12-15-2022

Full-Text Search Using Elasticsearch

Akash Shrestha
Grand Valley State University

Full-Text Search Using Elasticsearch

Akash Shrestha

A Project Submitted to

GRAND VALLEY STATE UNIVERSITY

In

Partial Fulfillment of the Requirements

For the Degree of

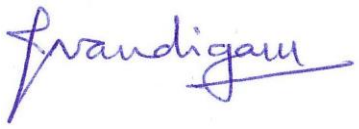
Master of Science in Applied Computer Science

School of Computing and Information Systems

December 2022

GRAND VALLEY STATE UNIVERSITY

The signatures of the individuals below indicate that they have read and approved the project of Akash Shrestha in partial fulfillment of the requirements for the degree of Master of Science in Applied Computer Science.



____ _____12/20/22_____

Dr. Jagadeesh Nandigam, Project Advisor Date

_____

Dr. Robert Adams, Graduate Program Director Date

_____

Dr. Paul Leidig, Unit head Date

Abstract

Search engines have changed the way we use the internet. They can search or filter out relevant and valuable content of interest to the users. But many of the applications we use today lack search or are just poor. So how can we leverage the same power of search engines in our applications? This project aims to look at “Full-Text Search” which allows us to do a text-based search in text-intensive data. The search will be performed by matching any, or all words of the query exactly or with some relevancy against the indexes created by the searching tool. The traditional approach to searching is performing a query on the database itself which usually returns the exact match. Searching directly on the database is not desirable because it’s not efficient for full-text search as it involves a large amount of textual data. In this project, I’ve used a search engine tool called Elasticsearch and integrated it with an existing marketplace application that is just a simple website where users would be able to post classified ads to sell their products. I have used Elasticsearch to perform searches on two domains of the application. The first one is the search feature of the website itself. This will be used by the users to search for relevant postings. The second one is the search for the application metrics. This will be used by the administrators. These metrics will contain analytics for the user-posted data, and also infrastructure-related data such as logs, all of which would be generated by performing full-text search and extraction of required data. Also, I’ve used Logstash to ingest data into Elasticsearch from my application, and Kibana for index browsing and data visualization, making use of the whole ELK stack. Elasticsearch is very rich and supports vast use cases. This project looks at the fundamental concepts of Elasticsearch and the features required for my application’s use cases only.

Introduction

In this project, I have created a search engine for an existing marketplace application. When I say search engine, it's not only a search feature in my application but also searching through all kinds of data our application has, for example, logs, metrics, etc. We know how search engines work on the internet and how they can search the whole internet. But the same capability is missing in many applications. This project aims to look at how this can be achieved, and I've used a Full-Text Search for it.

Full-Text Search is simply searching for words inside textual data. The textual data to search is usually extensive but works with texts as small as a word or sentences too. Some applications already have a very limited capability full-text search feature, for example, 'find' in word processing software, or selection queries in a database. This traditional search mostly looks for the substring of a given word or phrase in a document. But full-text search looks at the meaning and context as well, giving relevant results to a given search query. It performs searches with ranking, normalization, and linguistics, all with good performance. That's why I chose full-text search for this project.

I've used Elasticsearch for performing a full-text search which is based on the Apache Lucene search engine library. Elasticsearch provides a great interface over Lucene and is easier to implement in an application. It has real-time and scalable indexing of documents with one or more fields which can be configured using its mappings and text analyzers. It has its Query DSL to make very powerful search queries. My application has its database. So, I've used Logstash to ingest data from the database to Elasticsearch in real time. I've used Kibana to perform raw queries and visualize data from it. So, I'm making use of the whole ELK stack.

Project Management

The project work was carried out with a set of linear and sequential tasks, like the waterfall model. It was planned using the Gantt chart shown below which includes the tasks that were carried out. Most of the tasks not only involved technical work, but also some research activities. I used my local machine to setup the ELK stack. Since most of the work was done in the Elasticsearch domain, I spent the majority of time creating scripts and dashboards in the ELK stack, rather than coding in the Java application that I had.

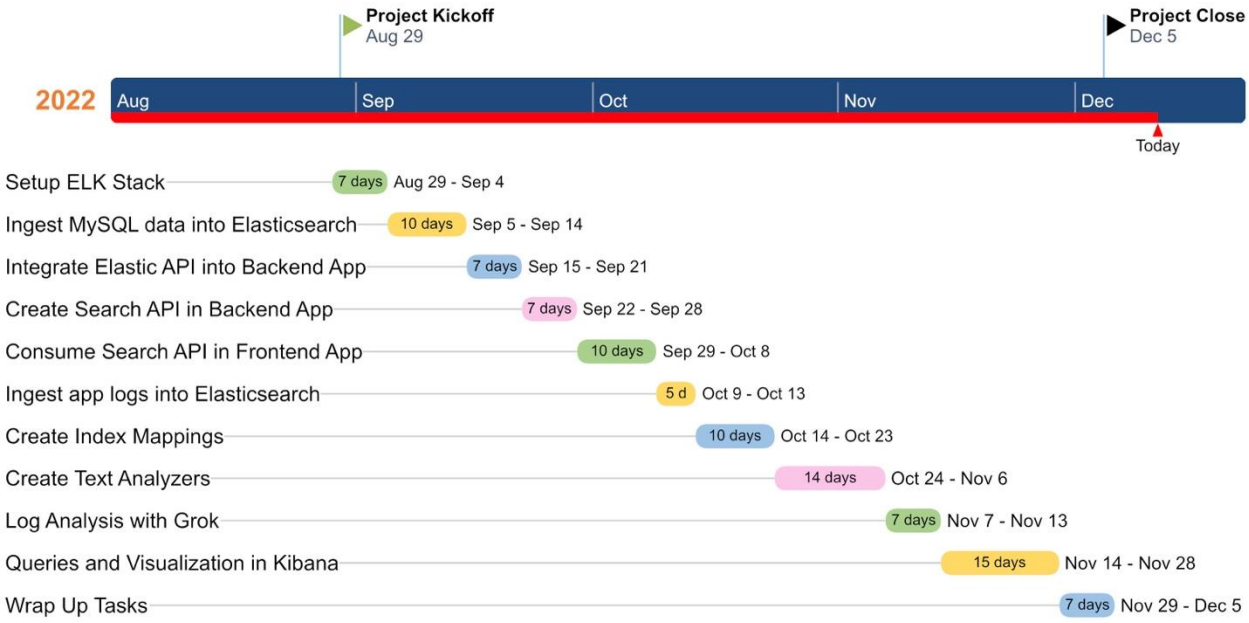
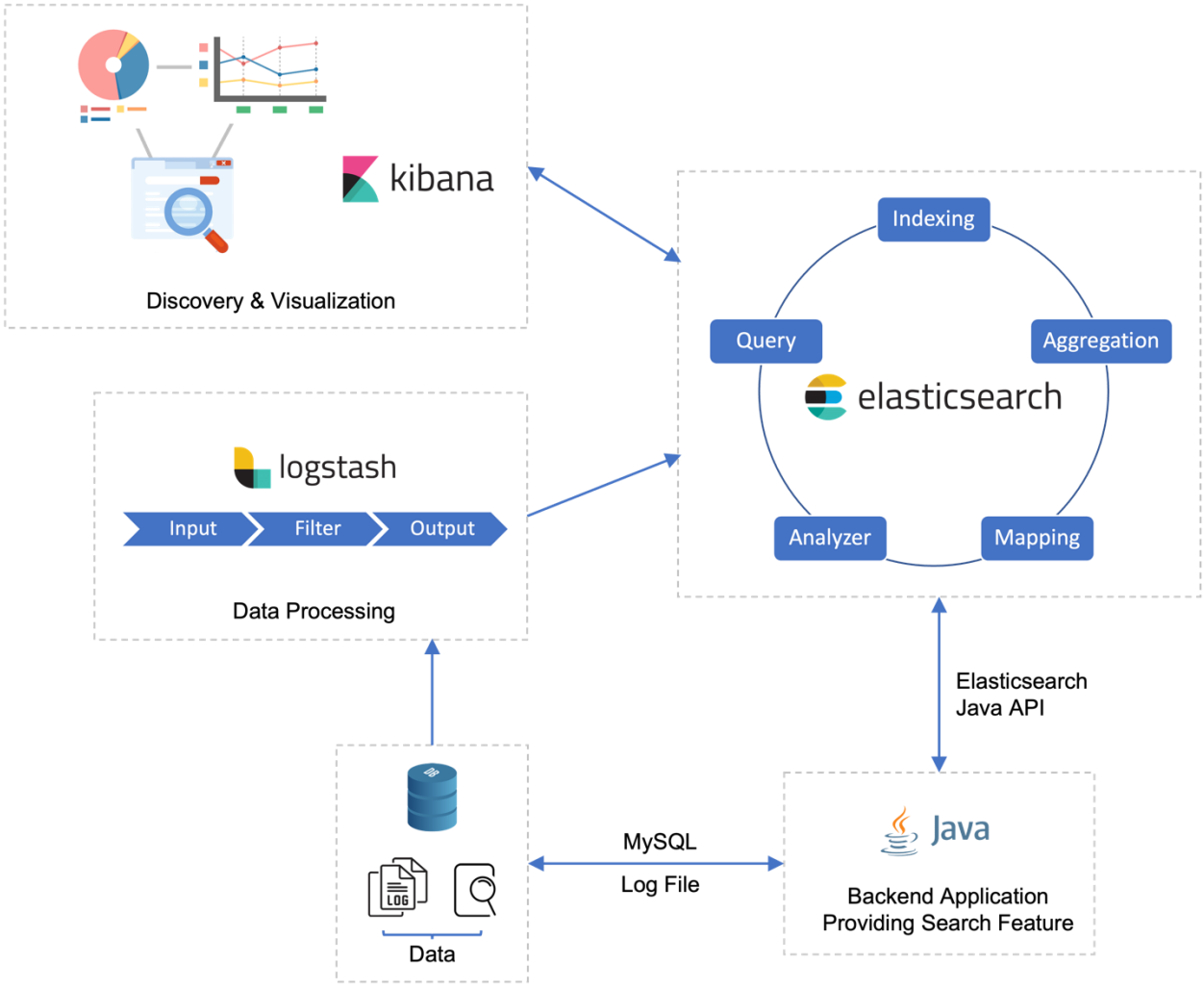


Figure: Gantt Chart for the project

Organization

The organization of my project's components is shown below. I already have a Java application using MySQL that needs to implement a search feature. I've used Logstash to ingest data from MySQL and log files into Elasticsearch. Within Elasticsearch, I've created indices using the mappings. These mappings use text analyzers to perform indexing and searching. I've also set up runtime fields for creating queries and performing the aggregation tasks, which are visualized using Kibana.



Elasticsearch Mappings

Mappings define how a document and its fields are stored and indexed. This document in my project is in JSON format. A mapping consists of properties and their datatypes, analyzers, additional fields, and other configurations. My application is a marketplace application that has posts containing fields such as titles and descriptions. I've used mappings for only these fields and since it's a text field, I must use an analyzer for them, which I've created custom as well.

```
{
   "id":     "123",
   "title": "Those Across The River, Horror Novel by
             Christopher Buehlman"
   "description": "Failed academic Frank Nichols and his
             wife, Eudora, have arrived in the sleepy
             Georgia town of Whitbrow..."
   "field1": "value1",
   "field2": "value2",
   ....
}
```

Document

Mapping configuration for 'title' field of above document:

```
PUT /my_index
{
  "mappings": {
    "properties": {
      "title": {
        "type": "text",
        "fielddata": true,
        "analyzer": "my_text_analyzer",
        "fields": {
          "keyword": {
            "type": "keyword",
            "ignore_above": 256
          },
          "terms": {
            "analyzer": "terms_analyzer",
            "type": "text",
            "fielddata":true
          }
        }
      },
      ...
    }
  }
}
```

'text' type must have 'analyzer'

a 'field' in an index can in turn have multiple fields to represent the data of that field, which can have different types and analyzers.

Text Analyzer

In Elasticsearch, for a text field, a text analyzer is a must. This text analyzer converts raw or unstructured text to a structured format that is optimized for search. It normalizes the data before indexing and also the query string that is provided while searching the indices. Search results are mostly based on this text analyzer's output giving the relevancy. It constitute of three parts: Character Filter, Tokenizer, and Token Filter.

1. **Character Filter**

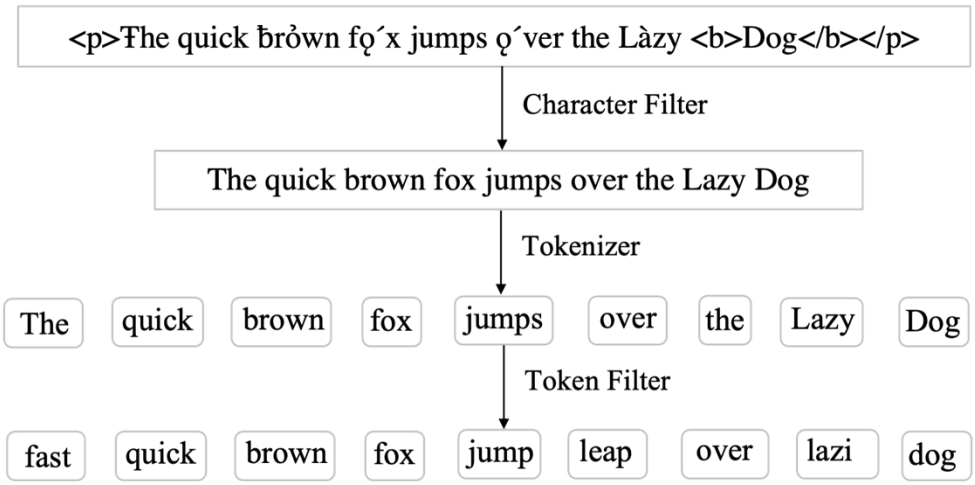
   Receives the string as a stream of characters and modifies each character.

2. **Tokenizer**

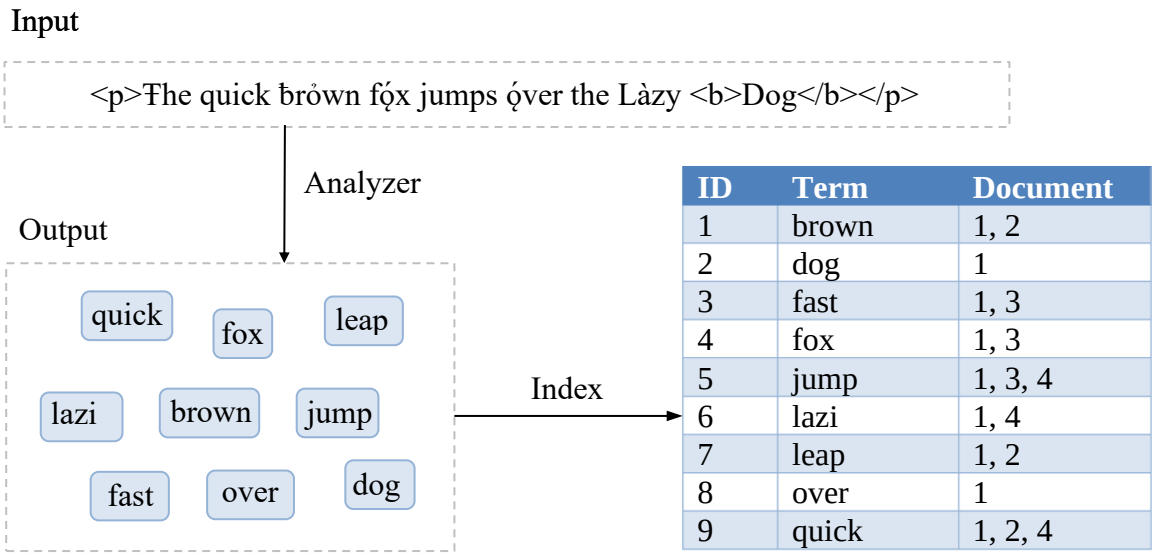
   Breaks up String into individual words (tokens) to index them separately.

3. **Token Filter**

   Takes a stream of tokens to modify, delete, or add tokens (normalization).



Input vs. Output



ID	Term	Document
1	brown	1, 2
2	dog	1
3	fast	1, 3
4	fox	1, 3
5	jump	1, 3, 4
6	lazi	1, 4
7	leap	1, 2
8	over	1
9	quick	1, 2, 4

Elasticsearch configuration for analyzer

```
PUT /my_index
{
  "settings": {
   "analysis": {
    "analyzer": {
     "my_text_analyzer": {
       "tokenizer": "standard",
       "filter": [
        "lowercase",
          "stop",
          "my_stemmer",
          "my_synonym"
       ]
     },
     "my_terms_analyzer": {
       "tokenizer": "standard",
       "filter": [
         "lowercase",
         "stop",
         "my_stemmer" ]
     }
    }
   }
  }
}
```

```
PUT /my_index
{
  "settings": {
   "analysis": {
    "filter": {
     "my_stemmer": {
      "type": "stemmer",
      "language": "english"
     },
     "my_synonym": {
       "type": "synonym_graph",
       "synonyms": [
         "car, automobile, auto",
         "romance, love, passion",
         "thriller, mystery, shocker",
         "horror, scary",
         "fantasy, fancy, illusion"
       ]
     }
    }
   }
  }
}
```

Searching Unstructured Text with Runtime Fields

When we want to search or perform some analysis on large texts such as logs where there is no association with linguistics or some structure, then we can use runtime fields to extract text. The runtime fields are the fields that are added in the runtime while performing the query. This means the fields are not added to the index. We can use a script to extract the data from the string. I've used a Grok pattern to extract some specific data as fields for my log messages. Then I performed some queries on those runtime fields to generate a visualization in Kibana.

Reflection

This project was very different from the projects that I've done previously. Mainly because this project didn't have a typical software development roadmap. Most of my time was spent on configuring the ELK stack from data ingestion, and indexing to searching. I didn't have to spend much time integrating it with my existing web application, which is why I have focused on Elasticsearch rather than integration with applications in this document as well as my presentation. There wasn't much coding involved because the Java API for Elasticsearch was simple to use once I got an understanding of how Query DSL worked. The linear approach to project management worked well for me, as the previous tasks were dependent on the next ones.

The result of the project turned out to be good, even when I had only implemented a simple use case with just a few token filters in the text analyzers for my search. But I got an understanding of how it could be expanded for complicated use cases in enterprise applications. The only reading resource that I used was the official documentation. I found the documentation to be very good and its examples to be self-explanatory. This is what led my project progress to be very effective and efficient as well. Elasticsearch has evolved and still is. So it may be difficult to keep track of version compatibilities for the mappings and queries we use.

Conclusions

The topic of my project was completely new to me and I learned a lot about the field of full-text search. The goal of this project was met with good results. Although my project was based on a particular technology called Elasticsearch, the fundamentals of full-text search it has are relevant to other search engine tools as well. For future work, a more complicated text analyzer could be built for the search feature of my application. We could also generate useful business analytics using the same indices in the future. We could expand the infrastructure monitoring beyond the logs analysis and make it insightful. Elasticsearch is a very performant system that could be distributed with multi-tenant capabilities. This feature could be explored by deploying it in remote servers ourselves, or we could use a managed service called Elastic Cloud, which offers SaaS-based services.

Appendices

1. Elastic documentation:

   https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html

2. Logstash documentation: https://www.elastic.co/guide/en/logstash/current/introduction.html

3. Kibana documentation: https://www.elastic.co/guide/en/kibana/current/introduction.html

4. Full Text Search, PostgreSQL documentation:

   https://www.postgresql.org/docs/current/textsearch-intro.html

5. What is Full-Text Search and How Does it Work?, MongoDB Documentation:

   https://www.mongodb.com/basics/full-text-search