Introduction

This document is written with future Uforia developers in mind. The following chapters will describe how Uforia and Elasticsearch interact with each other as well as with what purpose in mind the 'Elasticsearch builder[1] was created.

At the time of writing development is not yet complete and so this document may contain features not present in the version that the reader is looking at. This document will be updated when development is complete so that it accurately displays all current features and uses.

Elasticsearch itself has excellent documentation[2] and it's recommended the reader refer to this documentation to acquire the most up-to-date information on Elasticsearch and its functions. Most functions will only be explained in relation to the context. For more information about Elasticsearch functions and how to use them, please read the Elasticsearch documentation.

[1] Temporary name

[2] http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/index.html

Motivation

Elasticsearch allows direct interaction with its API via both REST and HTTP. The Elasticsearch manual gives examples using GET / PUT requests like so:

Create the 'uforia' index:

```
curl -XPUT 'http://localhost:9200/uforia/'
```

Create a 'message_rfc822' mapping in the 'uforia' index:

```
curl -XPUT 'http://localhost:9200/uforia/message_rfc822/_mapping' -d '
            "message_rfc822": {
                "properties": {
                    "Bcc": {"type": "string"},
                    "Body": {"type": "string"},
                    "Cc": {"type": "string"},
                    "Content_Type": {"type": "string"},
                    "Date": {"format": "dateOptionalTime",
                             "type": "date"},
                    "From": {"type": "string"},
                    "Organization": {"type": "string"},
                    "Subject": {"type": "string"},
                    "To": {"type": "string"},
                    "XTo": {"type": "string"},
                    "Xbcc": {"type": "string"},
                    "Xcc": {"type": "string"},
                    "hashid": {"type": "string"}
                }
            },'
```

Commandline commands like curl generally prove no problem for the more technical oriented individuals. Giving users directly access to the Elasticsearch REST API however also assumes they know enough about Elasticsearch to:

- Check if the mapping already exists
- Update / Delete existing mappings
- Handle errors
- Back up existing data before making changes

The Uforia frontend was designed with less technical users in mind and the ability to customize.

Users should be able to select which metadata fields they would like mapped and generate this automatically without having to directly interface with the REST API.

This is where the 'Elasticsearch builder' comes in. It uses PyES[3] to interact with Elasticsearch in the background and allows developers to use Python to develop around it.

It's meant to allow it to be used from the command-line and a configuration file to decide which fields of which Uforia module should be mapped when it is run.

[3] Python ElasticSearch library

Elasticsearch builder

All code is within the 'uforia-browser' repository. Inside this repository is a folder named 'build_index'. All code about the builder will be contained within that folder. The file structure is as follows:

```
build_index/
      admin_output/          ← logs and config output from web interface
      databases/             ← modules for interacting with SQL databases
      include/               ← configuration files
      json_mappings/
            _old/            ← old json mapping files
                  phpfill/   ← old php scripts for filling mappings
      libraries/
            pyes/            ← contains the PyES library.
      uf_func/               ← mapping and parsing functions
      es_coretypes.py
      main.py
```

Most of the folders should require no more explanation, but some like the 'json_mappings' will likely be phased out once development is complete.

The 'json_mappings' folder is currently only included so that we still have something to quickly refill the rfc822 mapping (emails). This mapping is currently used with 500,000 emails of Enron to demonstrate Uforia capabilities and D3JS visualisation abilities.

Most of the code is contained within main.py. The file es_coretypes.py is used as part of a function that determines the 'coretype' based on what type the SQL database has it cast as.

Example usage

./main.py will display the help for the file.
In order to start over, in other worse clear the index set in the config file, the command can be run like so:

./main.py --delete-index

This will erase all data, no backups are made and no confirmation is requested. Be sure this is what you want to do. It will require Elasticsearch to re-index your data.

The file 'custom_uforia_mapping.cfg' that is often referred to in the help is not a file automatically generated by the builder. Instead, the easiest way to create this file is most likely running *--all-modules-config* instead and then renaming that file to 'custom_uforia_mapping.cfg'.

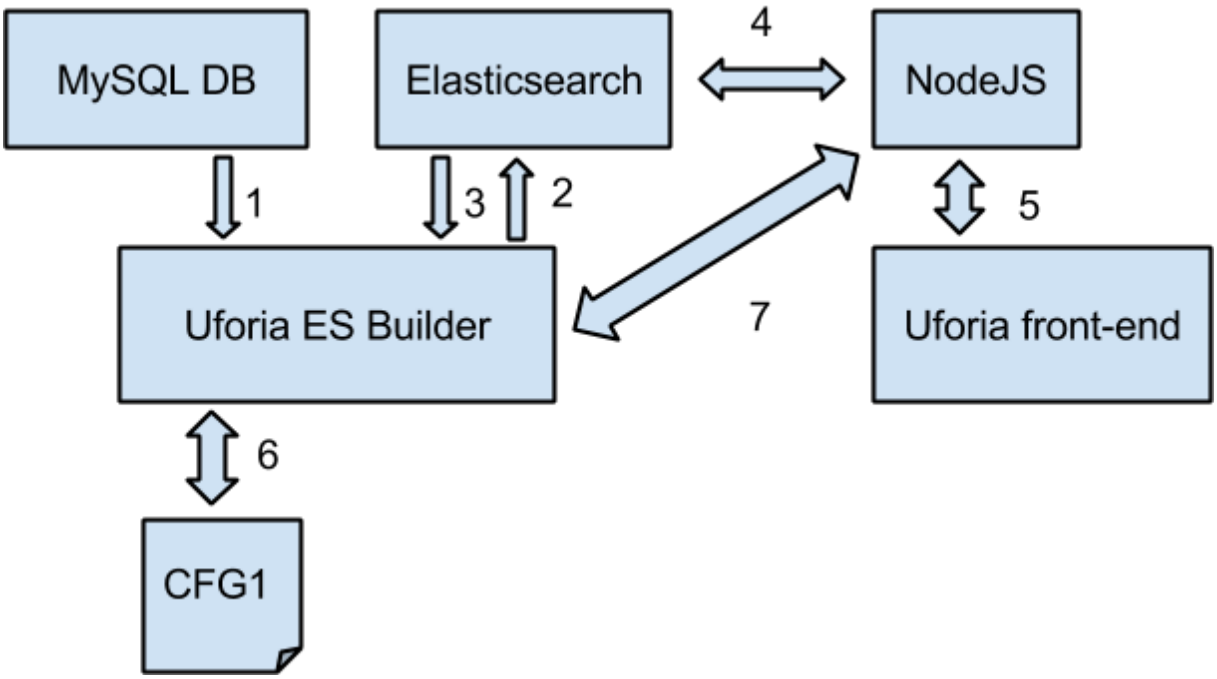
Afterwards, options such as *--gen-fields* can be run to add fields found in each table and then create the desires mapping(s) in the config file from there.

Purpose

Uforia's database changes the more/less modules are enabled and is not designed with the intent of being 'human friendly' to read. The 'builder' was being designed to allow somebody to quickly and automatically create Elasticsearch mappings and automatically fill the created mappings.

This has since further developed into something that should also allow the front-end to direct the builder through a configuration file.

The following diagram will be used to further explain what the script does and how it interacts with everything:



1 The Builder is only capable of requesting information from the MySQL. Nothing will be written to the MySQL database.

2 There are four things the builder can do when sending information to Elasticsearch.

1. Create a (new) index
2. Create a new mapping
3. Fill a mapping with information from the SQL database.
4. Update a mapping with new data

3 In order to write data to Elasticsearch a connection needs to be established. Before any of the functions that write data to Elasticsearch a connection check will be performed. The PyES library is capable of retrieving data from Elasticsearch but this false outside of its intended purposes.

4, 5 & 7 This document does not describe interactions between Elasticsearch and NodeJS, but it's important to understand that the front-end uses D3JS to visualize the data. The data D3JS uses is fed to it by NodeJS and NodeJS is the one to talk directly to Elasticsearch.

A user can designate which fields are of interest using the front-end. This information can then be fed to the builder through NodeJS. The script will in turn create a new mapping and fill the data accordingly.

6 Using the supported_mimetypes table that Uforia creates, the script goes through the each of the modules and adds each field to a default configuration file. This configuration file can then in turn be used to create mappings for each mime type automatically.

The code itself is well commented and will not be ‘picked apart’ in this document. Please refer to main.py for information about the actual code and what each individual function does and is meant to do.

Links

- http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/index.html (Elasticsearch documentation)
- http://www.uforia.nl/ (Uforia live demonstration)
- https://en.wikipedia.org/wiki/Enron_scandal (Information on the Enron scandal)
- https://pyes.readthedocs.org/en/latest/ (Python Elastic Search documentation)