ICEMD
INSTITUTO ECONOMÍA DIGITAL | ESIC

ID de alumno :

137915

PROYECTO

MBD Módulo 2

Caso Práctico individual

Diseño de arquitectura Big Data:

BiciMad

Índice

❖Introducción:

En las siguientes líneas, plantearemos una arquitectura Big Data, para la empresa pública BiciMad, detallando, los datos con los que contamos, las tecnologías utilizadas y como fluyen los datos a través de dicha arquitectura.

Uno de los principales casos de uso con los que daremos solución con la arquitectura planteada, será la generación de KPI´ s, para tener la capacidad de monitorizar el negocio, mediante cuadros de mandos.

Generaremos dos tipos de dashboards:

1. Alimentados por datos dinámicos, con los que controlaremos en tiempo real el estado y grados de ocupación de las estaciones, para además poder generar en un caso hipotético en el que los empleados de BiciMad usasen estos informes para resolver incidencias en las estaciones o mover bicis entre estaciones según el grado de ocupación.

2. Alimentados por datos estáticos, con estos elaboraremos estadísticas e informes, con el objetivo final de optimizar el negocio y ofrecer un servicio de calidad.

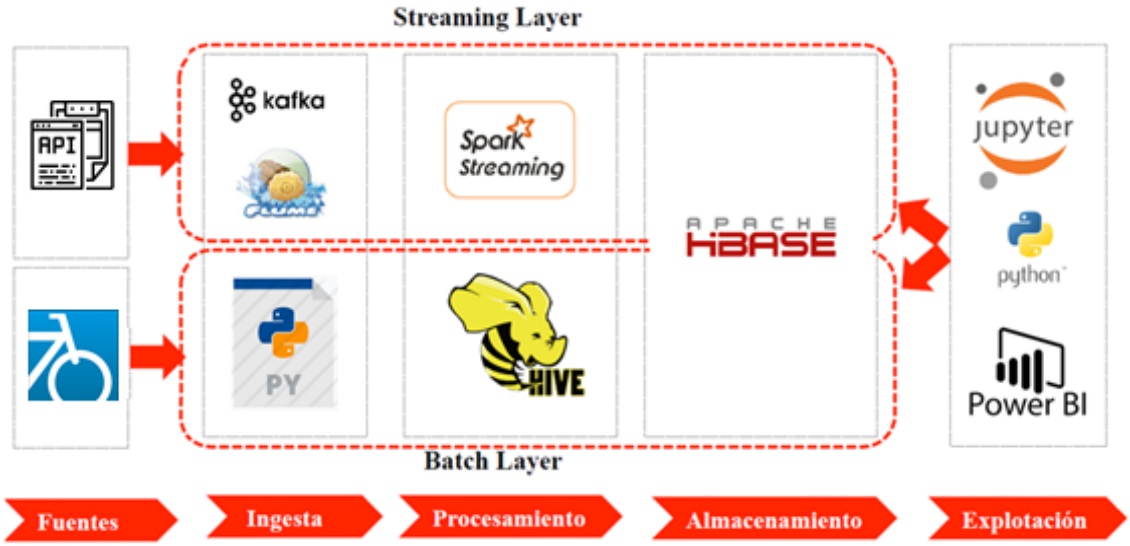
❖Modelo de Negocio:

BiciMad es un sistema público de trasporte basado en el alquiler de bicicletas eléctricas según pago por uso o “bike sharing”, donde se ponen a disposición de un grupo de usuarios una serie de bicicletas, para que sean compartidas como medio de transporte temporal.
El usuario puede acceder a las bicicletas en una de las estaciones que la empresa tiene esparcidas por el centro de Madrid y devolverla en otro punto.

Entre las características de BiciMad está emplear nuevas tecnologías con el fin de mejorar la experiencia del cliente, entre ellas está la capacidad del usuario de interactuar con el servicio desde la app, web o tótem de cada estación y a través de estos, poder reservar, y aprovechar las bonificaciones que ofrecen, pues si devuelves una bicicleta en una estación con “baja ocupación” o la retiran de una estación de “alta ocupación” bonifican al usuario con una reducción del coste del viaje, está es una forma de que el propio usuario autorregule el servicio, para evitar las acumulaciones de bicicletas en ciertas zonas y la falta de ellas en otras.

❖Arquitectura:

La arquitectura que se desarrollara posteriormente quedaría estructurada de la siguiente manera:



❖ Fuentes de datos:

Contaremos con datos de carácter estático y dinámico:

i) **<u>Datos dinámicos:</u>** Son datos generados en tiempo real por los sensores instalados en las estaciones de carga.
El acceso a estos datos será a través de la API, puesta a disposición por la EMT. Para tener acceso deberemos rellenar un formulario y se nos proporcionará Id de cliente y Pass Key.
A partir de la API, obtendremos acceso al estado y la disponibilidad tanto de las estaciones como de las bases de carga de BiciMad, en tiempo real.
Desde la documentación oficial de la API, se detallan las posibles invocaciones a la misma, pudiendo realizar dos:
-**get_stations:** Obteniendo como respuesta un XML con la relación de todas las estaciones de BiciMad y su estado.

-**get_single_station:** El formato devuelto y la información contenida será similar al anterior método, salvo que en este caso tendremos los datos de una única estación. Quedando de la siguiente manera:

<?xml version="1.0" encoding="ISO-8859-1"?>
<stdRet xmlns="http://schemas.datacontract.org/2004/07/BiciMadTools.Code" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
<code>0</code>
<data i:type="d2p1:string" xmlns:d2p1="http://www.w3.org/2001/XMLSchema">{ "stations": [{ "id": 145, "latitude": "40.4480662", "longitude": "-3.6952860", "name": "Orense 12", "light": 2, "number": "151", "address": "Calle Orense nº 12", "activate": 1, "no_available": 0, "total_bases": 24, "dock_bikes": 8, "free_bases": 16, "reservations_count": 0 }]}</data>
<description>ok</description>
<time>26-02-2019 00:29:00.674</time>
<version>3.0.0 (posBiciMad)</version>
<whoAmI>BiciMad</whoAmI>
</stdRet>

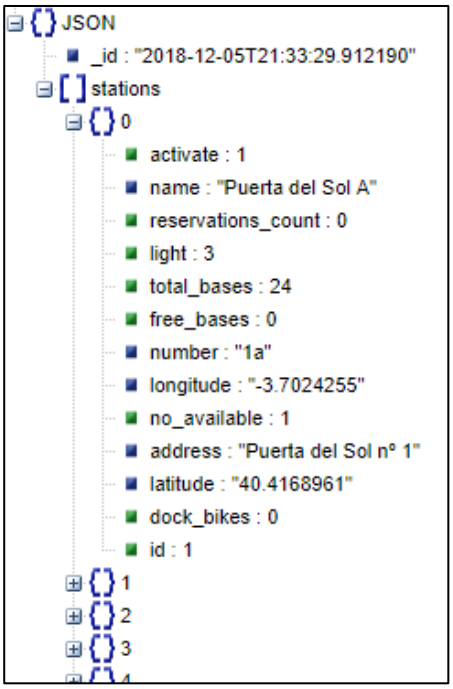
Este será la respuesta de la API a un get_single_station, desde donde obtendremos la siguiente información.

-id: Código de la Estación Base
-latitude: Latitud de la estación en formato WGS84
-longitude: Longitud de la Estación en formato WGS84
-name: Nombre de la Estación
-light: Grado de Ocupación (0=baja, 1=media, 3=alta)
-number: Denominación lógica de la Estación Base
-actívate: Estación activa (0=No activa, 1=activa)
-no_available: Disponibilidad de la Estación (0=disponible, 1=no disponible)
-total_bases: Número de bases de la estación
-dock_bikes: Número de bicicletas ancladas
-free_bases: Número de bases libres
-reservations_count: Número de reservas activas

Uso de los datos: -A partir de esta información será sobre la cual se generarán mapas, para plasmar el estado del servicio y la disponibilidad de bases y anclajes o generar alerta.

ii) **<u>Datos estáticos:</u>** Son datos almacenados en formato JSON, generados con carácter mensual y volcados en el portal de Open Data de EMT. Dentro de este repositorio existen dos tipos de datos disponibles:

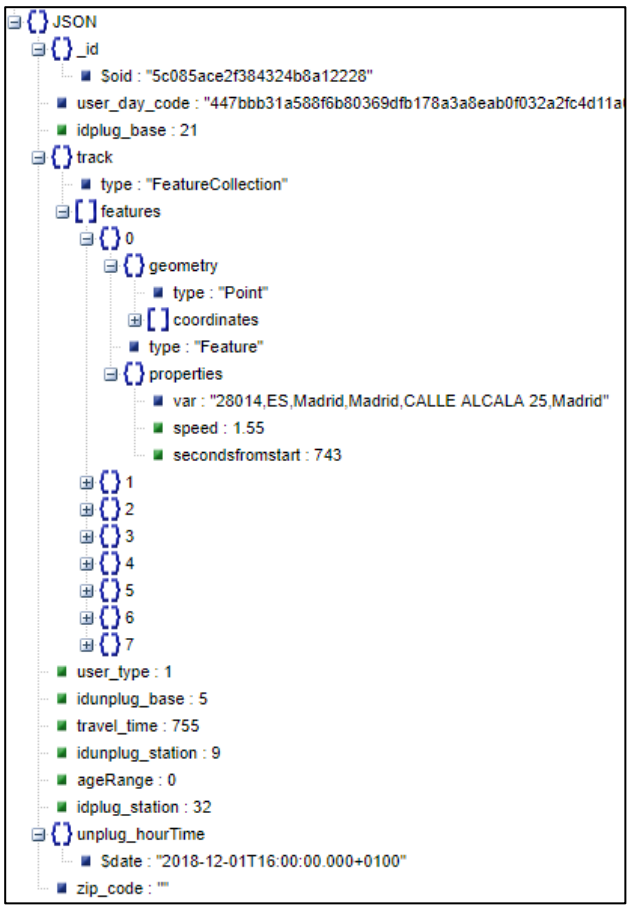
1) Situación estaciones BiciMad por día y hora: Con información acumulada del estado de las estaciones y cómo evolucionan a lo largo del mes.
La información es similar a la que se pone a disposición vía API, pero en este caso en vez de en tiempo real es acumulativa del mes, detallándose en cada una de las líneas del JSON, el "_id", que corresponde a la fecha y hora exacta, en la que se generó el pin.
El archivo contiene el estado de todas las estaciones del servicio de BiciMad.



2) Datos de uso: con información relativa a los movimientos de las bicicletas, incluyendo datos relativos al viaje y relativos al usuario, previamente anonimizados, para no violar su privacidad.

-Datos de usuario: id (anonimizado), franja de edad, perfil de usuario en base al tipo de uso y abono (anual, ocasional, empleado BiciMad).

-Datos de viaje: Fecha, estación y base de origen y destino, hora de inicio y duración del viaje y un track en formato GeoJSON, acerca del viaje.



Uso de los datos: Con esta tipología de datos de carácter estático, basado en la historificación, se pretende elaborar dashboards y aprendizajes acerca del servicio. Los datos serán modelados obteniendo propensiones de que estaciones tienen más incidencias y que rutas o estaciones son las más frecuentes, para anticiparnos a las incidencias o picos de demanda del servicio en algunas zonas y mejorar la experiencia de usuario.

❖Ingesta:

Una vez realizado un análisis exploratorio de los datos y comprobada su dimensión temporal, pasaremos a ingestarlos, conectando las herramientas del ecosistema Big Data con los entornos donde se encuentran los datos.

Para la parte de **datos dinámicos**, nos conectaremos a la API de BiciMad con Flume. Este es especialmente eficiente en la recolección y agregación de Logs, siendo clave la capacidad de conectarse a diversos orígenes de datos y tomar los datos. Entre otro de sus fuertes, destaca la escalabilidad vertical y horizontal.
Levantaremos un agente de Flume y que estará recolectando en tiempo real, cuyo source es la API pública de BiciMad, el destino de los datos, será Kafka, esto será especificado en el archivo de configuración de Flume.

Kafka se caracteriza por su escalabilidad y es especialista en gestionar grandes volúmenes de eventos, sirviendo de amortiguador entre el origen de los datos y el procesamiento, además de esto, Kafka, garantizando la entrega de mensajes, garantizando una persistencia en el origen de los

datos. Kafka es el bus o canal encargado en conectar productores (Flume) con consumidores (Procesamiento) garantizando la entrega, en una latencia baja, sin la necesidad de una infraestructura dedicada.

Los **datos estáticos** son liberados mensualmente en el portal de Open Data de la EMT, como hemos comentado vienen estructurados en formato JSON y serían ingestados por un proceso desarrollado en Python que invocaremos puntualmente y volcaría estos datos en nuestra arquitectura BigData siendo directamente almacenados, para llevar a cabo un procesamiento posterior en Batch.

❖ Procesamiento :

En este punto de la arquitectura los datos procedentes de las herramientas de ingesta, son manipulados y tratados para producir información significativa.
En la línea de lo expuesto, tendremos dos tipos de procesamiento:

1. En real time, donde se da un procesado continuado de la información según es generada, el procesamiento tiene lugar en memoria, previo a ser almacenado en disco.
2. Después de ser almacenada en un sistema distribuido, se da un segundo tipo de procesamiento, Batch, para hacer consultas más específicas, que posteriormente viajarán hacia los sistemas de explotación para mostrar las variaciones en los KPI´ s.

Para el **procesamiento en tiempo real** utilizaremos Spark Streaming, esta es la API de Spark para el procesamiento de datos en tiempo real, de forma escalable, con alto rendimiento y tolerante a fallos. Entre los beneficios de Spark también destacan, la velocidad de procesamiento, siendo 100 veces más rápido que Hadoop en el procesamiento en memoria y 10 veces más veloz en procesamiento en disco, frente al icónico MapReduce de Hadoop.

Spark Streaming se conectará con Kafka, que es el sistema distribuido basado en colas de mensajes, la API de Spark, es la encargada de gestionar el flujo de datos, coordinando la creación de trabajos, pues Spark Streaming trabaja en micro batches inferiores a 0.5 segundos y lanza los trabajos sobre la core de Spark, que es la encargada realmente del procesado de los datos. La conexión con Kafka es nativa, incluyendo las siguientes librerías en nuestro código en Scala:

```
import org.apache.spark.streaming.kafka._
  val kafkaStream = KafkaUtils.createStream(streamingContext,
    ZK quorum], [consumer group id], [per-topic number of Kafka partitions to consume])
```

En la arquitectura de BiciMad en ese procesamiento en real time que va a realizar Spark Streaming, se ejecutaran acciones de tratamiento de datos en base a la agregación de información, procedente del estado de las estaciones, que es la información dinámica que recibimos procedente de la API.
Desde el propio Spark, seremos capaces de generar KPI´ s en tiempo real, del estado de las estaciones y del grado de ocupación.

La información que será almacenada en un sistema distribuido, será una agregación del estado de las estaciones a modo de histórico (especificado más adelante), en base a las respuestas de la API y un cálculo simplificado del estado de cada estación y del número de bases disponibles, por cada consulta a la API.

La información dinámica, que nos parece más interesante será*: id, no_available, total_bases, dock_bikes, free_bases y reservations_count*, esta es la información que utilizaremos en la explotación para plasmar la situación del servicio y lanzar alertas a hipotéticos operarios en caso de incidencia.

El **procesamiento Batch**, se da tras almacenar la información, recordemos que esta información es liberada mensualmente, y nuestro proceso desarrollado en python, nos almacena los nuevos datos liberados en nuestro sistema distribuido, siendo desde aquí desde donde cargaremos los datos, a las tablas de Hive que especifiquemos.

Para esta información estática utilizaremos Hive, que facilita la ejecución de consultas sobre el almacenamiento pues sirve de frontal, en el cual escribir peticiones en lenguaje HiveQL, muy similar a SQl., tras esto, Hive transforma esta consulta en un comando MapReduce, ejecutado sobre el almacenamiento distribuido y por tanto paralelizando el trabajo.

Cargaremos en las tablas de Hive, de nuestro Database, los nuevos JSONs liberados en el OPENDATA, una de las funciones por la que elegimos Hive, es porque tiene la potencialidad de poder leer directamente de un JSON, sin la necesidad de delimitar los campos ni realizar otros procesamientos previos, mediante los comandos tipo "*SELECT GET_JSON_OBJECT …"*

Con Hive lanzaremos consultas sobre los datos históricos almacenados tanto de las estaciones como de los usuarios, para generar KPI´ s. Esta información histórica es complicada procesarla en real time, por ello resulta más sencillo, procesarla en Batch.

La información estática que consultaremos en Hive, con la que elaboraremos cuadros de mandos será, por un lado referente al perfil de usuario *(user_day_code, user_type, travel_time, age_range)*, al viaje *(track,*

id_unplugin_station, id_plugin_station, travel time) y por otro lado volveremos a realizar consultas acerca del estado del servicio, ya no a modo de alertas, si no con el fin de conocer las estaciones más propensas a saturarse, averiarse, etc.

❖ Almacenamiento :

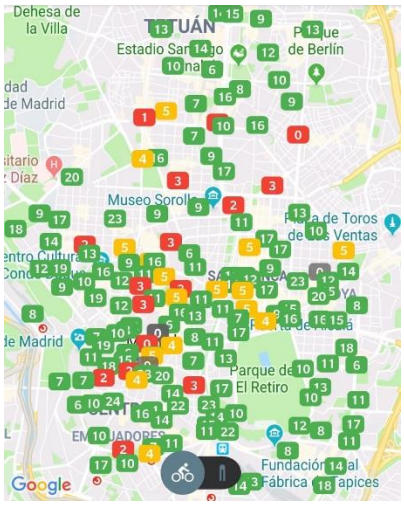
Tras procesar la información se almacenará en HBase, que es un tipo de Base de datos NoSQL de tipo clave-valor, organizando los datos en familias de columnas, con latencia baja, el cual comparte el sistema de archivos de Hadoop, pues almacena los datos en HDFS, por lo que por debajo lo hace dependiente de Hadoop.
Las bondades de HBase residen en esa capacidad de consultar información near real time y en la capacidad de almacenar grandes volúmenes de datos.

En HBase se volcará la información procesada por Spark Streaming acerca del estado de las estaciones y los anclajes, además de la información estática mensual, de datos de uso y situación de las estaciones.

Con el fin de no sobrecargar HBASE, borraremos recurrentemente parte de la información dinámica que nos iba llegando. Recordemos que nos conectamos periódicamente a la API para conocer el estado de las estaciones, tras esto íbamos agregando la información, pero estos datos del estado de las estaciones son los mismos que nos llegan en uno de los JSONS, como datos estáticos, pues el JSON de "Situación de las estaciones por fecha y hora" es el mismo que la agregación que realizamos, por lo que para evitar duplicaciones innecesarias (más allá de la replicación que realiza el propio sistema), la información duplicada será eliminada, con los convenientes tiempos para asegurar no perder información.

❖ Explotación:

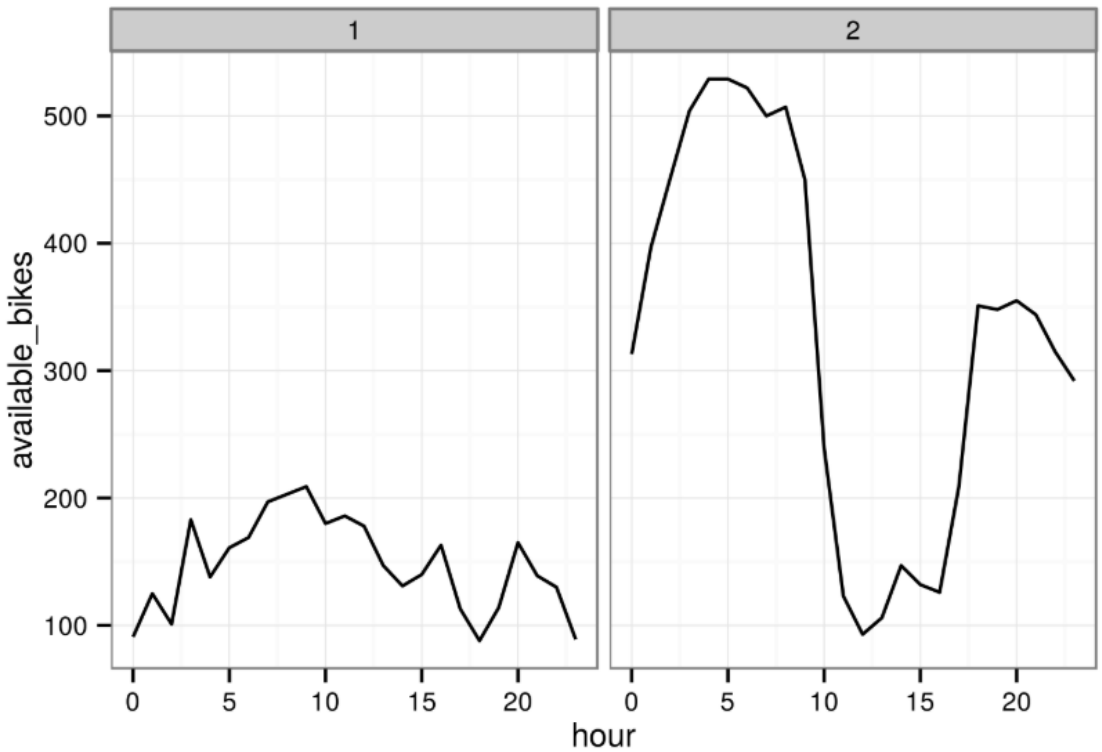
Con los **<u>datos dinámicos,</u>** se propondrá una monitorización en tiempo real del servicio, con el fin de conocer el número de bicicletas disponible para cada estación y el número de anclajes libres. Para este tipo de análisis, utilizaremos Power BI, por la capacidad de conectarse a múltiples orígenes de datos, pudiendo conectarse directamente con Spark o con Hadoop (sobre el que se monta HBase) y por lo intuitiva que es la herramienta,



volviéndose relativamente sencillo implantar cuadros de mando.

Por otra parte, con la información que extraemos de los datos podríamos montar un sistema de alertas, para de manera hipotética, poder avisarnos de cuando hay estaciones caídas o incidencias con el servicio.

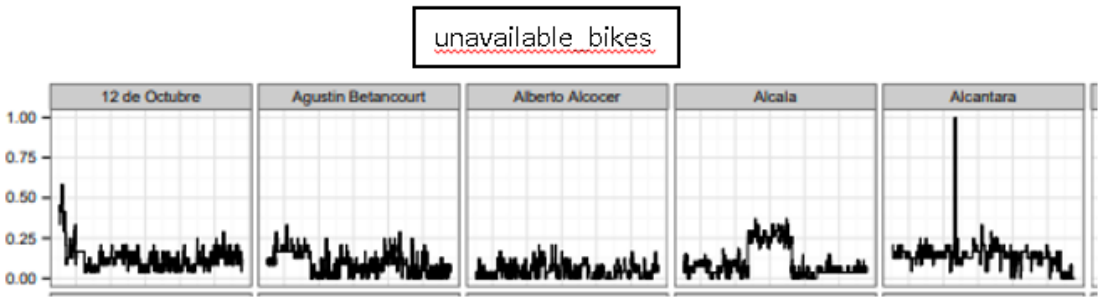
Por el camino de los **datos estáticos**, se propondrá montar también con Power BI, cuadros para el reporte del negocio. Dado que la herramienta tiene la capacidad de conectarse con Hive, volcaremos sobre ella el resultado de nuestras consultas, elaborando una monitorización del negocio, extrayendo del mismo conclusiones como:
-Número de usuarios diarios del servicio.
-Índice de repetición (en el mismo día) del servicio.
-Establecer picos de demanda, en base al número de usuarios por momentos.
-Viajes más frecuentes.
-Top de las estaciones con mayor/menor grado de ocupación.
-Índices de usos por tipologías de usuario.



Esta información parece muy interesante en el caso de que estuviésemos en la posición de BiciMad, primero para entender nuestro negocio y segundo para intentar mejorarlo y anticipar picos de demandas.

Con la misma finalidad, montaremos análisis, mas ad-hoc para ir un poco más allá acerca del servicio, para ello utilizaremos Jupyter, que lo programaremos con Python e implementaremos librerías de machine learning, siendo uno de los fines tratar de encontrar correlaciones entre

distintos eventos que ocurren en la ciudad y picos o descensos de demanda del servicio, como pudiesen ser conciertos, partidos, lluvias, etc. Además de una monitorización más profunda del servicio, pudiendo encontrar patrones o agrupaciones de estaciones en función del tipo de uso, con la capacidad de bajar más al detalle.



❖ Conclusiones:

La arquitectura Big Data planteada, es capaz de dar solución de manera consistente a los problemas planteados, pues es lo suficientemente flexible como para por un lado (streaming) recolectar la información del estado del servicio en tiempo real y por otro (Batch), realizar análisis del negocio en base a los datos de uso y la situación del servicio de manera agregada.

Además de esto, con esta arquitectura, seriamos capaces de pivotar hacia otros posibles casos de uso, siendo la arquitectura lo suficientemente flexible para ello.

❖ Bibliografía:

→ https://datos.madrid.es/FWProjects/egob/Catalogo/Transporte/Bici/Ficheros/Bicimad_Estructura_FICHERO_DATOS.pdf
→ https://opendata.emtmadrid.es/Documentos/Servicios-y-estructuras-Bicimad-V1-1.aspx
→ https://opendata.emtmadrid.es/getdoc/2f7fdbf1-f849-4357-9778-cbd5c4ebc27c/default.aspx
→ https://spark.apache.org/docs/2.1.2/streaming-kafka-0-8-integration.html
→ https://rinzewind.org/blog-es/2015/analisis-de-datos-de-bicimad.html
→ https://blog.cloudera.com/blog/2014/11/flafka-apache-flume-meets-apache-kafka-for-event-processing/
→ https://www.esmadrid.com/sites/default/files/documentos/bicimad24.pdf
→ https://webcache.googleusercontent.com/search?q=cache:SLSbeC5fTCcJ:https://profile.es/blog/creando-microservicios-de-tiempo-real-con-kafka-streams/+&cd=10&hl=es&ct=clnk&gl=es
→ https://sg.com.mx/revista/52/arquitectura-lambda-combinando-lo-mejor-dos-mundos
→ https://ocw.unican.es/pluginfile.php/2396/course/section/2473/tema%203.2%20Arquitecturas%20y%20tecnologi%CC%81as%20para%20el%20big%20data.pdf

Camino de Valdenigrales, s/n • 28223 Pozuelo de Alarcón
Tel 902 918 912 • Fax 91 351 56 20

w w . i c e m d . c o m