FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Advanced Database Systems

NI-PDB

Elasticsearch Overview

Author:
Tomáš PATRO

January 7, 2021

Abstract

In today's world of software development, it is often required to implement some form of fast search in our applications. The requirement is connected to customers and business stakeholders' need to query complex data in a fast, scalable, and reliable manner. This is the case where Elasticsearch technology can help us to solve this problem. The engine offers us a way to index our documents in a schema-free way and parse full-text and other types of data to search them in near real-time. This article looks at the basic concepts of Elasticsearch, its query DSL, and presents the Elastic Stack (ELK).

Keywords – Elasticsearch, search, real-time, REST, API, Kibana, Logstash, Beats, index, shard, document, cluster, node, query, DSL

Contents

Introduction

Nowadays, software engineers often have to work with large amounts of data. Traditional relational databases are often an excellent choice to store the data, make transformations over them, and query them. The problem starts to be obvious once there is a requirement to implement a fast and reliable search over a large amount of data. We might come to a point where the traditional databases would start to struggle to meet our expectations on a fast data search.

In this article, we will look at the *Elasticsearch* technology and *Elastic Stack (ELK)*. Elasticsearch seems to be good at solving the problem we described above. It takes advantage of its powerful engine, enabling us to search over the documents in near real-time. We will also look at the query DSL we use to work with the engine and ELK, which is a set of tools designed to work with Elasticsearch.

1 What is Elasticsearch?

You can find a variety of answers on the internet. Many senior software developers may find it difficult to provide a common answer. It is a result of the variety of use cases where we can implement and use this tool:

- Storing and analyzing logs, metrics, security events, and other types of data.
- Searching in the app using search box.
- Using machine learning to automatically model the behavior of the data in real-time.
- Automating business workflows using Elasticsearch as a storage engine.
- Using Elastichsearch for spatial information (Geographic Information System [GIS]).
- Storing and processing genetic data in the bioinformatics field. [1]

The authors of Elasticsearch characterize it as the distributed search and analytics engine. The engine is built over the *Apache Lucene* [2], a Java-based search and indexing library. The engine is implemented in a way that provides near real-time search and analytics for various types of data. In the following section, we will look at the main concepts and characteristics of this tool. We take most of the information from the official *Elasticsearch Reference* [1]. If the information is from another source, we put there an extra citation.

The main characteristic of Elasticsearch is the way it stores the data. The storage itself is schema-free. It means that we do not define how the data should be structured inside the tables or similar data structures. The data are stored in the *JSON* documents. It is up to an application itself to define how it handles and stores the data inside the engine.

When we store a document, the engine indexes it, and it is fully searchable in near real-time – within 1 second. The engine uses a data structure called an *inverted index* to store text fields which enables it to search the data so fast. An inverted index finds every unique word which appears throughout the storage and identifies all of the documents each word occurs in. We can see how it works in Figure 1. The numeric and geo fields are stored in BKD trees. The ability to use the per-field data structures to assemble and return search results is what makes Elasticsearch so fast.

We can look at the data from different levels of abstraction. In the Elasticsearch, there are more or less these levels of abstractions / concepts when we talk about data and structure:

1. fields
2. documents
3. mapping
4. mapping types (legacy)
5. index
6. shards, replicas, nodes, cluster
7. analyzers [4]

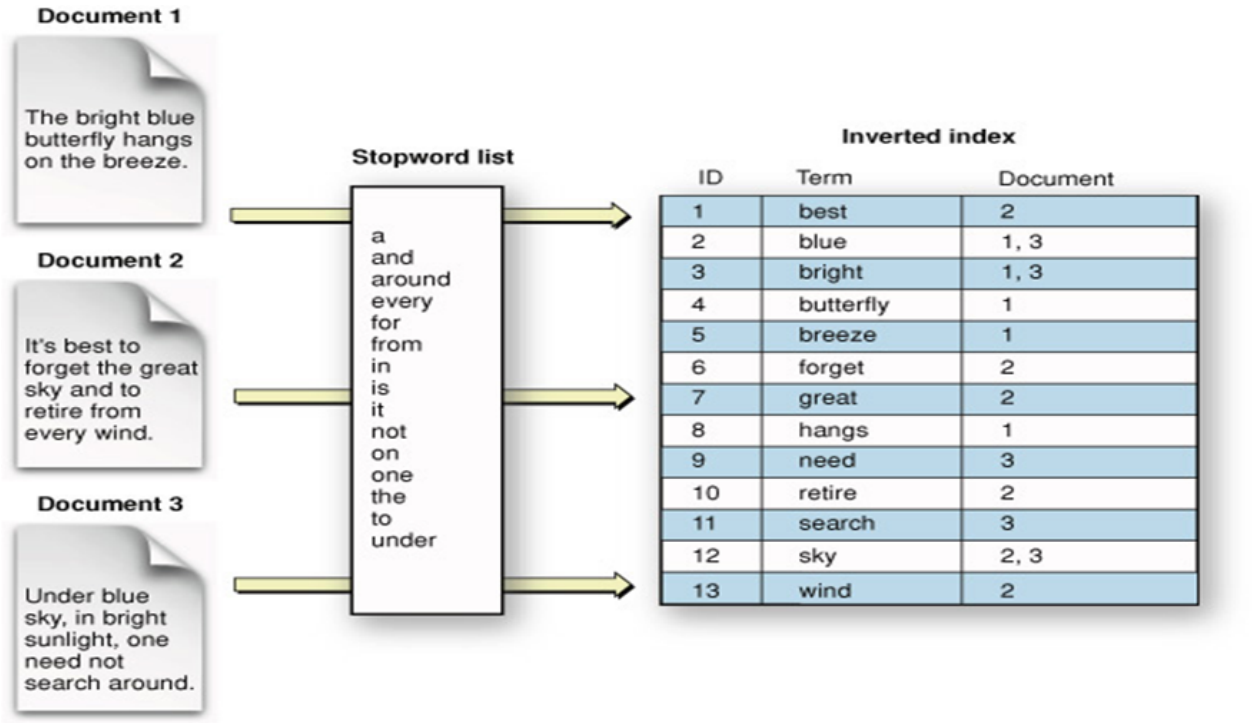

Figure 1: Inverted index in Elasticsearch. [3]

1.1 Fields

Fields are the smallest units of data in Elasticsearch. A field is simply a key inside the JSON document, which has a corresponding value. The value can be of various *field data types*. There are numerous data types in Elasticsearch. From simple `boolean` or `integer` to spatial data types like `geo_point`. If we do not define a mapping for the fields, the data types are derived from the JSON document itself. We will talk about this more in Section 1.3.

A single field can have multiple data types. This is a concept of *multi-fields* in Elasticsearch. For example, a `string` field could be mapped as a `text` field for full-text search, and as a `keyword` field for sorting and aggregations.

1.2 Documents

Documents are considered as the base unit of the Elasticsearch storage. A document is simply a JSON object which is stored within an Elasticsearch index. We can compare the document to a single row inside the relational database table. The document consists of user-defined fields. Some reserved fields are used as the document metadata. For example, the `_id` field is automatically generated sequential identifier of a document. The metadata fields start with the underscore. A simple document could look like this:

```
{
  "_index" : "bank",
  "_type" : "_doc",
  "_id" : "1",
```

```
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "account_number" : 1,
    "balance" : 39225,
    "firstname" : "Amber",
    "lastname" : "Duke",
    "age" : 32,
    "gender" : "M",
    "address" : "880 Holmes Lane",
    "employer" : "Pyrami",
    "email" : "amberduke@pyrami.com",
    "city" : "Brogan",
    "state" : "IL"
  }
}
```

Code 1: A simple Elasticsearch document.

1.3 Mapping

Mapping is a process of defining the data types of the user-defined fields within an index. There are two types of mappings in Elasticsearch:

1. dynamic mapping
2. explicit mapping

Dynamic mapping is an automatic process of data types definition. The engine automatically detects and assigns the data types to the new fields. It uses default rules for this process, which can be configured by the users to precisely meet their expectations. For example, we could customize the way the engine detects date formats based on our preferences.

Explicit mapping is a process where the user explicitly defines the data types of the particular fields within an index. For example, we could define that an email should be of the `keyword` type since we do not need to use full-text search for the emails:

```
{
  "mappings": {
    "properties": {
      "email":  { "type": "keyword"  }
  }
}
```

Code 2: An example of the explicit mapping.

1.4 Mapping Types

The mapping types were used to represent a type of indexed document. They were removed in version 7 of Elasticsearch. For this reason, we will not cover them in this article. If you want to learn more, please, refer to the *Removal of mapping types* article [5].

1.5 Index

The index is considered the largest unit of data in Elasticsearch. It is a logical partition of documents and can be compared to a database or schema in the world of relational databases. The index can separate different types of documents and apply a specific configuration for the optimized collection of documents within an index. You can define as many indexes as you want. The indexes hold documents that are unique to each index.

1.6 Shards, Replicas, Nodes & Cluster

There is no limit to how many documents you can store in one *index*. A common issue with Elasticsearch is when your index becomes big enough, so it starts to exceed your machine's physical storage. This is a moment when we use a *shard*.

There is an option to partition an index or multiple indexes into the shards. We call it horizontal partitioning. The concept is illustrated in Figure 2.

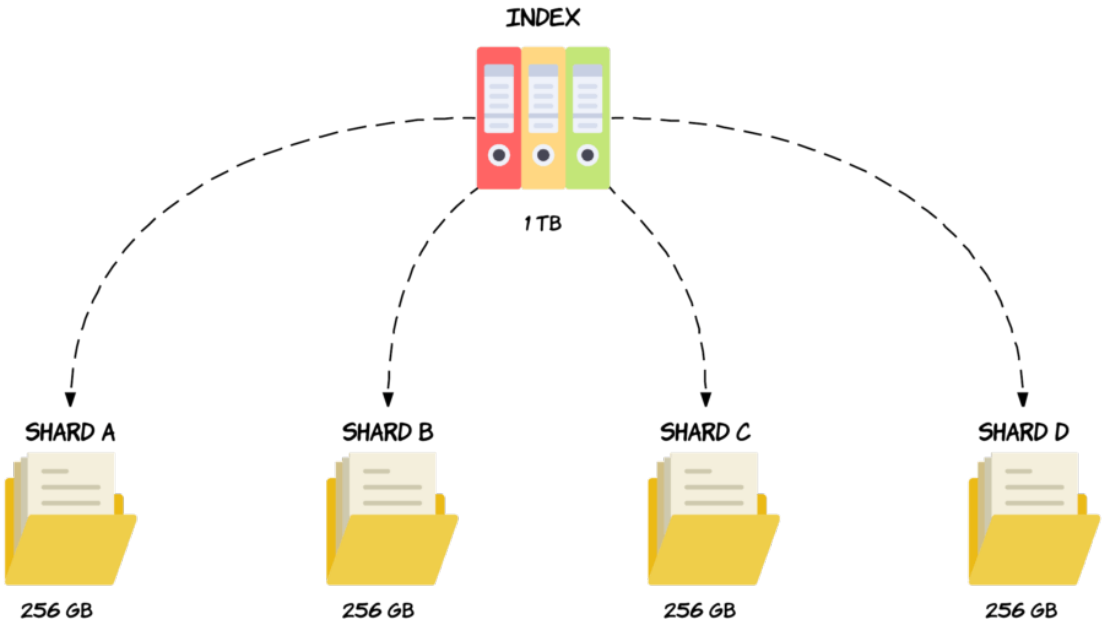

Figure 2: Index split into multiple shards. [6]

Another logical partition is something called *node*. Node is practically an instance of the Elasticsearch process. A collection of nodes is called *cluster*. The partitioning is illustrated in Figure 3.

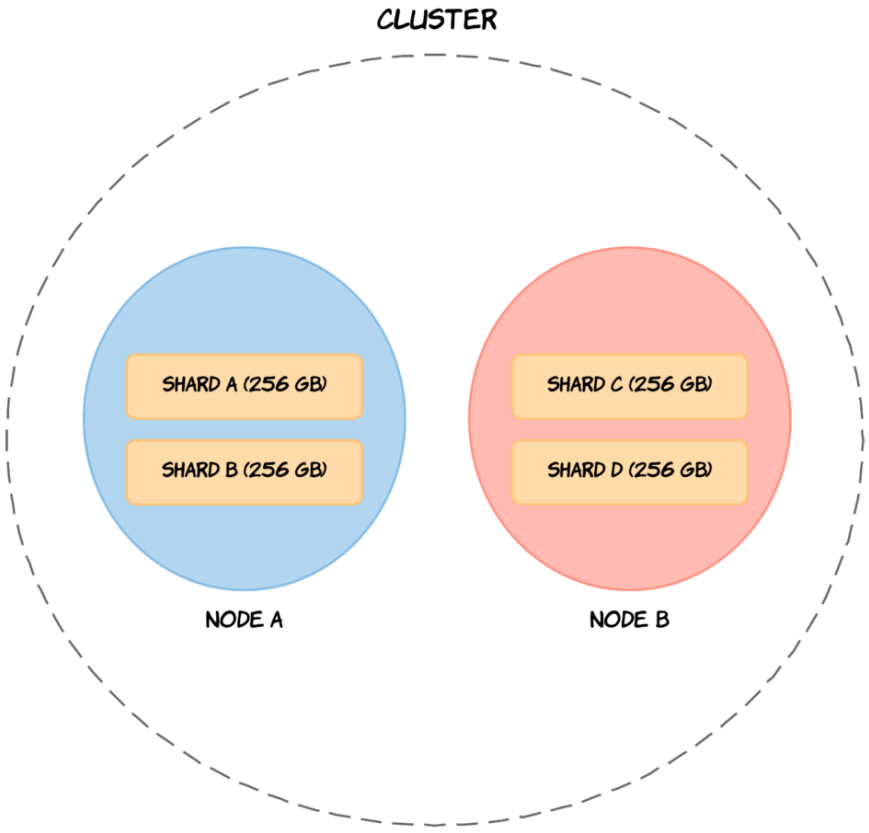

Figure 3: Cluster split into multiple nodes. [6]

One of the benefits of the nodes is the ability to distribute your system across multiple machines or physical storages. This is tightly connected with something called *replica*. Replicas are copies of your index's shards. To ensure high availability, a replica is not stored on the same node as an original shard. This mechanism enables us to implement fail-safe mechanisms to prevent data loss in case of a failure.

All nodes know about all the other nodes in the cluster and can forward client requests to the appropriate node. There are multiple types of nodes in the engine which have different roles – data node, machine learning node…

1.7 Analyzers

Analyzers are not directly connected to the structure of the Elasticsearch but are very important. They are utilities used during the indexing process to parse the full-text based on some grammar or tokenizer. By default, Elasticsearch applies *Standard Analyzer* which is able to parse a common English text. The advantage is that you can define your analyzer or use some other already defined analyzers, e.g., if you tried to parse a text in the Klingon language.

2 Query DSL

Elasticsearch provides us with a simple (HTTP) REST API to manage our cluster, index, and search our data. It is a Domain Specific Language (DSL) we use to query the engine and work with it. To start using it, you can take advantage of any tool for sending HTTP requests, e.g., *cURL* tool. Let's look at some simple examples of the API.

To add (index) some data, we simply run a `PUT` request with the payload we need:

```
PUT /languages/_doc/1
{
    "name": "Python"
}
```

Code 3: An example of indexing some data.

The request creates the index `languages` if it doesn't already exist. The `_doc` is an important part of the path. In combination with the `PUT` method, we tell Elasticsearch that we want to create a new document. To query the document, we use the same path and `GET` method. We get the following response:

```
GET /languages/_doc/1
{
  "_index" : "languages",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 10,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "name": "Python"
  }
}
```

Code 4: The result of querying the data.

If we want to search the `languages` index and sort the results, for example, by the name, we can use a simple query for it:

```
GET /languages/_search
{
  "query": { "match_all": {} },
  "sort": [
    { "name": "asc" }
  ]
}
```

Code 5: Searching and sorting the data.

This will return the languages sorted by their name. By default, the engine returns only the first 10 documents that match the search criteria. We can adjust this behavior by adding

additional parameters to the query. You can note that we define the request body in the `GET` method. Many of you could argue that this goes against the REST principles, but it is the way the DSL is defined.

If you want to learn more about the DSL, please refer to the *Query DSL* [7] section of the official Elasticsearch documentation.

3 Elastic Stack (ELK)

Elastic Stack (ELK) is a set of tools that are often used to enhance the work with Elasticsearch. In addition to the engine itself, you can find here the tools like Kibana, Logstash, and Beats.

Kibana is an open-source analytics and visualization platform designed to work with Elasticsearch. We can use it to search, index, view, and generally interact with our data stored in Elasticsearch indexes. It enables us to visualize data, generate reports with a variety of charts, tables, maps…

Logstash is a powerful data parsing and transformation tool. It is a recommended addition to Elasticsearch if you want to do complex and advanced transformations over your data.

Beats are open source data shippers that you install as autonomous agents/services on your OS. They send operational data to Elasticsearch. Each Beat can be installed separately. We can use them, for example, to collect the system metrics (CPU, RAM…) and send them to Elasticsearch. We can also use them in combination with Logstash to parse or transform the data before sending them to the engine.

To learn more about ELK, please refer to the *Getting started with the Elastic Stack* [8] section of the official Elasticsearch documentation.

Conclusion

Elasticsearch and *ELK* are powerful tools that enable us to use near real-time full-text search over the documents. The main characteristic is the schema-free way of how the engine stores the data. It gives us the freedom to manipulate large sets of heterogeneous data. It also obligates us to handle the data manipulation on the application level properly.

In the first section, we described the main structure and principles of the Elasticsearch engine. We showed different levels of abstractions from the particular fields of the document, through the index, up to the cluster itself composed of different nodes, shards, indexes…

In the second section, we made a brief preview of the basic syntax of the query DSL. We described that the engine can be queried through the specific REST API and showed how we can take advantage of the HTTP protocol, methods, and DSL to work with the engine.

In the last section, we looked at the Elastic Stack (ELK). ELK is a set of tools that are designed to work smoothly with Elasticsearch. We described *Kibana* (analytics and visualization platform), *Logstash* (parsing and transformation tool), and *Beats* (set of data shippers).

This article aimed to show the basic concepts, characteristics, and tools connected with Elasticsearch. It should serve you as a kick-off and inspiration for your next project where you would think about implementing a fast, near real-time search.

References

1. ELASTICSEARCH B.V. *Elasticsearch Reference* [online]. © 2020 [visited on 2020-12-27]. Available from: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html.
2. THE APACHE SOFTWARE FOUNDATION. *Apache Lucene* [online]. © 2011–2020 [visited on 2020-12-27]. Available from: https://lucene.apache.org/.
3. PRYIOMKA, Alex. *Elasticsearch index sharding explanation* [online]. © 2017 [visited on 2020-12-27]. Available from: https://stackoverflow.com/questions/47003336/elasticsearch-index-sharding-explanation.
4. BERMAN, Daniel. *10 Elasticsearch Concepts You Need to Learn* [online]. © 2019 [visited on 2020-12-27]. Available from: https://logz.io/blog/10-elasticsearch-concepts/.
5. ELASTICSEARCH B.V. *Removal of mapping types* [online]. © 2020 [visited on 2020-12-27]. Available from: https://www.elastic.co/guide/en/elasticsearch/reference/current/removal-of-types.html.
6. ANDERSEN, Bo. *Understanding Sharding in Elasticsearch* [online]. © 2017 [visited on 2021-01-06]. Available from: https://codingexplained.com/coding/elasticsearch/understanding-sharding-in-elasticsearch.
7. ELASTICSEARCH B.V. *Query DSL* [online]. © 2020 [visited on 2021-01-06]. Available from: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html.
8. ELASTICSEARCH B.V. *Getting started with the Elastic Stack* [online]. © 2020 [visited on 2021-01-06]. Available from: https://www.elastic.co/guide/en/elastic-stack-get-started/7.10/get-started-elastic-stack.html.