Modern perspective of Search Engines Systems Empowering Enterprise Applications

Alejandro Pimentel (ap41)
CS 410: University of Illinois at Urbana-Champaign
November 5th, 2020

Introduction.

Information Retrieval concepts have been maturing for many years mainly in the academic context. We can go back to notable milestones like the publication of the article *"As We May Think"* by *Vannevar Bush* in 1945[1]. There, the author plays with the idea of using computers to search relevant information in documents.

In the 1960s many key developments were made in the field of *IR*, to name a few: *Cranfield Evaluations* done by *Cyril Cleverdon* which are evaluation methods for *IR* systems used even today; the *SMART* system created by Gerald Salton which allowed researchers to experiment and improve search concepts and ideas[1].

The 1970s and 80s had important developments as well, most notably, the TREC[2] conferences provided large sets of data to test various *IR* technics. The resulting algorithms from TREC were the first used to search the *World Wide Web*.

With the massive growth of data availability that came with the modern Web, the *IR* concepts and ideas from previous decades paved the way for the surge of modern Search Engines and Data Analytics Systems. These modern systems are what we will be analyzing in this article. Concretely, this article focuses on *ElasticSearch* as a case of study to dissect topics like:

- Underlying *IR* concepts used.
- Architecture considerations and components.
- Enabling an Application Ecosystem.
- Prototypical use case in the enterprise.

Case of study: *ElasticSearch*

ElasticSearch underlying *IR* concepts.

At its core, *ElasticSearch* is powered by *Apache Lucene*. Implemented in Java as a light library with no dependencies, *Lucene* is used by *ElasticSearch* to index documents and make them searchable.

For ranking documents uses a scoring function called *Practical Scoring Function*. This similarity function is based on *Term Frequency* and Inverse *Document Frequency* in the *Vector Space Model*[3]. Below is a general description of the function:

```
relevance(q,d) =
            queryNorm(q)
          * coord(q,d)
          * SUM (
                tf(t in d),
                idf(t)²,
                t.getBoost(),
                norm(t,d)
                        ) (t in q)
```

- relevance(q,d) the relevance score of document d for query q.

- queryNorm(q) query normalization factor.
- coord(q,d) coordination factor.
- SUM() is the sum of the weights for each term t in the query q for document d.
    - tf(t in d) is the term frequency for term t in document d.
    - idf(t) is the inverse document frequency for term t.
    - t.getBoost() is the boost that has been applied to the query.
    - norm(t,d) is the field-length normalization.

Covering each of the weights used in the function is beyond the scope of this article, but we can easily observe elements that conform most modern search engines such as: *TF-IDF* and special case of ***DLN based on fields*** (sections of a document built of 2 parts: name and value).

Another important element of *Apache Lucene* is the inverted index used to store all the terms' details. This data structure and its variations are at the core of many modern search engines. The inverted index stores key information used in the ranking documents process, to name a couple:

- Term Frequency in a document.
- Positions where the terms appear in a document.

Since *ElasticSearch* is a distributed search engine, it splits the inverted index into shards. These shards can affect how relevance is calculated, but in general, *ElasticSearch* via *Lucene* tries to minimize that side effect of having a distributed inverted index.

All in all, after a quick review of what is at the heart of *ElasticSearch* we can easily identify concepts that have been maturing in the past decades. Let's now zoom out and look at the components and ideas around a Search Engine System.

Architecture considerations and components.

To provide real life usability, a modern search engine system must cover many important technical considerations. Here we analyze some of them.

1) It should propose a *document-oriented* structure as oppose to a strict structure to store the data. To represent documents, often systems like *ElasticSearch* make use of a flexible representational format like *JSON*.

2) *Full-text search* is the core concept of any modern Search Engine. In the case of *ElasticSearch*, users can search any text in the whole corpus that conforms the distributed inverted index. *Lucene* is what enables this full-text search capability.

3) An API that allows access to other tools and programming languages is key to enable a mature ecosystem around a modern Search Engine. In the case of *ElasticSearch* this is provided by a very rich *REST* (Representational State Transfer) *API*.

4) Horizontal scalability is paramount for use cases where the amount of data handled tends to be large. Adding nodes in the search engine cluster must be transparent and require a minimum real time configuration if any. *ElasticSearch* can be deployed as a **containerized** cluster orchestrated by technologies like *Kubernetes*, in such case, auto scaling nodes should be transparent as the load in the system increases.

5) *Near-real time (NRT)* data availability. As soon as new data is produced, it should be indexed and made available for search in a matter of a few seconds.

6) Failure detection and correction. If nodes in the cluster become unavailable, other nodes should be assigned to cover the specific tasks. In the case of *ElasticSearch*, if a data node is lost, other data nodes should use their replicated data to cover the gap.

To fulfill all those requirements, cluster nodes should cover different responsibilities. Typically, nodes in a Search Engine System can be divided as described below:

- Data nodes. Nodes that execute search and aggregation tasks over data.
- Client nodes. Nodes that expose an API to take requests to different parts in the cluster. Requests can be administrative or for specific search tasks.
- Master nodes. Cluster configuration tasks are orchestrated by the master nodes. Scaling up and down the cluster is a clear example of a task.
- Data processing nodes. As more data is being generated, it should be pre-process in a real time stream before adding it to the inverted index.

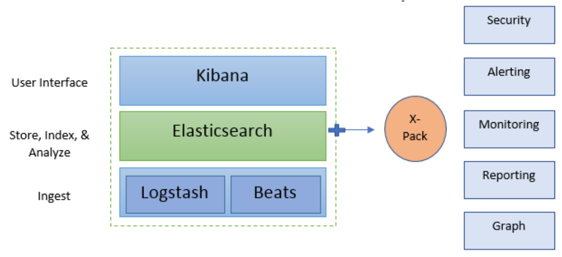
So far, we mentioned what is at the core of a modern Search Engine System and some architecture considerations to be overtaken. But, how can a system become widely used and empower new applications that ultimately help users to achieve different kind of tasks?

Enabling an Application Ecosystem.

Depending on the use we want to make of a Search Engine System, users should be able to plug-in different components to facilitate particular use cases. Such components can be developed by third parties or users can develop their own.

Of course, providing a rich *API* is the principal factor that enables the existence of these applications, that conform an ecosystem of solutions for the user. Those applications share similarities across different Search Engine Systems, let us look at some notable examples in the Enterprise Software world.

The diagram[4] below shows the basic components of the *ElasticSearch* ecosystem called *Elastic Stack*.



Kibana. It is a visualization tool where the user can create graphs, dashboards and export reports. As other visualization tools it aggregates data selected by the user to form a visualization.

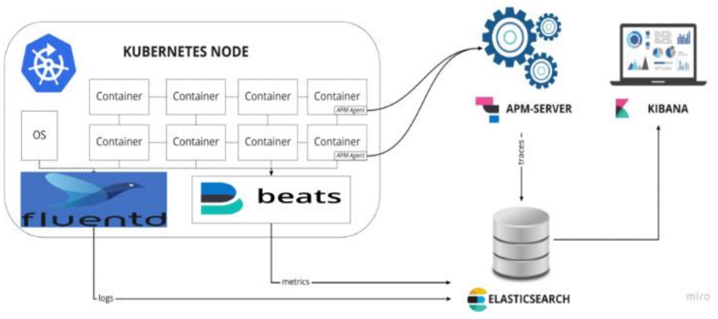
Logstash. It processes streams of data coming from different sources. *Logstash* consists of a plugin-oriented architecture. This allows the community to create ad-hoc processing pipelines that can be reused and shared.

Beats. While Logstash is a server-side component, *Beats* is installed on the client side to transfer data from the source. Depending on the data's nature there are different implementations of Beats, namely: *AudioBeat*, *FileBeat* and *MetricBeat*.

X-Pack. This is the non-free (commercial) part of the *Elastic Stack*. It adds various benefits, from security to alerting and monitoring. Most notably, it includes a *Machine Learning* module that specializes in *Anomaly Detection in Time Series Data*[5].

Given all these components, how a typical use case would look like? We often talk about Search Engine in the scope of the Web content, but, what about environments like enterprise applications in Telecom or other industries?

Prototypical use case in the enterprise.



More and more, software applications in the enterprise are required to respond to massive demand and load. One revolutionary change in last years has been the transition to *Microservices*, the idea is simply, instead of having a monolith including all the functionality, it is broken down into self-sufficient and independently deployed *microservices* that interact with each other to achieve the same goal. These *microservices* are deployed in containers, depending on the load, these containers are automatically replicated to the order of hundreds or even thousands.

Among other benefits, this *mesh* of *microservices* is much easier to horizontally scale, nevertheless, environments like these have a different set of challenges. Namely, how to monitor metrics and search for errors in the massive number of logs produced by hundreds of containers. This is one classic use case that modern *Search Engine Systems* attempt to resolve.

In the case of *ElasticSearch*, we can see in the image above [6], some complementary components like *fluentd*, *beats* and *APM Server* collect data from a *mesh* of containers and feed it to the *ElasticSearch* engine. Once the data is processed and indexed, users can search and analyze logs or other metrics from a Visualization Tool like *Kibana*.

Of course, solutions like these often don't stop at simply search for data produced by the applications, components like *X-Pack* can be leveraged to run analytics or even predict potential application outages.

Conclusion.

Search Engines as a derivation of the *Text Information* and *Information Retrieval* concepts, have been evolving into very complex systems aiming to fulfil a wide variety of requirements in the broad IT and research fields.

Often, we tend to see Search Engines embed in the more popular context of Information Retrieval for the Web, but the applications in the most unexpected fields are quite relevant. Furthermore, these systems are being enriched by the *Machine Learning* revolution. It is very notable how far we have come from the groundbreaking achievements in the past decades to a plethora of systems dedicated to solving specific Information Retrieval needs.

References and resources.

[1] Singhal, Amit (2001). "Modern Information Retrieval: A Brief Overview"

[2] https://trec.nist.gov/overview.html

[3] Scoring function in *Lucene*. https://lucene.apache.org/core/4_0_0/core/org/apache/lucene/search/package-summary.html#scoring

[4] *Elastic Stack* 7.0 by Pranav Shukla, Sharath Kumar

[5] Effective Approaches for Time Series Anomaly Detection by Aditya Bhattacharya.

https://towardsdatascience.com/effective-approaches-for-time-series-anomaly-detection-9485b40077f1

[6] EFK 7.4.0 Stack on Kubernetes by Jaspreet Singh

https://blog.opstree.com/category/devops/elasticsearch/elasticsearch-cluster/