IMF Business School

TRABAJO FIN DE MÁSTER

ARQUITECTURA BIG DATA

MÁSTER UNIVERSITARIO EN BUSINESS ANALYTICS Y BIG DATA

Docente. Juan Manuel Moreno Lamparero

Curso académico 2018-2020

Contenido

1. INTRODUCCIÓN

Hoy en día muchas empresas tecnológicas trabajan para optimizar los recursos y costes de las empresas y de la misma forma buscan una manera para mejorar la ecología global. Si solo nos fijamos en los camiones de logística optimizando sus rutas no solo podemos mejorar las horas trabajadas, consumo de combustible, ahorro en recursos, sino también la contaminación de la atmósfera.

Muchos analistas de datos trabajan con los conocidos algoritmos de los problemas de rutas de vehículos (Vehicle Routing Problem - VRP) utilizando las librerías de **OpenStreetMap OSRM** y **OR-TOOLS** de **Google,** pero para planificar las rutas optimizadas tenemos que predecir el tráfico para calcular mejor dichas rutas. ¿Podemos predecir el tráfico que vamos a encontrar dentro de un mes? Nos es una tarea fácil ni rápida, necesitamos el histórico de datos de trafico de los años anteriores meses, semanas horas e incluso minutos.

2. OBJETIVO

Mi objetivo principal con este proyecto no es la adquisición, procesado y analítica de datos, es crear la arquitectura Big data distribuida y tolerante a fallos, decido obtener los datos en tiempo real cueles puedo compartir con mis compañeros de trabajo una vez terminado mi proyecto.

3. FUENTES DE DATOS

En el portal OpenData oficial de DGT encuentro los datos históricos desde 2013 cuales puedo descargar para las predicciones, también encuentro datos en el tiempo real (según la DGT:
la información se actualiza casi en tiempo real, con una periodicidad de unos 5 minutos, que es el tiempo mínimo de varios ciclos de semáforo, necesarios para dar una medición real, y que no se vea afectada la medición por si el semáforo está abierto o cerrado.), con 0estos datos podemos mantener nuestros datos siempre actualizados.

4. PLANTEAMIENTOS INICIALES

Antes de la solución definitiva, tenía varias ideas y diseños que finalmente fueron desestimados por diferentes razones. Se detallan brevemente los más relevantes.

4.1 ARQUITECTURA LAMBDA VS ARQUITECTURA KAPPA

Existen dos tipos de arquitecturas que son las más comunes en proyectos con tecnologías Big Data. Por un lado, están las arquitecturas Lambda, en las que se plantea una parte para el tratamiento de la información en batch, y otra para la parte online. En la primera tendríamos la información en crudo, sin modificar, e iríamos acumulando los datos, pudiendo cargarlos con “snapshots” o situaciones a final del día desde el sistema actual, o directamente con streaming, pero sin procesar, mientras que en la segunda nos centraríamos en la información online, los datos nuevos, y que en esta parte procesaríamos.

Por otro, tenemos las arquitecturas Kappa, en las que se prescinde de la parte batch, y procesamos únicamente datos en streaming continuo.

En un principio he pensado en el diseño de una arquitectura Lambda, que permitiera el almacenamiento de la información sin tratar, en un sistema de ficheros distribuido, como HDFS, y que se utilizará para procesos distintos (analítica y predicción) a los que explotarían la capa de streaming (métricas e informes necesarios para el negocio).

Finalmente, esta solución se descartó, ya que implicaba varios problemas:

- Sincronización de la información en todos los puntos: Al tener el sistema original, la capa online, y la capa batch, obligaba a establecer mecanismos y controles para evitar inconsistencias y pérdida de calidad de los datos.
- La complejidad de la arquitectura se incrementaba: Esta solución obligaba al uso de más herramientas, más esfuerzo y más recursos.

Este planteamiento podría realizarse en un equipo con más recursos.

4.2 BBDD ONLINE & BBDD HISTÓRICA

En la mayoría de los sistemas suelen existir dos niveles de explotación de la información, una primera que requiere de los datos más actuales, y que aglutina un alto porcentaje de las consultas, y otra que puede hacer uso de datos históricos, y que generalmente es para accesos más residuales. Por ello, inicialmente se trató de llevar esos dos niveles a la arquitectura diseñada, estableciendo una BBDD para la información reciente, y otra para el histórico.
Al igual que pasó con la arquitectura Lambda, se descartó, por las siguientes razones:

- Aumento de costes y el uso de herramientas con posibilidad de soporte, la opción de dos BBDD, con más licencias necesarias, aumenta el coste de la solución.
- Al igual que el punto anterior, más herramientas implicaba más esfuerzo en la implantación.

Sin embargo, este punto si es interesante, no para este proyecto, pero si para evaluarlo sobre una solución más amplia a largo plazo.

4.3 HERRAMIENTAS REAL TIME

Para ello he decido utilizar dos componentes principales: un gestor de colas que permita contener y no saturar el soporte final dónde se almacenarán los datos, y un framework de procesamiento distribuido que ofrezca la posibilidad de tratar eso datos previamente a su almacenamiento sin retardo en la inserción final. Una vez decidida la naturaleza de las herramientas, analizo diversas alternativas, decidiendo finalmente el uso de Apache Kafka como gestor de colas (por ser una de las más potentes y cuyo uso es bastante extendido) y Apache Spark como motor de procesamiento (permite programación en Python).

4.4 ALMACENAMIENTO

Para ello decido utilizar Elasticsearch es almacenamiento de documentos distribuidos. En lugar de almacenar información como filas de datos en columnas, Elasticsearch almacena estructuras de datos complejas que se serializa como documentos JSON.

Elasticsearch es un motor de analítica y análisis distribuido y open source para todos los tipos de datos, incluidos textuales, numéricos, **geoespaciales**, estructurados y **desestructurados**. La velocidad y escalabilidad de Elasticsearch y su capacidad de indexar muchos tipos de contenido significan que puede usarse para una variedad de casos de uso.

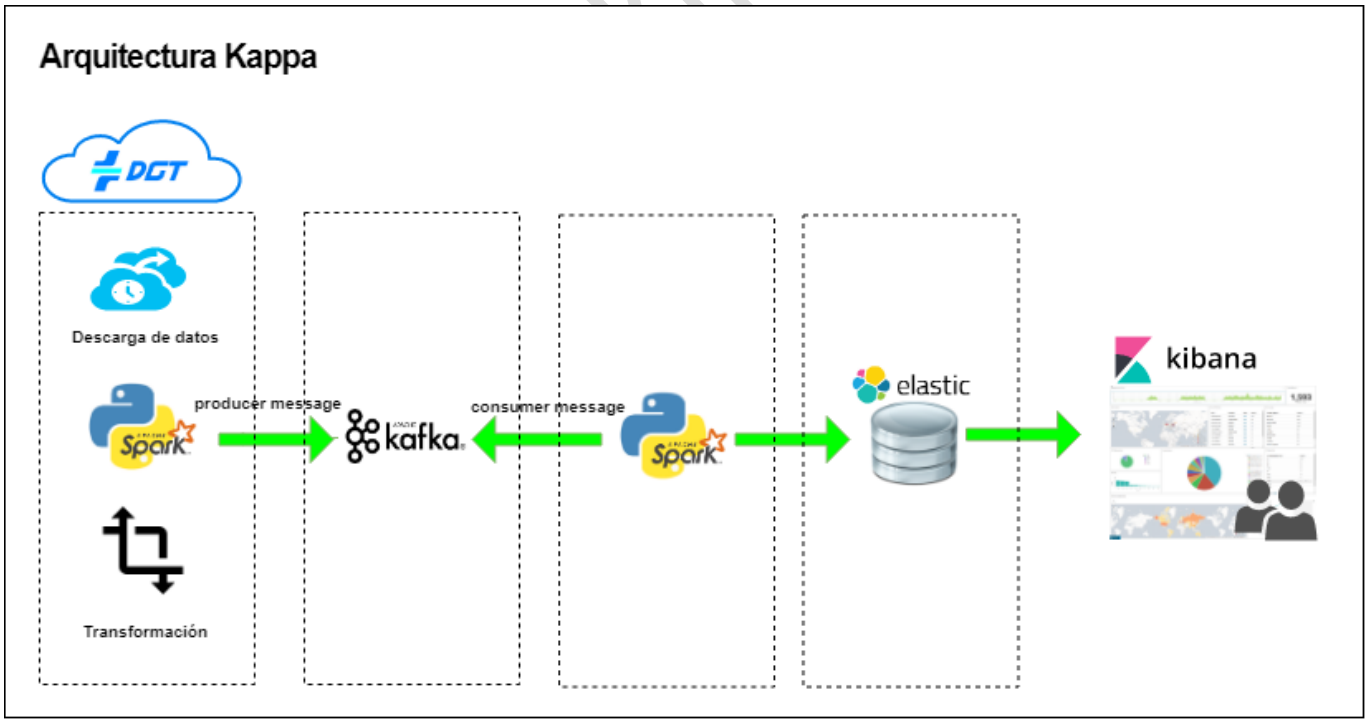
- Está muy bien integrada con otras dos aplicaciones de la misma compañía es rápido
- Distribuido por naturaleza
- Puede dar respuesta a múltiples clientes.
- Los documentos y configuración se pueden modificar y manipular mediante consultas HTTP. y, además, trabaja con ficheros con formato JSON.

4.5 VISUALIZACIÓN

Como Elasticsearch está muy bien integrada con la aplicación de la misma compañía, en este caso no hoy duda la decisión está clara: Kibana.

Kibana es una herramienta de visualización y gestión de datos para Elasticsearch que brinda histogramas en tiempo real, gráficos circulares y mapas. Kibana también incluye aplicaciones avanzadas, como Canvas, que permite a los usuarios crear infografías dinámicas personalizadas con base en sus datos, y Elastic Maps para visualizar los datos geoespaciales.

4.6 RESUMEN DE LA DECISIÓN TOMADA SOBRE LA ARQUITECTURA Y LA LAS HERRAMIENTAS A UTILIZAR EN EL PROYECTO.



Img 1. Arquitectura Kappa. Elaboración propia.

2. ANÁLISIS DE SOLUCIONES CLOUD

Una vez se decide la definición de la arquitectura en un entorno Cloud, comienzo un periodo de análisis y pruebas sobre los recursos ofrecidos por algunos de los proveedores.

Partiendo de un presupuesto de cero euros, lo que implica aprovechar algunas de las opciones de cuentas gratuitas que me ofrecen, y ver si con dichas cuentas es suficiente para dar soporte a las necesidades del proyecto.

Se evalúan: AWS, Azure y Google Cloud Platform, ya que son las que actualmente tienen más clientes, al constar del más amplias funcionalidades ofrecidas.

2.1 MICROSOFT AZURE

Esta plataforma dispone de un amplio abanico de servicios y plantillas de arquitecturas para proyectos Big Data. En esta plataforma hay la posibilidad de obtener una cuenta gratuita con un crédito de 170 € los 30 primeros días, para montar un clúster en Azure no requiere ningún esfuerzo, solo hay que seguir los pasos, y el Azure todo hace por ti (configuración de los nodos). **Mi objetivo es configurar un clúster real desde 0**, configurar manualmente para entender mejor su funcionamiento probar que ocurre cuando uno de los nodos se cae, por ejemplo.

2.2 AWS

Esta plataforma se ha utilizado en clase para la realización de algunas prácticas ofrece todas las funcionalidades necesitadas. Se descarta solo por el motivo de haberlo visto a lo largo del curso.

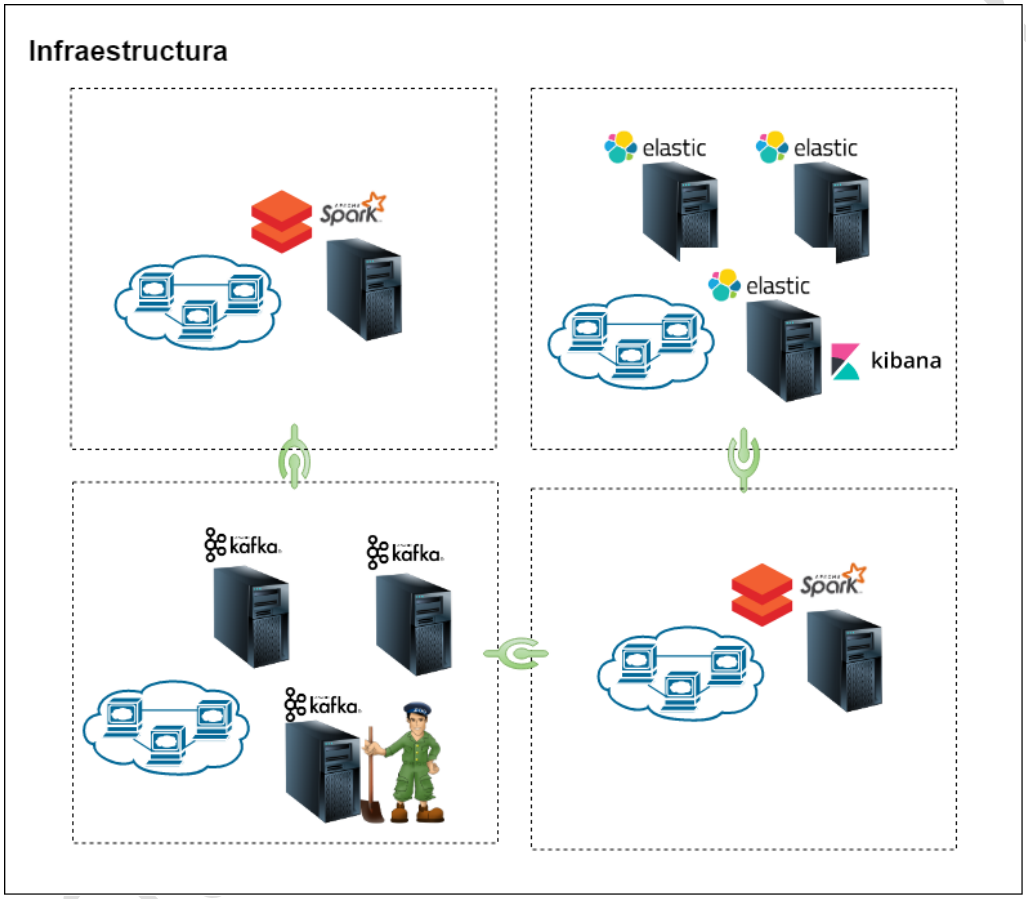
2.3 GOOGLE CLOUD PLATFORM

GCP ofrece multitud de servicios y recursos para el desarrollo de aplicaciones y sistemas. Con las cuentas gratuitas con un crédito de 300$ por año, puedo, a través de Google Compute Engine, definir las máquinas virtuales con los recursos que necesito.

Al realizar las pruebas de concepto el único punto a tener en cuenta es que todas las maquinas debo de pararlas en el momento en el que dejo de trabajar con ellas, para evitar consumir el saldo de la cuenta. Esto implica la renovación de las IPs externas, por lo que algunas configuraciones existentes se han intentado establecer con IPs internas, aunque no siempre ha sido posible.

Por otro lado, existe alguna restricción en cuanto a los recursos que se pueden utilizar, con las cuentas gratuitas, en las diversas zonas geográficas, por lo que mis máquinas están repartidas por todo el mundo. Esto también ha incrementado el coste de la solución, ya que el precio no es el mismo en todas ellas.

2.3 Resumen de la decisión tomada sobre infraestructura.



Img 2. Infraestructura. Elaboración propia.

Finalmente decido montar dos Clústeres en la plataforma Google Cloud Platform.

Para ello, y tras numerosas pruebas y reajustes, la infraestructura es la siguiente:

- **Tres nodos para Apache Kafka**, uno de los cuales compartirá recursos con **Apache Zookeeper**.
- **Tres nodos para Elastic** uno de los cuales compartirá recursos con **Kibana**
- Para los scripts de **Producer y Consumer de Kafka** utilizare los clústeres de Spark en **DataBricks** para ahorrar el tiempo, utilizare dos cuentas diferentes para ver la ejecución de los scripts en paralelo.

3. RESULTADOS ESPERADOS

1. **Montar los clústeres desde cero en la nube**, configurar correctamente, en caso de Clúster de Kafka con Zookeeper configurar correctamente los Brokers, Productores y Consumidores. Tener visibilidad entre los nodos del mismo Clúster y tener acceso a los servicios desde un cliente externo.
2. Con PySpark, hacer una pequeña **transformación de datos** obtenidos desde el portal de la DGT, los datos solo se pueden descargar en el formato XML con las coordinadas en el formato UTM del centroide que representa al punto de medida en el fichero georreferenciado, por lo cual es necesario pasar el fichero XML a JSON (en mi caso geoJson) y convertir los datos geográficos a EPSG (European Petroleum Survey Group) con los Sistemas de Referencia de Coordenadas habitualmente usados en los **Servicios Web de Mapas (WMS)**.
3. Una vez esta convertido al objeto geoJson **publicar en un Topic**.
4. En paralelo orto script **Spark Streaming** tiene que ser capaz de **recibir los mensajes** de mismo Topic e insertar en Elastic.
5. Finalmente crear un **informe en Kibana** para visualización y análisis de los datos obtenidos.

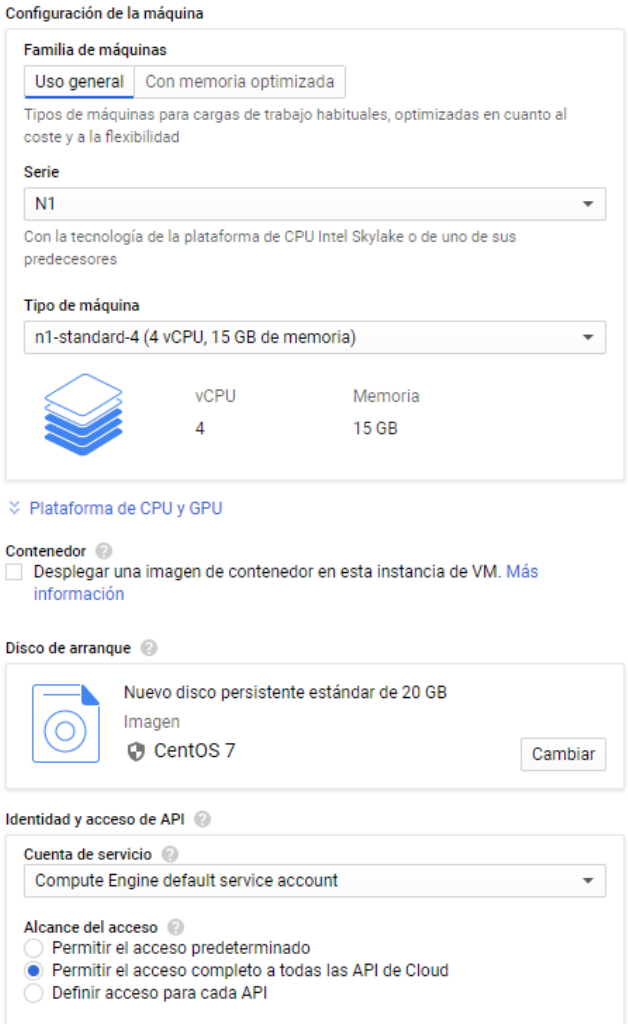
4. CREACIÓN Y CONFIGURACIÓN DE MV

Creamos una cuenta gratuita en Google Cloud Platform https://cloud.google.com/. Desde la consola Menu → Compute Engine → Instancias de VM

4.1 CREACIÓN DE CLÚSTER KAFKA + ZOOKIPER

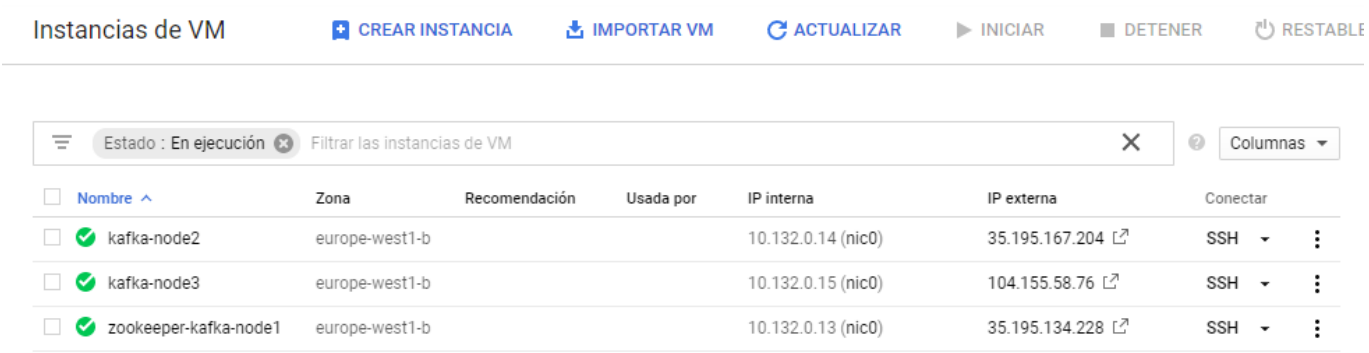
4.1.1 Crear Instancias GCP

Creamos nueva instancia como se muestra en la siguiente imagen



Img 3 Creación de la nueva instancia.

Creamos 3 instancias como se muestra en la Img 3, una vez terminado INICIAMOS las 3 instancias.



Img 4 Instancias Iniciadas.

Clúster	Nodo GCP	Herramientas	CPUs	RAM	Zona	Disco	S.O.
Apache Kafka	zookeeper-kafka-nodo1	Zookeeper + Kafka	4	15	europe-west1-b	10 GB	Centos 7
Apache Kafka	kafka-nodo2	Kafka	4	15	europe-west1-b	10 GB	Centos 7
Apache Kafka	kafka-nodo3	Kafka	4	15	europe-west1-b	10 GB	Centos 7

4.1.2 Instalación Apache Kafka

Entramos a la consola de las 3 máquinas pulsando el botón SSH, y primero modificamos el fichero hosts en las 3 MV con el comando vim /etc/hosts

Y añadimos en las 3 máquinas

```
10.132.0.13 zookeeper-kafka-node1.europe-west1-b.c.proyectotfm-
268719.internal zookeeper-kafka-node1
10.132.0.14 kafka-node2.europe-west1-b.c.proyectotfm-268719.internal kafka-
node2
10.132.0.15 kafka-node3.europe-west1-b.c.proyectotfm-268719.internal kafka-
node3
```

Instalamos Apache Kafka en las 3 Instancias

```
$ sudo yum -y install java-1.8.0-openjdk
$ sudo yum -y install wget
$ wget http://apache.uvigo.es/kafka/2.4.1/kafka_2.11-2.4.1.tgz
tar -xzf kafka_2.11-2.4.1.tgz
```

Modificamos el bash_profile en las 3 MV

```
$ vim .bash_profile
      :$HOME/kafka_2.11-2.4.1/bin
$ source .bash_profile
```

4.1.3 Configuración Zookeeper

Configuramos **Zookeeper**, para ello modificamos fichero de configuración de **Zookepeer**, solo en la maquina donde vamos a arrancar el **Zookeeper**

```
$ vi kafka_2.11-2.4.1/config/zookeeper.properties
```

Cambiamos la propiedad

```
 dataDir= /home/a_l_es/zookeeper_data
$ mkdir /home/a_l_es/zookeeper_data
$ mkdir /home/anastasia_lukina_es/zookeeper_data
```

Arrancamos el **Zookeeper**

```
$ zookeeper-server-start.sh kafka_2.11-2.4.1/config/zookeeper.properties
```

Como se comentado anteriormente para ahora los costes de consumo de crédito gratuito tengo que apagar más instancias una vez que dejo de trabajar con ellas. Para no volver a arrancar a mano cada vez que arranco la instancias, añado al fichero rc.local

```
$ sudo vi /etc/rc.d/rc.local
```

Añadimos siguiente línea:

```
/home/a_l_es/kafka_2.11-2.4.1/bin/zookeeper-server-start.sh
/home/a_l_es/kafka_2.11-2.4.1/config/zookeeper.properties > /dev/null 2>&1 &
```

```
$ sudo chmod +x /etc/rc.d/rc.local
$ sudo systemctl enable rc-local
$ sudo systemctl start rc-local
```

4.1.4 Configuración Kafka Brokers

Configuramos **Kafka Brokers** en las 3 MV

A diferencia de **Zookeeper**, cambiaremos más propiedades de configuración para los **Kafka Brokers**.

Configuración	Kafka-node1	Kafka-node2	Kafka-node3
broker.id	**0**	**1**	**2**
broker.rack	**RACK1**	**RACK1**	**RACK2**
listeners	**PLAINTEXT://zookeeper-kafka-node1:9092**	**PLAINTEXT://kafka-node2:9092**	**PLAINTEXT://kafka-node3:9092**
advertised.listeners	**PLAINTEXT://35.195.134.228:9092**	**PLAINTEXT://35.195.167.204:9092**	**PLAINTEXT://35.195.167.204:9092**
log.dirs	**/home/a_l_es/kafka_data**		
offsets.topic.num.partitions	**3**		
offsets.topic.replication.factor	**2**		
default.replication.factor	**2**		
zookeeper.connect	**10.132.0.13:2181**		

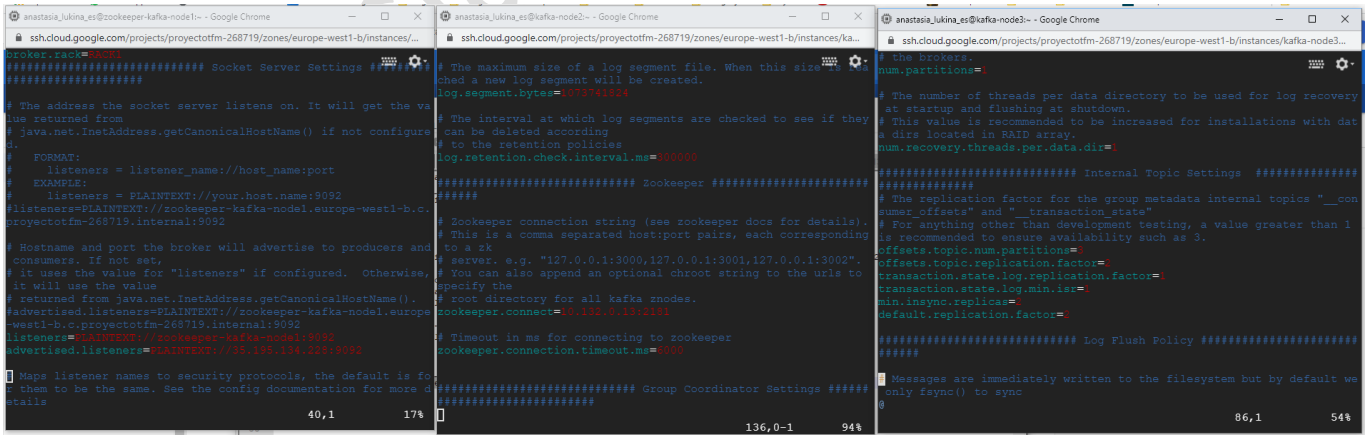
* **advertised.listeners** IPs Externas (una de las primeras problema encontradas)

Creamos un directorio para guardar los logs

```
$ mkdir /home/anastasia_lukina_es/kafka_data
```

Modificamos el fichero de configuración de Kafka server.properties

```
$ vi kafka_2.11-2.4.1/config/server.properties
```



Img 5. config/server.properties de las 3 MV

Arrancamos Kafka, manualmente y añadimos al rc.local para no arrancar manualmente cada vez que apagamos y arrancamos las MV

```
$ kafka-server-start.sh kafka_2.11-2.4.1/config/server.properties
$ kafka-server-stop.sh  kafka_2.11-2.4.1/config/server.properties
$ sudo vi /etc/rc.d/rc.local
```

Añadimos al rc.local en las 3 MV

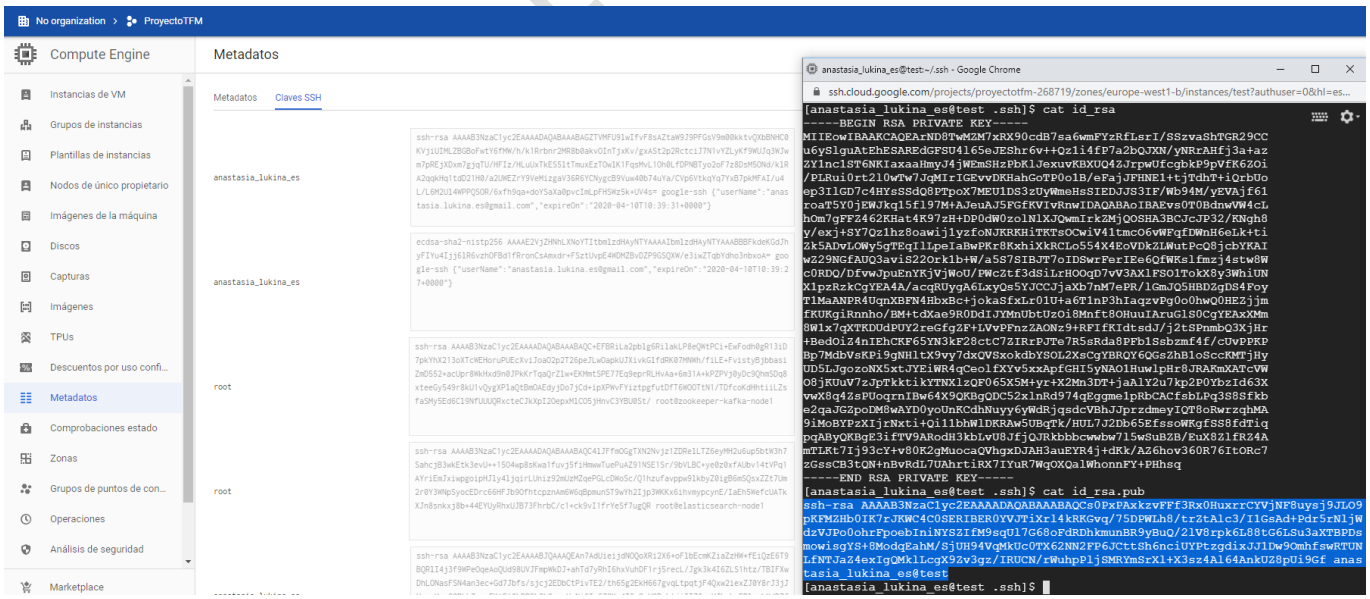
```
/home/a_l_es/kafka_2.11-2.4.1/bin/kafka-server-start.sh
/home/a_l_es/kafka_2.11-2.4.1/config/server.properties > /dev/null 2>&1 &
```

```
$ sudo chmod +x /etc/rc.d/rc.local
$ sudo systemctl enable rc-local
$ sudo systemctl start rc-local
```

Antes de comprobar la instalación tenemos que establecer la conexión SSH entra las máquinas, para ello hay que generar la clave pública y privada

```
$ cd .ssh
$ ssh-keygen
$ cat id_rsa.pub
```

Copiamos la clave y añadimos a metadatos Claves SSH



Img 6. Añadir Clave Publica

Antes Ahora ya podemos comprobar la instalación de **Apache Kafka**

Ahora ya podemos comprobar la instalación de Apache Kafka

```
$ zookeeper-shell.sh zookeeper-kafka-node1:2181 ls /brokers/ids
```

```
[anastasia_lukina_es@kafka-node3 ~]$ vi kafka_2.11-2.4.1/config/server.properties
[anastasia_lukina_es@kafka-node3 ~]$ sudo systemctl restart rc-local
[anastasia_lukina_es@kafka-node3 ~]$ zookeeper-shell.sh zookeeper-kafka-node1:2181 ls /broker
s/ids
Connecting to zookeeper-kafka-node1:2181

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[1]
[anastasia_lukina_es@kafka-node3 ~]$ vi kafka_2.11-2.4.1/config/server.properties
[anastasia_lukina_es@kafka-node3 ~]$ sudo systemctl restart rc-local
[anastasia_lukina_es@kafka-node3 ~]$ zookeeper-shell.sh zookeeper-kafka-node1:2181 ls /broker
s/ids
Connecting to zookeeper-kafka-node1:2181

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[0, 1, 2]
[anastasia_lukina_es@kafka-node3 ~]$
```

Img 7. Brokers Kafka

4.1.5 Creation de Topic

Creamos un Topic

```
$ sudo /home/anastasia_lukina_es/kafka_2.11-2.4.1/bin/kafka-console-
producer.sh --broker-list localhost:9092 --topic dgt
```

Hasta aquí todo va bien, vamos a probar a conectar con un cliente para comprobar si puedo conectar a Kafka con un cliente externo.

Para poder de conectar con un cliente externo tenemos que configurar las reglas de cortafuegos. Para ello desde la consola de Google Cloud Platform (GCP) desde menu → Red VPC → Reglas de cortafuegos, Creamos dos nuevas reglas.

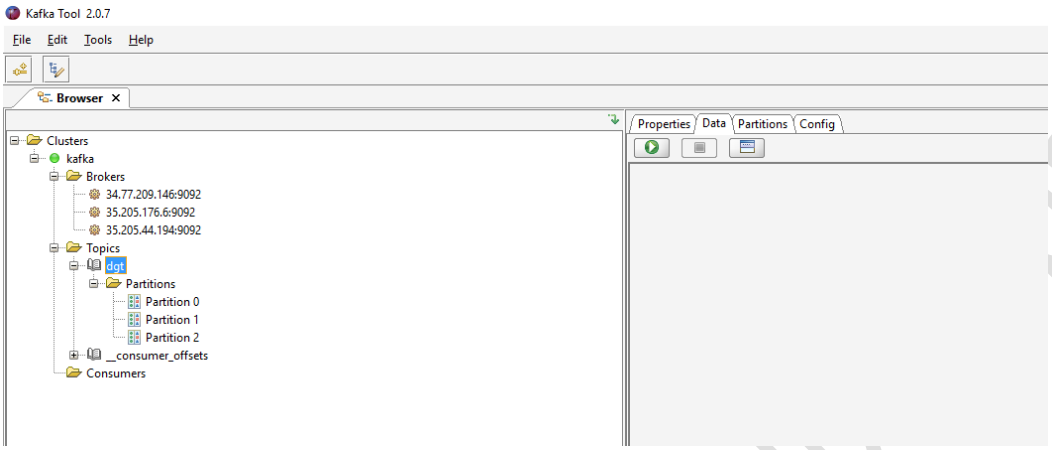
tcp2181 y tcp 9092

☐	kafka	Entrada	http-server	Intervalos de IPs: 0.0.0.0/0	tcp:2181	Permitir	1000	default
☐	kafkaproducer	Entrada	http-server	Intervalos de IPs: 0.0.0.0/0, 10.132.0.0/24	tcp:9092	Permitir	1000	default

Img 8. Reglas de cortafuegos

4.1.6 Clientes Apache Kafka

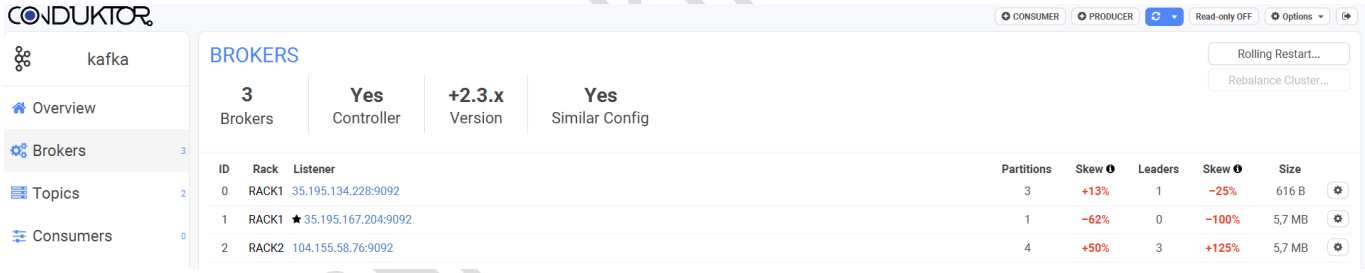
Vamos a descargar Kafka Tool una aplicación GUI para administrar y usar clústeres de **Apache Kafka**



Img 9. Kafka Tool.

Como podemos comprobar tenemos nuestro Clúster con **3 Brokers**, nuestro **Topic dgt** con **3 particiones**.

Vamos a descargar Conduktor Cliente de Apache Kafka de Escritorio https://www.conduktor.io/



Img 10. App Conduktor

Ya tenemos nuestro Clúster bien configurado, arrancado y listo para poder utilizar desde cualquier cliente.

¡NOTA! La seguridad no ha sido un punto en el que se ha hecho especial hincapié, en cuanto a configuraciones establecidas en este proyecto, pero sí lo son para futuras implementaciones en entornos corporativos.

4.2 CREACIÓN DE CLÚSTER ELASTIC + KIBANA

4.2.1 Creación Instancias GCP

Creamos 3 nuevas instancias con las mismas características como se muestra en la **Img 3. Creación de la nueva instancia.**

Clúster	Nodo GCP	Herramientas	CPUs	RAM	Zona	Disco	S.O.
Elastic	elasticsearch-node1	elasticsearch	4	15	europe-west2-c	10 GB	Centos 7
Elastic	elasticsearch-node1	elasticsearch	4	15	europe-west1-b	10 GB	Centos 7
Elastic + Kibana	kibana-node3	Elasticsearch + Kibana	4	15	europe-west1-b	10 GB	Centos 7

Cambiamos la password de usuario root (en las 3 instancias creadas)

```
$ sudo passwd
```

Y cambiamos a usuario root

```
$ su -
```

Modificamos el fichero hosts en las 3 MV con el comando

```
$ vim /etc/hosts
```

Y añadimos en las 3 maquinas

```
10.154.0.5 elasticsearch-node1.europe-west2-c.c.proyectotfm-268719.internal
elasticsearch-node1
10.132.0.10 elasticsearch-node2.europe-west1 b.c.proyectotfm-268719.internal
elasticsearch-node2
10.132.0.12 kibana-node3.europe-west1-b.c.proyectotfm-268719.internal kibana-
node3
```

4.2.2 Instalación ElasticSearch

Seguimos las instrucciones de instalación da Elastic de la página
https://www.elastic.co/guide/en/elasticsearch/reference/current/rpm.html

Descarga e instalación la clave de la clave pública

```
$ rpm --import https://artifacts.elastic.co/GPG-KEY-elasticsearch
```

Creacion de archivo **elasticsearch.repo** en el repositorio **/etc/yum.repos.d/**

```
$ cd /etc/yum.repos.d/
$ touch elasticsearch.repo
$ vim elasticsearch.repo
```

```
[elasticsearch]
name=Elasticsearch repository for 7.x packages
baseurl=https://artifacts.elastic.co/packages/7.x/yum
gpgcheck=1
gpgkey=https://artifacts.elastic.co/GPG-KEY-elasticsearch
enabled=0
autorefresh=1
type=rpm-md
```

Instalamos Elastic

```
$ sudo yum install --enablerepo=elasticsearch elasticsearch
```

Para configurar que Elasticserche se inicie automáticamente cuando se inicie la MV ejecutamos los siguientes comandos

```
$ sudo /bin/systemctl daemon-reload
$ sudo /bin/systemctl enable elasticsearch.service
```

4.2.3 Configuración ElasticSearch

Vamos a modificar el fichero de configuración

```
$ vim /etc/elasticsearch/elasticsearch.yml
```

Configuración elasticsearch-node1 /elasticsearch.yml

```
cluster.name: elastic-cluster
node.name: elasticsearch-node1
node.master: true
network.host: elasticsearch-node1
http.port: 9200
discovery.seed_hosts: ["10.132.0.10", "10.132.0.12"]
cluster.initial_master_nodes: ["elasticsearch-node1"]
```

Configuración elasticsearch-node2 /elasticsearch.yml

```
cluster.name: elastic-cluster
node.data: true
xpack.security.audit.enabled: true
node.name: elasticsearch-node2
network.host: elasticsearch-node2
http.port: 9200
discovery.seed_hosts: ["10.154.0.5", "10.132.0.12"]
cluster.initial_master_nodes: ["elasticsearch-node1"]
```

kibana-node3 /elasticsearch.yml

```
cluster.name: elastic-cluster
node.data: true
node.name: kibana-node3
network.host: 10.132.0.12
discovery.seed_hosts: ["10.154.0.5", "10.132.0.10"]
cluster.initial_master_nodes: ["10.154.0.5"]
```

Arrancamos ElasticSearch

```
$ systemctl start elasticsearch.service
$ systemctl status elasticsearch.service
```

```
[root@elasticsearch-node1 ~]# systemctl status elasticsearch.service
● elasticsearch.service - Elasticsearch
   Loaded: loaded (/usr/lib/systemd/system/elasticsearch.service; enabled; vendor preset: disab
led)
   Active: active (running) since Fri 2020-04-10 17:45:47 UTC; 1h 29min ago
     Docs: http://www.elastic.co
 Main PID: 1385 (java)
   CGroup: /system.slice/elasticsearch.service
           ├─1385 /usr/share/elasticsearch/jdk/bin/java -Des.networkaddress.cache.ttl=60 -De...
           └─1826 /usr/share/elasticsearch/modules/x-pack-ml/platform/linux-x86_64/bin/contr...

Apr 10 17:45:20 elasticsearch-node1 systemd[1]: Starting Elasticsearch...
Apr 10 17:45:26 elasticsearch-node1 elasticsearch[1385]: OpenJDK 64-Bit Server VM warning: ....
Apr 10 17:45:47 elasticsearch-node1 systemd[1]: Started Elasticsearch.
Hint: Some lines were ellipsized, use -l to show in full.
```

Img 11. Elasticsearch Running

4.2.4 Instalación Kibana

Instalamos Kibana en el tercer nodo kibana-node3, igualmente siguiendo las instrucciones de instalación https://www.elastic.co/guide/en/kibana/current/rpm.html

Descarga e instalación la clave de la clave pública

```
$ rpm --import https://artifacts.elastic.co/GPG-KEY-elasticsearch
```

Creación de archivo **kibana.repo** en el repositorio **/etc/yum.repos.d/**

```
[kibana-7.x]
name=Kibana repository for 7.x packages
baseurl=https://artifacts.elastic.co/packages/7.x/yum
gpgcheck=1
gpgkey=https://artifacts.elastic.co/GPG-KEY-elasticsearch
enabled=1
autorefresh=1
type=rpm-md
```

Instalamos **Kibana**

```
$ sudo yum install kibana
```

4.2.5 Configuración Kibana

Modificamos fechero de configuración de **Kibana**

```
server.port: 5601
```

```
server.host: "10.132.0.12"
elasticsearch.hosts: ["http://10.154.0.5:9200"]
```

Para poder de conectar con un cliente externo tenemos que configurar las reglas de cortafuegos. Para ello desde la consola de Google Cloud Platform (GCP) desde menu → Red VPC → Reglas de cortafuegos, Creamos dos nuevas reglas.

Tcp5601 y tcp 9092

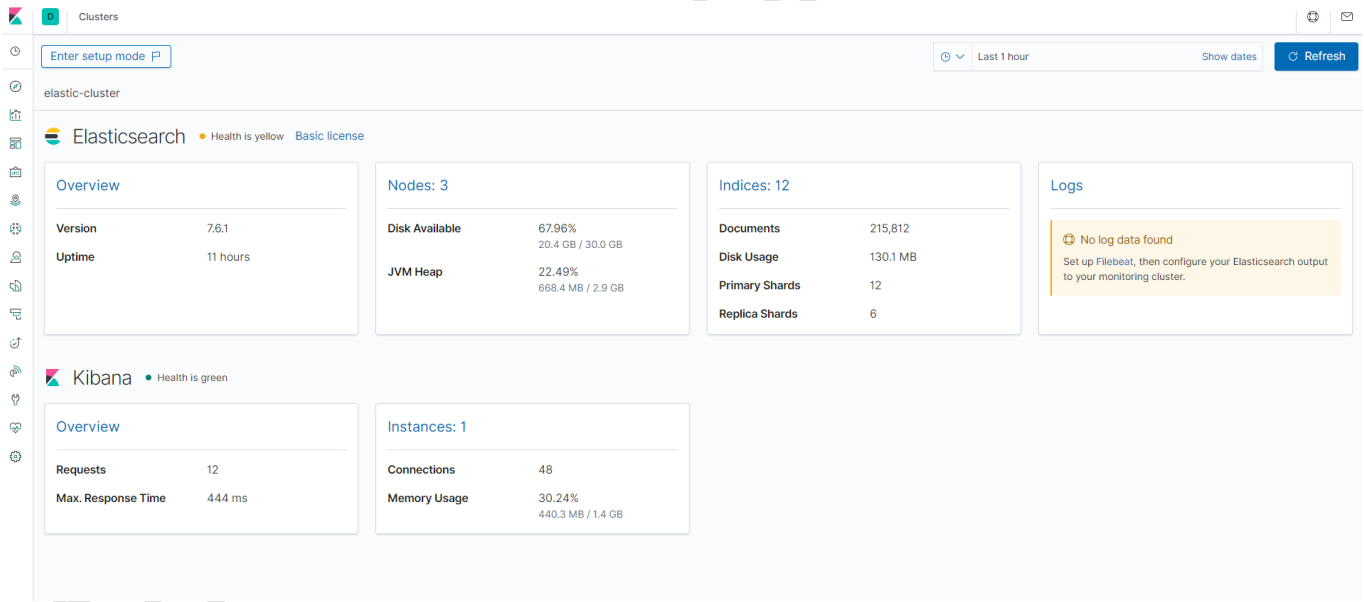
☐	kibana	Entrada	http-server	Intervalos de IPs: 0.0.0.0/0	tcp:5601	Permitir	1000	default

Img 12. Corta fuegos Kibana

☐	elasticsearch	Entrada	http-server	Intervalos de IPs: 0.0.0.0/0	tcp:9200	Permitir	1000	default

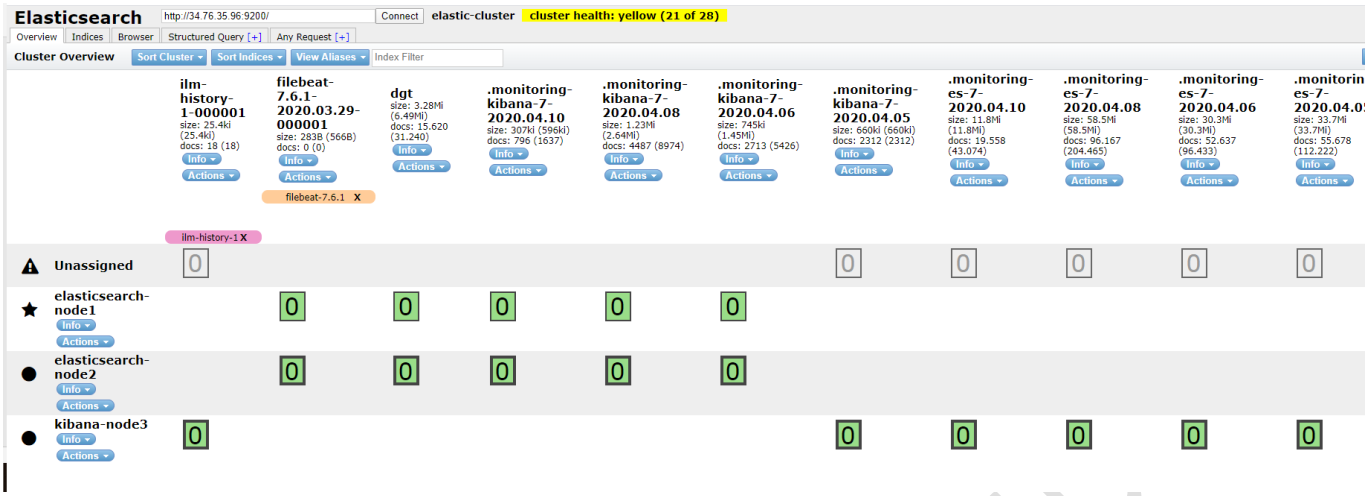
Img 13. Corta fuegos Elasticsearch

Probamos entrar a **Kibana**



Img 14. Kibana Monitoring

Probamos con un plugin de Google Chrome, añadimos el plugin de Elasticserch



Img 15. ElasticSearch pluginChrome

Hemos terminado con las configuraciones de los dos Clúster, comprobamos que los dos Cluasters están configurados correctamente, tenemos visibilidad desde fuera.
En el caso del Clúster Kafka para poder publicar y consumir el Topic con un cliente externo hemos instalado dos aplicaciones de escritorio Constructor y Kafka Tools.
En el caso de Elasticserch y Kibana hemos comprobado que nuestro Clúster tiene 3 nodos y el nodo 1 es el nodo maestro, tal y como hemos configurado.

La seguridad no ha sido un punto en el que se ha hecho especial hincapié, en cuanto a configuraciones establecidas en este proyecto, pero sí lo son para futuras implementaciones en entornos corporativos.

5. DESARROLLO Y EJECUCIÓN DE SCRIPTS

5.1 DATOS.

Tal y como se ha mencionado anteriormente vamos a descargar los datos de tráfico casi en tempo real, la información se actualiza casi en tiempo real, con una periodicidad de unos 5 minutos, que es el tiempo mínimo de varios ciclos de semáforo, necesarios para dar una medición real, y que no se vea afectada la medición por si el semáforo está abierto o cerrado.
La sensorización del tráfico se efectúa por medio de diversos equipamientos que permiten la realización del conteo de vehículos junto con la obtención del grado de ocupación de la vía. De esta manera se puede conocer el nivel de servicio, que se corresponde con los cuatro escalones típicos: tráfico fluido (nivel 0), tráfico lento (nivel 1), retenciones (nivel 2) y congestión (nivel 3).

Estos sistemas de detección son, en su mayoría, lazos electromagnéticos que se colocan debajo del pavimento y detectan la masa metálica de los vehículos que pasan sobre ellos, siendo sistemas de gran calidad y precisión. Tienen las limitaciones que se circunscriben a la toma de datos en un único punto, además de no disponer de visión de la zona para verificar o conocer la causa de los datos que suministran.
El objetivo principal de este conjunto de datos es proporcionar información del tráfico, en tiempo real, de la ciudad de Madrid.
En la ciudad de Madrid existen alrededor de 7.800 detectores de vehículos, de los que alrededor de unos 200 son sistemas ópticos de visión artificial con control desde el Centro de Gestión de la Movilidad, unos 1.400 son específicos de vías rápidas y acceso a la ciudad y, el resto, unos 6.200, son sistemas básicos de control de semáforos.
Estos 7.800 detectores conforman en la ciudad, aproximadamente, unos 4.100 puntos de medida, de los cuales 300 puntos están ubicados en M – 30 y se diferencian de los del resto de la ciudad porque disponen de un sistema para el control de la velocidad, caracterización de los vehículos y doble lazo de lectura.
Respecto a la información en tiempo real, ésta se obtiene por medio de un fichero en formato .xml que presenta, en tiempo real, los valores del tráfico medidos por los sistemas de control. El formato de presentación de la información es el estándar XML. El acceso a estos datos es libre desde el Portal de Datos Abiertos del Ayuntamiento de Madrid.

5.2 CAMPOS DEL CONJUNTO DE DATOS

Cabecera con la fecha y hora en que se obtuvieron los datos:

CAMPO	DESCRIPCIÓN
fecha_hora	fecha y hora en que se obtuvieron los datos

Para cada uno de los puntos de medida de tráfico urbano, los campos que se proporcionan son los siguientes:

CAMPO	DESCRIPCIÓN
idelem	Identificador del punto de medida. Se corresponde con el campo “idelem” presente en el fichero georreferenciado y que permite su posicionamiento sobre plano e identificación del vial y sentido de la circulación
descripcion	Denominación del punto de medida
accesoAsociado	Código de control relacionado con el control semafórico para la modificación de los tiempos
intensidad	Intensidad de número de vehículos por hora. Un valor negativo implica la ausencia de datos.
ocupación	Porcentaje de ocupación del punto de control por los vehículos. Un valor negativo implica la ausencia de datos.
carga	Parámetro de carga del vial en función de la intensidad, ocupación y características de la infraestructura. Un valor negativo implica la ausencia de datos.
nivelServicio	Nivel de servicio
intensidadSat	Intensidad de saturación de la vía en veh/hora y que se corresponde con el máximo número de vehículos que pueden pasar en el acceso a la intersección manteniéndose la fase verde del semáforo
error	Código de control de la validez de los datos del punto de medida

subarea	Identificador de la subarea de explotación de tráfico a la que pertenece el punto de medida
st_x	Coordenada X UTM del centroide que representa al punto de medida en el fichero georreferenciado
st_y	Coordenada Y UTM del centroide que representa al punto de medida en el fichero georreferenciado

Para poder almacenar los datos en **Elacticsearch** tenemos que transformar el XML a GeoJson y también transformar las coordenadas de tipo UTM del centroide a WKT Point para esto vamos a utilizar la **librería pyproj de Python.**

Well Known Text

La representación Well Known Text o de texto conocido (también llamado WKT en su acrónimo inglés) es una codificación o sintaxis en formato ASCII estandarizada diseñada para describir objetos espaciales expresados de forma vectorial. Los objetos que es capaz de describir el formato WKT son los siguientes:

- Puntos.
- Multipuntos.
- Líneas.
- Multilíneas.
- Polígonos.
- Multipolígonos.
- Colecciones de geometría.
- Puntos en 3 y 4 dimensiones.

Su especificación ha sido promovida por un organismo internacional Open Geospatial Consortium (OGC), siendo su sintaxis muy fácil de utilizar, de forma que su uso se encuentra muy generalizado en la industria geoinformática. De hecho, WKT es la base de otros formatos más conocidos como el KML utilizado en Google Maps y Google Earth.

La mayoría de las bases de datos espaciales, y en especial PostgreSQL a través de su extensión para el almacenamiento y análisis espacial PostGIS, utiliza esta codificación.

Existe una variante de este lenguaje, pero expresada de forma binaria, denominada Well Know Binary o binario conocido (WKB en su acrónimo inglés), la cual es también utilizada por estos gestores espaciales, pero con la ventaja de que al ser compilada en forma binaria la velocidad de proceso es muy elevada.

A efectos prácticos la sintaxis WKT consta de una descripción de los vértices que componen la geometría. Para que esta forma de especificar las geometrías tenga sentido deben de acompañarse de una indicación de la referencia espacial o proyección cartográfica utilizada en dicho vector.

Fuente https://es.wikipedia.org/wiki/Well_Known_Text

Para ahorrar el tiempo y coste del proyecte vamos a utilizar el Cluster de DataBriks. Para descargar los datos de la DGT vamos a ejecutar nuestro código cada minuto, en este caso lo hare con un Timer. Los ficheros descargados en el formato XML voy a guardar en **DBFS** en un sistema de archivos distribuido montado en un área de trabajo de Azure Databricks.

```
%fs ls dbfs:/DGT_MADRID/
```

path	name	size
dbfs:/DGT_MADRID/05_04_2020_23_36_intensidad.xml	05_04_2020_23_36_intensidad.xml	1822822
dbfs:/DGT_MADRID/06_04_2020_00_00_intensidad.xml	06_04_2020_00_00_intensidad.xml	1822765
dbfs:/DGT_MADRID/06_04_2020_00_01_intensidad.xml	06_04_2020_00_01_intensidad.xml	1822718
dbfs:/DGT_MADRID/06_04_2020_00_07_intensidad.xml	06_04_2020_00_07_intensidad.xml	1822747
dbfs:/DGT_MADRID/06_04_2020_00_14_intensidad.xml	06_04_2020_00_14_intensidad.xml	1822724
dbfs:/DGT_MADRID/06_04_2020_00_31_intensidad.xml	06_04_2020_00_31_intensidad.xml	1822674
dbfs:/DGT_MADRID/06_04_2020_00_35_intensidad.xml	06_04_2020_00_35_intensidad.xml	1822674
dbfs:/DGT_MADRID/06_04_2020_00_36_intensidad.xml	06_04_2020_00_36_intensidad.xml	1822691

Command took 5.65 seconds -- by imf.anastasia@gmail.com at 8/4/2020 23:27:49 on cluster_tfm

Img 16. Ficheros XML guardados en DBFS

5.3 SCRIPT DESCARGA DE DATOS Y PUBLICACIÓN EN KAFKA

1. Descargamos los datos
2.Comprobamos la fecha y hora de obtención de los datos
3. Transformamos los campos de localización
4. Convertimos **XML** a **GeoJson**
5. Publicamos el **GeoJson** en **Kafka**

```
import requests
import time
from datetime import datetime
import os
from pyproj import Proj
import pandas as pd
import numpy as np
from pyspark.sql.functions import UserDefinedFunction
from pyspark.sql.functions import *
from pyspark.sql.functions import udf
from pyspark.sql.types import StructType, StructField, DoubleType, StringType

from kafka import KafkaProducer
import json,time
from kafka.errors import KafkaError
```

Command took 0.44 seconds -- by imf.anastasia@gmail.com at 8/4/2020 20:24:59 on cluster_tfm

Img 16. Script Import de las Librerías

```
#1.Dascargamos el fichero xml de la DGT
def download_file(url):
    local_filename="/tmp/trafico-intensidad.xml"
    # NOTE the stream=True parameter
    r = requests.get(url, stream=True)
    with open(local_filename, 'wb') as f:
        for chunk in r.iter_content(chunk_size=1024):
            if chunk: # filter out keep-alive new chunks
                f.write(chunk)
                f.flush()
    return local_filename
```

Img 17. Script Función de descarga del fichero XML

```
def convert_toDouble(data):
  udf = UserDefinedFunction(lambda x: x.replace(",","."), StringType())
  data_out = data.withColumn("st_x", udf(col("st_x")).cast(DoubleType()))\
          .withColumn("st_y", udf(col("st_y")).cast(DoubleType()))
  return data_out
```

Command took 0.02 seconds -- by imf.anastasia@gmail.com at 8/4/2020 20:25:06 on cluster_tfm

Cmd 4

```
myProj = Proj("+proj=utm +zone=30 +ellps=intl +towgs84=0.0000,0.0000,0.0000,0.000000,0.000000,0.000000,0.00000000 +units=m +no_defs")
def convert_lon_lat(x, y):
  lon, lat = myProj(x,y, inverse=True)
  return [lon, lat]
```

Command took 0.02 seconds -- by imf.anastasia@gmail.com at 8/4/2020 20:25:08 on cluster_tfm

Cmd 5

```
schema = StructType([
    StructField("lon", DoubleType(), False),
    StructField("lat", DoubleType(), False)
])
```

Command took 0.02 seconds -- by imf.anastasia@gmail.com at 8/4/2020 20:25:13 on cluster_tfm

Cmd 6

```
lon_lat_udf = udf(lambda row: convert_lon_lat(row[0], row[1]), schema)
```

Img 18. Script Funciones para convertir las coordenadas de tipo UTM del centroide a WKT Point

```
def read_data(dbfs_name):
  data = spark.read\
    .format("com.databricks.spark.xml")\
    .option("rowTag", "pm")\
    .load(dbfs_name)
  data = convert_toDouble(data)
  df = data.withColumn("point", lon_lat_udf(struct([col('st_x'), col('st_y')])))\
    .select('*', 'point.*')
  pd_df= df.select("*").toPandas()
  return pd_df

```

Command took 0.03 seconds -- by imf.anastasia@gmail.com at 8/4/2020 20:25:23 on cluster_tfm

Img 19. Script función para convertir XML a DataFrame

```
#Producer Kafka
def on_send_success(record_metadata):
    #print(record_metadata.topic)
    #print(record_metadata.partition)
    #print(record_metadata.offset)
    print("Send Kafka Ok")

def on_send_error(excp):
    log.error('I am an errback', exc_info=excp)
    # handle exception

def send_kafka(data):
  producer = KafkaProducer(bootstrap_servers=['35.195.134.228:9092','35.195.167.204:9092','104.155.58.76:9092'],value_serializer=lambda m:
json.dumps(m).encode('utf-8'))
  producer.send('dgt', data).add_callback(on_send_success).add_errback(on_send_error)
```

Img 20. Script funciones Kafka Producer

```
def df_to_geojson(df, properties,fecha, lat='lat', lon='lon'):
    geo_obj = {}
    for _, row in df.iterrows():
      geo_obj['location'] = "POINT(" + str(row[lon]) + " " + str(row[lat]) + ")"
      geo_obj['fecha']= datetime.strptime(fecha, "%d/%m/%Y %H:%M:%S").strftime("%Y-%m-%d %H:%M:%S")
      for prop in properties:
        geo_obj[prop] = row[prop]
      #Enviar a Kafka
      #print(geo_obj)
      send_kafka(geo_obj)
```

Img 21. Script función convertir DF a GeoJson

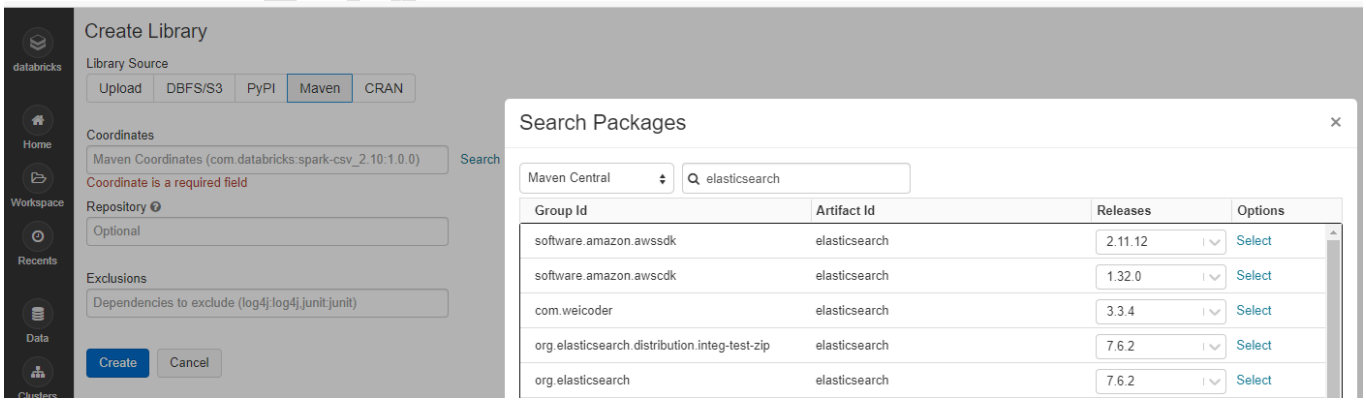
```python
def sleeper():
  fecha_ant = None
  while True:
    try:
      #1.Descargar el fichero xml desde la web DGT
      url = "http://informo.munimadrid.es/informo/tmadrid/pm.xml"
      res = download_file(url)
      print(res)
      #2.Subir los ficheros a dbfs
      now = datetime.now()
      dbfs_name= "dbfs:/DGT_MADRID/" + now.strftime("%d_%m_%Y_%H_%M") + "_intensidad.xml"
      print(dbfs_name)
      dbutils.fs.cp("file://" + res, dbfs_name)
      #3.Leer fecha del fichero
      df_fecha = spark.read\
                .format("com.databricks.spark.xml")\
                .option("rowTag", "pms")\
                .load(dbfs_name)
      if fecha_ant !=  df_fecha.select("fecha_hora").collect()[0]:
        # actualizamos la var fecha ant
        print('Fecha anterior: %s' % fecha_ant)
        fecha_ant = df_fecha.select("fecha_hora").collect()[0]
        print('Fecha actual: %s' % fecha_ant)
        # Leemos el xml, convertire a json y mandar a kafka
        df_data=read_data(dbfs_name)
        df_data= df_data.replace(np.nan, -1)
        df_data["color"] = [nivel_to_color(x) for x in df_data['nivelServicio']]
        cols = ['accesoAsociado','carga','descripcion','error','idelem','intensidad','intensidadSat','nivelServicio','ocupacion','subarea','velocidad','color']
        geojson = df_to_geojson(df_data, cols, fecha_ant.fecha_hora)
      #else:
        #Eliminas el fichero del dbfs
      dbutils.fs.rm(dbfs_name)
      print('Se ha elinado: %s' % dbfs_name)
      os.remove(res)
      print('Se ha elinado: %s' % res )
    except Exception as ex:
      print(ex)
      continue

    # Run our time.sleep() command,
    # and show the before and after time
    print('Before: %s' % time.ctime())
    time.sleep(60)
    print('After: %s\n' % time.ctime())
```

Img 22. Script función principal Timer.

Script completo formato html

GetData_DGT.html

Para siguiente script voy a tener que crear una nueva cuenta en **DataBricks**, para poder ejecutar los dos scripts al mismo tiempo y a de más hay que abrir en deferentes navegadores para que no se haga el autologin.
Este script creamos con **pysparkStreaming**
Agregamos las librerías necesarias



Img 22. Instalación de librerías en DataBricks

Img 23. Librerías necesarias para Elastic y Kafka Comsumer

5.4 SCRIPT CONSUMER KAFKA E INSERT EN ELASTICSEARCH

1. Suscribirse al Topic
2. Insertar en Elastic

Script Kafka Consumer

```
import sys
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils
from elasticsearch import Elasticsearch
import json
```

```
Command took 0.25 seconds -- by anastasia.lukina.es@gmail.com at 8/4/2020 22:49:54 on cluster_tfm (clone)
```

Img 24. Script Import Librerias

```
#Las IpsExternas para conectar a Elastic
es = Elasticsearch(["34.77.57.255:9200", "35.241.242.33:9200", "35.242.150.103:9200"])
```

```
Command took 0.07 seconds -- by anastasia.lukina.es@gmail.com at 8/4/2020 22:49:56 on cluster_tfm (clone)
```

```
Cmd 3
```

```
#Inser en Elastic
def send_elastic(data):
  es.index(index="dgt", doc_type="_doc", body=data)

```

Img 25. Script función Insert en Elastic

```
if __name__ == "__main__":
  try:
    ssc = StreamingContext(sc, 2)
    #Conector a Kafka
    kvs = KafkaUtils.createDirectStream(ssc, ["dgt"], {"metadata.broker.list": '35.195.134.228:9092'})
    lines = kvs.map(lambda x: x[1])
    #Mapeamos a Json
    parsed = kvs.map(lambda x: json.loads(x[1]))
    def function(x):
      y = x.collect()
      for d in y:
        data= json.dumps(d)
        print(data)
        send_elastic(data)
    #Recorremos RDD
    parsed.foreachRDD(lambda x :function(x))
    ssc.start()
    ssc.awaitTermination()
  except Exception as ex:
    print(ex)
    pass
```

Img 26. Script main

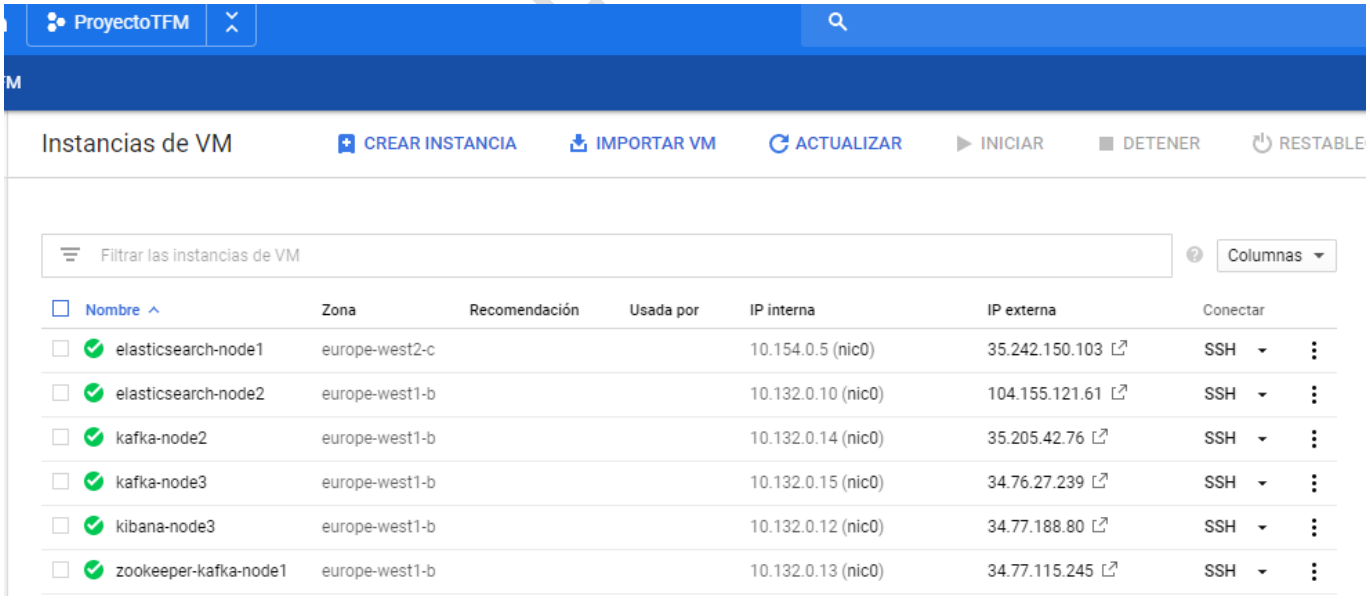
Script completo en el formato html

KafkaConsumer.html

6. RESULTADOS

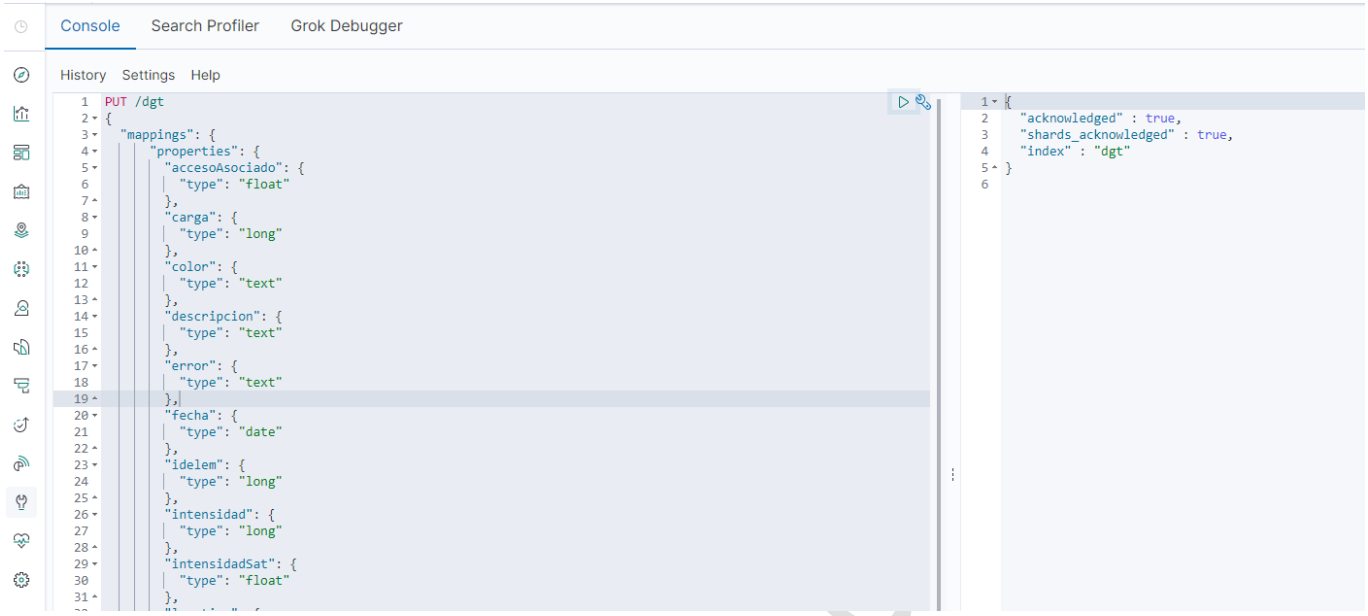
Ya están todos los pasos completados, ahora podemos proceder a realizar las pruebas del flujo completo.

1. Arrancamos los Clústeres en Google Cloud Platform



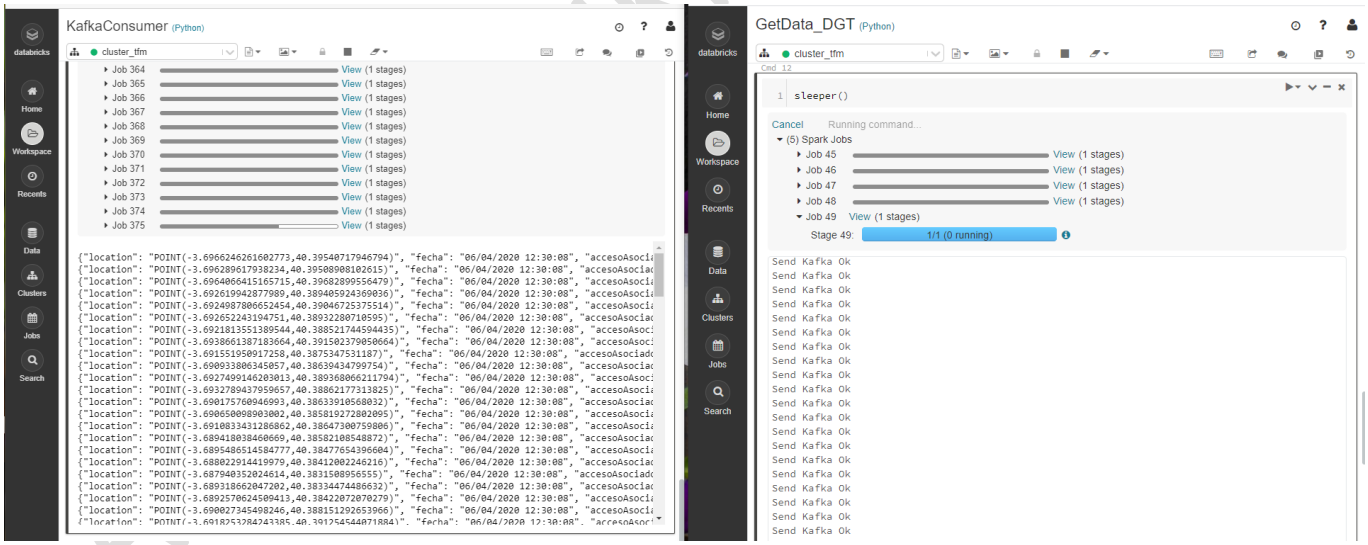
Img 27. MV Google Cloud Platform

2. Creación de Index con sus tipos en Elasticsearch



Img 28. Creación de Index en Elastic

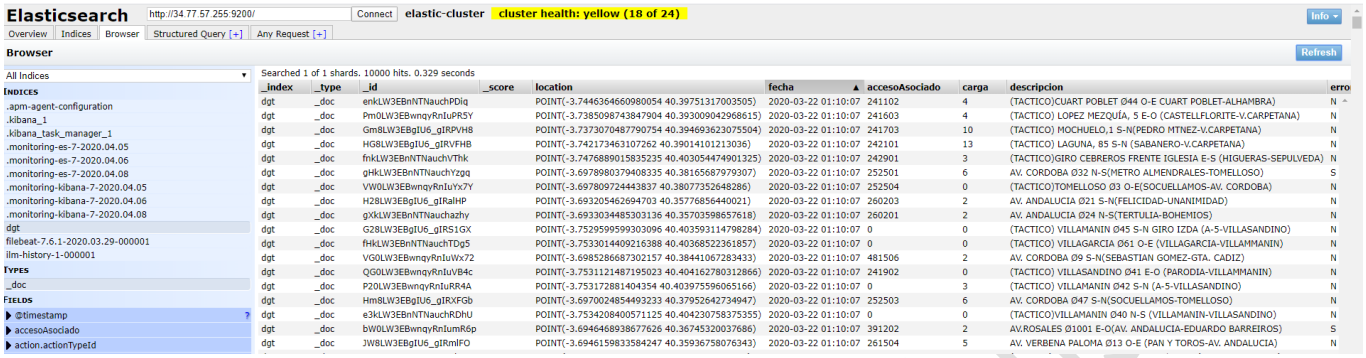
3. Ejecutamos los dos scripts al mismo tiempo para comprobar que justo al publicar recibimos los mensajes en tiempo real.



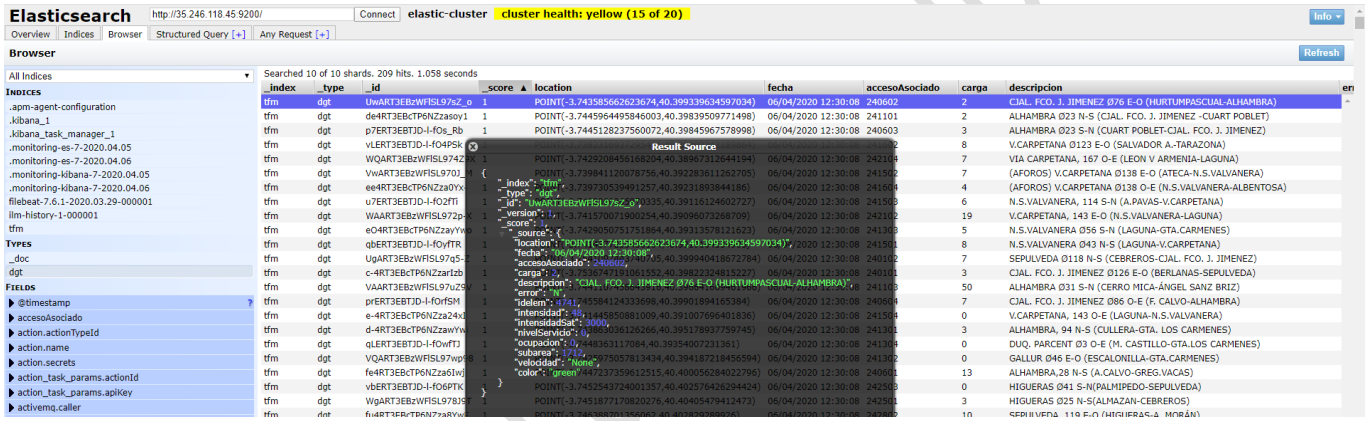
Img 29. Ejecución de los scripts

Como podemos observar en la Img 28, enviamos y recibimos los mensajes de Kafka en Tiempo Real.

4. Comprobar la inserción de los datos en Elastic.

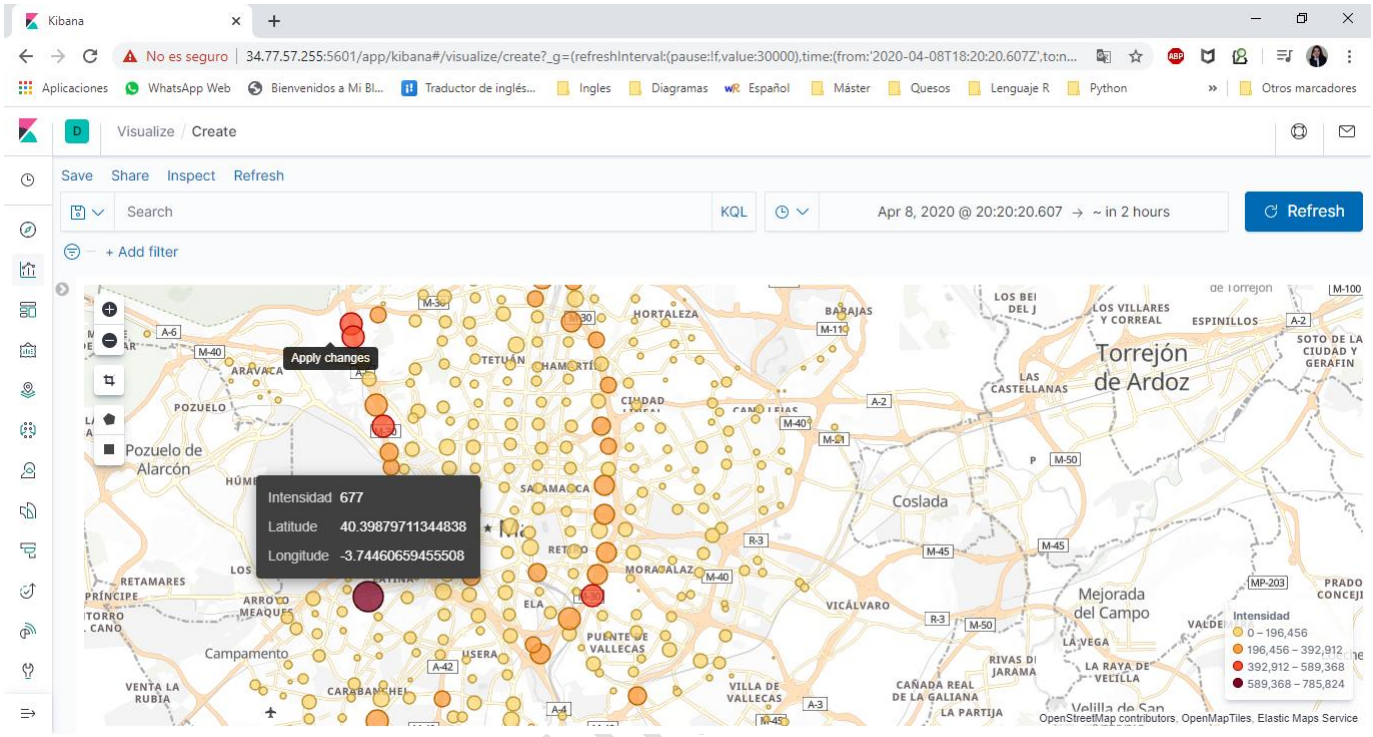


Img 30. Browser Plugin Chrome

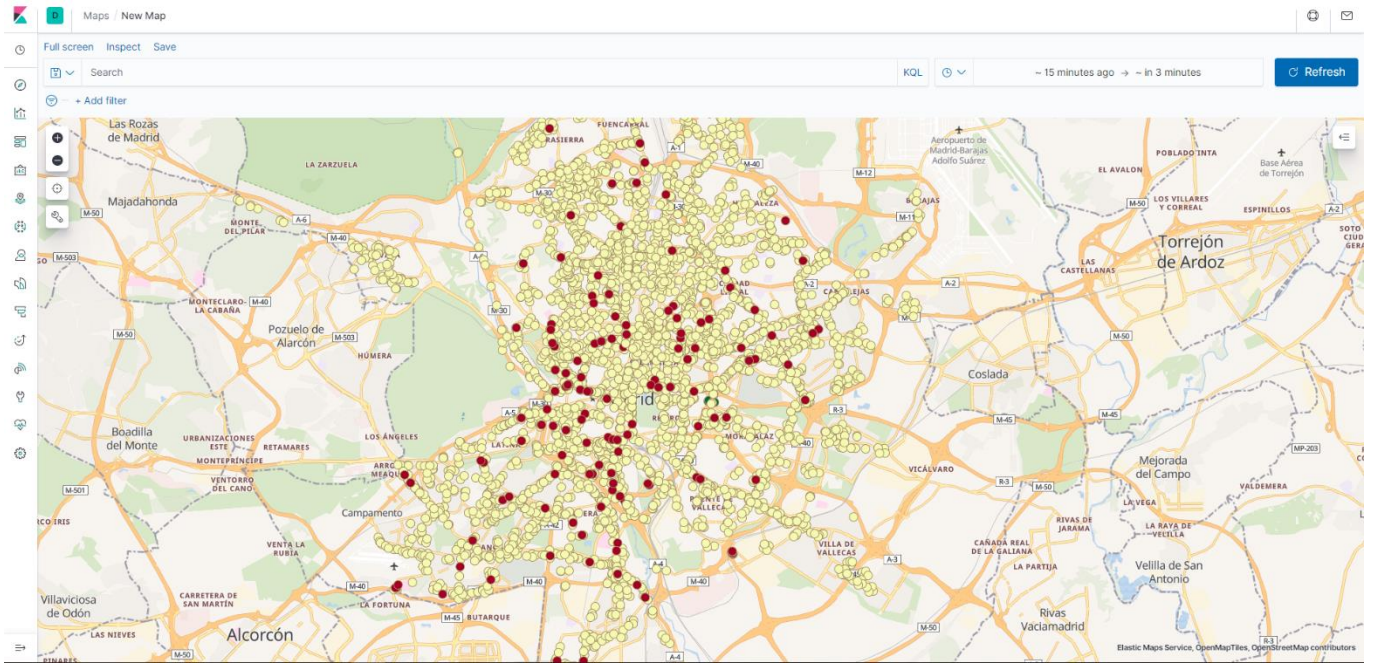


Img 31. Visualización de los datos con el Plugin Chrome

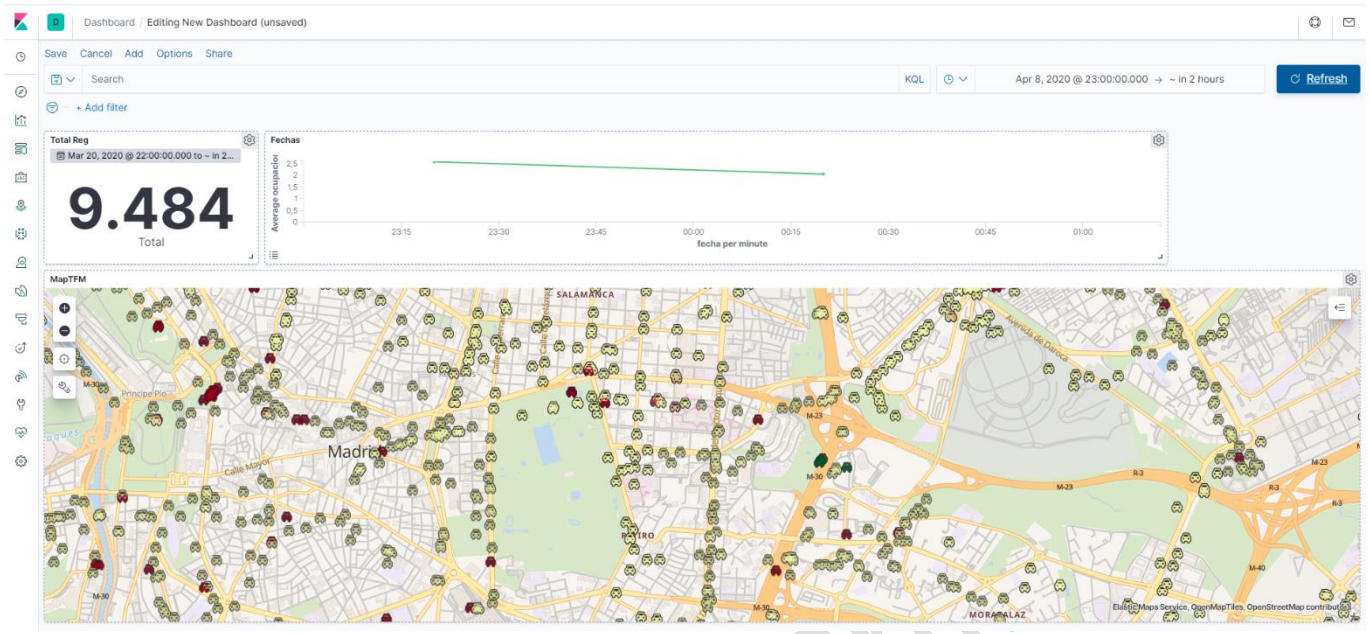
5. Visualización de datos en Kibana
   En la siguiente imagen podemos ver los datos en tiempo real según están insertando los datos se van pintando en la pantalla



Img 32. Vista1



Img 33. Vista2



Img 34. Vista3

7. CONCLUSIONES

1. Problemas encontrados:

1.1 A la hora de montar los **Clústeres** he encontrado varios problemas:
1.2 Poca información de como montar y configurar un **Clúster rea**l (con las plataformas AWS, Azure y GCP existen los clústeres preconfigurados lo único que hay que seguir los pasos donde nos quedamos sin saber cómo se configura por debajo)
1.3 En caso de **Clúster de Kafka** la configuración de Publicador y Consumidor tenía que poner los IPs externos, y cada vez que arrancaba de nuevo las MV las IPs externas cambiaban, por lo cual siempre tenía que cambiar los ficheros de configuración en los 3 nodos. La configuración del **Clúster Kafka en mi caso me ha resultado más costoso del todo el trabajo**.
1.4 Al terminar todo proceso por primera vez no podría crear las capas geográficas en **Kibana**, porque no encontraba documentos de tipo geoespaciales.
1.5 Al utiliza el clúster de **DataBrikcs** solo se podría ejecutar un script y para realizar la prueba completa de publicar y recibir los mensajes **de Kafka en tiempo real** y que el ejemplo se acerca más al caso real, he tenido que crear otra cuenta de prueba con otro correo electrónico y abrir en otro navegador.

2. Resultados:

Finalmente, después de miles de pruebas he conseguido realizar el trabajo tal y como se planteaba desde el principio.

8. ASIGNATURAS/MÓDULOS RELACIONADOS

Parra desarrollar el proyecto propuesto me he basado en los siguientes módulos/asignaturas:

1. **Mod 6. Infraestructura Big Data**
   1.1 Procesamiento de datos con Spark
   1.2 Arquitecturas En Streaming
   1.3 Componentes De Arquitectura En Streaming
   1.4 Plataformas Y APIS En La Nube
2. **Mod 7. Almacenamiento e Integración de Datos**
   1.1 Adquisición De Datos
3. **Mod 8. Valor y Contexto de la Analítica Big Data**
   1.1 Proyectos De Big Data
4. **Mod 9. Aplicaciones Analíticas**
   1.1 Caso De Estudio De Técnicas De Recuperación De Información

9. MÉTODOS, MATERIALES Y TECNOLOGÍAS

1. **Google Cloud Platform (GCP) – Networking**
   1.1. https://youtu.be/3ioh96OWolI
   1.2. https://youtu.be/BPBUAn9EGh0
   1.3. https://cloud.google.com/compute/docs/instances/connecting-to-instance
2. **Apache Kafka**
   2.1. https://kafka.apache.org/
   2.2. https://docs.bitnami.com/oci/infrastructure/kafka/administration/create-cluster/
   2.3. Kafka => Sparkstreaming => Elastic https://hadoopist.wordpress.com/2016/06/17/kafka-sparkstreaming-elastic/
   2.4. https://www.datasciencecentral.com/profiles/blogs/setting-up-your-first-kafka-development-environment-in-google
   2.5. https://www.digitalocean.com/community/tutorials/how-to-install-apache-kafka-on-centos-7
3. **Elasticsearch + kibana**
   3.1. https://www.elastic.co/guide/en/elasticsearch/reference/current/rpm.html
   3.2. https://www.elastic.co/guide/en/kibana/current/rpm.html
   3.3. https://www.elastic.co/guide/en/elasticsearch/reference/current/cluster-health.html
4. **Datos del tráfico en tiempo real**
   4.1. https://datos.madrid.es/
5. **Obtención y tratamiento de datos**

   5.1. geo-point elastic https://www.elastic.co/guide/en/elasticsearch/reference/7.6/geo-point.html#geo-point
   5.2. Códigos EPSG (European Petroleum Survey Group) https://www.mapa.gob.es/es/cartografia-y-sig/ide/directorio_datos_servicios/caracteristicas_wms.aspx
6. **Otros recursos Cursos Udemy**
   6.1. Hadoop Big Data desde cero https://www.udemy.com/course/monta-un-cluster-hadoop-big-data-desde-cero/learn/lecture/4858040#overview

6.2. Procesando el Big Data con Apache Spark (en español) https://www.udemy.com/course/programacion-con-apache-spark/learn/
6.3. Domina Apache Spark 2.0 con Scala, curso intensivo https://www.udemy.com/course/apache-spark-y-scala-curso-intensivo-apache-spark/learn/
6.4. Apache Kafka Series - Learn Apache Kafka for Beginners v2 https://indra.udemy.com/course/apache-kafka/learn/
6.5. Apache Kafka - Real-time Stream Processing (Master Class)
6.6. https://indra.udemy.com/course/kafka-streams-real-time-stream-processing-master-class/learn/

7. Herramienta para crear diagramas gratis

7.1 https://app.diagrams.net/