiones, ejecución de acciones en función del contexto, reconocimiento inteligente, posicionamiento, monitorización... [1]

Las tecnologías que permiten IoT se pueden categorizar en:

* Tecnologías que adquieren información contextual.
* Tecnologías que permiten el procesamiento de la información contextual adquirida
* Tecnologías para mejorar la seguridad y la privacidad.

Las 2 primeras son los bloques funcionales que permiten dar inteligencia a las cosas.

El tercer bloque, aunque no es funcional, es una necesidad de facto para permitir el uso de IoT [1].

Podemos ver que el IoT no es una sola tecnología, sino una mezcla de distintas tecnologías software y hardware. Es una gran mezcla de tecnologías que necesitan ser adaptadas para permitir aplicaciones de IoT como la eficiencia energética, velocidad seguridad, confianza [1].

Estas tecnologías se agrupan por capas dando lugar a la arquitectura de IoT, donde el número de capas puede variar según como se realice la agrupación, pero es muy común la agrupación en 4 capas: capa de dispositivos y sensores, capa de gateways y redes, capa de gestión de servicios y capa de aplicación [2]. Las capas se comunican con sus capas adyacentes. De la más baja a la más alta tenemos:

* **Capa de dispositivos y sensores**. Conexión entre el mundo físico y el digital, capturan la información mediante sensores y puede realizar un pequeño procesamiento sobre ésta. También se encarga aplicar las decisiones sobre los elementos físicos mediante actuadores.
* **Capa de gateways y redes.** Se encarga del transporte de la información capturada y tiene que lidiar con la heterogeneidad de los dispositivos mediante la unificación protocolos.
* **Capa de gestión de servicios.** Se encarga del almacenamiento, análisis de la información capturada, controles de seguridad, modelado de procesos y gestión de dispositivos.
* **Capa de aplicación.** Aplica toda la información al dominio del problema en cuestión.

El PoC se encontraría en la capa de gestión de servicios, realizando el análisis de los datos y generando nueva información a partir de estos

## BigData y arquitecturas

La idea del Big Data está fuertemente relacionada el abaratamiento del coste de almacenamiento de datos y la gran cantidad de datos estructurado, semiestructurados y no estructurados que generan las aplicaciones actuales o que las empresas tienen ya almacenados [3].

Trabajar con estos datos y obtener conocimiento de ellos es muy complicado ya que los conjuntos de datos se pueden encontrar en distintas escalas, pueden ser adquiridos de distintas formas, pueden tener distinta naturaleza (estructurados, semiestructurados, y no estructurados) [3]. El big data pueden caracterizarse, por tanto, por la creación de datos y su almacenamiento, análisis y recuperación, tal y como se define en 5 V

* **Volumen.** Denota la gran cantidad de datos generado en poco tiempo que determina el valor y el potencial que se deben considerar y requiere de plataformas de computación especial en orden de poder analizarla.
* **Velocidad.** Se refiere a la velocidad en la que los datos son generados y procesados para poder superar los retos y la demanda que reside en el camino del desarrollo y crecimiento
* **Variedad.** Se refiere al tipo de contenido para el análisis de datos. Los datos provienen de distintas fuentes y por tanto es heterogéneo en su contenido.
* **Veracidad.** Para algunas organizaciones la calidad de los datos **variables** es básica.
* **Valor** Se enfoca en lo significativos que son los datos almacenados para su análisis.

El Big Data no se pude alcanzar con herramientas convencionales como las bases de datos relacionales, sino que requiere la combinación de varias herramientas formando una arquitectura para el almacenamiento y análisis de los datos [4].

Existen múltiples arquitecturas y nuevas arquitecturas están surgiendo con el uso de la Nube, pero las más antiguas y extendidas son Kappa y su versión simplificada, Lambda [3] [4].

En la arquitectura Lambda se busca poder ejecutar consultas genéricas sobre todos los datos disponibles, tanto históricos como los que se van adquiriendo en tiempo real. Como el volumen de datos es muy grande, una manera de reducir el tiempo de las consultas es calcular los datos previamente y creando vistas sobre las que se realizan las consultas, pero como como son muchos datos y la creación de las vistas lleva tiempo, es muy probable que nuevos datos lleguen al sistema mientras se crean las vistas. Para hacer frente a este problema la arquitectura Kappa se divide en tres capas:

* Capa Batch. Que crea las vistas con los datos precalculados para las consultas mediante procesos por lotes y que también se encarga de persistir los datos según entran en el sistema.
* Capa de Tiempo real. Que se encarga de crear vistas incrementales sobre el flujo de datos que entra en tiempo real, que no son procesados por la capa batch. Estás vistas son temporales, ya que el procesamiento en tiempo real suele ser menos precisos por la naturaleza dinámica de los datos en tiempo real. Las vistas serán sustituidas por las vistas de la capa Batch cuando el proceso por lotes se vuelva a lanzar.
* Capa de Servicio. Contiene las vistas para ser consultadas.

La generación de las vistas en la capa Batch es muy precisa ya que dispone de todos los datos para poder crearlas y permite la tolerancia al fallo humano, ya que si el proceso por lotes para la creación de las vistas no era correcto se podría volver a lanzar, algo imposible en sistemas incrementales como las bases de datos relacionales [4].

El problema de la arquitectura Lambda es que exige el mantenimiento de dos repositorios de código, uno para la capa Batch y otro para la capa de Tiempo real. La complejidad radica en que las estructuras de datos de ambas capas deben permanecer sincronizadas. Para solucionar este problema surge la arquitectura Kappa que busca simplificar la arquitectura Lambda al quitar la capa Batch y quedarse solo con las capas de Tiempo real y Servicios, y por tanto solo existe un repositorio, simplificando las tareas de desarrollo como depuración o el mantenimiento. Por el contrario, arquitectura Kappa carece de la potencia de cómputo que tiene la capa Batch, y tareas como el entrenamiento de modelos de Machine Learning o la regeneración de las vistas se ven penalizados [5].

La elección de que arquitectura usar depende del tipo de uso que se le vaya a dar [3][5].

En el caso del análisis de vehiculos mediante la información obtenida por el puerto OBDII, como es el caso del PoC, el uso tanto de la capa de tiempo real (Streaming) como de la capa Batch permite obtener el mayor conocimiento de los datos ya que la ejecución de procesos de por lotes al poder trabajar sobre todos los datos históricos [8], por tanto la arquitectura Lambda sería la que mejor se ajustaría.

## Procesamiento Batch y Streaming

Sacar del proyecto, ver las referencias

## Spark

Streaming -> <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#output-modes>

## Kafka



Figura 2‑1: Logo EPS

# Diseño

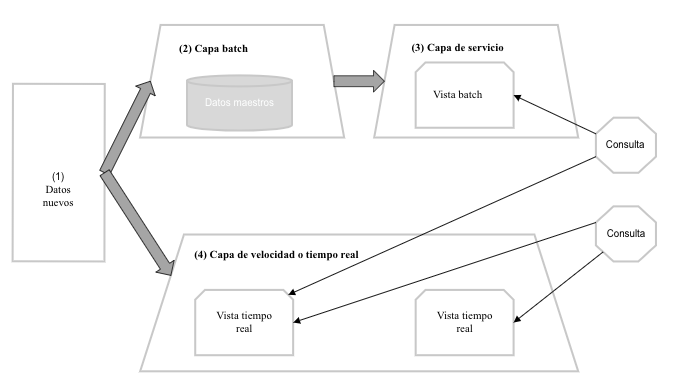
## Introducción

A continuación, se detalla la arquitectura Lambda, enumerando sus componentes y como estos intervienen en el procesamiento de los datos.

También se expondrán los casos de uso para realizar la prueba de concepto que probará la implementación de esta arquitectura con Spark.

Finalmente unimos los dos puntos anteriores explicando como aplicamos la arquitectura Lambda con Spark para llevar a cabo la prueba de concepto.

## Arquitectura Lambda



Aunque ya se ha introducido la arquitectura Lambda en apartado Estado del arte, al ser la arquitectura elegida para implementar la prueba de concepto, vamos a describirla en mayor detalle explicando lo que hace cada capa y sus componentes.

La arquitectura se compone de 2 flujos de datos distintos, uno corresponde a un flujo de datos potencialmente infinitos sobre el que se aplica un procesamiento en Streaming, y el segundo sobre un conjunto de gran volumen datos estáticos ya asentados al que se le aplica un procesamiento por lotes o batch. La combinación de ambos flujos permite trabajar con volúmenes de datos propios del BigData con latencias bajas. Los flujos son complementarios, ya que el procesamiento batch requiere de mucho tiempo y por tanto tiene una latencia alta que impide que mantener los datos actualizados, para solventar este problema se usa el procesamiento en Streaming sobre los datos que van llegando al sistema mientras aún se está ejecutando el procesamiento en batch. Aunque el procesamiento en Streaming puede ser imprecisos por la naturaleza de un sistema distribuido donde los datos se procesan en tiempo real, sus resultados son una buena aproximación hasta que estos datos son tenidos en cuenta por el procesamiento en batch y sustituidos por los resultados de éste. Para conseguirlo, la arquitectura Lambda se separa en 3 capas. Las capas Capa batch y Capa de servicio se ocupan del flujo de procesamiento sobre el gran volumen los datos estáticos, y la Capa de tiempo real se ocupa del flujo de procesamiento de los datos en tiempo real.

### Capa batch

Es responsable tanto de persistir los datos maestros o datos brutos asi como de ejecutar los procesos batch para crear las vistas que se almacenarán en la Capa de servicio para su posterior consulta. La persistencia de los datos maestros es inmutable, una vez guardados no se puede modificar, para evitar la corrupción de estos. Es fundamental que no se puedan cambiar porque así se previenen fallos humanos sobre el procesamiento para crear las vistas. Si el procesamiento no se implementó correctamente, las vistas se pueden borrar y volver a crear con una versión corregida.

Para la persistencia se suelen usar sistemas de archivos distribuidos como HDFS

Para el procesamiento batch se solía utilizar Hadoop MapReduce aunque ahora es más común usar Databricks, Apache Spark o Apache Flink.

### Capa de servicio

Se encarga de guardar las vistas generadas en la Capa batch y permitir su consulta de forma eficiente. Se suelen usar bases de datos NoSQL que fomenten la lectura de datos por encima de la escritura ya que los datos no se actualizarán, si es necesario se borrarán y se volverán a crear. Algunos ejemplos son: Cassandra, CouchDB ...

### Capa de tiempo real

Esta capa se encarga de procesar los datos que entran en el sistema en tiempo real para estos se puedan consultar mientras la Capa batch está generando las vistas que aún no están disponibles en la Capa de servicio para ser consultadas. Algunas de las tecnologías que se pueden usar en esta capa son: Spark (Structured Streaming), Flink, Storm, Kafka Stream

## Prueba de concepto

Para mostrar cómo sería la implementación de la arquitectura Lambda usando Spark vamos a basarnos en aplicación IoT en la que se instalan, al puerto OBDII de vehiculos de combustión, unos dispositivos que leen los datos disponibles de los propios sensonres de telemétria del vehículo y transminten esos datos en un formato unificado basado en JSON. La unidad de información la llamaremos Frame y está compuesta por los valores actuales de los sensores, la fecha de recolección por parte del dispositivo y la fecha de entrada en el sistema. Más detalle en el apendice, apartado Modelo de datos, sección Frames. El tamaño medio de los frames es 1,4KB, dependiendo de los sensores disponibles en el vehículo, que se suelen transmitir con una frecuencia de uno por minutos aproximadamente, generando unos 2 MB al día por vehículo.

El análisis lo realizaremos sobre la manera de conducir del conductor, para ello definiremos los trayectos que ha realizado el conductor, y los etiquetaremos para poder comparar los trayectos con la misma etiqueta entre sí. Finalmente crearemos eventos en base a la conducción uno de ellos según el porcentaje de aceleración y otro en base al consumo brusco de combustible.

### Casos de uso

#### Creación de trayectos

Un trayecto queda determinado por los cambios del sensor de la llave de ignición. Cuando ésta pasa de apagado a encendido se considera que se ha iniciado el trayecto, cuando pasa de encendido a apagado se considera que ha terminado. Dentro del trayecto no interesa:

* La localización y fecha del inicio del trayecto.
* La localización y fecha del final del trayecto.
* La distancia total recorrida.
* El consumo total realizado.

#### Etiquetado de trayectos.

El usuario etiqueta una serie de trayectos ya realizado para entrenar un modelo de Machine Learning que será usado para etiquetar trayectos futuros.

#### Creación de evento de exceso de aceleración.

Se usará el valor del sensor correspondiente al acelerador para determinar que no se supera un umbral.

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

Se usará el valor del sensor del nivel de combustible para determinar que no sufre una bajada mayor a un determinado umbral dentro de un rango de tiempo.

## Arquitectura aplicada

METER DIAGRAMA COMO SE RELACIONAN. ¿EXPLICAR LO QUE SON Y SU RELACIÓN?

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.

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

Después, se detalla la implementación de cada caso de uso. Por cada caso de uso se indica la capa implicada de la arquitectura, así como el origen y destino de los datos, la estructura de datos de datos de entrada y salida de cada transformación, y la lógica de la transformación y sus tests unitarios que ayudan a su compresión. Se hace referencia al repositorio que contiene un archivo docker-compose para levantar el entorno y una serie de scripts para ejecutar la prueba, todo ello detallado en el README.md del repositorio.

Una vez que has contado la arquitectura en el apartado anterior, aquí detallas qué has ido haciendo para implementarla

Incluyes también la recopilación de datasets, el análisis de datos que hayas hecho sobre ellos, y todos los pasos que hayas necesitado para tener listo el sistemaSi has hecho algo que no haya funcionado bien por alguna razón (y que hayas tenido que cambiar), también lo cuentas aquí, porque es trabajo realizado

## 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 de tiempo real

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.

## Pruebas

???

## Resultados

En la parte de resultados lo que tienes que sacar es estadísticas (si son en forma de gráficas mejor) de las pruebas que hayas hecho. Si has entrenado un modelo, valores de precisión, etc sobre el dataset de test

# 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. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. IEEE Communications Surveys Tutorials, 17(4) Pag:2347-2376. <https://ieeexplore.ieee.org/document/7123563>
2. Keyur K Patel, Sunil Patel, Carlos Salazar. Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges. IJESC Volumen 6 Issue 5. Pag:6122-6131 <https://www.researchgate.net/publication/330425585>
3. Singh, Kamakhya & Behera, Rajat & Mantri, Jibendu. (2019). Big Data Ecosystem: Review on Architectural Evolution: Proceedings of IEMIS 2018, Volume 2 <https://www.researchgate.net/publication/327389927_Big_Data_Ecosystem_Review_on_Architectural_Evolution_Proceedings_of_IEMIS_2018_Volume_2>
4. Nathan Marz. Big Data: Principles and best practices of scalable realtime data systems. Manning Publications. Pag: 25-61
5. Martin Feick, Niko Kleer, and Marek Kohn (Hrsg.): SKILL 2018, Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2018 <https://dl.gi.de/bitstream/handle/20.500.12116/28983/SKILL2018-05.pdf?sequence=1&isAllowed=y>
6. Ian L.Dryden, David J Hodge. Estimating Vehicle Fuel Consumption and Emissions using GPS BigData. Int. J. Environ. Res. Public Health 2018, 15(4), 566.
7. Ali, Muhammad and Kamal, Muhammad D. and Tahir, Ali and Atif, Salman. Fuel Consumption Monitoring through COPERT Model—A Case Study for Urban Sustainability,,2021, 13, 21
8. Ian L.Dryden, David J Hodge, Journeys in big data statistics, Statistics & Probability letters, Volume 136, May 2018, PAges 121-125.
9. C. Prehofer and S. Mehmood, "Big Data Architectures for Vehicle Data Analysis," *2020 IEEE International Conference on Big Data (Big Data)*, Atlanta, GA, USA, 2020, pp. 3404-3412

# Apéndice

## Modelos de datos

### Frame de información de los dispositivos IoT

Corresponde con un archivo JSON, donde se pueden encontrar 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”. Identificador 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 recibió 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.

## Entorno – Docker

## Tests de aceptación

## Test unitarios

## Codigo