ITBA - Proyecto Final

Informe - Visualización de noticias

Agustina Fainguersch | Juan José Marinelli | Lucas Moscovicz

Tutor: Lic. Ariel Aizemberg

1. Índice

2. Abstract

El objetivo de este trabajo es el análisis e implementación de tecnologías tanto de frontend como de backend para el desarrollo de visualizaciones interactivas de noticias.

En el transcurso del trabajo se estudiaron diversos frameworks tanto para realizar las visualizaciones como para optimizar las consultas a la base de datos donde se encuentran almacenadas las noticias, en el transcurso de este informe se detallarán las mejoras realizadas al sistema actual utilizando los frameworks mencionados.

Por otro lado, se realizaron nuevas visualizaciones de las noticias para expandir el análisis hecho hasta el momento sobre los datos en cuestión.

Finalmente, sobre las visualizaciones ya hechas en el sistema actual, se realizaron mejoras en el backend para optimizar su performance.

1.1 Consigna formal del proyecto

La visualización de grandes volúmenes de datos es un desafío a la hora de brindar la información de manera útil para los usuarios.

Para este proyecto se cuenta con una base de datos de noticias (title + summary) de más de 70 medios de la Argentina, generando 5000 registros por día. Hay muchas posibilidades de mostrar estas noticias de una manera no tradicional, mediante el uso de visualizaciones interactivas que nos permitan optimizar el tiempo de lectura.

El objetivo de este proyecto es trabajar sobre las diversas modalidades para explorar esta base de datos y visualizarla de la mejor manera posible: los temas del día, los temas según los medios de comunicación, las noticias que tienen más impacto en las redes sociales, descubrimiento de tendencias y ciclos, medición de la cantidad de menciones a un candidato o una marca, entre muchas más opciones.

Cada uno de estas visualizaciones deberá tener una representación interactiva en la web, utilizando librerías de visualización de datos (d3js, p5js, etc.)

3. Propuesta funcional del proyecto

Previa realización del proyecto, se realizó una propuesta funcional formal para establecer el alcance del proyecto y las implementaciones que se realizarían en el mismo. A continuación se presenta la propuesta funcional:

Se propone la implementación del sistema en 2 partes, por un lado la mejora del tiempo de respuesta del servidor de provisión de datos, implementado hoy en día en **PHP** junto con una base de datos **Postgres.** Para ello se agregarán los siguientes módulos al proyecto:

Una instancia de **ElasticSearch**, un indexador de bases de datos que permite optimizar la búsqueda de texto mejorando así la calidad de las visualizaciones ya existentes en la aplicación del tutor.

Una API hecha en **Java**, paralela a la API existente en PHP para realizar algunas de las consultas extra para obtener información para las nuevas visualizaciones a realizar.

Por otro lado, se desarrollarán nuevas visualizaciones interactivas basadas en las noticias de la base de datos existente usando tanto el backend existente como el que se desarrollará durante el transcurso del proyecto.

Una vez implementadas las optimizaciones sobre el servidor, se realizará un trabajo para demostrar y documentar los beneficios del trabajo realizado. Para eso se realizará lo siguiente:

- *Recolección de datos: Tiempo medio de respuesta del servidor para obtener:*
  - *Palabras para generar el word cloud con diferentes fechas y medios. Con y sin **ElasticSearch**.*
  - *Tiempo de respuesta de la API desarrollada en Java para realizar las nuevas visualizaciones.*

- *Documentación del proyecto: Descripción detallada de las implementaciones realizadas para facilitar su reproducción una vez terminado el proyecto.*
  - *API Desarrollada en Java*
  - *Nuevas Visualizaciones*
  - *Instancia de ElasticSearch con las mejoras en la API desarrollada*

4. Implementación

4.1 Backend

4.1.1 API existente en PHP (API 1.0) y base de datos Postgres

Al inicio de este proyecto, se contaba ya en la universidad tanto con una base de datos donde se encontraban almacenadas las noticias con las que se trabajó como una API de datos escrita en el lenguaje PHP que realizaba la conexión con la misma y la conversión de los datos leídos a un formato adecuado para su visualización usando los frameworks de frontend correspondientes (d3.js, dc.js, etc).

De dicha API se tomaron métricas temporales para evaluar su performance y, tal como fue detallado en la propuesta funcional anteriormente, se decidió como parte del proyecto a realizar, mejorar esa performance para que las visualizaciones del sistema pudieran funcionar de manera más eficiente logrando un mayor nivel de interactividad.

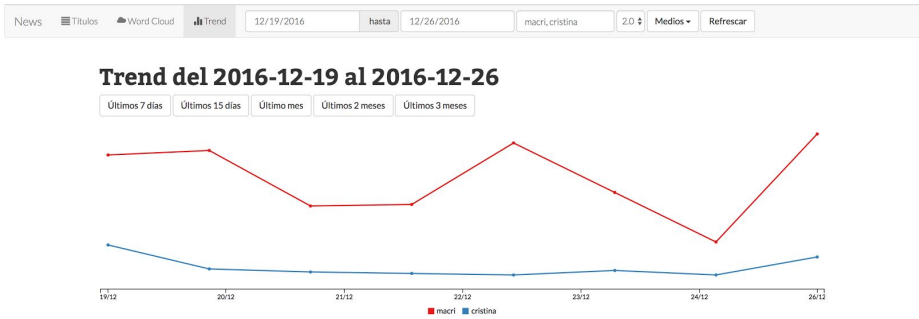
El problema fue dividido en dos partes diferentes, explicados cada uno en una de las secciones subsiguientes. En primer lugar se evaluó el uso de un nuevo lenguaje de mayor eficiencia para mejorar los tiempos de respuesta y en segundo lugar se agregó una capa intermedia en el acceso a datos para lograr una mayor eficiencia.

Las consultas implementadas en la nueva API (existentes en el sistema previo) fueron tres:

- **Titles:** Dado un rango de fechas y un conjunto de palabras clave retorna los principales títulos de dichas fechas que contienen dichas palabras clave.

- **Trend:** Dado un rango de fechas y un conjunto de palabras clave retorna la cantidad de apariciones diarias de dichas palabras clave en los medios en

función de la palabra clave más mencionada en un día, considerando dicha cantidad de apariciones como un 100% y retornando el resto de las apariciones en función de dicho valor.



- **WordCloud:** Dado un rango de fechas y un conjunto de palabras clave retorna un conjunto de palabras junto con su cantidad de apariciones en los medios en esas fechas, excluyendo un conjunto de palabras denominadas “stopwords” de la respuesta debido a su irrelevancia (por ejemplo: palabras como “a”, “de”, “en”, “después”, etc).

4.1.2 API nueva en Java (API 2.0)

La API nueva fue desarrollada utilizando el lenguaje Java 1.8, para su implementación fue necesario el uso de los siguientes frameworks:

- Spring *[https://spring.io/]*
- Guava *[https://github.com/google/guava]*
- StringTemplate *[http://www.stringtemplate.org/]*

Para manejar las dependencias de la misma se uso el gestor de dependencias **Maven** *[https://maven.apache.org/]*.

Para la implementación de la API se realizó un modelo descriptivo en Java de la base de datos junto con 3 capas principales de aplicación que se encargan cada una de una funcionalidad diferente:

Conector con ElasticSearch: Es el encargado de realizar la conexión al servicio de ElasticSearch corriendo en el puerto 9200 del servidor donde se encuentra funcionando la aplicación.

Parser y transformador de la información: Se encarga de interpretar la respuesta de ElasticSearch y transformarla para poder devolver al Frontend información propiamente estructurada para su fácil visualización con el uso de los frameworks mencionados.

Controlador principal: Es el encargado de recibir las requests externas, llamar a las otras capas y coordinar la ejecución principal de la API.

Para la realización del modelo se implementó una clase en Java para representar el formato de respuesta de cada tipo de visualización implementada.

4.1.2.1 Documentación de los endpoints de la API 2.0

Titles

Devuelve la información básica de las noticias en cada uno de los días del rango de fechas entre date1 y date2. Esta información comprende el título, resumen, link a la noticia original, e información del medio que la publicó.

URL: http://infovis2.it.itba.edu.ar/news-visualization/api/titles
Método: GET
Parámetros requeridos:
key=[string]
Parámetros opcionales:
date1=[date]
date2=[date]
keyword=[List<String>]
medios=[List<Integer>]
limit=[integer]
offset=[integer]
Consulta ejemplo:
http://infovis2.it.itba.edu.ar/news-visualization/api/titles?key=21232f297a57a5a743894a0e4a801fc3&date1=2016-05-01&date2=2016-05-08&keyword=macri,cristina

Trend

Devuelve la cantidad de apariciones de cada una de las keywords pasadas como parámetro en cada una de las fechas comprendidas en el rango entre los parámetros date1 y date2. Los valores devueltos por este endpoint están comprendidos en un intervalo entre 0 y 100 siendo 100 la escala correspondiente a la mayor cantidad de apariciones de cualquiera de los keywords en un día.

URL: **http://infovis2.it.itba.edu.ar/news-visualization/api/trend**
Método: GET
Parámetros requeridos:
key=[string]
keyword=[List<String>]
Parámetros opcionales:
date1=[date]
date2=[date]
medios=[List<Integer>]
Consulta ejemplo:
http://infovis2.it.itba.edu.ar/news-visualization/api/trend?key=21232f297a57a5a743894a0e4a801fc3&date1=2016-05-01&date2=2016-05-08&keyword=macri,cristina

Word-Cloud

Devuelve la información necesaria para generar un word cloud usando el contenido de las noticias comprendidas en el rango de fechas entre date1 y date2. Esta información consiste en cada una de las palabras que aparecen en dichas noticias junto con la cantidad de veces que la misma aparece en el conjunto de noticias ordenadas de mayor a menor.

URL: **http://infovis2.it.itba.edu.ar/news-visualization/api/word-cloud**
Método: GET
Parámetros requeridos:
key=[string]
Parámetros opcionales:
date1=[date]
date2=[date]
keyword=[List<String>]
medios=[List<Integer>]
limit=[integer]
offset=[integer]
minfreq=[integer]
Consulta ejemplo:
http://infovis2.it.itba.edu.ar/news-visualization/api/word-cloud?key=21232f297a57a5a743894a0e4a801fc3

4.1.2.2 Documentación de los parámetros de la API 2.0

A continuación se enumeran los parámetros disponibles en la API 2.0 con una breve descripción de su uso:

Parámetros de acotación de resultado

- **limit** [Integer]: Cantidad máxima de resultados a ser devueltos.
- **offset** [Integer]: Cantidad de registros a ser salteados en la respuesta una vez generada la misma.
- **minfreq** [Integer]: Frecuencia mínima de apariciones que un elemento de la respuesta debe tener para ser devuelto.

Parámetros de búsqueda

- **date1** [Date(Y-m-d)]: Cota inferior para la búsqueda en la base de datos (de no pasarse se asume la fecha de hoy)
- **date2** [Date(Y-m-d)]:Cota superior para la búsqueda en la base de datos (de no pasarse se asume igual a **date1**)
- **keyword** [List<String>]: Lista de palabras clave a usar como filtro en la búsqueda dependiendo de la consulta realizada
- **medios** [List<Integer>]: Lista de ids de los medios a ser tenidos en cuenta para la búsqueda (si se quiere usar todos los medios simplemente se pasa el parámetro vacío)
- **key** [String]: Clave a ser usada para acceso a la base de datos, la misma puede asimismo tener asociadas keywords o medios por defecto a la hora de realizar diversas búsquedas

4.1.3 ElasticSearch

Sobre la base de datos existente se agregó una nueva capa para lograr un más eficiente acceso a los datos usando un servicio de ElasticSearch. ElasticSearch es una herramienta de indexamiento y búsqueda open source, accesible a través de una API REST.

Previo a levantar el servicio de ElasticSearch para ponerlo a correr sobre la base de datos, se deben generar los índices necesarios para que las consultas al mismo pueden ejecutar más rápidamente usando todas las ventajas que provee. Para ello se utilizó un **JDBC Importer** *(https://github.com/jprante/elasticsearch-jdbc)*, un pequeño aplicativo en Java que permite hacer consultas predefinidas a una base de datos indexando sus resultados en ElasticSearch. La consulta utilizada para generar los índices fue muy simple dado que la

tabla a indexar (entries) contenía toda la información necesaria para la rápida realización de las consultas. La consulta a indexar fue:

```
SELECT e.id, e.tmp, e.media, e.source, e.fecha, e.title, e.summary, c.medio_id, m.nombre,
m.url_favicon FROM entries AS e, categoria AS c, medios AS m WHERE e.source = c.source AND
c.medio_id = m.id
```

Una vez generados los índices se levantó el servicio de ElasticSearch el cual abre el puerto 9200 de localhost para escuchar y contestar requests mediante su API REST. Las requests se realizan en formato JSON y los parámetros a usar se encuentran definidos en la documentación de Elastic según el tipo de consulta que se quiera realizar (*https://www.elastic.co/guide/en/elasticsearch/reference/current/_introducing_the_query_language.html*).

Una vez corriendo este servicio y la API realizada, se pueden realizar las consultas para obtener la información estructurada necesaria para las visualizaciones existentes en el sistema, con una performance superior a la obtenida con la API existente previamente.

Un instructivo detallado de cómo poner en funcionamiento tanto la API en Java como la instancia de ElasticSearch se encuentra en el anexo de este informe.

4.2 Frontend - Nueva Visualización Agregada

Además de las mejoras a las visualizaciones existentes, se implementó una nueva visualización en el sistema actual. La misma se basa en un subconjunto de la información disponible seleccionado por Google News.

La idea principal de esta nueva visualización es observar la polarización de los diferentes medios respecto del tipo de noticias que deciden publicar. Para ello se genera, en base a algún medio en particular, un gráfico de radar usando la librería *chart.js* en el cual cada arista del radar es una categoría de noticias (Deportes, Economía, Política, etc) y en el centro se dibuja un polígono el cual se extiende más hacia una arista u otra según cuantas noticias de ese tipo haya emitido Google News de ese medio en dicha categoría.

Es posible acceder a la nueva visualización a través de la siguiente URL:

http://infovis.it.itba.edu.ar/news/radars.php?key=21232f297a57a5a743894a0e4a801fc3

A continuación se muestran ejemplos de la visualización realizada:


Ole - Olé
Nacional
Internacional
Salud
Ciencia y Tecnología
Espectáculos
Deportes
Noticias destacadas
Economía


Página 12 (Registro)
Nacional
Internacional
Salud
Ciencia y Tecnología
Espectáculos
Deportes
Noticias destacadas
Economía


LA NACION (Argentina)
Nacional
Internacional
Salud
Ciencia y Tecnología
Espectáculos
Deportes
Noticias destacadas
Economía


Infobae.com
Nacional
Internacional
Salud
Ciencia y Tecnología
Espectáculos
Deportes
Noticias destacadas
Economía


canchallena.com
Nacional
Internacional
Salud
Ciencia y Tecnología
Espectáculos
Deportes
Noticias destacadas
Economía


Olé
Nacional
Internacional
Salud
Ciencia y Tecnología
Espectáculos
Deportes
Noticias destacadas
Economía

5. Resultados

5.1 Performance del nuevo Backend

Para medir la performance del nuevo Backend se hicieron múltiples requests a ambas APIs promediando sus tiempos de respuesta.

Las mediciones fueron realizadas en el rango de 1 semana, 1 mes y 1 año para comparar la escalabilidad de las mismas.

5.1.1 Comparaciones de tiempos para ambas APIs

Para realizar la comparación de tiempos se hizo un pequeño script en bash que, utilizando una lista de palabras hace un request via *curl* a ambas APIs utilizando cada una de las palabras como término para rangos de fechas diferentes de 1 semana, 1 mes y 1 año. Luego de correr el script y obtener el tiempo para cada una de estas consultas se obtuvo la media y desvío estándar de los tiempos obtenidos para realizar las métricas comparativas.

Lista de palabras usada:

["macri", "cristina", "river", "boca", "tornado", "tinelli", "mirtha", "rusia", "barcelona", "messi", "marley", "salud", "ministerio", "china", "trump", "clinton", "tenis", "cine", "ladrillo", "droga", "inseguridad"]

Consulta de Titles:

URL Usada para tests de la API de PHP:

`http://infovis.it.itba.edu.ar/news/titulos_data.php?key=2b9c1508a73a9e713a05420d9887cf13&date1=<date>&date2=<date>&keyword=<keyword_seleccionada>`

URL Usada para tests de la API de Java:

`http://infovis2.it.itba.edu.ar/news-visualization/api/titles?key=2b9c1508a73a9e713a05420d9887cf13&date1=<date>&date2=<date>&keyword=<keyword_seleccionada>`

	API 1.0 (PHP)	API 2.0 (Java + ElasticSearch)
1 semana	$\bar{x}(media)$: 5.6161 $\sigma^2(desvio\ estandar)$: 0.369	$\bar{x}(media)$: 0.0748 $\sigma^2(desvio\ estandar)$: 0.023
1 mes	$\bar{x}(media)$: 5.5714 $\sigma^2(desvio\ estandar)$: 0.108	$\bar{x}(media)$: 0.0599 $\sigma^2(desvio\ estandar)$: 0.017
1 año	$\bar{x}(media)$: 5.5736 $\sigma^2(desvio\ estandar)$: 0.132	$\bar{x}(media)$: 0.0909 $\sigma^2(desvio\ estandar)$: 0.036

Consulta de Trend:

URL Usada para tests de la API de PHP:
`http://infovis.it.itba.edu.ar/news/trend_data.php?key=2b9c1508a73a9e713a05420d9887cf13&date1=<date>&date2=<date>&keyword=<keyword_seleccionada>`

URL Usada para tests de la API de Java:
`http://infovis2.it.itba.edu.ar/news-visualization/api/trend?key=2b9c1508a73a9e713a05420d9887cf13&date1=<date>&date2=<date>&keyword=<keyword_seleccionada>`

	API 1.0 (PHP)	API 2.0 (Java + ElasticSearch)
1 semana	$\bar{x}(media)$: 3.4540 $\sigma^2(desvio\ estandar)$: 0.09	$\bar{x}(media)$: 0.1510 $\sigma^2(desvio\ estandar)$: 0.091
1 mes	$\bar{x}(media)$: 4.3573 $\sigma^2(desvio\ estandar)$: 0.024	$\bar{x}(media)$: 0.4723 $\sigma^2(desvio\ estandar)$: 0.323
1 año	$\bar{x}(media)$: 17.9340 $\sigma^2(desvio\ estandar)$: 0.128	$\bar{x}(media)$: 5.2995 $\sigma^2(desvio\ estandar)$: 3.649

Consulta de WordCloud:

URL Usada para tests de la API de PHP:
`http://infovis.it.itba.edu.ar/news/wc_data.php?key=2b9c1508a73a9e713a05420d9887cf13&date1=<date>&date2=<date>&keyword=<keyword_seleccionada>`

URL Usada para tests de la API de Java:
`http://infovis2.it.itba.edu.ar/news-visualization/api/word-cloud?key=2b9c1508a73a9e713a05420d9887cf13&date1=<date>&date2=<date>&keyword=<keyword_seleccionada>`

	API 1.0 (PHP)	API 2.0 (Java + ElasticSearch)
1 semana	$\bar{x}(media)$: 5.5526 $\sigma^2(desvio\ estandar)$: 0.136	$\bar{x}(media)$: 0.0841 $\sigma^2(desvio\ estandar)$: 0.019
1 mes	$\bar{x}(media)$: 5.47 $\sigma^2(desvio\ estandar)$: 0.079	$\bar{x}(media)$: 0.0839 $\sigma^2(desvio\ estandar)$: 0.026
1 año	$\bar{x}(media)$: 5.5169 $\sigma^2(desvio\ estandar)$: 0.212	$\bar{x}(media)$: 0.1310 $\sigma^2(desvio\ estandar)$: 0.038

6. Conclusiones

6.1 Performance de la API 2.0 + ElasticSearch

En la mayoría de los casos la performance de las consultas incrementó de manera significativa gracias al uso de la nueva API junto con ElasticSearch. Esto se debe a que el

índice utilizado para las consultas en cuestión es el levantado por ElasticSearch en memoria y se logra acceder a la información presente en la base de datos de forma mucho más ágil y rápida; sobre todo cuando se trata de búsquedas textuales como en el caso del word cloud.

Por otro lado, en ciertos casos ElasticSearch no provee una mejoría de igual magnitud cuando se presentan cierto tipo de consultas específicas para las cuales ElasticSearch aún no se encuentra optimizado. En particular esto se puede observar en el endpoint de **trend** el cual no responde con la misma velocidad que los otros endpoints realizados debido a que en dicha consulta se requieren agregaciones las cuales dificultan la performance de ElasticSearch. En particular, esto se puede notar al pasar diferentes términos sobre los cuales realizar las agregaciones, por ejemplo, al pasar “Macri” o “Cristina” que son términos los cuales aparecen en la mayoría de las noticias, el tiempo de respuesta es mayor que al pasar un término poco común. Esto se debe a que **ElasticSearch** realiza en primer lugar un filtro bajo dichos términos el cual se hace velozmente y luego realiza las agregaciones que toman una cantidad de tiempo mucho mayor.

Finalmente, se puede observar que la primera vez que se realiza una consulta en particular a la nueva API esta tarda un tiempo mayor que al realizar la misma consulta por segunda vez y de allí en adelante. Este fenómeno se debe a que ElasticSearch guarda sus índices en archivos y, al momento de realizar una consulta, levanta la porción necesaria de los mismos a memoria y los deja allí por un cierto tiempo. Por eso al volver a realizar una consulta similar, en caso de encontrarse los índices para ella aún en memoria, ElasticSearch responde de manera mucho más veloz que la primera vez.

6.2 Comparativa de diversos frameworks de visualización utilizados

Las principales librerías utilizadas para experimentar con visualizaciones fueron por un lado **d3.js** y **dc.js** y por otro **chartjs**.

Por un lado, realizar diversos tipos de gráficos **d3.js** o **dc.js** puede ser mucho más complejo a nivel código que **chartjs** ya que este permite una customización mucho mayor de las visualizaciones realizadas. En cambio, **chartjs** permite realizar visualizaciones mucho más rápidamente con una menor cantidad de código pero el resultado de las mismas es mucho más estándar que las anteriores ya que la cantidad de parámetros que se pueden modificar es menor para este framework.

Una diferencia a nivel técnico entre **d3.js** y **chartjs** es el hecho de que **d3.js** permite alterar el DOM y generar las visualizaciones usando distintos tipos de componentes HTML. Uno de los componentes principales que usa para realizar sus visualizaciones es *svg*, pero no se restringe a ello.

Chartjs, por otro lado dibuja exclusivamente sobre el elemento *canvas* haciéndolo más restrictivo a la hora de su uso; sin embargo, posee varios gráficos pre-configurados

que, de querer usarse para una visualización en particular, pueden resultar útiles ya que son muy fáciles de implementar.

Para este proyecto se terminó usando **chartjs** para la visualización implementada ya que lo que se deseaba mostrar coincidía con una de las visualizaciones pre-configuradas de dicho framework. En otro caso hubiese sido pertinente usar **d3.js** o **dc.js** ya que estos permiten implementar visualizaciones más complejas y customizables.

Para **dc.js** también se experimentó con la librería **crossfilter.js** la cual permite realizar filtros multidimensionales para las visualizaciones. Con el mismo se pueden cambiar dinámicamente diversos rangos de la visualización actualizando solamente dicha información sin necesidad de refrescar la vista en la cual se encuentra contenida. Esta herramienta permite agregar un nivel de interactividad mayor a las visualizaciones realizadas.

7. Anexo

7.1 Instructivo de instalación de la API Java y ElasticSearch

7.1.1 Configuración de ElasticSearch

1. **Descargar ElasticSearch de https://www.elastic.co/**
2. **Antes de ejecutar elastic, podemos ver si el mismo ya se encuentra levantado haciendo un GET a localhost:9200 con el siguiente comando:**

```
curl -XGET “http://localhost:9200/?pretty”
```

En caso de estar ejecutando, la respuesta obtenida será un JSON con el número de versión de ElasticSearch y algunos datos básicos de su configuración general. Por otro lado, si el mismo no se encuentra levantado, simplemente no podremos conectarnos al puerto 9200.

3. **Para correr ElasticSearch, ir a la carpeta donde se encuentra extraído y ejecutar:**

```
ES_JAVA_OPTS="-Xms5g -Xmx5g" ./bin/elasticsearch
```

Si se quiere dejar corriendo el proceso se puede ejecutar el comando:

```
ES_JAVA_OPTS="-Xms5g -Xmx5g" ./bin/elasticsearch &
```

Seguido del comando:

```
disown
```

El comando *disown* es un comando nativo de linux. El mismo sirve para desligar la ejecución del último proceso ejecutado en la consola del proceso que corre dicha consola; por ende, al cerrar la terminal desde la cual se levantó, en este caso, el servicio de ElasticSearch esto no matará dicho proceso y ElasticSearch seguirá corriendo.

4. **Clonar en otra carpeta el repositorio con el script para indexar los datos de la base de datos en ElasticSearch**

```
git clone git@bitbucket.org:xdatapf1/pf-jdbc-importer.git
```

En la carpeta *bin* habrá un script llamado *postgresql-prod.sh* donde se deben configurar las credenciales de la base de datos junto con la consulta que se desee indexar en ElasticSearch. Un ejemplo de configuración para este archivo para este proyecto sería:

```
{
    "type" : "jdbc",
    "jdbc" : {
        "url" : "jdbc:postgresql://localhost:5432/<database>",
        "user" : "<username>",
        "password": "********",
        "sql" : "select
e.id,e.tmp,e.media,e.source,e.fecha,e.title,e.summary,c.medio_id,m.no
mbre,m.url_favicon from entries as e, categoria as c, medios as m
where e.source = c.source and c.medio_id = m.id order by fecha asc",
        "elasticsearch" : {
            "host" : "localhost",
            "port": "9300"
        }
    }
}
```

Dicho script se deberá ejecutar con el comando:

```
./bin/postgresql-prod.sh
```

Este comando tarda unos minutos en correr (dependiendo de la cantidad de tuplas) y genera todos los índices necesarios para que ElasticSearch pueda operar de forma eficiente.

5. **Una vez terminada la migración de ElasticSearch, se puede comprobar la cantidad de tuplas indexadas con el siguiente comando:**

```
curl -XGET “http://localhost:9200/jdbc/jdbc/_count”
```

Dicho comando debería devolver el número de tuplas que retorna la consulta configurada previamente para ser indexada. Si el script corrió exitosamente, entonces todas las tuplas de dicha consulta deberían estar ya indexadas por ElasticSearch. Un ejemplo de respuesta sería:

```
{"count":100,"_shards":{"total":5,"successful":5,"failed":0}}
```

Indicándonos que hay 100 tuplas indexadas de la base de datos.

6. **Configurar la ejecución del cron para indexar nuevas tuplas**

Una vez indexadas todas las tuplas deseadas, solo resta configurar el cron que indexará las nuevas tuplas entrantes cada cierto tiempo. Para ello, dentro del mismo repositorio se encuentra el siguiente script:

```
bin/postgresql-cron.sh
```

El cual contiene la lógica necesaria para indexar las nuevas tuplas a medida que las mismas van entrando a la tabla. Lo único que requiere este archivo, es un archivo llamado *date.txt* en la carpeta principal del usuario donde se guardará la última fecha en la que se corrió el cron para indexar solamente las tuplas ingresadas a la base de datos luego de esa fecha.

Para verificar si el cron está instalado en el servidor podemos ejecutar el siguiente comando:

```
crontab -l
```

Dicho comando nos muestra una lista de todos los crones que se encuentran configurados en el servidor. De haber una línea referenciando a nuestro archivo *postgresql-cron.sh* significa que el mismo ya esta configurado para correr cada cierto tiempo, de lo contrario, podemos ejecutar el siguiente comando:

```
crontab -e
```

Esto nos permitirá editar el archivo con los crones configurados en el servidor. Para agregar el cron simplemente agregamos la siguiente línea al final del archivo:

```
0 * * * * <path al repositorio>/bin/postgresql-cron.sh >> <path al repositorio>/logs/cron.log 2>&1
```

Esta configuración indica que el cron se correrá en el minuto 0 de cada hora y hará un append del output al archivo *cron.log* en caso de que se desee tener algún registro de cada vez que el cron es ejecutado.

7.1.2 Ejecución de la API de Java

1. **Clonar el repositorio donde se encuentra la API de Java**

```
git clone git@bitbucket.org:xdatapf1/news-visualization-api.git
```

2. **Antes de ejecutar la API, podemos ver si la misma ya se encuentra levantada haciendo un GET a localhost:9090 con el siguiente comando**

```
curl -XGET "http://localhost:9090/news-visualization/api/health"
```

En caso de estar ejecutando, la respuesta obtenida será un JSON con información sobre la configuración de runtime de la API. Por otro lado, si el mismo no se encuentra levantado, simplemente no podremos conectarnos al puerto 9090.

3. **En la carpeta donde se clonó el repositorio hay un script para la ejecución de la API con los parámetros predefinidos. Ejecutar el mismo con el siguiente comando**

```
./run.sh
```

Esto dejará corriendo la API de Java escuchando en el puerto 9090 de localhost.

4. **Para probar la API y su correcto funcionamiento se pueden usar los siguientes comandos que realizan las consultas implementadas en la API de Java:**

Titles

```
curl -XGET
"http://infovis2.it.itba.edu.ar/news-visualization/api/titles?key=21232f297a57a5a743894a0e4a801fc3&date1=2016-05-01&date2=2016-05-08&keyword=macri,cristina"
```

Trend

```
curl -XGET
"http://infovis2.it.itba.edu.ar/news-visualization/api/trend?key=21232f297a57a5a743894a0e4a801fc3&date1=2016-01-01&date2=2016-01-30&keyword=macri,cristina"
```

WordCloud

```
curl -XGET
"http://infovis2.it.itba.edu.ar/news-visualization/api/word-cloud?key=21232f297a57a5a743894a0e4a801fc3&date1=2016-05-01&date2=2016-05-08"
```

8. Recursos

1. ElasticSearch *“organize data and make it easily accessible”* <http://elastic.co>
2. D3.js *“Data Driven documents”* <http://d3js.org>
3. DC.js *“Dimensional Charting”* <https://github.com/dc-js/dc.js>
4. Chartjs *“Simple HTML5 Charts using the canvas element”* <http://www.chartjs.org>