Bootcamp
Data Engineering

Module01
Elasticsearch

Exercise 00 - The setup.

Turn-in directory :	ex00
Files to turn in :	
Forbidden function :	None
Remarks :	n/a

Let's start simple:

- Download and install Elasticsearch.
  - Go to Elasticsearch download.
  - In the product filter select `Elasticsearch`.
  - Choose the version 7.5.2 and download the tar.gz file.
- Unzip the file
- You should have several directories:

Directory	Description
`/bin`	Binary scripts including elasticsearch to start a node and elasticsearch-plugin to install plugins
`/config`	Configuration files including elasticsearch.yml
`/data`	The location of the data files of each index and shard allocated on the node
`/jdk`	The bundled version of OpenJDK from the JDK maintainers (GPLv2+CE)
`/lib`	The Java JAR files of Elasticsearch
`/logs`	Elasticsearch log files location
`/modules`	Contains various Elasticsearch modules
`/plugins`	Plugin files location. Each plugin will be contained in a subdirectory

- Start your cluster by running the `./elasticsearch` in the `/bin` folder and wait a few seconds for the node to start.

Ok so now your cluster should be running and listening on `http://localhost:9200`. Elasticsearch works with a REST API, which means that to query your cluster you just have to send an HTTP request to the good endpoints (we will come to that).

Check you can access the cluster:

```
curl http://localhost:9200
```

You can do the same in a web browser.

You should see something like this:

```
{
  "name" : "e3r4p23.42.fr",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "SZdgmzxFSnW2IMVxvVj-9w",
  "version" : {
    "number" : "7.5.2",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "e9ccaed468e2fac2275a3761849cbee64b39519f",
    "build_date" : "2019-11-26T01:06:52.518245Z",
    "build_snapshot" : false,
    "lucene_version" : "8.3.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
```

```
  "tagline" : "You Know, for Search"
}
```

If not, feel free to look at the doc :) (or ask your neighbors, or google...) Elasticsearch setup.

Now stop the cluster (ctrl-c). Change the configuration so that your cluster name is `"my-cluster"` and the node name is `"node1"`.

Restart your cluster and check the new names with

```
curl http://localhost:9200
```

Exercise 01 - The CRUDité.

Turn-in directory :	ex01
Files to turn in :	create-doc.sh; ex01-queries.txt
Forbidden function :	None
Remarks :	n/a

Now we are going to see how to perform basic CRUD operation on Elasticsearch.

Create

I'm gonna make it easy for you: Here is a curl request that creates a document with id=1 into an index named "twitter" and containing 3 fields:

```
curl -X PUT "http://localhost:9200/twitter/_doc/1?pretty" -H 'Content-Type:
↪  application/json' -d'
{
    "user" : "popol",
    "post_date" : "02 12 2015",
    "message" : "trying out elasticsearch"
}
'
```

So, what do we have here?

HTTP PUT method (remember, Elasticsearch use a REST API) followed by:

`ip_of_the_cluster:9200/index_name/_doc/id_of_the_document`, then a header specifying the content-type as a json, and finally the json.
Every document in Elasticsearch is a json, every request to Elasticsearch is sent as a json within an HTTP request.

Try it out, you should get an answer from the server confirming the creation of the document.

Let's see another way to create a document: Modify the above request to create a document in the twitter index but this time without specifying the id of the document. The document shall have the following field:

```
{
    "user" : "popol",
    "post_date" : "20 01 2019",
    "message" : "still trying out Elasticsearch"
}
```

Hint: try POST instead of PUT

Run the following command and check you have two hits.

```
curl -XGET "http://localhost:9200/twitter/_search"\?pretty
```

Look at the `_id` of each document and try to understand why those value.

Save your curl request to a file named `create-doc.sh`. The file shall be executable for the correction and it shall create the two documents above.

Ok nice, you have just created your two first documents and your first index !!
However, using curl is not very convenient right... Wouldn't it be awesome to have a nice dev tool to write out those requests... Kibana !!
Kibana is the visualization tool of the Elastic Stack. What's the Elastic Stack? -> ELK stack.

Kibana Install

Let's install Kibana!

As you did for Elasticsearch, on the same link

- download Kibana v7.5.2
- Unzip the file with tar and run it.
- Wait until Kibana is started.

You should see something like :
`[16:09:00.957] [info][server][Kibana][http] http server running at http://localhost:5601`

- Open your browser and go to `http://localhost:5601`
- Click on the dev tool icon on the navigation pane (3rd before last)
  Here you can write your query to the cluster in a much nicer environment than curl. You should have a pre-made match_all query. Run it, in the result among other stuff, you should see the documents you have created.

Try to create the following two documents in Kibana, still in the twitter index:

```
{
    "user" : "mimich",
    "post_date" : "31 12 2015",
    "message" : "Trying out Kibana"
}
```

and:

```
{
    "user" : "jean mimich",
    "post_date" : "01 01 2016",
    "message" : "Trying something else"
}
```

Got it? Great! From now on, all queries shall be done in Kibana. Save every query you run in Kibana in the ex01-queries.txt file. You will be evaluated on this file.

Read

Now that we got the (very) basis of how to query Elasticsearch, I'm gonna let you search the answer on our own.

- Write a search query that returns all the documents contained in the 'twitter' index. You should get 4 hits
- Write a search query that returns all the tweets from 'popol'. You should get 2 hits
- Write a search query that returns all the tweets containing 'elasticsearch' in their message. You should get 2 hits
- A little more complicated: write a search query that returns all the tweets from 'mimich' (and only this user!).

You should get 1 hit.

Save all the queries in `ex01-queries.txt`.

Hints

- look for the keyword field ;)
- strings are dead long live strings

For help, please refer to the doc (or to your neighbors, or google) query dsl

Update

Update the document with id 1 as follow and change the value of the field "message" from "trying out elasticsearch" to "updating the document".
If you did this correctly when you update the document you should see "_version": 2 in the answer from the cluster

Save the query in `ex01-queries.txt`.

Delete

- Run the following command:

```
POST _bulk
{"index": {"_index": "test_delete", "_id":1}}
{"name": "clark kent", "aka": "superman"}
{"index": {"_index": "test_delete"}}
{"name": "louis XV", "aka": "le bien aimé"}
```

It is a bulk indexing algorithm, it allows to index several documents in only one request.

- Delete the document with id 1 of the test_delete index
- Delete the whole test_delete index

Save all the queries in `ex01-queries.txt`.

Exercise 02 - Your first Index Your first Mapping.

Turn-in directory:	ex02
Files to turn in:	ex02-queries.txt
Forbidden functions:	None
Remarks:	you have to put all the request you run in Kibana in ex02-queries.txt

At this point, you should have 4 documents in your twitter index from the previous exercise. You are now going to learn about the mapping.

Using NoSQL doesn't mean you should not structure your data. If you want to optimize your cluster you must define the mapping of your index. We will see why. You may have noticed that in the previous exercise, every time you created a document, Elasticsearch automatically created the index for you. Well, it also generates a default mapping for this index.

However, the default mapping is not ideal...

- We would like to retrieve all the tweets posted in 2016 and beyond. Try the following search query:

```
GET twitter/_search
{
  "query": {
    "range": {
      "post_date": {
        "gte": "01 01 2016"
      }
    }
  }
}
```

Do you have good results? No... there is a mapping issue.

- Your objective now is to create a new index called 'twitter_better_mapping' that contains the same 4 documents as the 'twitter' index but with a mapping that comply with those four requirements:

The following query should only return the tweet posted in 2016 and beyond (2 hits):

```
GET twitter_better_mapping/_search
{
  "query": {
    "range": {
      "post_date": {
        "gte": "01 01 2016"
      }
    }
  }
}
```

The following query should return only 1 hit.

```
GET twitter_better_mapping/_search
{
  "query": {
    "match": {
      "user": "mimich"
    }
```

```
  }
}
```

The mapping must be strict (if you try to index a document with a field not defined in the mapping, you get an error).

The size of the twitter_better_mapping index should be less than 5 kb (with four documents). What was the size of the original index?

Hints

- You can't modify the mapping of an existing index, so you have to define the mapping when you create the index, before indexing any document in the index.
- The easiest way to write a mapping is to start from the default mapping Elasticsearch creates. Index a document sample into a temporary index, retrieve the default mapping of this index and copy and modify it to create a new index. Here you already have the twitter index with a default mapping. Write a request to get this mapping and start from here.
- You will notice that by default ES creates two fields for every string field: my-field as "text" and my-field.keyword as "keyword" type. The "text" type takes computing power at indexing and costs storage space. The "keyword" type is light but does not offer all the search power of the "text" type. Some fields might need both, some might need just one... optimize your index!
- Once you have created the new index with a better mapping, you can index the documents manually as you did in the previous exercise or you can use the reindex API (see Elastic Doc).

Exercise 03 - Text Analyzer.

Turn-in directory:	ex03
Files to turn in:	ex03-queries.txt
Forbidden functions:	None
Remarks:	

So by now you already know that mapping a field as a keyword or as a text makes a big difference. This is because Elasticsearch analyses all the text fields at ingestion so the text is easier to search.

- Let's see an example. Ingest the two following documents in an index named school.

```
POST school/_doc
{
  "school": "42",
  "text" : "42 is a school where you write a lot of programs"
}

POST school/_doc
{
  "school": "ICART",
  "text" : "The school of art management and art market management"
}
```

We created an index that contains random schools. Let's look for programming schools in it.

- Try this request.

```
GET school/_search
{
  "query":
  {
     "match": {
       "text": "programming"
     }
  }
}
```

No results... and yet, you have probably noticed that there is a document talking about a famous programming school. It's a shame that we can't get it when we execute our request using the keyword `programming`.

- Your mission is to rectify this! Modify the school index mapping to create a shool_bis index that returns the good result to the following query:

```
GET school_bis/_search
{
  "query":
  {
     "match": {
       "text": "programming"
     }
  }
}
```

Hints

- Look for the text analyzer section in the documentation.
- There is a key notion to understand Elasticsearch: the **inverted index**. Take the time to understand how the analyzer creates **token** and how this works with the inverted index.

Exercise 04 - Ingest dataset

Turn-in directory:	ex04
Files to turn in:	
Forbidden functions:	None
Remarks:	n/a

Now that you know the basics of how Elasticsearch works, you are ready to work with a real dataset !! And to make this fun you gonna use the same dataset as for the SQL module so you can understand the differences between SQL and NoSQL.

There are many ways you can ingest data into Elasticsearch. In the previous exercise, you've seen how to create a document manually.
You could do this for every line of the CSV, with a python script for instance that parses the CSV and create a document for each line. There is an Elasticsearch client API for many languages that helps to connect to the cluster (to avoid writing HTTP requests in python): Elasticsearch client.

But there is an easier way: Logstash. Logstash is the ETL (Extract Transform Load) tool of the Elasticsearch stack. We don't want you to spend to much time learning how to use Logstash so we will guide you step by step:

- Download logstash
- Un-tar the file (still in your `/goinfre`).
- Move the 'ingest-pipeline.conf' to the `config/` in the logstash directory (unzip appstore_games.csv.zip if you have not already).

This file describes all the operations that logstash shall do to ingest the data. Let's take a look at the file:

The file is split into three parts :

- `input`: definition of the inputs.
- `filter`: operation to perform on the inputs.
- `output`: definition of the outputs.

```
input {
    file {
        path => "/absolute/path/to/appstore_games.csv"
        start_position => "beginning"
        sincedb_path => "sincedb_file.txt"
    }
}
```

- `file`: our input will be a file, could be something else (stdin, data stream, ...).
- `path`: location of the input file.
- `start_position`: where to start reading the file.
- `sincedb_path`: logstash stores its position in the input file, so if new lines are added, only new lines will be processed (ie, if you want to re-run the ingest, delete the sincedb_file).

```
filter {
    csv {
        separator => ","
        columns => ["URL","ID","Name","Subtitle","Icon URL","Average User Rating","User
↪   Rating Count","Price","In-app Purchases","Description","Developer","Age
↪   Rating","Languages","Size","Primary Genre","Genres","Original Release Date","Current
↪   Version Release Date"]
        remove_field => ["message", "host", "path", "@timestamp"]
        skip_header => true
  }
```

```
  mutate {
        gsub => [ "Description", "\\n", "
"]
        gsub => [ "Description", "\\t", "    "]
        gsub => [ "Description", "\\u2022", "•"]
        gsub => [ "Description", "\\u2013", "-"]
        split => { "Genres" => "," }
        split => { "Languages" => "," }
  }
}
```

- `csv`: we use the csv plugin to parse the file.
- `separator`: split each line on the comma.
- `column`: name of the columns (will create one field in the index mapping per column)
- `remove_field`: here we remove 4 fields, those 4 fields are added by logstash to the raw data but we don't need them.
- `skip_header`: skip the first line
- `mutate`: When logstash parse the field it escape any '' it found. This changes a'\n', '\t', '\u2022' and '\u2013' into a '\\n', '\\t', '\\u2022', '\\u2013' respectively, which is not what we want. The mutate plugin is used here to fix this.
- `gsub`: substitute '\n' by a new line and the '\\u20xx' by its unicode character.
- `split`: split the "Genres" and "Languages" field on the "," instead of a single string like "FR, EN, KR" we will have ["EN", "FR", "KR]

```
output {
    elasticsearch {
        hosts => "http://localhost:9200"
        index => "appstore_games_tmp"
    }
    stdout {
        codec => "dots"
    }
}
```

- `elasticsearch`: we want to output to an Elasticsearch cluster.
- `hosts`: Ip of the cluster.
- `index`: Name of the index where to put the data (index will be created if not existing, otherwise data are added to the cluster).
- `stdout`: we also want an output on stdout to follow the ingestion process.
- `codec => "dots"`: print one dot '.' for every document ingested.

So, all we do here is create one document for each line of the csv. Then, for each line, split on comma and put the value in a field of the document with the name defined in 'columns'. Exactly what you would have done with Python but in much less line of code.

Now, let's run Logstash:

- To run logstash you will need to install a JDK or JRE. On a 42 computer, you can do this from the MSC.
- Edit the ingest-pipeline.conf with the path to the appstore_games.csv
- `./bin/logstash -f config/ingest-pipeline.conf`

You should have 17007 documents in your index.

Exercise 05 - Search - Developers

Turn-in directory:	ex05
Files to turn in:	ex05-queries.txt
Forbidden functions:	None
Remarks :	

Let's start with some queries you already did for the SQL module.

We are looking for developers involved in games released before 01/08/2008 included and updated after 01/01/2018 included.

- Write a query that returns the games matching this criterion.

Your query shall also filter the "_source" to only return the following fields: `Developer`, `Original Release Date`, `Current Version Release Date`.

You should get 3 hits.

Hints

- You might need to adjust the mapping of your index
- Create a new index and use the reindex API to change the mapping rather than using logstash to re-ingest the CSV
- The "bool" query will be useful ;)

Exercise 06 - Search - Name_Lang

Turn-in directory :	ex06
Files to turn in :	ex06-queries.txt
Forbidden function :	None
Remarks:	

We are looking for the Name and Language of games between 5 and 10 euros.

- Write a query that returns the games matching this criterion.

Your query shall filter the "*source*" to return the following fields only: "Name", "Languages", "Price".

You should get 192 hits.

Hints

- You might need to adjust the mapping of your index
- Create a new index and use the reindex API to change the mapping rather than using logstash to re-ingest the CSV.

Exercise 07 - Search - Game

Turn-in directory:	ex07
Files to turn in:	ex07-queries.txt
Forbidden functions:	None
Remarks :	

Elasticsearch was initially designed for full-text search, so let's try it.

- I'm looking for a game (and no other genre will be accepted). I'm a big fan of starcraft and I like real-time strategy. Can you write a query to find me a game?

It's a good time to look at how Elasticsearch scores the documents so you can tune your query to increase the result relevance.

There isn't one good answer to this exercise, many answers are possible. Just make sure the top hits you got are relevant to what I'm searching for!

Exercise 08 - Aggregation

Turn-in directory :	ex08
Files to turn in :	ex08-queries.txt
Forbidden function :	None
Remarks :	

Let's do some aggregation!

- Write a query that returns the top 10 developers in terms of games produced.
- Set the size to 0 so the query returns only the aggregation, not the hits.

Exercise 09 - Aggregation in Aggregation

Turn-in directory :	ex09
Files to turn in :	ex09-queries.txt
Forbidden function :	None
Remarks :	

We would like to know what is the most represented game “Genre” (top 10) and for each of those “genre” the repartition of the “Average User Rating” with a bucket interval of one (ie: for each Genre the number of game with an Avg User Rating of 1, of 2, 3, 4, and 5).

- This must be done in a single query.

Exercise 10 - Kibana

Turn-in directory:	ex10
Files to turn in:	ex10-queries.txt
Forbidden functions:	None
Remarks:	

Time to explore Kibana a little bit more.

- Your goal is to create a Dashboard with the following visualizations:
  - A plot showing the number of games released (Y axis) over time (X axis)
  - A histogram that counts the number of games released each year, and for each year the count of the "average user rating" by an interval of 1
  - A Pie Chart showing the repartition of Genres
  - A cloud of words showing the top developers
- Once your dashboard has been created, explore the possibilities of Kibana (click on the top developer in the cloud of words for instance).

Your Dashboard should look like something like this:

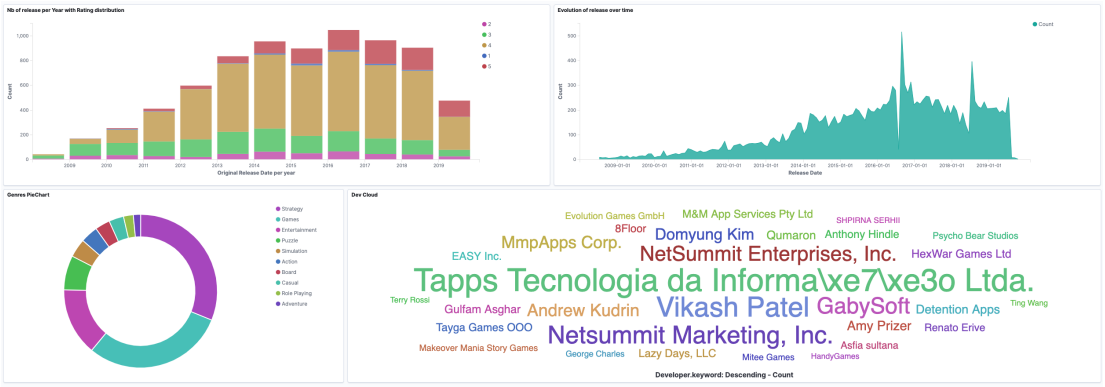

Figure 1: Dashboard

Hints

- You need to create an index pattern first (Go in the management menu, then index pattern).
- Create each visualization in the visualization tab and then create the dashboard in the dashboard tab.