Cloud Computing and Big Data

FIB – Barcelona School of Informatics

Barcelona, Spring 2015

Hands-on 4: Indexing, Searching and Visualizing

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Barcelona Supercomputing Center Centro Nacional de Supercomputación

Indexing, searching and visualizing data in the cloud

CLOUD COMPUTING AND BIG DATA

Facultat d'Informàtica de Barcelona (FIB)

UPC Barcelona Tech & Barcelona Supercomputing Center

Version 1.0 - 28 February 2015

Authors: Dani Cea & Jordi Torres

Table of Contents

1 Hands-on content 3
2 Introduction to Elasticsearch 3
3 Installing and configuring Elasticsearch 4
4 Using Elasticsearch 4
5 Retrieving and searching data 9
6 Installing Kibana 13
7 Displaying data with Kibana 14

1 Hands-on content

This hands-on is intended to help the student to get started with two important tools: Elasticsearch, a search server, and the data visualization engine Kibana. The commands used here work on bash and the python code uses the 2.7.9 version.

2 Introduction to Elasticsearch

Elasticsearch is an open source distributed search server based on Apache Lucene (http://lucene.apache.org) . It allows you to start with one machine and scale to hundreds, and supports distributed search deployed over Amazon EC2's cloud hosting (Hands-on 1).

It provides a distributed, multitenant-capable full-text search engine with a RESTful web interface and schema-free JSON documents. Elasticsearch is developed in Java and is released as open source under the terms of the Apache License

Elasticsearch can be used to index and search all kinds of documents, providing scalable, near real-time queries supporting multitenancy. It uses Apache Lucene as the query language, and tries to make all features of it available through the RESTful, JSON-based API.

Notable users of Elasticsearch include Wikimedia, StumbleUpon, MozillaQuora, Foursquare, Etsy, SoundCloud, GitHub, FDA, CERN, Stack Exchange, Center for Open Science and Reverb.

3 Installing and configuring Elasticsearch

First of all, we need to install and run a new instance of Elasticsearch. This part of the tutorial is intended for people outside FIB, for students in the lab, simply copy the following folder into your home directory, an then skip to the next chapter:

```
cp –r /assig/cc-mei/elasticsearch
```

Installing Elasticsearch is as easy as following the following tutorial: http://www.elasticsearch.org/guide/en/elasticsearch/guide/current/_installing_elasticsearch.html

We are going to use 2 Elasticsearch plugins: KOPF (https://github.com/lmenezes/elasticsearch-kopf) and Marvel (http://www.elasticsearch.org/guide/en/marvel/current/). The command for installing them are, within the elasticsearch folder:

```
./bin/plugin -i elasticsearch/marvel/latest
./bin/plugin --install lmenezes/elasticsearch-kopf/master
```

The configuration file can be found in the ./config/elasticsearch.yml file. In order to configure Elasticsearch, the following tutorial might be useful: http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/setup-configuration.html

4 Using Elasticsearch

Using the code from Hands-on 3 exercise 2, we are going to store and index all tweets received from the stream into an existing ElasticSearch instance.

To start running your Elasticsearch instance, simply go to the folder and run:

```
./bin/elasticsearch &
```

Once started, the system is present at the url **http://localhost:9200**. To see the status of the Database, we can use the KOPF plugin, located at url http://localhost:9200/_plugin/KOPF.

Let's start with the basis: How to access ElasticSearch API. Again, we could write our own functions that execute the REST calls needed, but there is an official Python library to do that. To install it, first we are going to create the virtual environment for this specific project:

```
$ virtualenv venv_lab4
```

```
$ source ./venv_lab4/bin/activate
```

Now, let's install the elasticsearch module from the pip repository:

```
$ pip install elasticsearch
```

Your will see the "Successfully installed elasticsearch-1.4.0 urllib3-1.10.2" message.

Let's get into the code. Open the second exercise from the Hands-on 3 , and add the elasticsearch import:

```
from elasticsearch import Elasticsearch
```

The Stream Handler has the following modifications:

```
class StdOutListener(tweepy.StreamListener):

        def on_data(self, data):

                global index

                tweet = json.loads(data)

                if tweet["geo"] is not None:

                        lat = tweet["geo"]["coordinates"][1]

                        lon = tweet["geo"]["coordinates"][0]

                        tweet["geo"]["coordinates"][0] = lat

                        tweet["geo"]["coordinates"][1] = lon

                es.index(index=index, doc_type='tweet', id=tweet["id_str"],
                body=tweet)

        def on_error(self, status):

                print status
```

The **on_error** function remains the same. The **on_data** function, instead of outputting the tweets, now indexes them into elasticsearch using the “elasticsearch.Elasticsearch.index” method, specifying as parameters the index where to store the tweet, the doc type, where we set “tweet” for all of them, the document ID, where we use the tweet ID, and finally the body, where we specify the entire tweet to store.

The tweepy configuration remains the same, but we have to add the following elasticsearch configuration. Add the following to config.py:

```
es_server = 'http://minerva.bsc.es:8089'
es_mapping = {
        "mappings" : {
                "tweet": {
                        "_timestamp": {
                                "enabled": True,
                                "store": True,
                                "index": "analyzed"
                        },
                        "_index": {
                                "enabled": True,
                                "store": True,
                                "index": "analyzed"
                        },
                        "properties": {
                                "geo": {
                                        "properties": {
                                                "coordinates": {
                                                        "type": "geo_point",
                                                        "index": "analyzed"
                                                }
                                        }
                                }
                        }
            }
    }
}
```

Then, back to app.py:

```
es = Elasticsearch([config.es_server])

index = None
```

Here is where we instantiate the elasticsearch module, pasing the server url as parameter (obtained from the config file). The second line creates a global variable that will hold the index name during the streaming process.

The stop function remains exactly the same, but the start function now needs to create the index where tweets are going to be stored:

```
def startStream(track):

        global stream

        global index

        index = track

        print ('Index %s created', index)

        es.indices.create(index=index, body=config.es_mapping)

        stream.filter(track=[track], async=True)
```

The index name is simply the track name. The index is created using the "elasticsearch.Elasticsearch.indices.create" method, where we pass as parameters the index name and the configuration, obtained from the config file.

The code is ready to go! Save the changes into exercise3.py and run the program:

```
$ python exercise3.py
```

Try to use a common word as the search filter (love, friend, Monday…) to obtain a lot of tweets in less time. Remember to write down the index name created, you will need it for the next step!

5 Retrieving and searching data

After a couple of minutes of your program running, you'll have enough data to play around. It's now time to search and retrieve some results using the previous concepts about the elasticsearch API.

To access the Elasticsearch API, you could use any software/tool capable of executing REST queries (GET, POST, PUT and DELETE), like curl. But to make things easier, we have prepared a nice, friendly interface that can be accessed online, using the Sense plugin for Elasticsearch at the url http://localhost:9200/_plugin/marvel/sense/index.html

From here, let's perform some queries in our data. Let's start retrieving all the stored tweets by using the _search endpoint (http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/search-request-body.html):

```
POST tweets-14240977515603235/_search
```

In the response we can see the number of tweets that match our query (marked as "hits.total"), the time that took to perform the query ("took" field), and the information of all tweets within the array "hits". By default only the first 10 results are shown, but we can configure that by simply adding:

```
POST tweets-14240977515603235/_search
{
   "size": 100
}
```

This will retrieve the first 100 tweets that match the query. Great! But isn't it very messy, with all the tweet fields that doesn't really matter us? We can select the fields to display by adding the following:

```
POST tweets-14240977515603235/_search
{
   "size": 100
   "fields": [
      "user.screen_name",
      "text"
   ]
}
```

This example will display only the username of the user who wrote it and the tweet text, which makes it easier to read.

Elasticsearch also provides sort capabilities over the fields. For example, we can order the results alphabetically by the username like this:

```
POST tweets-14240977515603235/_search
{
   "size": 100,
   "fields": [
      "text"
   ],
   "sort": {
     "user.screen_name" : "asc"
   }
}
```

For data searching capabilities, Elasticsearch offers queries and filters. Filters are used for binary or exact value searches, for instance, tweets containing a specific word, while queries are used when results depend on a relevance score, for instance, tweets near a specific geopoint.

There are tens of different queries and filters (http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/query-dsl.html), but we are going see 2 really interesting ones. First the “common terms filter”, used to obtain accurately a list of results ordered by relevance that match the queried text:

```
POST tweets-14240977515603235/_search
{
   "size": 1000,
   "fields": [
      "text"
   ],
   "query": {
     "common": {
        "text": {
           "query": "Love and friends",
           "cutoff_frequency": 0.1
        }
     }
   }
}
```

This query will obtain, ordered by relevance (field _score), the tweets whose text match the text “Love and friends”. The algorithm differentiates between relevant words (love, friends) and irrelevant words (and) to obtain more precise results. The cutoff_frequency is used as the frontier between what a relevant and an irrelevant word is. You can read more about it at http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/query-dsl-common-terms-query.html

Finally, let’s see an example of an interesting filter, and one of the most powerful capabilities on Elasticsearch: geospatial indexes. Using the geo_distance filter, we are capable to obtain tweets within a specific distance from a geo point. Let’s add a geo_distance filter to the previous query:

```
POST tweets-14240977515603235/_search
{
   "query": {
      "filtered": {
         "query": {
          "common": {
            "text": {
               "query": " Love and friends ",
               "cutoff_frequency": 0.1
            }
          }
         },
         "filter": {
            "geo_distance": {
               "distance": "3000km",
               "geo.coordinates": [
                  -101,
                  40
               ]
            }
         }
      }
   },
   "fields": [
      "place.full_name",
      "text"
   ]
}
```

First, we use a "filtered" query (http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/query-dsl-filtered-query.html) to combine a query and a filter. Within the query, we include common terms query we used before. As for the filter, we create a geo_distance filter (http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/query-dsl-geo-distance-filter.html) that will filter out all tweets located outside the radius: 3000km around the (40, -101) coordinate, which is approximately the center of USA. The filter is quite restrictive, as most of tweets will have no geoposition and, thus, will be filtered out. Feel free to adjust either the query or the filter. You can check the coordinates of a specific point at http://www.mapcoordinates.net/en

6 Installing Kibana

Kibana is Elasticsearch's data visualization engine, allowing you to natively interact with all your data in Elasticsearch via custom dashboards. Kibana's dynamic dashboard panels are savable, shareable and exportable, displaying changes to queries into Elasticsearch in real-time. You can perform data analysis in Kibana's beautiful user interface using pre-designed dashboards or update these dashboards in real-time for on-the-fly data analysis.

In order to run Kibana, it is necessary to configure a web server. For those outside FIB, refer to the official documentation present at http://www.elasticsearch.org/overview/kibana/. For FIB students, there is a way to configure an apache server easily with the following script:

```
apache2inst
```

The script configures the apache server, tells you the necessary commands to start and stop it, and also the root folder. Navigate to the apache folder, and execute the following command to copy the Kibana onto the web server folder:

```
cp –r /assign/cc-mei/Kibana ./www/htdocs/
```

Then, start the server with the command:

```
./sbin/apache2ctl start
```

Kibana will be reachable at the url http://localhost:8080/kibana

7 Displaying data with Kibana

Let's start with a blank dashboard, and start configuring some panels to visualize the data we recently extracted.

http://localhost:8080/kibana/index.html#/dashboard/file/blank.json

On the top of the interface, you can specify queries and filters, similarly to what we just did using elasticsearch API. In order to obtain results from elasticsearch, we need to tell kibana which index to look at to obtain the data. All tweets stored in elasticsearch include a _index, _type and _timestamp fields, so we can specify the following query:

```
_index:YOUR_INDEX_HERE
```

Replace YOUR_INDEX_HERE with your index name. Queries follow the apache Lucene syntax (http://lucene.apache.org/core/).

Time to create some panels! Kibana interface is organized in rows, each row containing 12 length units. Panels are contained inside rows, and take at least 1 unit and up to 12. Each panel displays data for a specific field. Let's create our first data panel, that shows the most common words from the tweet message:

- Create a new row clicking the "Add a row" button. Set as row name "Words", and 250px as Height. Press Save when done.
- On the left sidebar within the row, click the "Add panel" button (the green cross).
- Select "terms" as the Panel Type.
- Put "Most common words" as tittle, span 4 (that's the number of units that is gonna take within the row, so 4/12 = 1/3 of the row).

- In Parameters, we have to specify the field to display data from. In our case, tweet message is contained in the "text" field.
- In the view options, you can select which kind of graphic you'd like to use. Feel free to change between pie or bar, and position the legend as you wish.
- Finally, in Queries, select "selected" then click on the query you just created before. This will tell kibana to display the data results using that query.

Congratulations! You just created your first Kibana panel. The information display is obtained using a regular elasticsearch API call, that can be seen using the "inspect" button (the small i). You can edit the panel using "configure" (the small cog). Panel can also be moved, resized and deleted.

Time for you to play! Try to create the following panels. You can always check all tweet fields in the "Table" panel at the bottom of Kibana:

- Using the bettermap panel, create a map displaying the tweet geoposition. Set "geo.coordinates" as the coordinate field, and "text" as the tooltip field.
- Using map panel, create a "tweets per country" panel. The tweet country code is in the "place.country_code" field. You can choose Europe, USA or worldwide maps.
- Using a terms panel, create a pie chart showing the most used languages. The tweet field is "lang".
- Using the histogram panel, create a tweet count in 1 minute intervals. The time field is in the _timestamp field.
- Using terms panels, create the most common hashtags and users, similarly to the most common words. Fields are "entities.hashtags.text" and "entities.user_mentions.screen_name"

If you struggle to create any panel, do not worry! You have them all created here:

http://localhost:8080/kibana/

Remember to set the query to point to your index!

Finally, it’s important to note that all panels act also as filters. Try to click on a terms panel, click & drag on a histogram, or click on a country within the map panel. You’ll see how the corresponding filter is created under the “Filtering” tab, at the top.