Elastic Search fundamentals

Overview

Elastic Search(ES) is a distributive full-text search and analytics engine. The magic that allows the efficient search of ES is called inverted index. This documentation aims at helping members of our team to get familiar with Elastic Search.

Table of Contents

1. **Basic Concepts**:
    - Index
    - Mapping Type
    - Shards & Replicas
    - REST API
2. **Installation**
3. **Entry point of ES in Python**
4. **Indices API**
    1. **Create indices**
        - ***Index-level Settings***
        - ***Mapping types***

Basic Concepts:

We first introduce several basic concepts so that we know what we are talking about.

- *Index*

  Index is the single most important concept of ES when used as a noun. An index is a collection of documents that has similar characteristics where a document is a basic unit of information. The verb form of **Index** means the action of including a document in an index.

  For example, you can create an index of legal contracts. Each legal contract in the index has a client name, creation date and contract content. The following table represents such a sample index containing two documents.

Client Name	Contract Created Date	Document Content
David	2019-01-01	This is a mock contract
John	2019-01-02	This is a mock contract 2

- *Mapping Type*

  The **mapping type** of an index defines how ES processes a document and integrates it as a part of the index. *Fields* and *parameters*(which will be made clear in an example later) in the mapping type directly affect how the index and search process will be performed.

  Every field of an index has a `type` which is either inferred dynamically from the documents or explicitly specified. For example, we can specify the data type of the field `Contract Created Date` as `date` so that ES knows that it is a date field. Aside from `type`, there are many parameters for a field that one can set.

- *Shards & Replicas*

  The distributive nature of ES originates from how ES read documents into an index and perform search on the index. To be brief, ES stores an index across several nodes and each piece is called a **shard**. ES also saves several copies of every shard and each copy is called a **replica**.

  If you are interested, you can also read this.

- *REST API*

  A RESTful API is an API that complies with the REST architecture standards for web services. For ES, the REST APIs specify how an ES client can talk to ES to do indexing and searching. Many programming languages have their own packages built above REST APIs

of ES to provide communication with ES. For example, the package we are going to use in the project is the Python Elasticsearch Client.

The following REST APIs for ES will be used most frequently:

1. Indices API is used to form an index and index documents into an index.
2. Search API is used to search in an index.

Installation

We refer the installation process of ES to readers which is stated quite clearly in the Installation section in the documentation and the Elastic Search start kit by Zhikun Cui.

After installation of ES, we have to install the Python Elasticsearch Client. The instruction is here.

Entry point of ES in Python

We will be using Python scipt to talk to ES. The package we are using is Python Elasticsearch Client. Instead of having to use HTTP calls to talk to ES, the Python Elasticsearch Client provides a very handy `elasticsearch.Elasticsearch` class to talk to ES using Python script.

For example, we can easily initialize such an instance `es` using a list of nodes. All indexing and searching will be performed through this instance.

```
from elasticsearch import Elasticsearch
# hosts is a list of nodes. If you have >=1 nodes, the list should contains >=1 elements
# host can also be specifed by URL or other format and can have more options, for example, SSL
# see also https://elasticsearch-py.readthedocs.io/en/master/api.html#elasticsearch.Elasticsearch
es = Elasticsearch(hosts=[{"host":"localhost", "port":9200}])
```

Indices API

Indices API provides interfaces to create, update, delete and manage an index. Let's have a look at it.

*1. **Create indices***

An index can be created by directly starting to feed documents or created manually. In the first situation we just start indexing documents and let ES dynamically infer the mapping type of the index. We take the second approach here because that will illustrate how we can specify the mapping type of an index.

We create the index `legal_contract` in the following code. Don't worry, we are going to explain it.

```
# setting and mapping type of the index
requestbody = {
    "settings":{  # index-level setting
        "number_of_shards" : 3,
        "number_of_replicas" : 2
    },
    "mappings": {  # mapping type
        "contract": {   # name of the mapping type
                        # may contain metafields
            "properties": {   # properties of fields of the index
                "ClientName": {    # field name
                    "type": "keyword",  # type of the field
                },
                "ContractCreatedDate": {   # field name
                    "type": "date",        # type of the field
                    "format": "yyyy-MM-dd" # format of the date field
                },
                "DocumentContent": {   # field name
                    "type": "text",    # type of the field
                    "term_vector": "with_positions_offsets"   # more optional parameters of the field
                }
            }
        }
    }
}
# create the legal_contract index
es.indices.create(index='legal_contract',body=requestbody)
```

All requests can be communicated with ES using a Query language based on JSON. That is the syntax supported by REST APIs of ES. An example of such request is the `requestbody` above. There are two parts: `settings` and `mappings`.

Index-level Settings

`settings` specifies the **index-level setting** of the index.

In the above example, `number_of_shards` and `number_of_replicas` specify the number of shards and replicas of `legal_contract`. There are more index-level settings that can be specified. See this part of the documentation.

Mapping types

The `mappings` specifies the **mapping type** of the index. Let's dive deeper into the **mapping type** by telling you what does every part mean in the `mappings` above.

1. Name of the mapping type of legal_contract is `contract`, which is specified in `mappings`. It can be any customized name you want it to be.

2. The mapping type `contract` can contain zero or several `meta_field`s that tells ES how to deal with metadata of the document created by ES. In the above example, there is no `meta_field` in `contract`. For more about `meta_field`s, see metafields.

3. The mapping type `contract` contains a `properties` which specifies all data fields and their properties. For example, `ClientName`, `ContractCreatedDate` and `DocumentContent` are data fields of `legal_contract`.

4. Every data field has a `type` which specifies their data type. The `type` of a data field tells the data type and has an effect on how ES treats a data field. For example, `keyword` asks ES to match the exact string when search is performed on `ClientName` . The `text` asks ES to do a full text search when search is performed on `DocumentContent`. Similarly, `date` asks ES to consider `ContractCreatedDate` as a date field and there are several things ES can do to such a data type. For more about data type, you can find it here.

5. Aside from `type`, more optinal parameters can be specified on properties of a data field. For example, the parameter `term_vector` is set to be `with_positions_offsets` for the data field `DocumentContent`, which tells ES to store position of terms(words) and character offsets of the terms appearing in the `DocumentContent` when a document is indexed. This will allow us to identify the position of a particular word when we are searching for it. Different data types have different optional parameters associated to it. For more information, see here.