

ID de alumno :
137915

PROYECTO

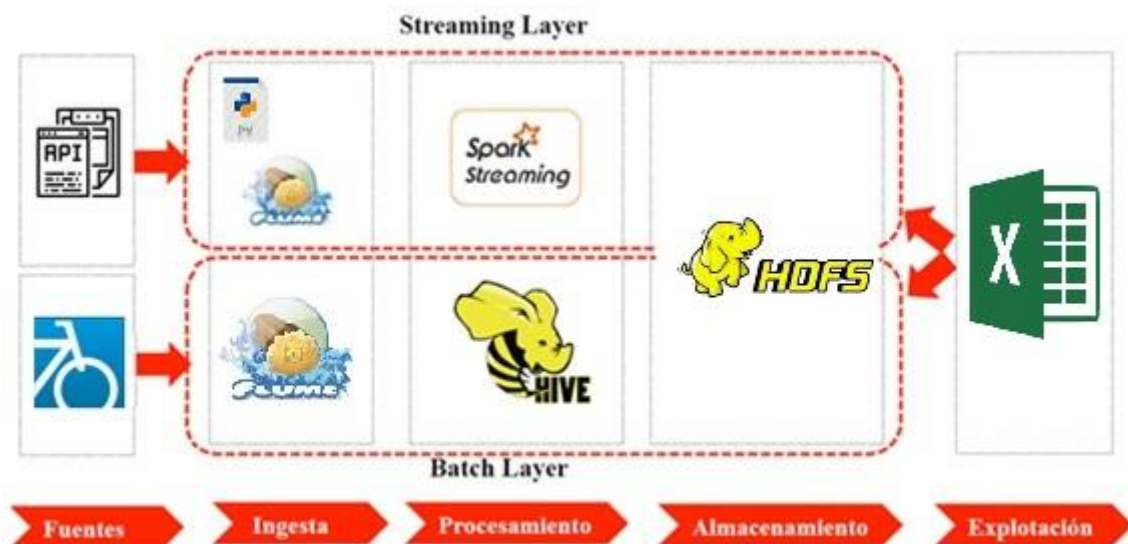
MBD Módulo 3

Caso Práctico individual

Diseño de arquitectura Big Data:
BiciMad

Anexo

La descripción de la nueva arquitectura será la siguiente:



En el funcional se explica y argumenta con pantallazos como se describe el flujo de datos, pero a grandes rasgos hemos excluido kafka y añadido Flume en el batch Layer.

HBase, que en principio era nuestro sistema de almacenamiento, ha sido sustituido por HDFS, pero en próximos módulos, espero poder volver a incluirlo una vez entendido su funcionamiento.

Explotación que en principio iba a ser con Jupyter y Power BI, ha sido sustituido por Excel, esto es debido a que era la única herramienta en la que no daba problemas de permisos al crear una conexión con Hive, vía ODBC.

Los cuadros de mando son creados de igual manera en esta herramienta y son mostrados en el funcional, aunque no los incluiremos en el archivo por el límite de 25MB que tenemos para subir archivos. Funcionan igual que lo harían en Power BI trabajando con un modelo de datos por detrás haciendo dinámicos los datos y disponiendo de botoneras para poder sacar más insights de los datos.

Funcional

Batch Layer:

Descargamos los datos estáticos de la web mediante un wget, pero para no saturar el cluster descargamos solo un mes de los datos de uso y estaciones.

En el caso de tener permisos **sudo** y suficiente espacio, configuraríamos un cron en el que mediante un **wget** descargue todos los archivos zip de la web y revise diariamente la web en busca de nuevos archivos.

Para no saturar el cluster, descargaremos únicamente un archivo de uso de estaciones y otro de datos de uso. Estos serán almacenados en el nodo frontera del cluster, en las carpetas:

sftp://35.242.236.241/home/acerrato/m3_indiv/def_data/station
sftp://35.242.236.241/home/acerrato/m3_indiv/def_data/usage

La arquitectura creada, será propuesta de tal manera, que la actualización de datos sea un proceso relativamente sencillo.

1. Ingesta:

Los archivos almacenados serán ingestados por Flume y crearemos dos configuraciones, para que se dejen en carpetas independientes.

Los trataremos de manera diferente, para no mezclar los datos y optimizar la ingesta, los archivos de configuración serán:

flume_bicimad_usage.conf

Agent1.sources = spooldir-source
Agent1.channels = memory-channel
Agent1.sinks = hdfs-sink

###Describe/configure Source
Agent1.sources.spooldir-source.type = spooldir
###Carpeta de origen
Agent1.sources.spooldir-source.spoolDir = /home/acerrato/m3_indiv/def_data/usage
###Longitud maxima de una linea
Agent1.sources.spooldir-source.deserializer.maxLineLength = 100000
###Codigo para dejarlo en formato JSON y no ensuciar el contenido del archivo
Agent1.sources.spooldir-source.hnadler = org.apache.flume.source.http.JSONHandler

###Describe the sink
Agent1.sinks.hdfs-sink.type = hdfs
###Carpeta de destino en HDFS
Agent1.sinks.hdfs-sink.hdfs.path = /user/acerrato/bicimad_m3/data/usage
###Dejados en formato Text, fundamental para despues ser tratados con Hive
Agent1.sinks.hdfs-sink.hdfs.writeFormat = Text
###Tamaño max de archivo 128MB aprox, lo optimo para HDFS
Agent1.sinks.hdfs-sink.hdfs.rollSize = 128000000
###Numero max de filas, en funcion del archivo, optimizado para que se aproxime
###tambien a 128MB, tambien podria haber tomado valor 0
Agent1.sinks.hdfs-sink.hdfs.rollCount = 60000
###Nombre del archivo de salida
Agent1.sinks.hdfs-sink.hdfs.filePrefix = usage-
Agent1.sinks.hdfs-sink.hdfs.round = true
Agent1.sinks.hdfs-sink.hdfs.roundValue = 10
Agent1.sinks.hdfs-sink.hdfs.roundUnit = minute

###Use a channel which buffers events in memory
Agent1.channels.memory-channel.type = memory
Agent1.channels.memory-channel.byteCapacity = 6912212
Agent1.channels.memory-channel.transactionCapacity = 100

###Bind the source and sink to the channel
Agent1.sources.spooldir-source.channels = memory-channel
Agent1.sinks.hdfs-sink.channel = memory-channel

flume_bicimad_stations.conf

Agent1.sources = spooldir-source
Agent1.channels = memory-channel
Agent1.sinks = hdfs-sink

###Describe/configure Source
Agent1.sources.spooldir-source.type = spooldir
###Carpeta de origen
Agent1.sources.spooldir-source.spoolDir = /home/acerrato/m3_indiv/def_data/station
###Longitud maxima de una linea
Agent1.sources.spooldir-source.deserializer.maxLineLength = 100000
###Codigo para dejarlo en formato JSON y no ensuciar el contenido del archivo
Agent1.sources.spooldir-source.hnadler = org.apache.flume.source.http.JSONHandler

###Describe the sink
Agent1.sinks.hdfs-sink.type = hdfs
###Carpeta de destino en HDFS
Agent1.sinks.hdfs-sink.hdfs.path = /user/acerrato/bicimad_m3/data/stations
###Dejados en formato Text, fundamental para despues ser tratados con Hive
Agent1.sinks.hdfs-sink.hdfs.writeFormat = Text
###Tamaño max de archivo 128MB aprox, lo optimo para HDFS
Agent1.sinks.hdfs-sink.hdfs.rollSize = 128000000
###Numero max de filas, en funcion del archivo, optimizado para que se aproxime
###tambien a 128MB, tambien podria haber tomado valor 0
Agent1.sinks.hdfs-sink.hdfs.rollCount = 4000
###Nombre del archivo de salida
Agent1.sinks.hdfs-sink.hdfs.filePrefix = station-
Agent1.sinks.hdfs-sink.hdfs.round = true
Agent1.sinks.hdfs-sink.hdfs.roundValue = 10
Agent1.sinks.hdfs-sink.hdfs.roundUnit = minute

###Use a channel which buffers events in memory
Agent1.channels.memory-channel.type = memory
Agent1.channels.memory-channel.byteCapacity = 6912212
Agent1.channels.memory-channel.transactionCapacity = 100

###Bind the source and sink to the channel
Agent1.sources.spooldir-source.channels = memory-channel
Agent1.sinks.hdfs-sink.channel = memory-channel

executeFlume_usage.sh

#!/usr/bin/env bash

export JAVA_OPTS="-Xms100m -Xmx2000m -Dcom.sun.management.jmxremote"
.. apache-flume-1.9.0-bin bin flume-ng agent \
-f /home/acerrato/m3_indiv/flume_indiv/ejercicio1/conf/flume_bicimad_usage.conf \
-name Agent1 \
-Dflume.root.logger=INFO,console

Para enviar datos:

\$s nc localhost 1937
hblblalalabla
OK
dlsakjdlsakjdsalk
OK
#####

executeFlume_stations.sh

#!/usr/bin/env bash

export JAVA_OPTS="-Xms100m -Xmx2000m -Dcom.sun.management.jmxremote"
.. apache-flume-1.9.0-bin bin flume-ng agent \
-f /home/acerrato/m3_indiv/flume_indiv/ejercicio1/conf/
flume_bicimad_stations.conf \
-name Agent1 \
-Dflume.root.logger=INFO,console

Para enviar datos:

\$s nc localhost 1937
hblblalalabla
OK
dlsakjdlsakjdsalk
OK
#####

Tras ser ingestados, estos se localizan en :
/user/acerrato/bicimad_m3/data/usage
/user/acerrato/bicimad_m3/data/stations

2) Procesamiento:

Iniciamos Hive y nos disponemos a cargar en tablas los archivos que ingestó Flume, con los siguientes comandos cargamos los archivos en tablas y los estructuramos a nuestro gusto, para ser tratados después.

-Datos de uso:

```
hive
hive_usage
API_bicimad_2.py
1 show databases;
2
3 use cerrato;
4
5 show tables;
6
7 CREATE EXTERNAL TABLE usage_bicimad (data String);
8
9 LOAD DATA INPATH "/user/acerrato/bicimad_m3/data/usage" INTO TABLE usage_bicimad;
10
11 SELECT id, day, start_station, end_station, travel_time, user_type, age
12 from usage_bicimad
13 LATERAL VIEW json_tuple (data, 'id','unplug_hourTime','user_day_code','idunplug_station','idplug_station','travel_time','user_type','ageRange')
14 basic as idd, da, user_day, start_station, end_station, travel_time, user_type, age
15 LATERAL VIEW json_tuple (basic.idd, 'soid')
16 bb as id
17 LATERAL VIEW json_tuple (basic.da, '$date')
18 d as day
19 limit 5;
INFO : Completed executing command(queryId=hive_20190324152750_7c6c43f7-522d-4387-b879-b2c6754dfbf9); Time taken: 0.011 seconds
INFO : OK
+-----+-----+-----+-----+-----+-----+-----+
| id | day | start_station | end_station | travel_time | user_type | age |
+-----+-----+-----+-----+-----+-----+-----+
| 5c4b07ea2f38432e007daab8 | 2019-01-01T00:00:00.000+0100 | 82 | 83 | 162 | 1 | 0 |
| 5c4b07ea2f38432e007daaba | 2019-01-01T00:00:00.000+0100 | 133 | 36 | 415 | 1 | 5 |
| 5c4b07ea2f38432e007daabb | 2019-01-01T00:00:00.000+0100 | 25 | 90 | 545 | 1 | 0 |
| 5c4b07ea2f38432e007daabd | 2019-01-01T00:00:00.000+0100 | 105 | 155 | 417 | 1 | 3 |
| 5c4b07ea2f38432e007daabe | 2019-01-01T00:00:00.000+0100 | 51 | 44 | 448 | 1 | 4 |
+-----+-----+-----+-----+-----+-----+-----+
```

Esto lo volcaremos en una tabla, mediante el comando

```
22 CREATE TABLE usage_def AS
23 SELECT id, day, start_station, end_station, travel_time, user_type, age
24 from usage_bicimad
25 LATERAL VIEW json_tuple (data, 'id','unplug_hourTime','user_day_code','idunplug_station','idplug_station','travel_time','user_type','ageRange')
26 basic as idd, da, user_day, start_station, end_station, travel_time, user_type, age
27 LATERAL VIEW json_tuple (basic.idd, 'soid')
28 bb as id
29 LATERAL VIEW json_tuple (basic.da, '$date')
30 d as day;
```

-Datos de estaciones

```
hive      hive_usage      API_bicimad_2.py x
CREATE EXTERNAL TABLE station_bicimad (data String);

LOAD DATA INPATH "/user/acerrato/bicimad_m3/data/stations" INTO TABLE station_bicimad;

SELECT * FROM station_bicimad limit 1;

select j.id as status_day,
get_json_object (data, concat('$.stations[',stat.a,'].id')) as station_id,
get_json_object (data, concat('$.stations[',stat.a,'].name')) as station name,
get_json_object (data, concat('$.stations[',stat.a,'].activate')) as activate,
get_json_object (data, concat('$.stations[',stat.a,'].no_available')) as no_available,
get_json_object (data, concat('$.stations[',stat.a,'].light')) as light,
get_json_object (data, concat('$.stations[',stat.a,'].total bases')) as total bases,
get_json_object (data, concat('$.stations[',stat.a,'].free_bases')) as free_bases,
get_json_object (data, concat('$.stations[',stat.a,'].reservations_count')) as reservations_count,
get_json_object (data, concat('$.stations[',stat.a,'].dock_bikes')) as dock_bikes
from station_bicimad
lateral view json_tuple (data, '_id') j as id
lateral view posexplode (split(get_json_object (data, '$.stations[*]'),' ',' '))stat as a, x limit 172
where get_json_object (data, concat('$.stations[',stat.a,'].id')) as station_id is not null

CREATE TABLE station_def AS
select j.id as status_day,
get_json_object (data, concat('$.stations[',stat.a,'].id')) as station_id,
get_json_object (data, concat('$.stations[',stat.a,'].name')) as station name,
get_json_object (data, concat('$.stations[',stat.a,'].activate')) as activate,
get_json_object (data, concat('$.stations[',stat.a,'].no_available')) as no_available,
get_json_object (data, concat('$.stations[',stat.a,'].light')) as light,
get_json_object (data, concat('$.stations[',stat.a,'].total bases')) as total bases,
get_json_object (data, concat('$.stations[',stat.a,'].free_bases')) as free_bases,
get_json_object (data, concat('$.stations[',stat.a,'].reservations_count')) as reservations_count,
get_json_object (data, concat('$.stations[',stat.a,'].dock_bikes')) as dock_bikes
from station_bicimad
lateral view json_tuple (data, '_id') j as id
lateral view posexplode (split(get_json_object (data, '$.stations[*]'),' ',' '))stat as a, x limit 172
```

status_day	station_id	station_name	activate	no_available	light	total bases	free_bases	reservations_count	dock_bikes
2019-01-26T03:05:38.412321	1	Puerta del Sol A	1	0	2	24	15	0	7
2019-01-26T03:05:38.412321	2	Puerta del Sol B	1	0	2	24	10	0	12
2019-01-26T03:05:38.412321	3	Miguel Moya	1	0	2	24	11	0	10
2019-01-26T03:05:38.412321	4	Plaza Conde Suchil	1	0	2	18	7	0	8
2019-01-26T03:05:38.412321	5	Malasaña	1	0	2	24	12	0	7
2019-01-26T03:05:38.412321	6	Fuencarral	1	0	0	27	19	1	4
2019-01-26T03:05:38.412321	7	Colegio Arquitectos	1	0	2	24	14	0	10
2019-01-26T03:05:38.412321	8	Hortaleza	1	0	0	21	18	0	1
2019-01-26T03:05:38.412321	9	Alonso Martínez	1	0	0	24	22	0	0
2019-01-26T03:05:38.412321	10	Plaza de San Miguel	1	0	2	24	10	0	14
2019-01-26T03:05:38.412321	11	Marques de la Ensenada	1	0	0	24	24	0	0
2019-01-26T03:05:38.412321	12	San Andrés	1	0	2	24	11	0	13
2019-01-26T03:05:38.412321	13	San Hermenegildo	1	0	2	24	14	0	7
2019-01-26T03:05:38.412321	14	Conde Duque	1	0	1	24	3	0	20
2019-01-26T03:05:38.412321	15	Ventura Rodríguez	1	0	2	24	7	0	16
2019-01-26T03:05:38.412321	16	San Vicente Ferrer	1	0	2	21	13	0	6
2019-01-26T03:05:38.412321	17	San Bernardo	1	0	2	21	7	0	14
2019-01-26T03:05:38.412321	18	Carlos Cambrero	1	0	2	24	7	0	16
2019-01-26T03:05:38.412321	19	Plaza de Pedro Zerolo	1	0	2	24	11	0	11
2019-01-26T03:05:38.412321	20	Prin	1	0	0	24	23	0	1

3)Explotación:

En una herramienta de explotación conectaremos con hive y nos importaremos los datos. En este caso, por tema de permisos, solo me permitía conectar con Excell, aunque el objetivo inicial era con Power BI o MicroStrategy.

Aun así, Excell con las herramientas de PowerQuery y PowerPivot, nos permite establecer conexión con hive y importar los datos a un modelo de datos para poder tratarlos desde ahí.



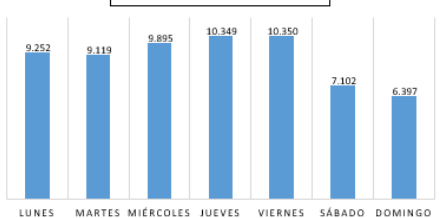
Cuadro de mando USO DEL SERVICIO

Tiempo	
(en blanco)	+1h
0 - 15 min	16 - 30 min
31 - 45 min	46 - 60 min

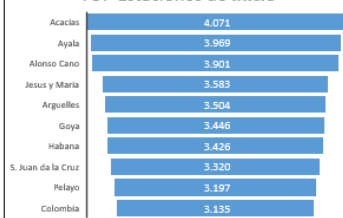
Profile	
Annual	
BiciMad Empleado	
No Identificado	
Ocasional	

Edad	
0-16	17-18
19-26	27-40
41-65	66 -
No Identificado	

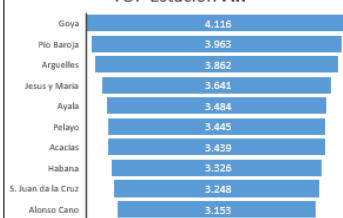
PROMEDIO DE USUARIOS



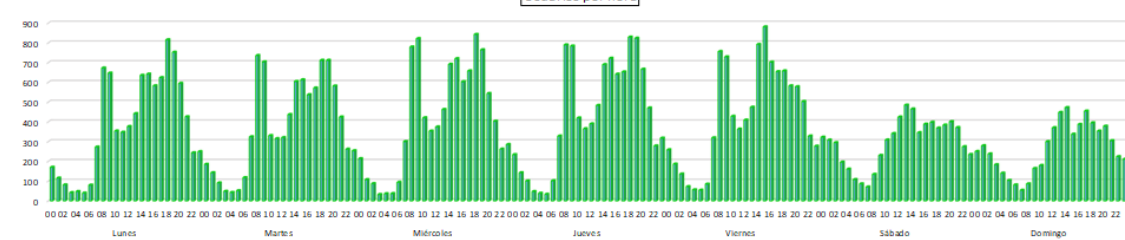
TOP Estaciones de Inicio



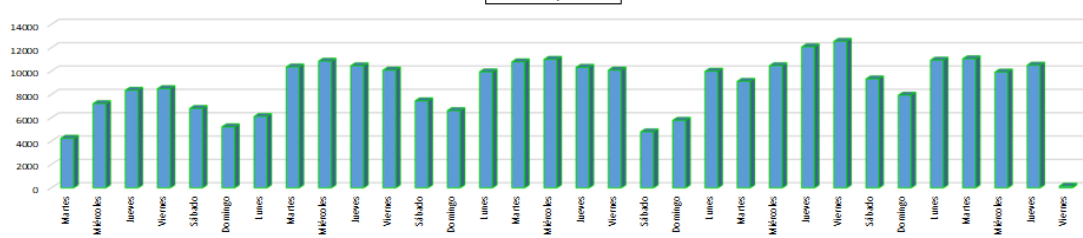
TOP Estación Fin



Usuarios por hora



Usuarios por hora



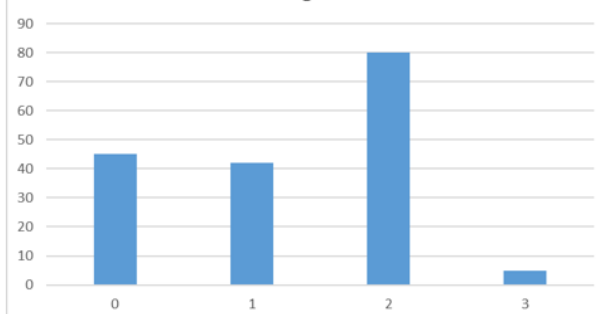
Ambos cuadros de mando serían dinámicos, con botoneras, para poder sacarle más partido a los datos.



Cuadro de mando ESTACIONES

Estado - Name	Numero de estaciones
0	167
1	5
Carretas	1
Colón B	1
Conde Peñalver	1
Red de San Luis A	1
Red de San Luis B	1
Total general	172

Light

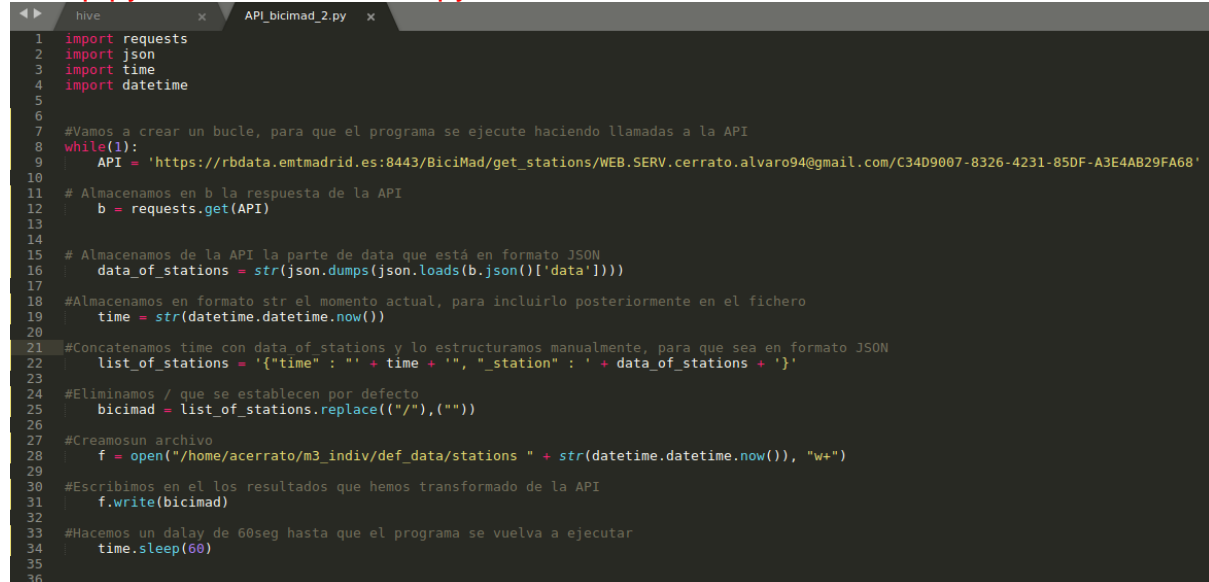


Streaming Layer:

Creamos un programa en python que estará llamando a la API y transformando los datos, dejándolos en formato JSON, que luego agregaremos con flume.

El programa se quedará corriendo en segundo plano con el comando:

nohup python API_bicimad_2.py &



```
1 import requests
2 import json
3 import time
4 import datetime
5
6
7 #Vamos a crear un bucle, para que el programa se ejecute haciendo llamadas a la API
8 while(1):
9     API = 'https://rbddata.emtmadrid.es:8443/BiciMad/get_stations?WEB.SERV.cerrato.alvaro94@gmail.com/C34D9007-8326-4231-85DF-A3E4AB29FA68'
10
11 # Almacenamos en b la respuesta de la API
12 b = requests.get(API)
13
14
15 # Almacenamos de la API la parte de data que está en formato JSON
16 data_of_stations = str(json.dumps(json.loads(b.json()['data'])))
17
18 # Almacenamos en formato str el momento actual, para incluirlo posteriormente en el fichero
19 time = str(datetime.datetime.now())
20
21 #Concatenamos time con data_of_stations y lo estructuramos manualmente, para que sea en formato JSON
22 list_of_stations = '{"time" : "' + time + '", "station" : ' + data_of_stations + '}'
23
24 #Eliminamos / que se establecen por defecto
25 bicimad = list_of_stations.replace("/","")
26
27 #Creamos un archivo
28 f = open("/home/acerrato/m3_indiv/def_data/stations " + str(datetime.datetime.now()), "w+")
29
30 #Escribimos en el los resultados que hemos transformado de la API
31 f.write(bicimad)
32
33 #Hacemos un delay de 60seg hasta que el programa se vuelva a ejecutar
34 time.sleep(60)
35
36
```

Almacenará los datos de la API en el nodo frontera del cluster en la carpeta
sftp://35.242.236.241/home/acerrato/m3_indiv/def_data/dinamic_data

Dejaremos abierto un agente de flume que irá llevando a HDFS cada uno de los archivos que se irán generando via API

1. Ingesta

De igual maner

```
*flume_bicimad_streaming.conf
~/bicimad_m3/Streaming

File Edit View Search Tools Documents Help

Agent1.sources = spooldir-source
Agent1.channels = memory-channel
Agent1.sinks = hdfs-sink

###Describe/configure Source
Agent1.sources.spooldir-source.type = spooldir
###Carpeta de origen
Agent1.sources.spooldir-source.spoolDir = /home/acerrato/m3_indiv/def_data/
dynamic_data
###Longitud maxima de una linea
Agent1.sources.spooldir-source.deserializer.maxLineLength = 100000
###Codigo para dejarlo en formato JSON y no ensuciar el contenido del archivo
Agent1.sources.spooldir-source.hnadler = org.apache.flume.source.http.JSONHandler

###Describe the sink
Agent1.sinks.hdfs-sink.type = hdfs
###Carpeta de destino en HDFS
Agent1.sinks.hdfs-sink.hdfs.path = /user/acerrato/bicimad_m3/data/streaming
###Dejados en formato Text, fundamental para despues ser tratados con Hive
Agent1.sinks.hdfs-sink.hdfs.writeFormat = Text
###Tamaño max de archivo
Agent1.sinks.hdfs-sink.hdfs.rollSize = 12800000
###Ingstará linea a linea, pues la consulta a la API es una linea por min|
Agent1.sinks.hdfs-sink.hdfs.rollCount = 1
###Nombre del archivo de salida
Agent1.sinks.hdfs-sink.hdfs.filePrefix = usage-
Agent1.sinks.hdfs-sink.hdfs.fileType = DataStream
Agent1.sinks.hdfs-sink.hdfs.round = true
Agent1.sinks.hdfs-sink.hdfs.roundValue = 10
Agent1.sinks.hdfs-sink.hdfs.roundUnit = minute

###Use a channel which buffers events in memory
Agent1.channels.memory-channel.type = memory
Agent1.channels.memory-channel.byteCapacity = 6912212
Agent1.channels.memory-channel.transactionCapacity = 100

###Bind the source and sink to the channel
Agent1.sources.spooldir-source.channels = memory-channel
Agent1.sinks.hdfs-sink.channel = memory-channel

Plain Text Tab Width: 8 Ln 23, Col 75 INS
```

a, realizaremos la ingesta con flume, aunque no es lo óptimo, se generará un archivo por cada respuesta de la API, con el fin de poder consultar los datos actualizados y no tener que esperar a que se llene un fichero de 128MB. Se trabajará con una configuración muy similar a la del los anteriores a diferencia de que el *rollCount* = 1 para generar un archivo por cada respuesta de la API, es decir, uno por minuto.

2) Procesamiento

Con Spark leeremos de la carpeta de hdfs los archivos JSON, la idea inicial era conseguir transformar el Json, y con Spark SQL, consultar los datos para que genere los KPIs, donde únicamente buscaríamos conocer el estado de las estaciones que no están operativas, y que esta se mostrase por terminal, sin

almacenar archivos tratando de conseguir que por cada nuevo archivo recibido se re-ejecute el programa y arrojase el resultado del último archivo. Por desgracia no he conseguido modelar el JSON, como para que mostrar los datos en local.

```
package processing

import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.{col, explode}

object Tweets {

  def main(args: Array[String]): Unit = {

    val logger = Logger.getLogger(name = "org.apache.spark")
    logger.setLevel(Level.ERROR)

    val spark = SparkSession
      .builder()
      .appName(name = "Processing_api")
      .master("local[*]")
      .getOrCreate()

    val data = spark.read.json(path = "hdfs:///user/acerrato/bicimad_m3/data/streaming/*")
    //val data = spark.read.text("./src/main/scala/resources/usage-.1553461984272")

    //transformamos el JSON para extraer los campos
    val BM = data.select(col = "time", cols = "_station")
    val bicimad = BM.withColumn(colName = "time", explode(col(colName = "_station")))

    val def_bicimad = bicimad.select(col = "time", cols = "_station.id", "_station.light")
    def_bicimad.show()

  }

}
```

Streaming no tiene explotación pues generariamos el resumen de los KPIs por terminal, conociendo en todo momento que inactivas y tomar medidas sobre ellas.



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

www.icemd.com