Mini Labs: ElasticSearch

Contents

Mini Labs: ElasticSearch 1

Create Index 3

Create Index 3

Response 3

Add data 3

Request Body 3

Response 3

Response 4

Document APIs 5

Index API 5

Automatic Index Creation 5

Versioning 6

Internal Versioning 6

External Versioning 6

Operation Type 7

Automatic ID generation 7

Get API 8

Delete API 10

Update API 10

Search APIs 11

Multi-Index 11

URI Search 12

Request Body Search 13

Index APIs 14

Create Index 14

Delete Index 15

Get Index 15

Index Exist 16

Index Settings 16

Index Stats 17

Flush 17
Mapping 18
Field Data Types 18
    Core Data Types 18
    Complex Data Types 18
Removal of Mapping Types 19

Create Index

In this mini-lab, let us learn how to add some index, mapping and data to Elasticsearch. Note that some of this data will be used in the examples explained in this lab.

Create Index

You can use the following command to create an index −

```
PUT school
```

Response

If the index is created, you can see the following output −

```
{"acknowledged": true}
```

Add data

Elasticsearch will store the documents we add to the index as shown in the following code. The documents are given some IDs which are used in identifying the document.

Request Body

```
POST school/_doc/10
{
   "name":"Virginia Tech", "description":"Atlantic Coast
Conference",
   "street":"Dawarka", “city”:”Blacksburg”, “state”:”Virginia”,
"zip":"33312",
   "location":[28.5733056, 77.0122136], "fees":5000,
   "tags":["Good Faculty", "Great Sports"], "rating":"4.5"
}
```

Response

```
{
   "_index" : "school",
   "_type" : "_doc",
   "_id" : "10",
   "_version" : 1,
   "result" : "created",
   "_shards" : {
      "total" : 2,
      "successful" : 1,
      "failed" : 0
   },
   "_seq_no" : 2,
   "_primary_term" : 1
}
```

Here, we are adding another similar document.

```
POST school/_doc/16
{
   "name":"University of Virginia", "description":"Atlantic Coast
Conference",
   "street":"Tonk Road",
   "city":"Charlottsville", “state”:”Virginia”,
"zip":"17614","location":[26.8535922,75.7923988],
   "fees":2500, "tags":["Well equipped labs"], "rating":"4.4"
}
```

Response

```
{
   "_index" : "school",
   "_type" : "_doc",
   "_id" : "16",
   "_version" : 1,
   "result" : "created",
   "_shards" : {
      "total" : 2,
      "successful" : 1,
      "failed" : 0
   },
   "_seq_no" : 9,
   "_primary_term" : 7
}
```

In this way, we will keep adding any example data that we need for our working in the upcoming mini-labs.

Elasticsearch provides single document APIs and multi-document APIs, where the API call is targeting a single document and multiple documents respectively.

Index API

It helps to add or update the JSON document in an index when a request is made to that respective index with specific mapping. For example, the following request will add the JSON object to index schools and under school mapping −

```
PUT schools/_doc/5
{
   name":"Old Dominion University", "description":"ICSE",
"street":"West End",
   "city":"Norfolk",
   “state”:”Virginia”, "zip":"25002", "location":[28.9926174,
77.692485],
   "fees":3500,
   "tags":["fully computerized"], "rating":"4.5"
}
```

On running the above code, we get the following result −

```
{
   "_index" : "schools",
   "_type" : "_doc",
   "_id" : "5",
   "_version" : 1,
   "result" : "created",
   "_shards" : {
      "total" : 2,
      "successful" : 1,
      "failed" : 0
   },
   "_seq_no" : 2,
   "_primary_term" : 1
}
```

Automatic Index Creation

When a request is made to add JSON object to a particular index and if that index does not exist, then this API automatically creates that index and also the underlying mapping for that particular JSON object. This functionality can be disabled by changing the values of following parameters to false, which are present in elasticsearch.yml file.

```
action.auto_create_index:false
index.mapper.dynamic:false
```

You can also restrict the auto creation of index, where only index name with specific patterns are allowed by changing the value of the following parameter −

```
action.auto_create_index:+acc*,-bank*
```

Note − Here + indicates allowed and – indicates not allowed.

Versioning

Elasticsearch also provides version control facility. We can use a version query parameter to specify the version of a particular document.

```
PUT schools/_doc/5?version=7&version_type=external
{
   "name":"Nova Southeastern University", "description":"CBSE
Affiliation", "street":"Sharks",
   "city":"Fort Lauderdale", "state":"Florida", "zip":"17615",
"location":[31.8955385, 76.8380405],
   "fees":2200, "tags":["Senior Secondary", "beautiful campus"],
"rating":"3.3"
}
```

On running the above code, we get the following result −

```
{
   "_index" : "schools",
   "_type" : "_doc",
   "_id" : "5",
   "_version" : 7,
   "result" : "updated",
   "_shards" : {
      "total" : 2,
      "successful" : 1,
      "failed" : 0
   },
   "_seq_no" : 3,
   "_primary_term" : 1
}
```

Versioning is a real-time process and it is not affected by the real time search operations.

There are two most important types of versioning −

Internal Versioning

Internal versioning is the default version that starts with 1 and increments with each update, deletes included.

External Versioning

It is used when the versioning of the documents is stored in an external system like third party versioning systems. To enable this functionality, we need to set version_type to external. Here Elasticsearch will store version number as designated by the external system and will not increment them automatically.

Operation Type

The operation type is used to force a create operation. This helps to avoid the overwriting of existing document.

```
PUT mini-lab/_doc/1?op_type=create
{
   "Text":"this is mini-lab one"
}
```

On running the above code, we get the following result −

```
{
   "_index" : "mini-lab",
   "_type" : "_doc",
   "_id" : "1",
   "_version" : 1,
   "result" : "created",
   "_shards" : {
      "total" : 2,
      "successful" : 1,
      "failed" : 0
   },
   "_seq_no" : 0,
   "_primary_term" : 1
}
```

Automatic ID generation

When ID is not specified in index operation, then Elasticsearch automatically generates id for that document.

```
POST mini-lab/_doc/
{
   "user" : "tpoint",
   "post_date" : "2018-12-25T14:12:12",
   "message" : "Elasticsearch Lab"
}
```

On running the above code, we get the following result −

```
{
   "_index" : "mini-lab",
   "_type" : "_doc",
   "_id" : "PVghWGoB7LiDTeV6LSGu",
   "_version" : 1,
   "result" : "created",
   "_shards" : {
      "total" : 2,
      "successful" : 1,
      "failed" : 0
   },
   "_seq_no" : 1,
   "_primary_term" : 1
}
```

Get API

API helps to extract type JSON object by performing a get request for a particular document.

```
pre class="prettyprint notranslate" > GET schools/_doc/5
```

On running the above code, we get the following result −

```
{
   "_index" : "schools",
   "_type" : "_doc",
   "_id" : "5",
   "_version" : 7,
   "_seq_no" : 3,
   "_primary_term" : 1,
   "found" : true,
   "_source" : {
      "name" : "Nova Southeastern University",
      "description" : "CBSE Affiliation",
      "street" : "Sharks",
      "city" : "Fort Lauderdale",
      "state" : "HP",
      "zip" : "176115",
      "location" : [
         31.8955385,
         76.8380405
      ],
      "fees" : 2200,
      "tags" : [
         "Senior Secondary",
         "beautiful campus"
      ],
      "rating" : "3.3"
   }
}
```

- This operation is real time and does not get affected by the refresh rate of Index.
- You can also specify the version, then Elasticsearch will fetch that version of document only.
- You can also specify the _all in the request, so that the Elasticsearch can search for that document id in every type and it will return the first matched document.
- You can also specify the fields you want in your result from that particular document.

```
GET schools/_doc/5?_source_includes=name,fees
```

On running the above code, we get the following result −

```
{
   "_index" : "schools",
   "_type" : "_doc",
   "_id" : "5",
   "_version" : 7,
   "_seq_no" : 3,
   "_primary_term" : 1,
   "found" : true,
   "_source" : {
      "fees" : 2200,
      "name" : "Nova Southeastern University"
   }
}
```

You can also fetch the source part in your result by just adding _source part in your get request.

```
GET schools/_doc/5?_source
```

On running the above code, we get the following result −

```
{
   "_index" : "schools",
   "_type" : "_doc",
   "_id" : "5",
   "_version" : 7,
   "_seq_no" : 3,
   "_primary_term" : 1,
   "found" : true,
   "_source" : {
      "name" : "Nova Southeastern University",
      "description" : "CBSE Affiliation",
      "street" : "Sharks",
      "city" : "Fort Lauderdale",
      "state" : "HP",
      "zip" : "176115",
      "location" : [
         31.8955385,
         76.8380405
      ],
      "fees" : 2200,
      "tags" : [
         "Senior Secondary",
         "beautiful campus"
      ],
      "rating" : "3.3"
   }
}
```

You can also refresh the shard before doing get operation by set refresh parameter to true.

Delete API

You can delete a particular index, mapping or a document by sending a HTTP DELETE request to Elasticsearch.

```
DELETE schools/_doc/4
```

On running the above code, we get the following result −

```
{
   "found":true, "_index":"schools", "_type":"school", "_id":"4",
"_version":2,
   "_shards":{"total":2, "successful":1, "failed":0}
}
```

Version of the document can be specified to delete that particular version. Routing parameter can be specified to delete the document from a particular user and the operation fails if the document does not belong to that particular user. In this operation, you can specify refresh and timeout option same like GET API.

Update API

Script is used for performing this operation and versioning is used to make sure that no updates have happened during the get and re-index. For example, you can update the fees of school using script −

```
POST schools/_update/4
{
   "script" : {
      "source": "ctx._source.name = params.sname",
      "lang": "painless",
      "params" : {
         "sname" : "City Wise School"
      }
   }
 }
```

On running the above code, we get the following result −

```
{
   "_index" : "schools",
   "_type" : "_doc",
   "_id" : "4",
   "_version" : 3,
   "result" : "updated",
   "_shards" : {
      "total" : 2,
      "successful" : 1,
      "failed" : 0
   },
   "_seq_no" : 4,
   "_primary_term" : 2
}
```

You can check the update by sending get request to the updated document.

Search APIs

This API is used to search content in Elasticsearch. A user can search by sending a get request with query string as a parameter or they can post a query in the message body of post request. Mainly all the search APIS are multi-index, multi-type.

Multi-Index

Elasticsearch allows us to search for the documents present in all the indices or in some specific indices. For example, if we need to search all the documents with a name that contains central, we can do as shown here −

```
GET /_all/_search?q=city:Norfolk
```

On running the above code, we get the following response −

```
{
   "took" : 33,
   "timed_out" : false,
   "_shards" : {
      "total" : 7,
      "successful" : 7,
      "skipped" : 0,
      "failed" : 0
   },
   "hits" : {
      "total" : {
         "value" : 1,
         "relation" : "eq"
      },
      "max_score" : 0.9808292,
      "hits" : [
         {
            "_index" : "schools",
            "_type" : "school",
            "_id" : "5",
            "_score" : 0.9808292,
            "_source" : {
               "name" : "Nova Southeastern University",
               "description" : "CBSE Affiliation",
               "street" : "Sharks",
               "city" : "Fort Lauderdale",
               "state" : "HP",
               "zip" : "176115",
               "location" : [
                  31.8955385,
                  76.8380405
               ],
               "fees" : 2200,
               "tags" : [
                  "Senior Secondary",
                  "beautiful campus"
               ],
               "rating" : "3.3"
            }
         }
      ]
   }
}
```

URI Search

Many parameters can be passed in a search operation using Uniform Resource Identifier –

S.No	Parameter & Description
1	**Q** This parameter is used to specify query string.
2	**lenient** This parameter is used to specify query string.Format based errors can be ignored by just setting this parameter to true. It is false by default.
3	**fields** This parameter is used to specify query string.
4	**sort** We can get sorted result by using this parameter, the possible values for this parameter is fieldName, fieldName:asc/fieldname:desc

5	**timeout** We can restrict the search time by using this parameter and response only contains the hits in that specified time. By default, there is no timeout.
6	**terminate_after** We can restrict the response to a specified number of documents for each shard, upon reaching which the query will terminate early. By default, there is no terminate_after.
7	**from** The starting from index of the hits to return. Defaults to 0.
8	**size** It denotes the number of hits to return. Defaults to 10.

Request Body Search

We can also specify query using query DSL in request body and there are many examples already given in previous mini-labs. One such example is given here −

```
POST /schools/_search
{
   "query":{
      "query_string":{
         "query":"up"
      }
   }
}
```

On running the above code, we get the following response −

```
{
   "took" : 11,
   "timed_out" : false,
   "_shards" : {
      "total" : 1,
      "successful" : 1,
      "skipped" : 0,
      "failed" : 0
   },
   "hits" : {
      "total" : {
         "value" : 1,
         "relation" : "eq"
      },
      "max_score" : 0.47000363,
      "hits" : [
         {
            "_index" : "schools",
            "_type" : "school",
            "_id" : "4",
            "_score" : 0.47000363,
            "_source" : {
               "name" : "City Best School",
               "description" : "ICSE",
               "street" : "West End",
               "city" : "Norfolk",
               "state" : "UP",
               "zip" : "250002",
               "location" : [
                  28.9926174,
                  77.692485
               ],
               "fees" : 3500,
               "tags" : [
                  "fully computerized"
               ],
               "rating" : "4.5"
            }
         }
      ]
   }
}
```

Index APIs

These APIs are responsible for managing all the aspects of the index like settings, aliases, mappings, index templates.

Create Index

This API helps you to create an index. An index can be created automatically when a user is passing JSON objects to any index or it can be created before that. To create an index, you just need to send a PUT request with settings, mappings and aliases or just a simple request without body.

```
PUT colleges
```

On running the above code, we get the output as shown below −

```
{
   "acknowledged" : true,
   "shards_acknowledged" : true,
   "index" : "colleges"
}
```

We can also add some settings to the above command −

```
PUT colleges
{
  "settings" : {
      "index" : {
         "number_of_shards" : 3,
         "number_of_replicas" : 2
      }
   }
}
```

On running the above code, we get the output as shown below −

```
{
   "acknowledged" : true,
   "shards_acknowledged" : true,
   "index" : "colleges"
}
```

Delete Index

This API helps you to delete any index. You just need to pass a delete request with the name of that particular Index.

```
DELETE /colleges
```

You can delete all indices by just using _all or *.

Get Index

This API can be called by just sending get request to one or more than one indices. This returns the information about index.

```
GET colleges
```

On running the above code, we get the output as shown below −

```
{
   "colleges" : {
      "aliases" : {
         "alias_1" : { },
         "alias_2" : {
            "filter" : {
               "term" : {
                  "user" : "pkay"
               }
            },
            "index_routing" : "pkay",
            "search_routing" : "pkay"
         }
      },
      "mappings" : { },
      "settings" : {
         "index" : {
            "creation_date" : "1556245406616",
            "number_of_shards" : "1",
            "number_of_replicas" : "1",
            "uuid" : "3ExJbdl2R1qDLssIkwDAug",
            "version" : {
               "created" : "7000099"
            },
            "provided_name" : "colleges"
         }
      }
   }
}
```

You can get the information of all the indices by using _all or *.

Index Exist

Existence of an index can be determined by just sending a get request to that index. If the HTTP response is 200, it exists; if it is 404, it does not exist.

```
HEAD colleges
```

On running the above code, we get the output as shown below −

```
200-OK
```

Index Settings

You can get the index settings by just appending _settings keyword at the end of URL.

```
GET /colleges/_settings
```

On running the above code, we get the output as shown below −

```
{
   "colleges" : {
      "settings" : {
         "index" : {
            "creation_date" : "1556245406616",
            "number_of_shards" : "1",
            "number_of_replicas" : "1",
            "uuid" : "3ExJbdl2R1qDLssIkwDAug",
            "version" : {
               "created" : "7000099"
            },
            "provided_name" : "colleges"
         }
      }
   }
}
```

Index Stats

This API helps you to extract statistics about a particular index. You just need to send a get request with the index URL and _stats keyword at the end.

```
GET /_stats
```

On running the above code, we get the output as shown below −

```
………………………………………………
},
   "request_cache" : {
      "memory_size_in_bytes" : 849,
      "evictions" : 0,
      "hit_count" : 1171,
      "miss_count" : 4
   },
   "recovery" : {
      "current_as_source" : 0,
      "current_as_target" : 0,
      "throttle_time_in_millis" : 0
   }
} ………………………………………………
```

Flush

The flush process of an index makes sure that any data that is currently only persisted in the transaction log is also permanently persisted in Lucene. This reduces recovery times as that data does not need to be reindexed from the transaction logs after the Lucene indexed is opened.

```
POST colleges/_flush
```

On running the above code, we get the output as shown below −

```
{
   "_shards" : {
      "total" : 2,
      "successful" : 1,
      "failed" : 0
   }
}
```

Mapping

Mapping is the outline of the documents stored in an index. It defines the data type like geo_point or string and format of the fields present in the documents and rules to control the mapping of dynamically added fields.

```
PUT bankaccountdetails
{
   "mappings":{
      "properties":{
         "name": { "type":"text"}, "date":{ "type":"date"},
         "balance":{ "type":"double"}, "liability":{
"type":"double"}
      }
   }
 }
```

When we run the above code, we get the response as shown below −

```
{
   "acknowledged" : true,
   "shards_acknowledged" : true,
   "index" : "bankaccountdetails"
}
```

Field Data Types

Elasticsearch supports a number of different datatypes for the fields in a document. The data types used to store fields in Elasticsearch are discussed in detail here.

Core Data Types

These are the basic data types such as text, keyword, date, long, double, boolean or ip, which are supported by almost all the systems.

Complex Data Types

These data types are a combination of core data types. These include array, JSON object and nested data type. An example of nested data type is shown below &minus

```
POST /tabletennis/_doc/1
{
   "group" : "players",
   "user" : [
      {
         "first" : "dave", "last" : "jones"
      },
      {
         "first" : "kevin", "last" : "morris"
      }
   ]
}
```

When we run the above code, we get the response as shown below −

```
{
   "_index" : "tabletennis",
   "_type" : "_doc",
   "_id" : "1",
   _version" : 2,
   "result" : "updated",
   "_shards" : {
      "total" : 2,
      "successful" : 1,
      "failed" : 0
   },
   "_seq_no" : 1,
   "_primary_term" : 1
}
```

Another sample code is shown below −

```
POST /accountdetails/_doc/1
{
   "from_acc":"7056443341", "to_acc":"7032460534",
   "date":"11/1/2016", "amount":10000
}
```

When we run the above code, we get the response as shown below −

```
{   "_index" : "accountdetails",
   "_type" : "_doc",
   "_id" : "1",
   "_version" : 1,
   "result" : "created",
   "_shards" : {
      "total" : 2,
      "successful" : 1,
      "failed" : 0
   },
   "_seq_no" : 1,
   "_primary_term" : 1
}
```

We can check the above document by using the following command −

```
GET /accountdetails/_mappings?include_type_name=false
```

Removal of Mapping Types

Indices created in Elasticsearch 7.0.0 or later no longer accept a _default_ mapping. Indices created in 6.x will continue to function as before in Elasticsearch 6.x. Types are deprecated in APIs in 7.0.