Search Suggestions Documentation:

Table of Contents:

Download Elasticsearch 2.4

- Download elasticsearch version - 2.4: (*https://download.elastic.co/elasticsearch/release/org/elasticsearch/distribution/tar/elasticsearch/2.4.0/elasticsearch-2.4.0.tar.gz*)
- Extract tar file
- Start elasticsearch: **cd <tar extract>/bin/; ./elasticsearch**
- Validate elasticsearch status **curl http://localhost:9200/**

Flask Application Installation

- Python 2.7 and Flask 1.0.2 is used for development
- Create virtual environment **virtualenv venv; source venv/bin/activate**
- Install all the requirements **pip install -r requirements.txt**
- To execute flask app **python app.py**

Flask Application Structure

- **app.py** (starting point, contains flask routes information)
- **controller**
    - **es_pull_scroll.py** (scroll api is used for querying es)
    - **es_push_bulk.py** (bulk api is used indexing es)
    - **config.py** (contains default elasticsearch configuration)
- **static**
    - **es_mapping.json** (contains ed index mapping)
    - **sample.learn.logs.2016.json** (raw file containing all the json documents)
- **templates**

- **index.html**

Flask Routes

- By default flask application starts at **http://127.0.0.1:5002**/
- 2 Routes are present in application
  - '**/**' -> Default route, contains text box for query
  - '**/search/<searchtext>**' -> GET `searchtext`, fuzzy match SearchText in Elasticsearch using scroll api, Renders `index.html` with Elasticsearch results using Jinja2

Indexing Data to Elasticsearch(**es_push_bulk.py**)

All the columns present in raw files are pushed (**indexed**) to elasticsearch but analysis is performed only on **search_term**, all other columns are `not_analyzed` during indexing. Also only documents with `**result_type**` **->** `**SR**` are indexed.

To Index data follow steps:

- Go to controller **cd <search_suggestion>/controller/**
- Execute python script **python es_push_bulk.py**

Data is Indexed using **Elasticsearch Bulk API**

Querying Data In Elasticsearch(**es_pull_scroll.py**):

- Fuzzy search Elasticsearch for the provided `search text` (`T`)
- Uses Elasticsearch Scroll API to get all the matching documents
- `preserve_order` is set to True, Documents are sorted in descending order of `Match Score` (`V`) [**Assumption**]
- Internally uses `**Damerau-Levenshtein Edit Distance**` to achieve Fuzzy search
- For Search Text `**T**` greater than 2 characters, Output Results are **at most 2** `**Edit Distance**` **away from** `**Search Text**`

Elasticsearch mapping(**es_mapping.json**):

Analysis:
The objective of this step is to convert or transform the document into an inverted index and store it into a shard segment.

Analyzer:

2 types:

1. Build in
    a. Simple analyzer
    b. Whitespace analyzer
2. Custom

In this project **Custom analyzer** is used during mapping.

Consists of two steps:

- Tokenization:
    a. Standard (splitting words on white space)
    b. **NGram** (**Trigram** is leveraged in this Project)
- Filter
    - **Removing stop words**
    - **Lowercasing**
    - **Stemming**
    - **Synonyms**

Mapping Analysis Used in this project:

```
"analysis": {
    "analyzer": {
      "custom_analyzer": {
        "tokenizer": "ngram_tokenizer",
        "filter": [
          "english_possessive_stemmer",
          "lowercase",
          "custom_english_stop",
          "custom_stemmer"
        ]
      }
    },
    "filter": {
      "custom_english_stop": {
        "type": "stop",
        "stopwords": "_english_"
      },
      "custom_stemmer": {
        "type": "stemmer",
        "language": "english"
      },
      "english_possessive_stemmer": {
        "type": "stemmer",
        "language": "possessive_english"
      }
    },
```

```
  "tokenizer": {
    "ngram_tokenizer": {
      "type": "nGram",
      "min_gram": "3",
      "max_gram": "3",
      "token_chars": [
        "letter",
        "digit"
      ]
    }
  }
}
```