

Análisis en tiempo real de las respuestas a invitaciones en Meetup

<https://github.com/LuisRomeroCasado/ProyectoFinMaster.U-TAD>



**Para ver código
pincha aquí**

Autor: Luis Romero Casado

Tutor: Iván de Prado Alonso

Fecha: 27/07/2016

Índice

1. INTRODUCCIÓN	3
2. ARQUITECTURA	4
2.1 PROCESO DE SELECCIÓN DE ARQUITECTURA.	4
2.2 MÓDULO DE ADQUISICIÓN DE DATOS.....	5
2.2.1 <i>Websocket</i>	6
2.2.2 <i>Apache Storm</i> :.....	7
2.2.3 <i>Apache Kafka</i> :.....	8
2.3 MODULO DE PROCESAMIENTO.	8
2.3.1 <i>Spark Streaming</i>	8
2.3.2 <i>Spark SQL</i>	9
2.4 MÓDULO DE ALMACENAMIENTO.....	10
2.5 MÓDULO DE VISUALIZACIÓN.	10
3. DESPLIEGUE DEL PROYECTO	11
3.1 CREACIÓN DE CLUSTER EN AWS SERVIES/EC2	11
4. DASHBOARD	17
4.1 REAL TIME:	17
4.1.1 <i>Respuestas Dashboard</i> :	17
4.1.2 <i>Trending topic Dashboard</i>	18
4.2 BATCH:	18
4.2.1 <i>Grupos Dashboard</i>	18
4.2.2 <i>Eventos Dashboard</i>	19
5. CONCLUSIONES	20
6. PUNTOS PENDIENTES PARA FUTURAS IMPLEMENTACIONES	21
6.1 ALMACENAR LOS DATOS EN BRUTO FUERA DE ELASTICSEARCH.	21
6.2 AÑADIR NUEVAS FUENTES.....	21
6.3 AÑADIR UN NUEVO COMPONENTE REDIS.	21
6.4 MEJORAR LOS DASHBOARD	21
6.5 PROFUNDIZAR EN EL TRATAMIENTO BATCH DE LA INFOMACIÓN.....	21
6.6 AÑADIR SPARKML	22
6.7 GENERACIÓN DE ALERTAS	22
7 BIBLIOGRAFÍA	23

1. Introducción



Meetup es un web que permite crear eventos públicos y que la gente participe en ellos. Publica una stream donde se puede seguir en tiempo real las respuestas a las invitaciones a los diferentes grupos.

El objetivo de este proyecto es usar la información provista por **Meetup** usando tecnologías Big Data de forma que se obtengan resultados interesantes.

Para ello se ha creado un dashboard donde se puede visualizar el análisis de la información:

- Real time:
 - o Análisis de respuestas a eventos.
 - o Análisis de trending topics.
- Procesamiento batch:
 - o Análisis de información sobre grupos.
 - o Análisis de información sobre eventos.

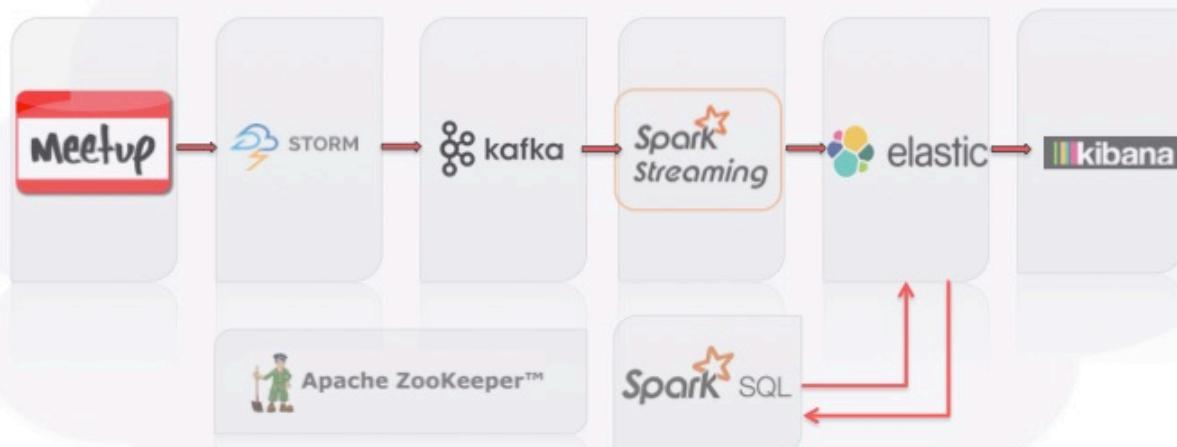
Una vez finalizado el proyecto, más que la información que hemos conseguido sacar por pantalla y el resultado del análisis, vemos que lo importante ha sido poder aplicar parte de los conceptos y tecnologías aprendidas a un entorno real. La mayor dificultad ha sido poder unir todas las piezas, y que puedan trabajar en un cluster distribuido.

Se ha seleccionado este proyecto por varios motivos:

- Posibilidad de trabajar tanto en “Real time” como con procesamiento Batch.
- Facilidad de acceso a la plataforma **Meetup**. Tiene a disposición de los usuarios una API muy bien documentada, que en mi opinión es muy buena para realizar prototipos como el que hemos hecho.
- Fuente de datos. Uno de los problemas a la hora de seleccionar proyectos era encontrar una fuente de datos que pudiera analizar, de un tamaño considerable, y que fuera pública.

2. Arquitectura

Meetup



2.1 Proceso de selección de arquitectura.

Lo primero fue analizar la fuente de datos y el API que proporciona **Meetup**. Para el procesamiento de respuestas en streaming recomienda el acceso a través de websocket, por lo que ese fue el primer punto a desarrollar.

Una vez realizado el test de acceso, la primera posibilidad fue conectarlo con Spark Streaming para comenzar a trabajar con los datos en tiempo real. También habíamos trabajado con Storm, pero para una arquitectura tipo Lambda me pareció mejor opción utilizar Spark con sus módulos Streaming y SQL.

El funcionamiento era correcto, pero surgía la duda de cómo poder manejar en un entorno productivo la posibilidad de tener que solucionar problemas o realizar actualizaciones sobre el módulo de procesamiento. Para solucionarlo se añadió una capa más, Apache Kafka. Trabaja como una cola de mensajes que nos permite poder parar el procesamiento sin perder datos.

Se desarrolló un módulo de Kafka con un consumer que accedía mediante el websocket a la fuente de datos, y con un producer conectado a Spark.

Comentando la idea con el tutor surgió el problema de trabajar en distribuido. El desarrollo realizado no escalaba, lo que podía suponer un cuello de botella en el acceso a datos. Me sugirió la idea de añadir un nuevo componente en la fase de adquisición, Storm.

Creando una tipología de Storm solucionaríamos el problema de la escalabilidad, además de dar la posibilidad de generar distintos Spouts para el acceso a varias fuentes. La fase de adquisición quedó definida como:

Source (Websocket) -> Strorm -> Kafka

La fase de procesamiento se había seleccionado Spark y continué con esa ida desarrollando primero el módulo de real time con Spark Streaming.

Para el almacenamiento la primera opción fue utilizar Cassandra. Además, para el acceso indexado a la información se añadiría Solr, que es el componente que Datastax tiene definido dentro de su DSE para el real time. El problema que surgió aquí era la visualización de los datos. No encontré una herramienta, por lo menos gratuita, que permitiera visualizar los datos de forma sencilla. Por tanto habría que desarrollar una web, por ejemplo en nodejs, lo que aumentaría el tiempo.

Lo más sencillo que vimos para la visualización fue Kibana, que permite la generación de un Dashboard con vistas de presentación muy sencillo. Para indexar los datos era mejor por tanto ElasticSearch.

En este punto tengo que decir que la arquitectura está incompleta, ya que ElasticSearch ha quedado como componente de almacenamiento, cuando creo que debería existir otro externo para los datos en bruto y ElasticSearch sólo para los indexados.

Pasamos a definir cada uno de los componentes por separado.

2.2 Módulo de adquisición de datos.

El proceso de adquisición de datos ha quedado definido de la siguiente forma:

Source (Websocket) -> Strorm -> Kafka



2.2.1 Websocket

Para el acceso al streaming de datos se ha utilizado un websocket desarrollado en Java, que se conecta al API de **Meetup**:

`ws://stream.meetup.com/2/rsvps`

La información sobre el API se puede encontrar en el enlace que indico a continuación:

http://www.meetup.com/es-ES/meetup_api/docs/stream/2/rsvps/?uri=%2Fmeetup_api%2Fdocs%2Fstream%2F2%2Frsvps%2F#websockets

Lenguaje: Java

El código desarrollado: Tfm-uta-stormMeetup -> com.tfm.utad.stormMeetup.websocket.

Los datos que obtenemos están en un Json con la siguiente estructura de ejemplo:

```

MeetupRsvp(
    1615875021,
    yes,
    0,
    1466357803713,
    public,
    RsvpEvent(232009717,
        Some(Poker night... (Intermediate level)),
        Some(http://www.meetup.com/OTPConnections/events/232009717/),
        Some(1469233800000)),
    RsvpGroup(19876020,
        OTP Connections,
  
```

```

Some(Alpharetta),
Some(us),
Some(GA),
Some(OTPConnections),
Some(34.12),
Some(-84.3),
List(RsvpGroupTopics(Some(Poker),Some(poker)),
     RsvpGroupTopics(Some(Camping),Some(camping)),
     RsvpGroupTopics(Some(Bicycling),Some(bike)),
     RsvpGroupTopics(Some(Wine),Some(wine)),
     RsvpGroupTopics(Some(Hiking),Some(hiking)),
     RsvpGroupTopics(Some(Dining Out),Some(diningout)),
     RsvpGroupTopics(Some(New In Town),Some(newintown)),
     RsvpGroupTopics(Some(Running),Some(running)),
     RsvpGroupTopics(Some(Social),Some(social)),
     RsvpGroupTopics(Some(Fun Times),Some(fun-times)),
     RsvpGroupTopics(Some(Adventure),Some(adventure)),
     RsvpGroupTopics(Some(Coffee and Tea Socials),Some(coffee-tea-socials)))),
RsvpMember(46028242,
            Some(Milton),
            None,
            Some(http://photos2.meetupstatic.com/photos/member/9/a/6/2/thumb\_185799522.jpeg)),
RsvpVenue(24506449,
          Some(Someone's Home),
          Some(34.075375),
          Some(-84.29409)))

```

2.2.2 Apache Storm:

Como se ha comentado, la idea de añadir este componente es dar la posibilidad de consumir varias fuentes de distintos tipos, siendo escalable la adquisición de datos. Para este proyecto sólo tendremos un punto de acceso, pero permite ampliarlo de forma sencilla creando nuevos Spouts.

En nuestro caso, y aunque esto no aporta eficiencia sino que más bien penaliza, hemos añadido una partición por id de los datos de entrada para poder paralelizar la lectura. Mi idea del proyecto no es crear una aplicación para producción sino aprender distintas formas de trabajar y probarlas en un entorno real.

La tipología está compuesta por un spout que accede a los datos, y un bolt que va a ser el encargado de enviar a Kafka. Se ha utilizado un tipo ya definido de bolt, KafkaBolt dado por la propia arquitectura de Storm (storm.kafka.bolt.Kafkabolt). de forma sencilla permite almacenar directamente los datos en kafka sin necesidad de utilizar el componente Kafka que habíamos desarrollado inicialmente.

Se ha definido un paralelismo de 2 Spouts, uno lee los id's pares y otro los impares. Y se conectan por shuffleGrouping a otros dos bolts de Kafka.

El código desarrollado:

Lenguaje: Java

El código desarrollado: Tfm-uta-stormMeetup -> com.tfm.utad.stormMeetup.storm.

2.2.3 Apache Kafka:

Cola de mensaje utilizada como muro de contención entre la fuente y la fase de procesamiento. Se ha creado un TOPIC (“TOPIC_RSVPS”), pero permitiría crear tanto topics como fuentes de datos necesitemos analizar. Por ejemplo, inicialmente probamos a crear una más para el procesamiento de comentarios con el API de streaming que también proporciona [Meetup](#).

En local utilizamos el zookeeper que viene con la distribución de Kafka, pero en el cluster definimos un componente aparte den una de las máquinas.

Para el desarrollo, como hemos comentado antes, hemos utilizado el KafkaBolt que proporciona Storm.

El código desarrollado:

Lenguaje: Java

El código desarrollado: Tfm-uta-stormMeetup -> com.tfm.utad.stormMeetup.storm

2.3 Modulo de procesamiento.

2.3.1 Spark Streaming

En este módulo procesamos la información en streaming que nos llega de la cola Kafka “TOPIC_RSVPS”, y la almacenaremos en elasticsearch una vez tratada.

Podemos diferenciar varias fases:

- Obtenemos el streaming de Kafka y parseamos el JSON que nos llega al modelo definido con la estructura que indica el API de meetup.

API: http://www.meetup.com/es-ES/meetup_api/docs/stream/2/rsvps/?uri=%2Fmeetup_api%2Fdocs%2Fstream%2F2%2Frsvps%2F#websockets

Estructura definida en el proyecto: Tfm-uta-sparkMeetup -> com.tfm.utad.sparkMeetup.model

- Almacenamiento de ese RDD en Elasticsearch. Se trataría del almacenamiento de los datos en bruto para luego poder ser procesados en batch (meetup_rsvps).
Esta parte estaría incompleta, veo la necesidad de utilizar otra BdD como puede ser Cassandra donde dejemos los datos estructurados para que puedan ser tratados tanto por el batch, como cuando tengamos la necesidad de regenerar los índices de Elasticsearch. Otra posibilidad también sería por ejemplo utilizar S3.

- Tratamiento del Stream y generación de un índice en Elasticsearch (meetup_by_country) donde obtenemos la información agrupada por país de las respuestas (yes/no), los asistentes y los invitados para una fecha.
- Tratamiento del Stream y generación de un índice en ElasticSearch (meetup_global_trending_topics) donde almacenamos la información de los trending topics agrupados por nombre.
- Tratamiento del Stream y generación de un índice en ElasticSearch (meetup_country_trending_topics) donde almacenamos la información de los trending topics agrupados por nombre y país.

Para estos dos últimos casos se utiliza las ventanas de SparkStreaming, ya que lo que buscamos es conocer los topics que son tendencia en determinados espacios de tiempo.

Lenguaje: Scala

El código desarrollado: Tfm-uta-sparkMeetup -> com.tfm.utad.sparkMeetup.streaming

2.3.2 Spark SQL

Para el procesamiento Batch utilizamos SparkSQL. Leemos los datos en bruto de Elasticsearch y hacemos algún tratamiento sobre la información para generar nuevos índices. La idea es definir cada cuanto tiempo queremos realizar este procesamiento Batch, pero obtener la información en bruto desde esa fecha y actualizar los índices.

- Leemos de ElasticSearch los datos a partir de una fecha y generamos una tabla temporal (RSVPS).
- Creamos dos tablas temporales con la información general de los grupos y los eventos (GROUPS y EVENTS).
- En otras dos tablas temporales tendremos la información procesada de estos grupos y eventos. GROUP_COUNT: información agrupada por id_grupo de la suma de miembros, eventos asociados a este grupo, y las sumas de respuestas yes/no, asistentes e invitados. EVENT_COUNT: información similar pero agrupada por evento.
- Hacemos el join de las tablas de cuentas y definiciones y generamos dos nuevos índices "meetup_groups" y "meetup_events"

Esta parte también ha quedado incompleta. La idea inicial era añadir algún punto más que dejamos pendiente para una aplicación cuando tengamos más tiempo:

- Profundizar más en el análisis de los datos de respuestas creando nuevas consultas.
- Análisis de los datos sacando información con la mezcla de otras fuentes como podría ser el stream de comentarios.
- Creación de un nuevo componente, posiblemente REDIS, donde definir diccionarios que nos sirvan para enriquecer la información que tratamos. Categorías, nombres de países, ...
- Hay definido en el proyecto un pequeño desarrollo en el que accedemos a otros servicios del API de meetup. Lo que se pretendía era acceder a la información de cada grupo procesado para obtener lo que en el stream no viene definido. Fecha creación, miembros,...

Conseguimos acceder, pero nos cortaba la conexión. Seguramente haya que conectarse con algún protocolo de autorización que no hemos tenido tiempo de investigar a fondo.
(Tfm-uta-sparkMeetup -> com.tfm.utad.sparkMeetup.HttpResolver)

- MachingLearning. Una de las partes que más me ha gustado era el módulo de SparkML. Tenía pensado aplicar algún algoritmo de predicción para ver por ejemplo una predicción de asistencia a eventos, de cancelaciones, o de recomendación de grupos.

Lenguaje: Scala, SQL

El código desarrollado: Tfm-uta-sparkMeetup -> com.tfm.utad.sparkMeetup.batch

2.4 Módulo de almacenamiento.

Para el almacenamiento había pensado en utilizar cassandra, pero como he comentado en la introducción, no encontré una manera de visualizar los datos de manera sencilla. Finalmente opté por ElasticSearch.

Se define un cluster e elasticsearch con varios nodos donde almacenamos la información indexada.

- meetup_rsvps: datos en bruto que procesamos del streaming
- meetup_by_country: análisis de las respuestas en streaming por país.
- meetup_global_trending_topics: análisis de las repuestas en streaming agrupando por los topics que son tendencia en una ventana determinada.
- meetup_country_trending_topics: igual que el anterior pero agrupando además por país.
- meetup_groups: información procesada en batch con información diaria (o como definamos que lanzamos el batch) para los grupos.
- meetup_events: información procesada en batch con información diaria (o como definamos que lanzamos el batch) para los eventos.

Definición índices:

https://github.com/LuisRomeroCasado/ProyectoFinMaster.U-TAD/blob/master/tfm-utad-elasticserach/elasticsearch_index.sh

2.5 Módulo de visualización.

Necesitaba una forma sencilla de mostrar los datos indexados en ElasticSearch, sobre todo para el real time. Kibana nos permitía de forma sencilla generar un Dahsboard accediendo a la información y poder realizar en tiempo real actualizaciones de la información. No permite actualizaciones inferiores a 5s, que haya visto, pero puedes generar gráficas y consultas sobre ellas de forma muy rápida.

Definición de Dashboard:

<https://github.com/LuisRomeroCasado/ProyectoFinMaster.U-TAD/blob/master/tfm-utad-kibana/tfm-utad-kibana.json>

3. Despliegue del proyecto



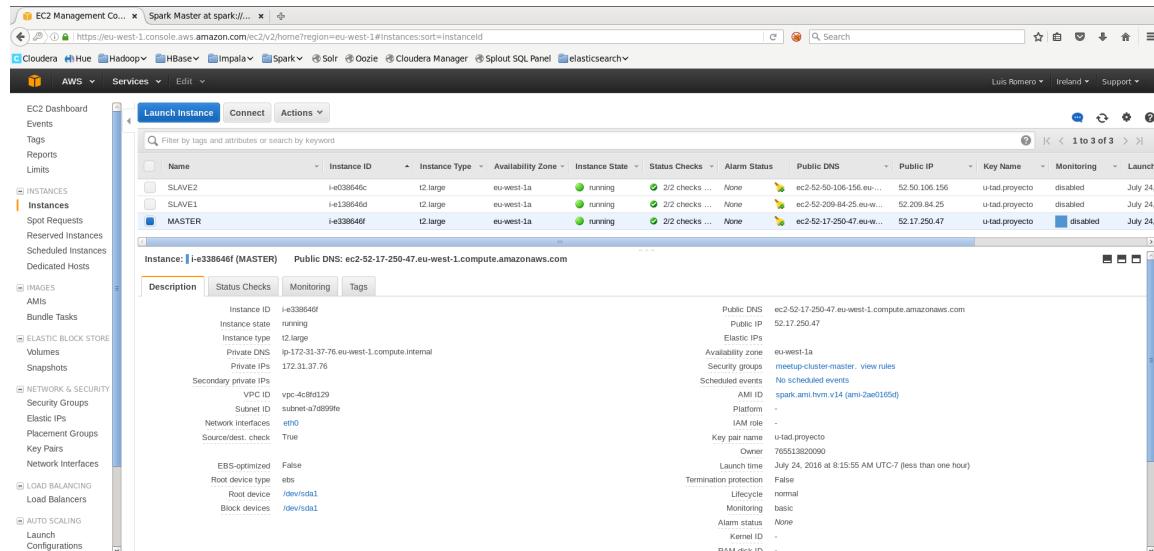
Uno de las mayores dificultades que he tenido durante el desarrollo del proyecto, ha sido intentar solucionar los problemas de infraestructura que por la capacidad de mi equipo iban surgiendo. Esto ha sido lo que me ha impedido profundizar más en el tratamiento de la información.

Aunque como punto positivo, te da la posibilidad de enfrentarte a problemas que en mi trabajo actual no surgen ya que existe un departamento de infraestructura dedicado sólo a esta parte.

Con la máquina virtual utilizada durante el master definía cada componente y más o menos conseguía hacerlo funcionar todo junto forzando al máximo la capacidad. El problema es que era imposible con la memoria actual crear un cluster, por lo que se ha optado por hacerlo en Amazon AWS.

Configuración de AWS.

3.1 Creación de cluster en AWS Services/EC2



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS	Public IP	Key Name	Monitoring	Launch
SLAVE2	i-e038646c	t2.large	eu-west-1a	running	2/2 checks ...	None	ec2-52-50-106-156.eu...	52.50.106.156	u-tad.proyecto	disabled	July 24,
SLAVE1	i-e139646d	t2.large	eu-west-1a	running	2/2 checks ...	None	ec2-52-209-84-25.eu...	52.209.84.25	u-tad.proyecto	disabled	July 24,
MASTER	i-e338646f	t2.large	eu-west-1a	running	2/2 checks ...	None	ec2-52-17-250-47.eu...	52.17.250.47	u-tad.proyecto	disabled	July 24,

Instance: i-e338646f (MASTER) Public DNS: ec2-52-17-250-47.eu-west-1.compute.amazonaws.com

Description	Status Checks	Monitoring	Tags
Instance ID: i-e338646f	2/2 checks ...	disabled	
Instance state: running			
Instance type: t2.large			
Private DNS: ip-172-31-37-76.eu-west-1.compute.internal			
Private IPs: 172.31.37.76			
Secondary private IPs:			
VPC ID: vpc-4c48d129			
Subnet ID: subnet-a7d899fe			
Network Interfaces:			
SourceDest check: True			
EBS-optimized: False			
Root device type: ebs			
Root device: /dev/sda1			
Block devices: /dev/sda1			

Public DNS: ec2-52-17-250-47.eu-west-1.compute.amazonaws.com
 Public IP: 52.17.250.47
 Elastic IP:
 Availability zone: eu-west-1a
 Security group: meetup-cluster-master, view_rules
 Scheduled events: No scheduled events
 AMI ID: spark.ami.hvm.v14 (ami-2ae0165d)
 Platform: -
 IAM role: -
 Key pair name: u-tad.proyecto
 Owner: 765513820090
 Launch time: July 24, 2016 at 8:15:55 UTC-7 (less than one hour)
 Termination protection: False
 Lifecycle: normal
 Monitoring: basic
 Alarm status: None
 Kernel ID: -
 RAM disk ID: -

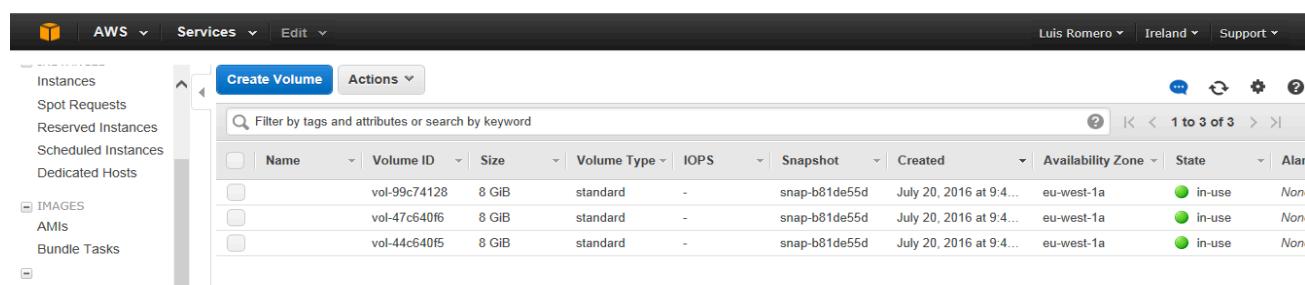
Se han creado 3 instancias:

- Máquina t2.large con Amazon Linux donde incluimos (MASTER):
 - o Nodo Master de Spark.
 - o Zookeeper.
 - o Nimbus de Storm y un supervisor.
 - o Nodo Kafka
 - o Nodo Elasticsearch.
 - o Kibana

- Dos Máquinas t2.large con Amazon Linux donde incluimos en cada una (SLAVE1/SLAVE2):
 - o Workers de Spark.
 - o Supervisor de Storm
 - o Nodo Kafka.
 - o Nodo Elasticsearch.

No es una arquitectura de cluster que se utilizaría en producción, pero la intención era crear un cluster que diera la posibilidad de trabajar en una arquitectura distribuida, con posibilidad de escalar fácilmente.

Con estas máquinas tenemos definidos los volúmenes correspondientes. Si fuera necesario, de forma sencilla se pueden añadir volúmenes EBS adicionales.



Name	Volume ID	Size	Type	IOPS	Snapshot	Created	Availability Zone	State	Alarms
vol-99c74128	8 GiB	standard	-	-	snap-b81de55d	July 20, 2016 at 9:4...	eu-west-1a	in-use	None
vol-47c640f6	8 GiB	standard	-	-	snap-b81de55d	July 20, 2016 at 9:4...	eu-west-1a	in-use	None
vol-44c640f5	8 GiB	standard	-	-	snap-b81de55d	July 20, 2016 at 9:4...	eu-west-1a	in-use	None

Se ha utilizado sparkEc2, que nos permite crear un cluster de spark en EC2 de forma sencilla. Se configura con las credenciales dadas de alta en amazon, y se define la máquina a utilizar, el número de workers, y la región donde quieras desplegarlo.

Crear cluster:

```
Create cluster: ./spark-ec2 --key-pair=u-tad.proyecto --identity-file=u-tad.proyecto.pem --region=eu-west-1 -  
-instance-type=t2.large -s 2 launch meetup-cluster
```

```
Master: ./spark-ec2 --key-pair=u-tad.proyecto --identity-file=u-tad.proyecto.pem --region=eu-west-1 get-  
master meetup-cluster
```

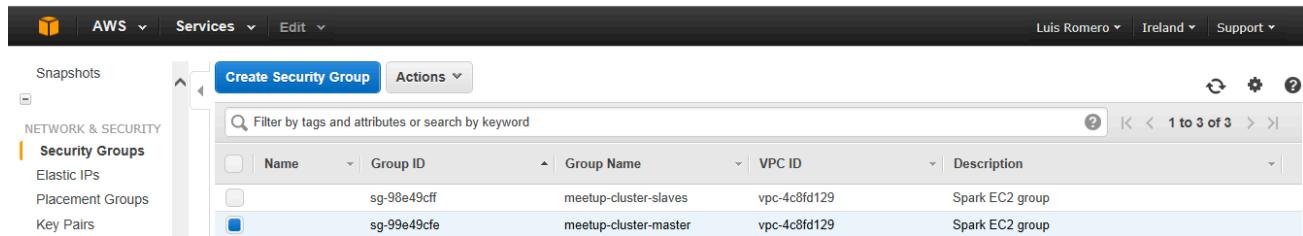
```
Login: ./spark-ec2 --key-pair=u-tad.proyecto --identity-file=u-tad.proyecto.pem --region=eu-west-1 login  
meetup-cluster
```

```
Start: ./spark-ec2 -i u-tad.proyecto.pem --region=eu-west-1 start meetup-cluster
```

```
Stop: ./spark-ec2 -i u-tad.proyecto.pem --region=eu-west-1 stop meetup-cluster
```

```
cloudera@quickstart:~/spark/ec2
File Edit View Search Terminal Help
[cloudera@quickstart ~]$ cd spark
[cloudera@quickstart spark]$ cd ec2
[cloudera@quickstart ec2]$ ls -l
total 80
drwxr-xr-x 3 cloudera cloudera 4096 Dec 21 2015 deploy.generic
drwxr-xr-x 3 cloudera cloudera 4096 Jul 18 12:18 lib
-rw-r--r-- 1 cloudera cloudera 184 Dec 21 2015 README
-rwxr-xr-x 1 cloudera cloudera 996 Dec 21 2015 spark-ec2
-rwxr-xr-x 1 cloudera cloudera 61436 Dec 21 2015 spark_ec2.py
-r----- 1 cloudera cloudera 1692 Jul 18 11:40 u-tad.proyecto.pem
[cloudera@quickstart ec2]$ export AWS_SECRET_ACCESS_KEY=4JuEGA1AMxvpiBPK6gkaLkef
BAeSDISgKMPyqljb
[cloudera@quickstart ec2]$ export AWS_ACCESS_KEY_ID=AKIAICV56PQDSGOKLKTQ
[cloudera@quickstart ec2]$ ./spark-ec2 -i u-tad.proyecto.pem --region=eu-west-1
start meetup-cluster
Searching for existing cluster meetup-cluster in region eu-west-1...
Found 1 master, 2 slaves.
Starting slaves...
Starting master...
Waiting for cluster to enter 'ssh-ready' state.....■
```

Configuración política de seguridad:

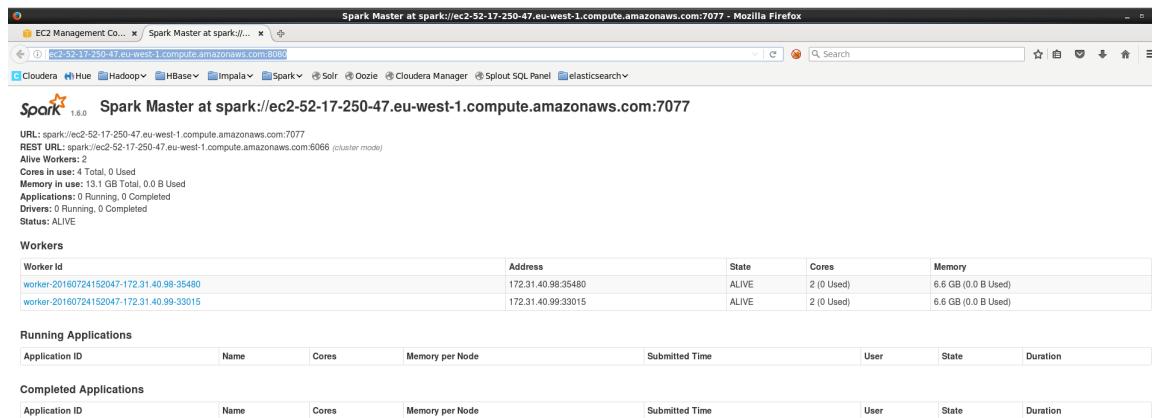


Name	Group ID	Group Name	VPC ID	Description
sg-98e49cff		meetup-cluster-slaves	vpc-4c8fd129	Spark EC2 group
sg-99e49fe		meetup-cluster-master	vpc-4c8fd129	Spark EC2 group

Añadimos los puertos necesarios para que puedan comunicarse cada uno de los componentes.

Una vez definido el cluster de Spark, se accede a cada máquina para añadir los distintos componentes y configurar la conexión entre ellos.

Cluster de Spark:



Spark Master at spark://ec2-52-17-250-47.eu-west-1.compute.amazonaws.com:7077 - Mozilla Firefox

URL: spark://ec2-52-17-250-47.eu-west-1.compute.amazonaws.com:7077
 REST URL: spark://ec2-52-17-250-47.eu-west-1.compute.amazonaws.com:6066 (cluster mode)

Alive Workers: 2

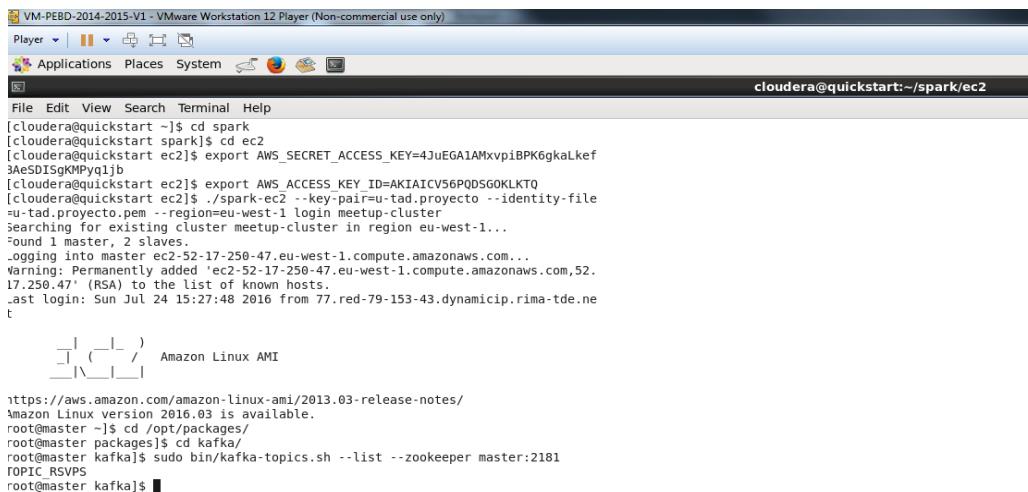
Cores in use: 4 Total: 0 Used
 Memory in use: 13.1 GB Total: 0.0 B Used
 Applications: 0 Running: 0 Completed
 Drivers: 0 Running: 0 Completed
 Status: ALIVE

Workers	Address	State	Cores	Memory
worker-20160724152047-172.31.40.98-35480	172.31.40.98.35480	ALIVE	2 (0 Used)	6.6 GB (0.0 B Used)
worker-20160724152047-172.31.40.99-33015	172.31.40.99.33015	ALIVE	2 (0 Used)	6.6 GB (0.0 B Used)

Running Applications	Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------------	----------------	------	-------	-----------------	----------------	------	-------	----------

Completed Applications	Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
------------------------	----------------	------	-------	-----------------	----------------	------	-------	----------

Kafka

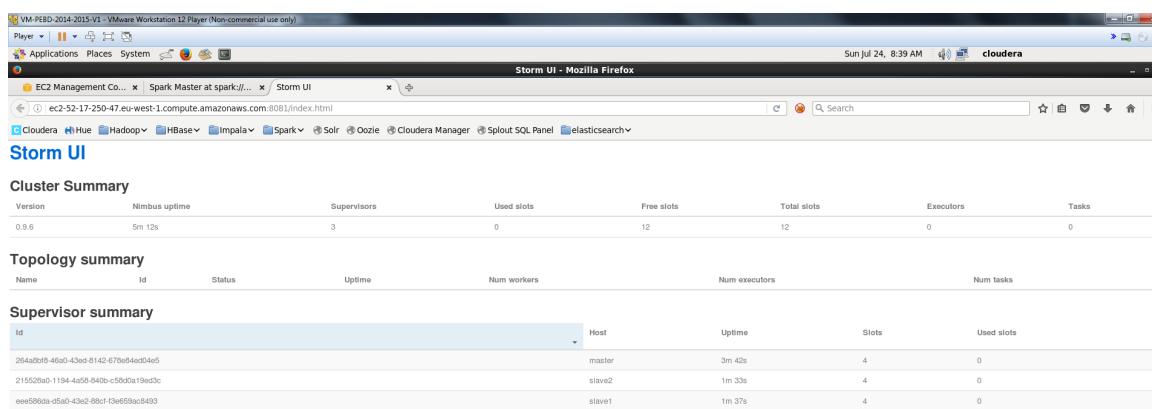


```

VM-PEBD-2014-2015-V1 - VMware Workstation 12 Player (Non-commercial use only)
Player | II | 
Applications Places System cloudera@quickstart:~/spark/ec2
File Edit View Search Terminal Help
[cloudera@quickstart ~]$ cd spark
[cloudera@quickstart spark]$ cd ec2
[cloudera@quickstart ec2]$ export AWS_SECRET_ACCESS_KEY=4JuEGA1AMxvpiBPK6gkaLkef
3aeSDISgkMPyq1jb
[cloudera@quickstart ec2]$ export AWS_ACCESS_KEY_ID=AKIAICV56PQDSGOKLKTQ
[cloudera@quickstart ec2]$ ./spark-ec2 --key-pair=u-tad.proyecto --identity-file
=u-tad.proyecto.pem --region=eu-west-1 login meetup-cluster
Searching for existing cluster meetup-cluster in region eu-west-1...
Found 1 master, 2 slaves.
Logging into master ec2-52-17-250-47.eu-west-1.compute.amazonaws.com...
Warning: Permanently added 'ec2-52-17-250-47.eu-west-1.compute.amazonaws.com,52
17.250.47' (RSA) to the list of known hosts.
Last login: Sun Jul 24 15:27:48 2016 from 77.red-79-153-43.dynamicip.rima-tde.ne
t
[cloudera@quickstart ~]$ ls
Amazon Linux AMI
[cloudera@quickstart ~]$ curl https://aws.amazon.com/amazon-linux-ami/2013.03-release-notes/
Amazon Linux version 2016.03 is available.
[cloudera@quickstart ~]$ cd /opt/packages/
[cloudera@quickstart packages]$ cd kafka/
[cloudera@quickstart kafka]$ sudo bin/kafka-topics.sh --list --zookeeper master:2181
TOPIC_RSVPs
[cloudera@quickstart kafka]$ 

```

Storm



Cluster Summary

Version	Nimbus uptime	Supervisors	Used slots	Free slots	Total slots	Executors	Tasks
0.9.6	5m 12s	3	0	12	12	0	0

Topology summary

Name	Id	Status	Uptime	Num workers	Num executors	Num tasks
------	----	--------	--------	-------------	---------------	-----------

Supervisor summary

Id	Host	Uptime	Slots	Used slots
264a0fb0-46a0-43ed-8142-67be84ed04e5	master	3m 42s	4	0
215520b0-1194-4a5b-84db-c58d0a11edc0	slave2	1m 33s	4	0
eef9990e-0fa0-43c0-88f1-12e6d0aa8493	slave1	1m 37s	4	0

ElasticSearch

master:

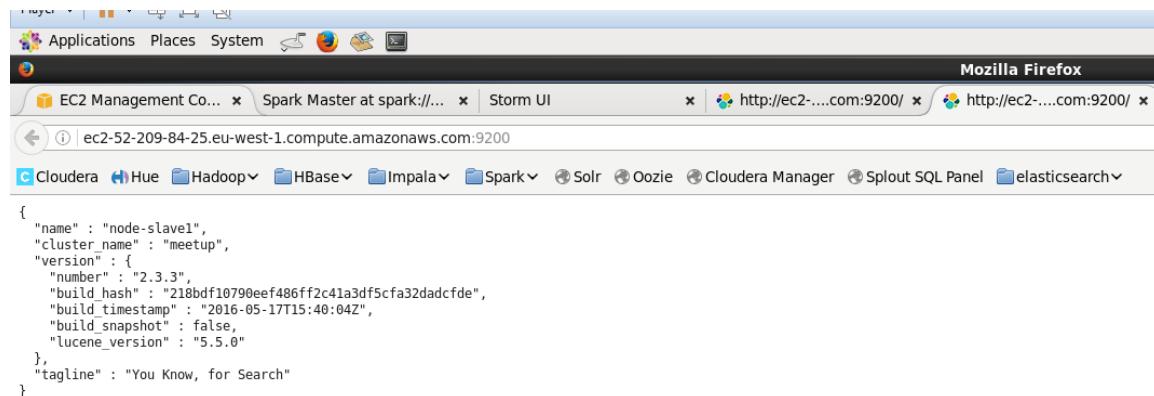


```

{
  "name" : "node-master",
  "cluster_name" : "meetup",
  "version" : {
    "number" : "2.3.3",
    "build_hash" : "218bdf107900ef486ff2c41a3df5cfa32dadcfde",
    "build_timestamp" : "2016-05-17T15:40:04Z",
    "build_snapshot" : false,
    "lucene_version" : "5.5.0"
  },
  "tagline" : "You Know, for Search"
}

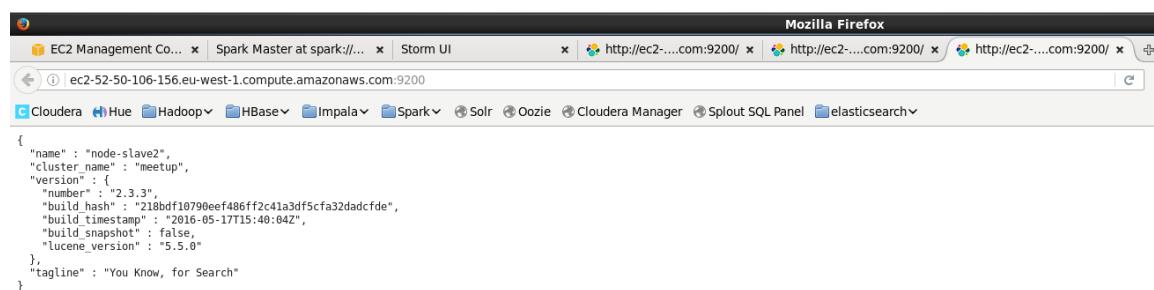
```

slave1



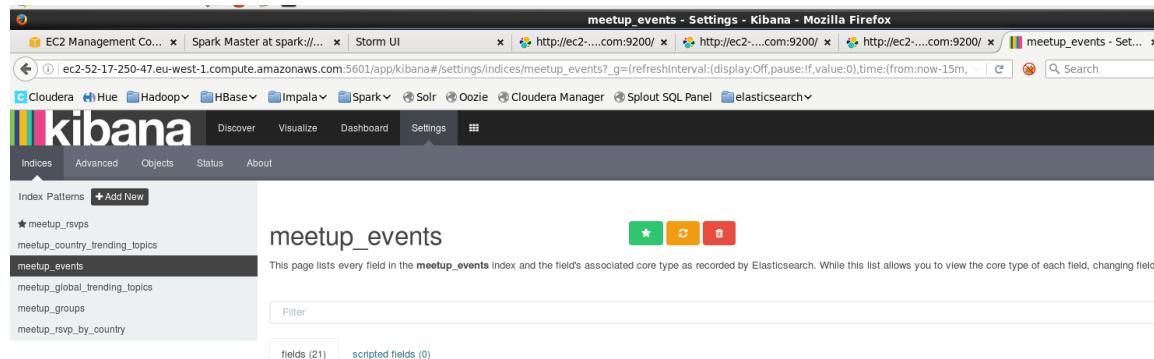
```
{
  "name" : "node-slave1",
  "cluster_name" : "meetup",
  "version" : {
    "number" : "2.3.3",
    "build_hash" : "218bdf10790eef486ff2c41a3df5cfa32dadcde",
    "build_timestamp" : "2016-05-17T15:40:04Z",
    "build_snapshot" : false,
    "lucene_version" : "5.5.0"
  },
  "tagline" : "You Know, for Search"
}
```

slave2



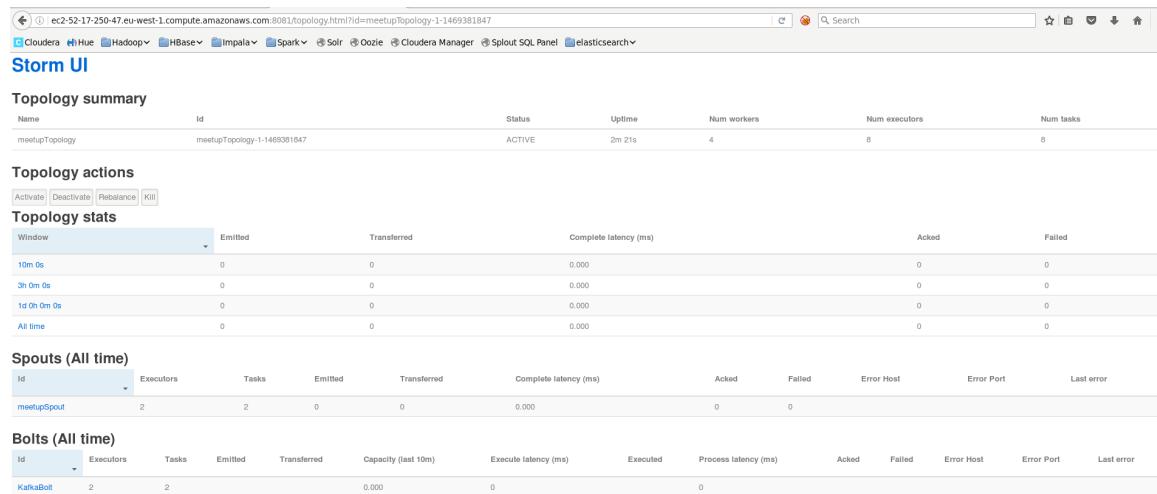
```
{
  "name" : "node-slave2",
  "cluster_name" : "meetup",
  "version" : {
    "number" : "2.3.3",
    "build_hash" : "218bdf10790eef486ff2c41a3df5cfa32dadcde",
    "build_timestamp" : "2016-05-17T15:40:04Z",
    "build_snapshot" : false,
    "lucene_version" : "5.5.0"
  },
  "tagline" : "You Know, for Search"
}
```

Kibana:



The screenshot shows the Kibana Settings page for the `meetup_events` index pattern. The left sidebar lists other index patterns: `meetup_rsvps`, `meetup_country_trending_topics`, `meetup_events` (which is selected), `meetup_global_trending_topics`, `meetup_groups`, and `meetup_rsvp_by_country`. The main panel displays the `meetup_events` index details. It shows 21 fields and 0 scripted fields. The fields listed include `_id`, `_index`, `_score`, `_type`, `city`, `country`, `description`, `group_id`, `group_name`, `host_id`, `host_name`, `lat`, `lon`, `name`, `start_time`, `tags`, `time`, `updated_time`, `venue`, `venue_id`, and `venue_name`.

Enviamos la topología al cluster de Storm



The screenshot shows the Apache Storm UI interface. At the top, there's a navigation bar with links to Cloudera, Hue, Hadoop, HBase, Impala, Spark, Solr, Oozie, Cloudera Manager, Splout SQL Panel, and Elasticsearch. Below the navigation bar, the main title is "Storm UI".

Topology summary:

Name	Id	Status	Uptime	Num workers	Num executors	Num tasks
meetupTopology	meetupTopology-1-1469381847	ACTIVE	2m 21s	4	8	8

Topology actions:

Buttons: Activate, Deactivate, Rebalance, Kill.

Topology stats:

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m Os	0	0	0.000	0	0
3h 0m 0s	0	0	0.000	0	0
1d 0h 0m 0s	0	0	0.000	0	0
All time	0	0	0.000	0	0

Spouts (All time):

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error
meetupSpout	2	2	0	0	0.000	0	0			

Bolts (All time):

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error
KafkaBolt	2	2	0	0	0.000	0	0	0	0	0			

4. Dashboard

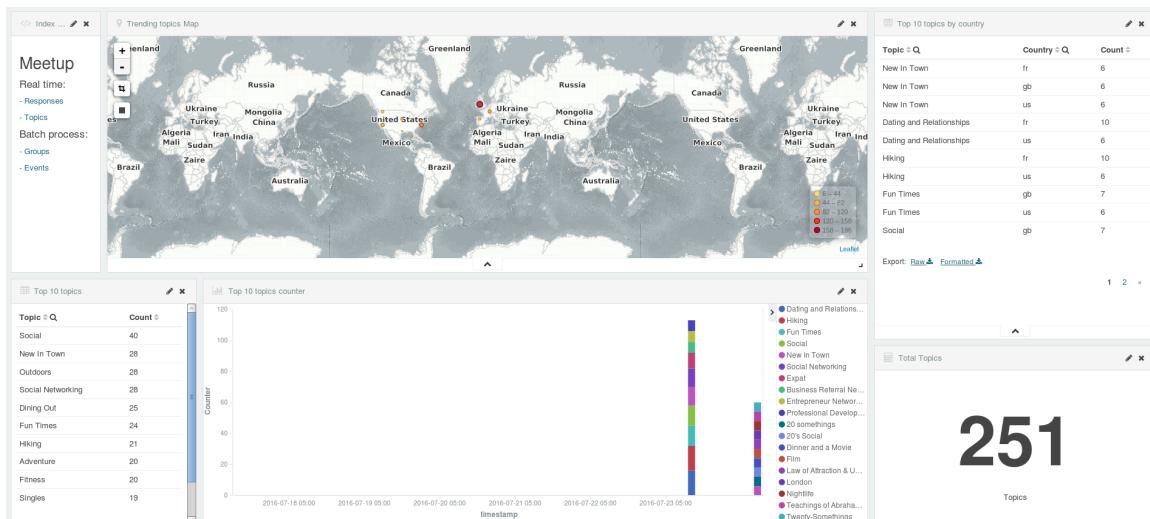


4.1 Real time:

4.1.1 Respuestas Dashboard:

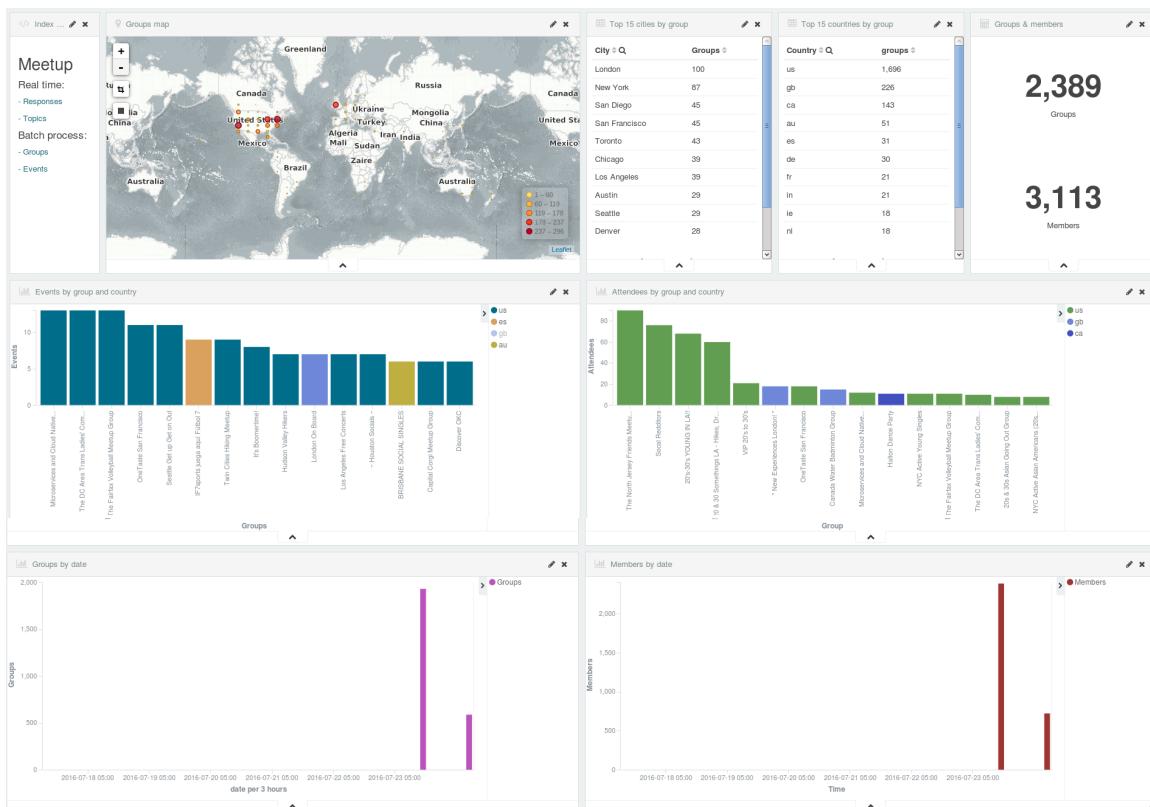


4.1.2 Trending topic Dashboard

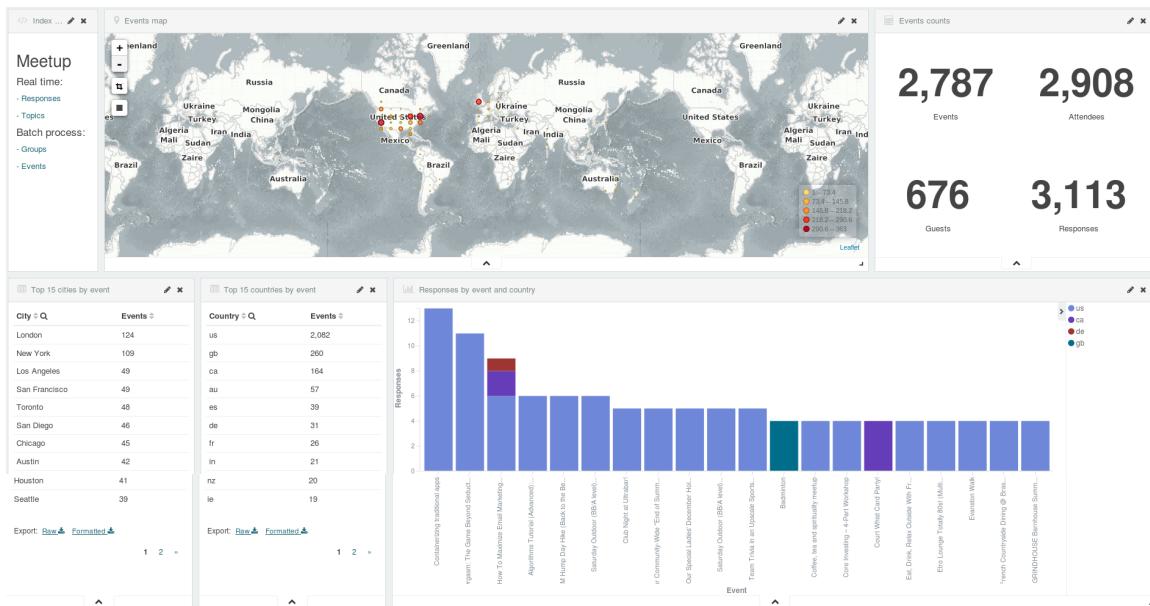


4.2 Batch:

4.2.1 Grupos Dashboard



4.2.2 Eventos Dashboard



5. Conclusiones



Aunque los resultados del análisis de los datos que he hecho creo no van a aportar ninguna información relevante al que los pueda revisar, sí que ha cumplido las expectativas, por lo menos por mi parte, cuando comencé el proyecto.

Durante el Master se han visto multitud de herramientas, y se ha profundizado no sólo a nivel teórico sino práctico. Pero creo que faltaba la parte de tener que decidir en función de un problema cuáles de ellas utilizar, y cómo unirlas en una arquitectura para que trabajen de forma conjunta. También la dificultad de tener que crear un cluster distribuido y desplegar en él las distintas patas de la arquitectura.

En este punto tengo que agradecer a mi tutor Iván, que me ha ido guiando con las dudas que me surgían a cada paso que intentaba dar.

6. Puntos pendientes para futuras implementaciones



6.1 Almacenar los datos en bruto fuera de ElasticSearch.

Creo que es no es necesario que los datos en bruto estén indexados en Elasticsearch. Sería una mejor opción dejarlos en cassandra o S3, por ejemplo, para poder acceder a ellos con el proceso batch, ya que el tiempo de respuesta no es tan importante como en el real time.

6.2 Añadir nuevas fuentes.

La arquitectura está preparada para poder añadir nuevas fuentes de datos sin que tengamos que reacerla por completo. Sería interesante tratar otras fuentes como puede ser el stream de **Meetup** de comentarios o eventos, o fuentes como twitter o Facebook.

6.3 Añadir un nuevo componente REDIS.

Me ha faltado alimentar la información procesada con los distintos servicios que proporciona el API de Meetup. Por ejemplo con la información completa de los grupos, las categorías,...

Una buena forma sería tener una BdD como REDIS (clave-valor) donde accederíamos con cada consulta para añadir la información que en el streaming no tenemos.

6.4 Mejorar los Dashboard

Se han definido algunas consultas, pero por falta de tiempo no todas sacan la información de la manera más clara. Además, cuando hemos conseguido tener un desarrollo más o menos claro, ya no había tiempo para generar un backend de datos lo suficientemente grande para mostrar buenas gráficas.

6.5 Profundizar en el tratamiento Batch de la información

Inicialmente el proyecto se orientó al tratamiento en streaming de las respuestas de **Meetup**. Una vez definida esa parte y funcionando se ha ido ampliando con el tratamiento en Batch. Se puede sacar mucha más información de la que actualmente dan los tratamientos realizados.

6.6 Añadir SparkML

Una de las partes que más se ha profundizado en el Master, siempre teniendo en cuenta que es de Big Data y no DataScience, es el módulo SparkML. Se han aprendido algoritmos que se podrían aplicar a la información que tenemos y sacar algún resultado.

6.7 Generación de alertas

Uno de los puntos que estuve viendo era la posibilidad de generar alertas. Revisando la documentación de elastic vi que existía el plugin Elastic Watcher que puede servir. Lo descargué pero no llegué a configurar alertas.

7 Bibliografía



- ❖ Storm: <http://storm.apache.org/releases/1.0.1/index.html>
- ❖ Kafka Producer API (java): <http://kafka.apache.org/documentation.html>
- ❖ Spark Streaming & Spark SQL
- ❖ <https://spark.apache.org/docs/latest/streaming-programming-guide.html>
- ❖ <https://spark.apache.org/docs/latest/sql-programming-guide.html>
- ❖ ElasticSearch: <https://www.elastic.co/guide/index.html>
- ❖ Kibana: <https://www.elastic.co/guide/en/kibana/current/index.html>
- ❖ AWS: <https://aws.amazon.com/es/documentation/>