ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

Elasticsearch

Advanced Topics on NoSQL databases

A4 - S8

ESILV
nicolas.travers (at) devinci.fr

Chapter 1
Setting Movies dataset

For this practice work, we will use the elasticsearch indexing software.

1.1 cURL

`cURL` is a small executable which sends/receives HTTP requests in command lines. You can specify headers by adding parameters to your command (XPUT/XGET/XPOST, -H"Content-Type...", –databinary...).

You can download `cURL` (already installed under Linux and MacOSX) here (better to use the SSL embedded version):
`https://curl.haxx.se/download.html`. It is natively installed on Linux, MacOSX and eventually Windows. ⚠the parameters are different on `PowerShell`, use "*Invite de commande*".

1.2 Elasticsearch & Kibana

The installation steps are a little bit straightforward[1]:

- Local installation: `https://www.elastic.co/downloads`
- Docker : name "elasticsearch-kibana" (@nshou v 6.5.4)

```
– docker pull nshou/elasticsearch-kibana:latest
```

  – For MacOSX with **M1 chipset** (2021), *amd64* VMs are not compatible. You should use *ARM64* VMs:
  `https://www.docker.elastic.co/r/elasticsearch/elasticsearch-oss:7.10.2-arm64`
  `https://www.docker.elastic.co/r/kibana/kibana:7.13.4-arm64`
  To achieve this, you should use a `docker-compose` to create a small cluster with both ES and Kibana[2].

1.3 Import dataset with cURL

- Download the `movies_elastic.json.zip` data file[3], and unzip it
- Import it in a **command line** with:

```
curl -XPUT localhost:9200/_bulk -H"Content-Type: application/json" --data-binary @movies_elastic.json
```

1.4 Test your database

1.4.1 Test on your browser, command line (cURL) or Kibana dev tools:

```
GET http://localhost:9200/movies/movie/1

GET http://localhost:9200/movies/_search
```

[1] A guide is online: `https://chewbii.com/elasticsearch/`
[2] An example here: `https://chewbii.com/elasticsearch/`
[3] `https://devinci-online.brightspace.com/d2l/le/content/15379/viewContent/18377/View`

Chapter 2
DSL queries

For each query, you must provide an URL with :

- A server to reach: `http://localhost:9200`
- The index (and with older version the "type"): `movies`
- A used service: `_search`
- This produces the URL: `http://localhost:9200/movies/_search`

You can write the query in three formats (when possible):

- With the HTTP GET method (web brower), with the q= parameter

```
http://localhost:9200/movies/_search?q=fields.title:foo+bar
```

- With the HTTP POST method, with the JSON document param "*query.json*" (called DSL queries):

```
curl -XPOST "localhost:9200/movies/movie/_search&pretty" -H"Content-Type: application/json" -d @query.json -o output.json
```

  And then open this JSON document in your Web browser.

- You can also make those queries directly in Kibana in the "Dev Tools"[1]

```
GET /movies/_search
{"query":{
        "match":{"fields.title":"foo bar"}}
}
```

2.1 Simple queries

2.1.1 Every movies which title matches 'Star Wars' (match query),
Correction : On the direct REST API (Web browser):

```
_search?q=title:Star+Wars
```

Or by POST http request on the REST API with an input document (curl/kibana):

```
{"query":{"match":{"fields.title":"Star Wars"}}}
```

The answer can be surprizing since "Bride Wars" or "Star Trek" match. In fact, a search engine gives a score by taking into account each term of the query, and the length of the title. The Cosine function based on $tf \times idf$ is used to produce this score.

2.1.2 Try with exact match (match_phrase),
Correction :

```
{"query":{"match_phrase":{"fields.title":"Star Wars"}}}
```

[1]Kibana dev tool: `http://localhost:5601/app/dev_tools#/console`

2.1.3 Star Wars movies and Directors equal to 'George Lucas' (boolean query),

Correction :

```
{"query":{
        "bool": {"should": [
                        { "match": { "fields.title": "Star Wars" }},
                        { "match": { "fields.directors": "George Lucas" }}
                ]
}}}
```

Elasticsearch is a textual search engine, so results are ranked according to the score of relevance, thanks to "should". You can require to make it mandatory with "must":

```
{"query":{
        "bool": {"should": [
                 { "match_phrase": { "fields.title": "Star Wars" }},
                 { "match_phrase": { "fields.directors": "George Lucas" }}
         ]
}}}
```

2.1.4 Movies were 'Harrison Ford' played,

Correction : `_search?q=fields.actors:Harrison+Ford`

```
{"query": {"match_phrase": {"fields.actors":"Harrison Ford"}}}
```

2.1.5 Movies were 'Harrison Ford' played with a plot containing 'Jones',

Correction :

```
{"query":{
        "bool": {
                "should": [
                        { "match_phrase": { "fields.actors": "Harrison Ford" }},
                        { "match": { "fields.plot": "Jones" }}
                ]
}}}
```

2.1.6 Movies were 'Harrison Ford' played with a plot containing 'Jones' but plots without containing 'Nazis'

Correction :

```
{"query":{
        "bool": {
                "must": [
                        { "match_phrase": { "fields.actors": "Harrison Ford" }},
                        { "match": { "fields.plot": "Jones" }}
                ],
                "must_not" : { "match" : {"fields.plot":"Nazis"}}
}}}
```

We can play is specific queries' semantic: should, must, should_not, must_not.

2.1.7 Movies of 'James Cameron' which rank is better than 1000 (boolean + range query)

Correction :

```
{"query":{
        "bool": {
                "must": [
                        { "match_phrase": { "fields.directors": "James Cameron" }},
                        { "range": { "fields.rank": {"lt":1000 }}}
                ]
}}}
```

2.1.8 Movies of 'James Cameron' which rating must be higher than 5 and which genre must not be 'Action' nor 'Drama'
Correction :

```
{"query":{
        "bool": {
                "should": { "match_phrase": { "fields.directors": "James Cameron" }},
                "must":{ "range": { "fields.rating": {"gt":5 }}},
                "must_not":[
                        {"match":{"fields.genres":"Action"}},
                        {"match":{"fields.genres":"Drama"}}
                ]
}}}
```

The list of "must_not" is preferable since we do not want to remove movies with both Action and Drama (a single match with a list), but neither of them.

2.1.9 Movies of 'J.J. Abrams' which released date is mandatory between 2010 and 2015 (**filtered** query on `release_date`)

Correction :

```
{"query": {
        "bool":{
                 "must": {"match": {"fields.directors": "J.J. Abrams"}},
                 "filter": {"range": {"fields.release_date": {
                                         "from": "2010-01-01",
                                         "to": "2015-12-31"}}}
}}}
```

You can see that "*filter*" is proper to a metadata selection (pre-filtering step) while "should" and "must" computes a relevance score. So, choose properly your query according to filters and scoring results.

2.2 Aggregate queries

We wish now to do some aggregate queries on the index in order to extract some statistics.

2.2.1 Complex queries

2.2.1 Give for each year the number of movies,
Correction :

```
{"aggs" : {
        "nb_per_year" : {
                "terms" : {"field" : "fields.year"}
}}}
```

2.2.2 For each category (genres), give the number of movies. To take into account the whole text, you need to use ”keyword” after the required field.
Correction : For the mapping, you must send a query on the ’movies’ index:

```
{"aggs" : {
        "nb_per_category" : {
                "terms" : {"field" : "fields.genres.keyword"}
}}}
```

2.2.3 Give the average rating of movies,
Correction :

```
{"aggs" : {
        "avg_rating" : {
                "avg" : {"field" : "fields.rating"}
}}}
```

2.2.4 Give the average rating of George Lucas’ movies,
Correction :

```
{
        "query" :{
                "match_phrase" : {"fields.directors" : "George Lucas"}
        }
        ,"aggs" : {
                "note_moyenne" : {
                        "avg" : {"field" : "fields.rank"}
}}}
```

2.2.5 Count the number of movies for the given ranges of rating: 0-1.9, 2-3.9, 4-5.9...),
Correction :

```
{"aggs" : {
        "group_range" : {
                "range" : {
                        "field" : "fields.rating",
                        "ranges" : [
                                {"to" : 1.9},
                                {"from" : 2, "to" : 3.9},
                                {"from" : 4, "to" : 5.9},
                                {"from" : 6, "to" : 7.9},
                                {"from" : 8}
                        ]
                }
}}}
```

2.2.6 Number of distinct directors in adventures movies,
Correction :

```
{
        "query":{
                "match" : {"fields.genres" : "Adventure"}
        },
        "aggs" : {
                "nb_distinct" : {
                        "cardinality" : {"field" : "fields.directors.keyword"}
                }
}}
```

2.2.2 Hard queries

2.2.1 Give the average rating per genre,
Correction :

```
{"size":0, "aggs" : {
        "group_genres" : {
                "terms" : {
                        "field" : "fields.genres.keyword"
                },
                "aggs" : {
                        "avg_rating" : {"avg" : {"field" : "fields.rating"}}}
}}}
```

“size” is used to avoid displaying top 20 results and show only the aggregated result.

2.2.2 Give min, max and average rating for each genre,
Correction :

```
{"size":0, "aggs" : {
        "group_genres" : {
                "terms" : {"field" : "fields.genres.keyword"},
        "aggs" : {
                "avg_rating" : {"avg" : {"field" : "fields.rating"}},
                "min_rating" : {"min" : {"field" : "fields.rating"}},
                "max_rating" : {"max" : {"field" : "fields.rating"}}
        }
}}}
```

2.2.3 Give the average ranking of movies per year and sort them increasingly,
Correction :

```
{"size":0,"aggs" : { "group_year" : {
        "terms" : {
                "field" : "fields.year",
                        "order" : { "avg_ranking" : "asc" }
                },
                "aggs" : {
                        "avg_ranking" : {
                                "avg" : {"field" : "fields.rank"}
}} }}}
```

Double aggregation is required.

2.2.4 Give average movie's rank and average movie's rating for each director. Sort the result decreasingly on average rating,
Correction :

```
{"size":0, "aggs" : {
        "group_director" : {
                "terms" : {
                        "field" : "fields.directors.keyword",
                        "size":10000,
                        "order" : { "avg_rating" : "desc" }
                },
                "aggs" : {
                        "avg_rank" : {"avg" : {"field" : "fields.rank"}},
                        "avg_rating" : {"avg" : {"field" : "fields.rating"}}
        }
}}}
```

A double aggregation is required to group by director, and then per rank and rating.

2.2.5 Give the terms occurrences extracted from each movie's title. The text value requires a specific mapping on the dataset stored in elasticsearch, see: `https://www.elastic.co/guide/en/elasticsearch/reference/current/fielddata.html`.
Correction : To be executed either on Kibana Dev Tools or command line, but using the service "_mapping"

```
PUT /movies/_mapping
{ "properties": {
        "fields.title": {
                "type":  "text",
                "fielddata": true
}}}
```

```
curl -XPUT http://localhost:9200/movies2 -H "Content-Type: application/json" -d @mapping.json
```

The `mapping.json` file contains the JSON document presented above (Kibana dev tools).

```
{"size":0, "aggs" : {
        "occ_per_term_in_title" : {
                "terms" : {"field" : "fields.title"}
}}}
```

The aggregation is made on each occurrence of the "title" words but Stop words are also diplayed in the result set (to tackle this issue, see Bonus chapter 3.2 for the correct analyzer)

2.2.6 Most significant terms in plots of George Lucas movies,
Correction :

```
PUT /movies/_mapping
{ "properties": {
        "fields.plots": {
                "type":  "text",
                "fielddata": true
}}}

GET /movies/_search
{
        "query" :{"match_phrase" : {"fields.directors" : "George Lucas"}},
        "aggs" : {
                "terms_significatifs" : {
                        "significant_terms" : {"field" : "fields.plot"}
        }},
        "size":1
}
```

The result with ”match” is interesting to look at.

Chapter 3

Bonus (More hard queries)

3.1 Synonyms

We can insert synonyms inside the search engine in order to change the mapping between words. For this, we need to define a new **analyzer**[1]

3.1.1 List of synonyms

First, create a file which contains the synonyms (called here "*SYNONYMS.TXT*"). The structure is detailed here: `https://www.elastic.co/guide/en/elasticsearch/guide/current/synonym-formats.html`

Here is an example:

```
"united states             => usa",
"united states of america => usa"
```

Put them in the following folder: `elasticsearch/config/analysis/SYNONYMS.TXT`

3.1.2 Change the analyzer

- Close your "movies" index:
  `POST movies/_close`
- Add a tokenizer:

```
PUT movies/_settings
{
    "settings": {
        "index" : {
            "analysis" : {
                "analyzer" : {
                    "MY_SYNONYMS" : {
                        "tokenizer" : "whitespace",
                        "filter" : ["graph_synonyms"]
                    }
                },
                "filter" : {
                    "graph_synonyms" : {
                        "type" : "synonym_graph",
                        "synonyms_path" : "analysis/SYNONYMS.TXT"
                    }
                }
            }
        }
    }
}
```

- Open your "movies" index:
  `POST movies/_open`

3.1.3 Query with synonyms

Use the analyzer

[1] `https://www.elastic.co/guide/en/elasticsearch/guide/current/using-synonyms.html`

```
POST movies/_search
{ "_source": {"includes":["..."]},
   "query": {
       "match" : {
           "fields.title": {
               "query" : "United States",
               "analyzer": "MY_SYNONYMS"
           }
       }
   }
}
```

To train, search for the *WordNet* dataset which contains a huge number of synonyms:
`https://www.kaggle.com/duketemon/wordnet-synonyms`
Change the file in the proper format, and import into ES. Then query your dataset with corresponding synonyms and see the different output.

3.2 Extra bonus

Other things you can look at:

- Use stemming and language analyzers:
  `https://www.elastic.co/guide/en/elasticsearch/reference/6.6/analysis-lang-analyzer.html`
- N-Grams tokenizers:
  `https://www.elastic.co/guide/en/elasticsearch/reference/6.6/analysis-ngram-tokenizer.html`
- Scripting with Painless:
  `https://www.elastic.co/guide/en/elasticsearch/painless/6.6/painless-examples.html`
- Add X-Pack plugins to elasticsearch:
  `https://www.elastic.co/guide/en/elasticsearch/reference/current/xpack-api.html`
- Integrate your **dataset** with *Logstash*:
  `https://www.elastic.co/guide/en/logstash/current/getting-started-with-logstash.html`