**CE306/CE706 (Spring 2022)**

**Lab script 1**

It is now time to get some practical experience with information retrieval tools. [Elasticsearch](https://www.elastic.co/) is one of the most powerful and most widely used open-source search engines. It is simple, scalable, and highly efficient and can manage structured as well as unstructured data. The examples look a bit like database examples but that is because the data comes with some structure. You will see that the pre-processing steps that you have come across in this week’s lecture will actually be performed on each field individually (check the [guide](https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis.html) to find out more).

Please do not stop working with Elasticsearch when you finish this lab session. Keep playing around with it, install it on your own computer, use it in your own project, submit code, and join the community, among others. [Here](https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html) is a starting point to explore the framework before you approach the steps below.

**Installation**

This lab assumes that you are using the Linux command line. Therefore, feel free to leave the Windows environment and reboot to Ubuntu on your machine. You can also install Elasticsearch on other operating systems, but Linux nicely fits the settings of Lab 1 and Lab 2. The instructions below assume that the command line shell you are using is either *bash* or *sh*. Therefore, feel free to leave the Windows environment and reboot to Ubuntu on your machine. You can use Terminal on Ubuntu.

*NOTE: Throughout the labs we are using version 6.5.1 of Elasticsearch (and Kibana) as a reference point because we know that this works with the CSEE Lab settings. Later versions have however been released and offer a lot of new and improved features. Feel free to install the latest release (at least when you install the software on your own computer). With every new release there might be come variations in the commands, so please feel free to explore which commands would work on your release by searching for the appropriate supported commands online.*

Let's set up a folder to work in. We're creating a temporary install to play with. In a full server environment this would likely be different. There are also services which offer Elasticsearch setup and ready to run.

To start, create a new directory in one of your folders and change to that directory.

mkdir search\_exercise

cd search\_exercise

We are going to need to download Elasticsearch (the search engine we will be using):

curl -L -O https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-6.5.1.tar.gz

We'll be using a tool called curl a lot in this lab. curl is a very handy tool for communicating with various servers including HTTP. The -L above follows HTTP redirects, and the -O tells curl to write the file locally using the same filename it has on the server (elasticsearch-6.5.1.tar.gz). The default behaviour is to print the contents of the file directly to the console.

Now we can extract the archive:

tar xf elasticsearch-6.5.1.tar.gz

To run Elasticsearch:

elasticsearch-6.5.1/bin/elasticsearch -d

The -d launches Elasticsearch as a daemon, so it will run in the background.

**Indexing Documents**

Grab a collection of documents from the Elasticsearch examples:

curl -L -O https://download.elastic.co/demos/kibana/gettingstarted/accounts.zip

Now unzip the file using:

unzip accounts.zip

You will get the accounts.json file

Take a peek inside the accounts.json file. It's in a JSON format.

There is an index line that specifies the id of the document, followed by the document itself.

Let's post this collection of accounts into Elasticsearch:

curl -H "Content-Type: application/json" -XPOST "localhost:9200/bank/account/\_bulk?pretty&refresh" --data-binary "@accounts.json"

Now check to see what indices Elasticsearch has:

curl 'localhost:9200/\_cat/indices?v'

You should see a table including a bank index containing 1000 documents. If plenty of content is printed, feel free to pipe the content using (|) and using more. More details [here](https://superuser.com/questions/198529/how-to-scroll-one-page-at-a-time-in-linux-at-the-command-line).

**Searching**

Let's start with a query that matches all the documents.

curl -XGET 'localhost:9200/bank/\_search?pretty' -H "Content-Type: application/json" -d'

'

You can see from the hits.total field we matched 1000 documents and, by default, the first 10 are shown. You can view this using pipe (|) and more in Linux.

If you find manipulating the large queries in the terminal using curl a little unwieldy and getting lots of errors feel free to try Kibana. The install is very quick and there are instructions at the end of the lab sheet. The rest of the lab will show only the query section, you are free to choose how to connect.

**Pagination**

The query we performed only showed 10 documents. We can show the next 10 as follows:

curl -XGET 'localhost:9200/bank/\_search?pretty' -H "Content-Type: application/json" -d'

{

  "query": { "match\_all": {} },

  "from": 10,

  "size": 10

}

'

from sets from which document we will start, size sets how many documents are shown.

**Querying Full Text**

A full text query will take multiple words, and search for all of them giving each document a score based on how close it was. Let's try an example:

curl -XGET 'localhost:9200/bank/\_search?pretty' -H "Content-Type: application/json" -d'

{

    "query": {

        "match" : {

            "address" : "national street"

        }

    }

}

'

The address bit of the query tells us which field we will be matching.

Looking at the results from this query it seems like we searched wrong. There isn't a "National Street", there's a "National Drive" though. Notice how results that contained both "national" and "street" were returned. match defaults to being an or query, so it will match documents containing either of the two terms. If we change the operator to and our "national street" search will return 0 results, because the terms "national" and "street" are not present together in any address field.

curl -XGET 'localhost:9200/bank/\_search?pretty' -H "Content-Type: application/json" -d'

{

  "query": {

    "match": {

      "address": {

        "query": "national street",

        "operator": "and"

      }

    }

  }

}

'

However, let's try "National Drive" with "and":

curl -XGET 'localhost:9200/bank/\_search?pretty' -H "Content-Type: application/json" -d'

{

  "query": {

    "match": {

      "address": {

        "query": "drive national",

        "operator": "and"

      }

    }

  }

}

'

I've deliberately reversed the terms. Note how the search still works. It's considered the terms independently, in any order, but they must both be in the address field for the document to be a hit.

But what if we really wanted to match "drive national" exactly.

**Matching Exact Phrases**

match\_phrase matches "National Drive" exactly. This gives only 1 result.

curl -XGET 'localhost:9200/bank/\_search?pretty' -H "Content-Type: application/json" -d'

{

  "query": {

    "match\_phrase": {

      "address": "national drive"

      }

    }

  }

'

Reversing the terms as we did in the previous example does not work here. This matches the exact phrase. Sometimes however, you only have part of the phrase.

**Matching Part of Phrases**

This type of search matches a phrase with a wildcard. An example, let's try and use this to make an autocomplete/search suggest. When the user starts typing, we could suggest what they may want to type.

For example, try searching for a firstname:

curl -XGET 'localhost:9200/bank/\_search?pretty' -H "Content-Type: application/json" -d'

{

  "query": {

    "match\_phrase\_prefix": {

      "firstname": "Jo"

      }

    }

}

'

You will notice you are shown lots of records with firstnames that start with Jo including **Jo**sephine, **Jo**sephina, **Jo**sie, and many others.

**Matching Multiple Fields**

It's common in a search engine, which you would want to match multiple fields with your query type. Let's say for example we typically search by lastname when looking up customer accounts, but sometimes we get given a name and we don't know whether it is a firstname, or a lastname. To improve our recall we want to search both fields.

To achieve this we can use a multimatch:

curl -XGET 'localhost:9200/bank/\_search?pretty' -H "Content-Type: application/json" -d'

{

  "query": {

    "multi\_match": {

      "query": "Francis",

      "fields": ["firstname","lastname"]

    }

  }

}

'

This hasn't quite worked though. "Francis Beck" came before "Kelli Francis". We can *boost* the lastname field in this search to make it more important:

curl -XGET 'localhost:9200/bank/\_search?pretty' -H "Content-Type: application/json" -d'

{

  "query": {

    "multi\_match": {

      "query": "Francis",

      "fields": ["firstname","lastname^2"]

    }

  }

}

'

Now "Kelli Francis" comes first. To match query in any fields simply assign an empty list to “fields”, i.e. “fields”:[].

**Sorting**

The query below sorts the results in descending order (desc) by balance.

curl -XGET 'localhost:9200/bank/\_search?pretty' -H "Content-Type: application/json" -d'

{

  "query": { "match\_all": {} },

  "sort": { "balance": { "order": "desc" } }

}

'

Try repeating the search suggest exercise from earlier sorted alphabetically by firstname.

**Filtering**

Filtering uses bool queries. These are queries that have scores of either 0 or 1. We can extend our earlier auto complete example.

Let's pretend we have a bank office in the state of Florida, so we are only interested in our search showing those records:

curl -XGET 'localhost:9200/bank/\_search?pretty' -H "Content-Type: application/json" -d'

{

  "query": {

    "bool": {

      "must": {

        "match\_phrase\_prefix": {

          "firstname": "Jo"

        }

      },

      "filter": {

        "term": {

          "state.keyword": "FL"

        }

      }

    }

  }

}

'

Now there are only two results.

It isn't just terms we can filter by, we can filter by numeric ranges. Let's pretend we are searching for someone with a name that starts with "Jo", but we are in the mortgages department in the bank, and only process customers that hold a balance of over 11,000 as the bank says we aren't allowed to mortgage unless they have less.

curl -XGET 'localhost:9200/bank/\_search?pretty' -H "Content-Type: application/json" -d'

{

  "query": {

    "bool": {

      "must": {

        "match\_phrase\_prefix": {

          "firstname": "Jo"

        }

      },

      "filter": {

        "range": {

          "balance": {

            "from": 11000

          }

        }

      }

    }

  }

}

'

Rather than 10 results, we now have 7, excluding the results that had a balance less than 11,000.

**Exercises**

**Exercise 1**

The LA office has bought you in as a consultant. They have lots of company accounts and most often customers call up quoting their company name/employer. They want to be able to search by that, firstname, and lastname. Then they would like all the results returned in alphabetical order by the company name. They don't want to see results from any other offices though.

Can’t find LA, so ended up using CA.

{

"query": {

"bool": {

"must": {

"multi\_match": {

"query": "Hall",

"fields": ["firstname","lastname"]

}

},

"filter": {

"term": { "state.keyword": "CA"}

}

}

},

"sort": {

"employer.keyword": {

"order": "desc"

}

}

}

**Exercise 2**

The bank HQ marketing department wants to run a promotion. They're really interested in marketing to their under 30 high-income customers. They'd like a report that shows only customers under 30, in descending order of balance.

**Exercise 3**

The customer records department are having a problem with their existing system. It keep check of all the addresses for customers. When they search "Clay", the system does the following search:

curl -XGET 'localhost:9200/bank/\_search?pretty' -H "Content-Type: application/json" -d'

{

    "query": {

        "multi\_match" : {

            "query" : "Clay",

"fields": []

        }

    }

}

'

But that brings up someone with the name "Clay" first. They would like to change it so that anything with the city Clay, or Clay in the address is shown before anyone with the name Clay.

**Kibana Install (in your own time)**

There is a visualation tool called Kibana that comes with, amongst other things, a dev console that can be used to connect to your Elasticsearch instance.

Installing and running this is very similar to Elasticsearch:

curl -L -O https://artifacts.elastic.co/downloads/kibana/kibana-6.5.1-linux-x86\_64.tar.gz

tar xf kibana-6.5.1-linux-x86\_64.tar.gz

kibana-6.5.1-linux-x86\_64/bin/kibana &

Kibana doesn't come with a built in switch to run as a daemon, so we have just added the & to the end of the command to run it in the background while we carry on working.

You can then open Kibana in your browser at <http://localhost:5601/>.

**Further Ideas**

This has been a quick introduction into the install and use of Elasticsearch. The next step is making Elasticsearch part of your wider application for your specific use. There are a wide range of libraries for different languages and frameworks that can assist you in passing queries to Elasticsearch and retrieving data that you can then display to your users. You may want to pick a library or framework and experiment with displaying some data as part of a web application.

**Acknowledgements**

This lab script was originally prepared by Chris Madge.