Bootcamp
Data Engineering

ARTIFICIAL INTELLIGENCE

Bootcamp Data Engineering

Day01 - NoSQL with Elasticsearch

Today, you will learn how to use a NoSQL database: Elasticsearch.
Wait.. Elasticsearch is a database ? Well not exactly it is more than that. It is defined as a search and analytics engine. But let's keep it simple for now, consider it as a database, we will see the rest later.

Notions of the day

Create an Elasticsearch cluster, create index and mappings, ingest document, search & aggregate, create visual with Kibana

General rules

- The exercises are ordered from the easiest to the hardest.
- Your exercises are going to be evaluated by someone else, so make sure that your variable names and function names are appropriate and civil.
- Your manual is the internet.
- You can also ask questions in the dedicated channel in the 42 AI Slack: 42-ai.slack.com.
- If you find any issue or mistakes in the subject please create an issue on our dedicated repository on Github: issues-url

Foreword

Did you know that Elasticsearch helps you find your soulmate (or ONS...) ?

With more than 26 million swipes daily, Tinder connects more people one-on-one and in real time than any other mobile app in the world. Behind the minimalist UI and elegant “swipe right, swipe left” that Tinder pioneered are tremendous challenges of data science, machine learning, and global scalability.
Hear how Tinder relies on the Elastic Stack to analyze, visualize, and predict not only a) which people a user will swipe right on, or b) which people will swipe right on that user, but c) when

there's a mutual swipe match. Tinder's VP of Engineering will describe how the service is growing into a global platform for social discovery in many facets of life.

Helper

- Your best friend for the day: Elasticsearch-documentation
- We recommand to use the /goinfre directory for this day as you will need ~3Go for the Elasticsearch cluster. But you are free to do as you whish. Note that using /sgonfre won't work.
- Keep in mind that the /goinfre is a local & temporary directory. So if you change computer you will lose your work, if you log out you might lose your work.

Exercice 00 - The setup.

Exercice 01 - CRUDité.

Exercice 02 - Your first Index Your first Mapping.

Exercice 03 - Text Analyzer

Exercice 04 - Eat them all! (ingest with logstash)

Exercice 05 - Search

Exercice 06 - Aggregation

Exercice 07 - Kibana & Monitoring

Exercice 08 - Security

Exercice 09 - Troubleshooting

Exercice 10 - Expand the cluster!

Exercise 00 - The setup.

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

Let's start simple:

• Download and install Elasticsearch.
Go to elasticsearch-download-url
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:

/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 second 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 end points. (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 neighbours, or goolge...)
elasticsearch-setup-url

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.

Turnin directory :	ex01
Files to turn in :	create-doc.sh; ex01-rendu.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 create 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 specifing the content-type as a json, and finally the json.

Every documents in Elasticsearch is a json, every request to Elasticsearch is sent as a json within a 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 specifiyng 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

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

Ok nice, you have just created your two first documents and your first index !!

However using curl is not very convinient right.. Wouldn't it be awesome to have a nice dev tool to write out those request.. Kibana !!

Kibana is the visualition tool of the Elastic Stack. What's the Elastic Stack ? -> elk-stack-url

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 someting 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, 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 the query shall be done in Kibana. Save every query you run in Kibana in the ex01-rendu.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 return all the document contained in the 'twitter' index.

You should get 4 hits

- Write a search query that return all the tweet from 'popol'.

You should get 2 hits

- Write a search query that return all the tweet containing 'elasticsearch' in their message.

You should get 2 hits

A little more complicated:

- Write a search query that return all the tweet from 'mimich' (and only this user!). Hint: look for the keyword field ;)

You should get 1 hits

Save all the query in ex01-rendu.txt

For help, please refer to the doc (or to your neighbours, or goolge) elastic-setup-url

Update

Update the document with id 1 as follow and change from "message" : "trying out elasticsearch" to "message" : "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-rendu.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, it allows to index several documents in one request.

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

Save all the query in ex01-rendu.txt

Exercise 02 - Your first Index Your first Mapping.

Turnin directory :	ex02
Files to turn in :	ex02-rendu.txt
Forbidden function :	None
Remarks :	you have to put all the request you run in Kibana in ex02-rendu.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.

Doing NoSQL doesn't mean you should not structure your data. When you want to optimize your cluster you must define the mapping of your index, we will see why. You have noticed that in the previous exercise, when you create a document, Elasticsearch automatically created the index for you. Well it also create a default mapping for this index.

However the default mapping is usually not ideal.

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

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

Do you have the 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:

1- 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"
     }
    }
}
}
```

2- The following query should return only 1 hit

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

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

4- The size of the twitter_better_mapping index should be less than 5 kb (with four documents).

Hint:

- You can't modify the mapping of an existing index, so you have to define the mapping when you create the index, prior to indexing any document in the index.
- The easiest way to write a mapping is to start from the default mapping Elasticsearch create. Index a document sample into a temporay 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 noticed 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 cost storage space. The "keyword" type is light but does not offer all the search power of the "text" type. Some field might need both, some might need just one... optimize your index !
- Once you have create the new index with a better mapping, you can index the documents manually as you did in previous exercise or you can use the reindex API (see Elastic Doc)

Exercise 03 - Text Analyzer.

Turnin directory :	ex03
Files to turn in :	ex03-rendu.txt
Forbidden function :	None
Remarks :	

Elasticsearch assumes that a text must be analyzed before it can be used.
Let's ingest the two following docuements 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"
}
```

Here we have an index that contains 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 progamming school.
it's a shame that we can't get it when we execute our request using the keyword programming.

Modify the shoold index mapping to create a shool_bis index that return the good result to the following query:

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

Hint:

- Look for text analyzer in the documention.

Exercise 04 - Ingest dataset

Turnin directory :	ex04
Files to turn in :	
Forbidden function :	None
Remarks :	n/a

Now that you know the basis 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 day so you can see differences between SQL and noSQL.

There are many way you can ingest data into Elasticsearch. In the previous exercise, you've seen how to create document manually.

You could do this for every line of the csv, with a python script for instance that parse the csv and create a document for each line. There is an Elasticsearch client API for many languages that help to connect to the cluster (to avoid writting http request in python): elastic-client-url

But there is an easier way: Logstash. Logstash is the ETL (Extract Transform Load, you will learn more on this in the next day) 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:

1- Download logstash:

https://www.elastic.co/downloads/logstash

2- Un-tar the file (still in your /goinfre)

3- move the 'ingest-pipeline.conf' to the config/ in the logstash directory

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 splitted in three parts :

input {} --> definition of the inputs

filter {} --> operation to perform on the inputs output {} --> definition of the outputs

```
input {
    file {
        path => "/path/to/the/dataset/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, ...etc)

path --> location of the input file

start_position --> where to start reading the file

sincedb_path --> logstash store its position in the input file, so if new lines are added, only new line 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", "\\u2022", "•"]
}
}
```

csv{} --> we use the csv pluging 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{} --> we use the mutate pluging to fix a nasty trick from logstash: in the comment field

the new line are written with '\n'. When logstash parse the field it escape any '' it found. This changes a '\n' into a '\n' which is not what we want. The mutate pluging is used here to fix this. gsub --> subtitute '\n' by a nwe line and the '\u2022' by its unicode character.

```
output {
    elasticsearch {
        hosts => "http://localhost:9200"
        index => "data"
    }
    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 creating one document for each line of the csv and 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:

- 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 - Senior

Turnin directory :	ex05
Files to turn in :	ex05-rendu.txt
Forbidden function :	None
Remarks :	

Let's start with a query you already did for the SQL day. Remember Ex07 ?

We are looking for developers involved in games released before 01/08/2008 included and update after 01/01/2018 included.
Write a query that returns the games matching this criteria.
Your query shall also filter the "_source" to only returns the following fields: "Developer", "Original Release Date", "Current Version Release Date".

Hint1: You might need to adjust the mapping of your index
Hint2: Create a new index and use the reindex API to change the mapping rather than using logstash to re ingest the csv
Hint3: "bool" query will be useful ;)

Exercise 05 - Search - Name_Lang

Turnin directory :	ex05
Files to turn in :	ex05-rendu.txt
Forbidden function :	None
Remarks :	

Let's start with a query you already did for the SQL day. Remember Ex06 ?

We are looking for the Name and Language of games strictly between 5 and 10 euros.
Write a query that returns the games matching this criteria.
Your query shall filter the "_source" to only returns the following fields: "Name", "Languages", "Price".

Hint1: You might need to adjust the mapping of your index
Hint2: Create a new index and use the reindex API to change the mapping rather than using logstash to re ingest the csv

Exercise 05 - Search - Name_Lang

Turnin directory :	ex05
Files to turn in :	ex05-rendu.txt
Forbidden function :	None
Remarks :	

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

I'm looking for a game. I'm a big fan of starcraft and I like real time strategy. Can you write a query to find me a game (and I will only accept game) ?

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

(There isn't one good answer for this exercise, many answer are possible)

Exercise 05 - Search - Name_Lang

Turnin directory :	ex05
Files to turn in :	ex05-rendu.txt
Forbidden function :	None
Remarks :	

Let's do some aggregation:

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

Exercise 05 - Search - Aggregation in Aggregation

Turnin directory :	ex05
Files to turn in :	ex05-rendu.txt
Forbidden function :	None
Remarks :	

We would like to know for the top 10 "Genre" what is the repartition of the "Average User Rating".
Write a query that returns the top 10 "Genre" and for each genre create a histogram of the Average User Rating with an interval of 0.5

Exercise 05 - Kibana

Turnin directory :	ex05
Files to turn in :	ex05-rendu.txt
Forbidden function :	None
Remarks :	

We will now explore a little Kibana.

Your goal is to create a Dashboard with the following visualization :

- A plot showing the number of game release (Y axis) over the time (X axis)
- An histogram that count the number of game release each year, and for each year the a count of the "average user rating" by interval of 1
- A Pie Chart showing the repartition of Primary Genre
- A cloud of words showing the top developers

Once your dashboard is created, explore the possibilies of Kibana (click on the top developer in the cloud of words for instance)