

PYTHON MEETS “ELASTIC APP SEARCH”

Build Your Own Search Solution

CS410 – Text Information System - Technical Review

By- Kushagra Soni – soni14@illinois.edu

1. INTRODUCTION

When an ever-growing organization grows bigger, one of the biggest challenges it faces in terms of data management along with data storage, is *Data Retrieval*. Every day there is ginormous amount of data being generated of which most of it is, *Text*, and we are not talking about the Internet here.

An average mid-size organization can generate Terabytes of Internal Text data that they crave for new technologies to help manage and retrieve it for their organizational usage.

Elastic, the company which leads the current search engine arena, has come up with its own solution of “Elastic Enterprise Search” which when combined with power of Flask, can serve as a personalized high-end search and data retrieval solutions for any organization.

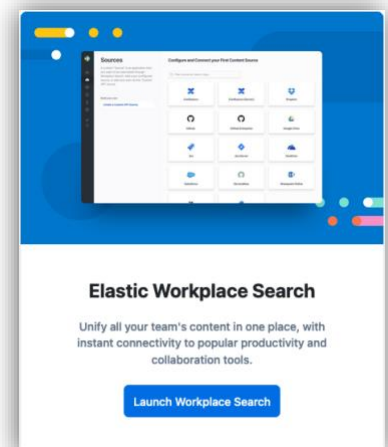
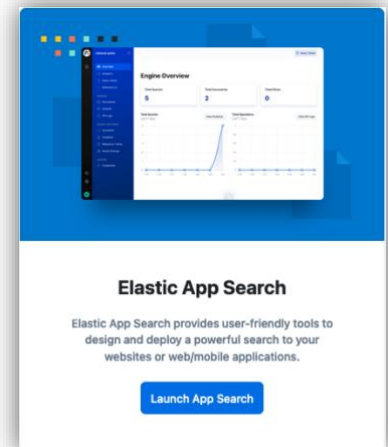
2. ELASTIC ENTERPRISE SEARCH

Elastic Search is now world’s leading open search and analytics solution. Powered by Apache Lucene it boasts of having excellent real-time index sharding and scaling capabilities. This makes it incredibly fast and reliable for all text-based searches.

Elastic, after revolutionizing with products like “Kibana”, “Logstash”, released a product named – Elastic “Enterprise Search” Suite in 2020.

Elastic “Enterprise Search Suite consists of 3 products, which are basically created on top of Elastic Search and cater different needs as per an organization.

1. Workplace Search – Unifies the data from internal content platforms like Google Drive, Slack, Salesforce, etc.
2. App Search – Using focused power of Elastic Search, it comes with Web crawler, various set of APIs, intuitive dashboards, and configurable relevance controls.



3. APP SEARCH

App Search is a ready-to-use search solution with user-friendly “relevance” tuning and provides built in analytics. Built on top of Elastic Search and Elastic Stack, App Search enables to user to easily add a powerful and customizable search experience to any application, website, or mobile app.

It comes in 2 versions, On-Prem and Cloud. The cloud version is hosted on top of Elastic Search server.

For the On-prem version usage, App Search can be installed directly via below URL.

<https://www.elastic.co/downloads/enterprise-search>

For the on-prem version of App Search needs to be connected to a running Elastic Search server already running on the system (or on cloud).

<https://www.elastic.co/downloads/elasticsearch>

4. FEATURES & USAGE

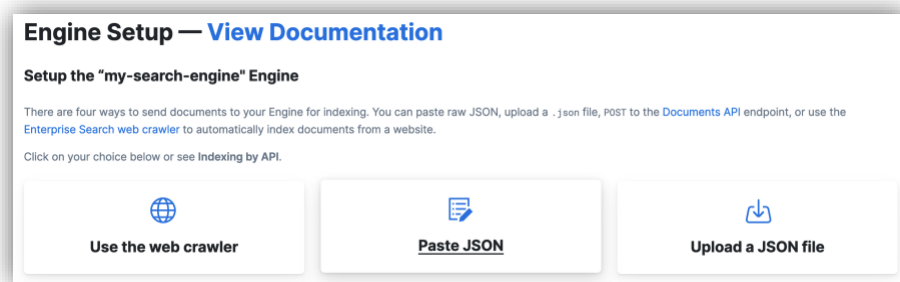
Below are the steps to use power of Elastic App Search and “Flask”

A. Create an Engine

App Search lets you create a search engine which serves as the repository to save and refactor your searched documents.

B. Setting Up the Engine

There are various ways of setting up the engine, using JSON upload, pasting a JSON or using a Web Crawler.



C. Indexing The Documents

Once the engine is setup and the documents are uploaded, they are indexed and stored by the document indexer in order make them search-optimized. It also creates a “Schema” of the indexed data, which can also be updated by user as per needs.

D. Web Crawler – A New Addition

App Search provides a web crawler which efficiently index and sync all your web content. All we need to provide is the domain and the part of the website you want to be scanned or “crawled”, and it does all the heavy lifting of indexing the web content.

E. Refining The Search Engine

Once the documents are indexed, the engine is already ready to be used. The search refinement is something which has been made easy with the below mention added functionality in the App Search.

- 1) **Relevance Tuning**: This let the user to further tune the search results based on the needs of the application being built. the consists of 2 core components: **Weights** and **Boosts**.
- 2) **Synonyms**: This let the user create set of such queries which contextually have the same meaning the in the dataset. A query can be a string made up of more than one word.
- 3) **Curations**: This let the user control the data set. They can promote or hide documents from the search.

5. APP SEARCH API ENDPOINTS & PYTHON CLIENT

All the features bundled in Elastic App Search also comes along with their respective API endpoints. these endpoints can be accessed to make use of the functionality and intuitively built relevant and useful search applications.

To access these endpoints programmatically through Python frameworks, Elastic has released, its Python API Client - *elastic_enterprise_search*. Through this client, we can access the App search client libraries.


```
=> python3.9
Python 3.9.6 (default, Jun 29 2021, 05:25:02)
[Clang 12.0.5 (clang-1205.0.22.9)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import json
>>> from elastic_enterprise_search import AppSearch
>>> with open("config.json") as json_data_file:
...     config = json.load(json_data_file)
...
>>> client = AppSearch(
...     config['appsearch']['base_endpoint'],
...     http_auth=config['appsearch']['api_key'])
```

(Instantiating a client object by passing the base endpoint and authentication key)

```
>>> client.create_engine("my-music-search")
{'name': 'my-music-search', 'type': 'default', 'language': None, 'document_count': 0}
>>>
```

(Creating a search engine directly from the client)

The same search engine can also be viewed and managed directly from the App Search UI as well.

Engines			
			Create an Engine
Name	Created	Documents	Fields
my-music-search	November 7, 2021	0	1
			Manage 

Similarly, there are various other methods which can be used to efficiently create, insert, and manage the documents directly through the Python client. This enables the developers to handle the creativity and functionality of their search application as per the needs.

6. CREATING A MUSIC SEARCH APPLICATION

Using the above-mentioned App Search client and Flask we can easily create a search engine and customize it.

Pre-Requisites:

- Already running “Elastic Search” and “App Search” server instances.
- Latest version of Python and Flask Frameworks installed through *pip* – *Python’s standard package manager*.
- Endpoints to a Music based API to source the Artist, Track and Lyrics data.
- The Elastic Search and App Search should be running on two different terminal consoles.
- A configuration json file containing the details of “app-search” endpoints, api key (setup during App Search server initiation), an engine name. the config should also contain the Music API URL and secret key details.

A. Creating the Basic Flask Framework

A basic Flask framework can be spun up by first importing the relevant libraries.

```
import json
from elastic_enterprise_search import AppSearch
from flask import Flask, render_template, request
import requests
import time

app = Flask(__name__)
```

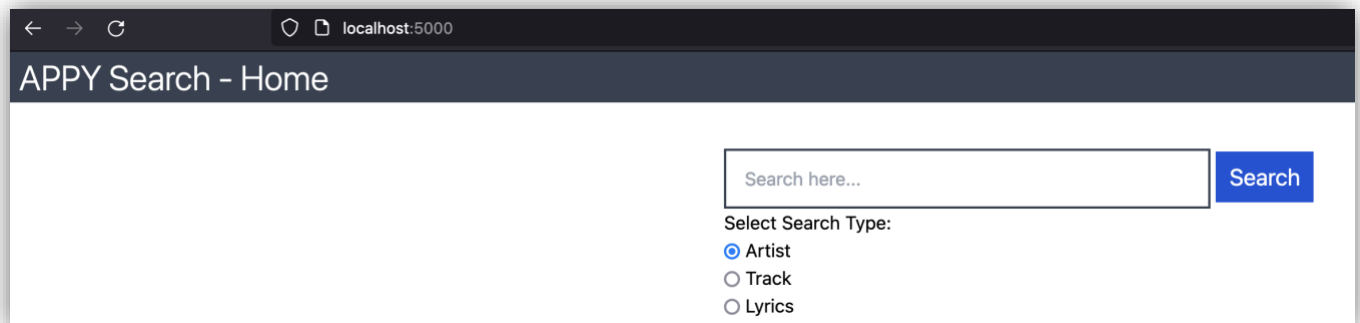
Then we initialize the App Search client, and other configurations.

```
client = AppSearch(  
    config['appsearch']['base_endpoint'],  
    http_auth=config['appsearch']['api_key'])  
  
engine_name = config['appsearch']['engine_name']  
api_url = config['mm_api']['url']  
api_key = config['mm_api']['key']  
api_format = config['mm_api']['format']
```

Creating an initial endpoint/ route for a Search bar using Flask framework.

```
@app.route("/")  
def home():  
    data = client.search(engine_name, body={  
        "query": ""  
    })  
    search_type = 'artist'  
    return render_template("index.html", data=data['results'],  
                           search_type=search_type)
```

Rendering a basic HTML page with a Search bar using Flask framework. This is done by using `<form>` element and adding 3 `<radio>` elements to select the type of search, namely - Artist, Track, Lyrics.



APPY Search - Home

Search here... Search

Select Search Type:

☒ Artist

☐ Track

☐ Lyrics

For displaying the results of the search, we are using a combination of `<div>` elements in a “Grid” display. For all the HTML styling we are using *tailwind* CSS framework. This will setup as up with our basic structure of search application.

B. Adding the Search functionality to the Framework

Initially, we don't have any documents in our App Search Index. So, we need a way using which we can keep updating our index as soon as user triggers a query.

We can add the functionality to fetch the Music data from the external Music API. for this, we will use "MusixMatch" open-source API for this purpose. This API provides a wide variety of API methods to fetch the Music related data, of which we will be using "track.search" which has inbuilt string search for Artists, tracks and Lyrics.

We have defined a `read()` function, to fetch the data from the API and save it in a JSON format. Along with that it will also, update our App Search Index, where each record will be used as one document and indexed automatically.

```
def read(query, search_type):
    matcher = "track.search"
    if search_type == 'artist':
        matcher_string = f'&q_artist={query}'
        rating_criteria = f'&s_{search_type}_rating'

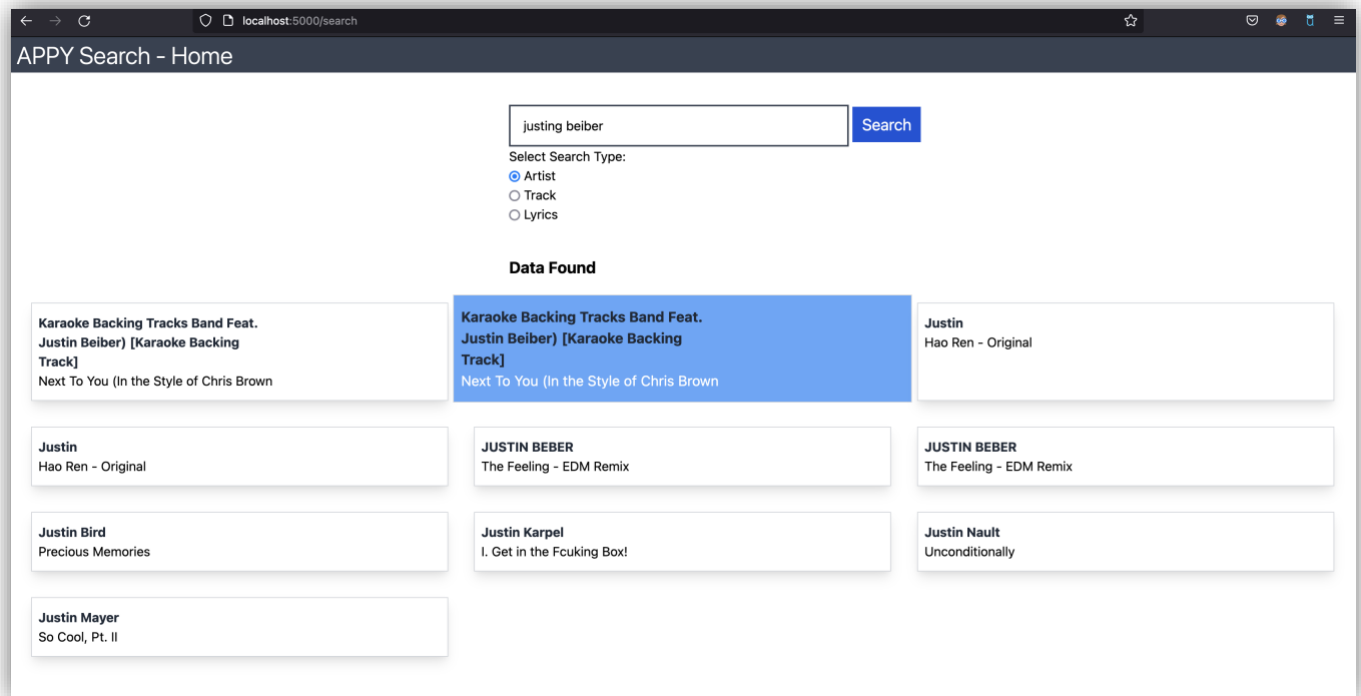
    elif search_type == 'track':
        matcher_string = f'&q_track={query}'
        rating_criteria = f'&s_{search_type}_rating'
    else:
        matcher_string = f'&q_lyrics={query}'
        rating_criteria = f'&s_track_rating'

    api_call = api_url + matcher + api_format + matcher_string + rating_criteria + api_key
    request = requests.get(api_call)
    data = request.json()
    data = [item['track'] for item in data['message']['body']['track_list']]
    #print(data)
    if data:
        with open('data.json', 'w') as outfile:
            json.dump(data, outfile)
        response = client.index_documents(engine_name, data)
        time.sleep(0.1)
        return response
    else:
        return []
```

And with that we are ready with our very own Music Search engine using Flask. Let's run it and see the results.

```
if __name__ == "__main__":
    app.run(debug=False)
```

Let's search for some tracks by Justin Bieber, we will only use "justin beiber" in lower caps as our query string. And there we go!!! We have our very own quick and simple Search application in just few steps.



7. CONCLUSION

The capabilities and usage of App Search are vast and diverse. Any organization or individual can create creative search solutions using it which can have a technological advantage over any existing solutions. The *relevance tuning* and *curation* functionalities do not require any programming expertise, compared to the other search solutions, where the developers need to adjust the algorithms to achieve the best relevance for the queries. This is a niche add-on which really brings the control of data retrieval configuration directly to the developers with minimal efforts.

8. REFERENCES

- <https://www.elastic.co/enterprise-search>
- <https://www.elastic.co/app-search/>
- <https://www.elastic.co/guide/en/enterprise-search-clients/python/7.15/app-search-api.html>
- <https://developer.musixmatch.com/documentation>
- <https://flask.palletsprojects.com/en/2.0.x/>
- All the code snapshots used are developed and written from scratch by the author.