Elasticsearch 데이터 처리

Elasticsearch는 데이터 저장 형식으로 **json** 도큐먼트를 사용합니다. 데이터 저장 형식 뿐 아니라 쿼리와 클러스터 설정 등 모든 정보를 json 형태로 주고받기 때문에 elasticsearch 의 사용을 위해서는 json 사용에 익숙해져야 합니다.

REST API

CRUD - 입력, 조회, 수정, 삭제

Elasticsearch에서는 단일 도큐먼트별로 고유한 URL을 갖습니다. 7 버전부터 도큐먼트에 접근하는 URL은 `http://<호스트>:<포트>/<인덱스>/_doc/<도큐먼트 id>` 구조이다. 이전 버전까지는 도큐먼트 타입 개념이 있었지만, 현재는 _doc 고정자로 접근해야 한다. 다음은 curl 도구를 이용해서 my_index 인덱스에 도큐먼트 id가 1인 데이터를 입력하는 예제이다.

```
$ curl -XPUT "http://localhost:9200/my_index/_doc/1" -H 'Content-Type: ap
plication/json' -d'
{
  "name": "Jongmin Kim",
  "message": "안녕하세요 Elasticsearch"
}'
{"_index":"my_index","_type":"_doc","_id":"1","_version":1,"result":"create
d","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":0,"_primary_ter
m":1}
```

이후부터는 elasticsearch의 REST 명령들은 Kibana의 Dev Tools 에서 입력하는 형식으로 설명하도록 하겠다. 입력은 `request` 탭, 그리고 응답은 `response` 탭에 표기하도록 하겠다.

입력(PUT)

데이터 입력을 할 때는 **PUT** 메서드를 이용한다. 다음은 Kibana 에서 my_index 인덱스에 도큐먼트 id가 1인 데이터를 입력하는 예제이다.

request

```
PUT my_index/_doc/1
{
```

```
  "name":"Suhwan Lee",
  "message":"안녕하세요 Elasticsearch"
}
```

response

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

처음으로 도큐먼트를 입력하면 결과에 `"result" : "created"` 로 표시가 된다. 동일한 URL에 다른 내용의 도큐먼트를 다시 입력하게 되면 기존 도큐먼트는 삭제되고 새로운 도큐먼트로 덮어씌워지게 된다. 이 때는 결과에 `created`가 아닌 `updated`가 표시된다.

동일안 도큐먼트id 로 서로 다른 내용을 PUT 명령으로 넣게 되면, 그 전 도큐먼트는 다 사라지게 되고 그 위에 새로운 도큐먼트가 덮어씌워지게 된다.

request

```
PUT my_index/_doc/1
{
  "name":"Suhwan Lee",
  "message":"안녕하세요 Kibana"
}
```

response

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

실수로 기존 도큐먼트가 덮어씌워지는 것을 방지하기 위해서는 입력 명령에 _doc 대신 **_create** 를 사용해서 새로운 도큐먼트의 입력만 허용하는 것이 가능합니다. 입력하려는 도큐먼트 id에 이미 데이터가 있는 경우 아래와 같이 입력 오류가 나게 된다.

request

```
PUT my_index/_create/1
{
  "name":"Suhwan Lee",
  "message":"안녕하세요 Elasticsearch"
}
```

response

```
{
  "error": {
    "root_cause": [
      {
        "type": "version_conflict_engine_exception",
        "reason": "[1]: version conflict, document already exists (current versi
on [2])",
```

```
        "index_uuid": "qYOJI9ELR2-HqVtgTeI9jw",
        "shard": "0",
        "index": "my_index"
      }
    ],
    "type": "version_conflict_engine_exception",
    "reason": "[1]: version conflict, document already exists (current version
[2])",
    "index_uuid": "qYOJI9ELR2-HqVtgTeI9jw",
    "shard": "0",
    "index": "my_index"
  },
  "status": 409
}
```

조회(GET)

GET 메서드로 가져올 도큐먼트의 URL을 입력하면 도큐먼트의 내용을 가져온다. 다양한 정보가 함께 표시되며 문서의 내용은 **_source** 항목에 나타난다.

requset

```
GET my_index/_doc/1
```

response

```
{
  "_index" : "my_index",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 2,
  "_seq_no" : 1,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "name" : "Suhwan Lee",
    "message" : "안녕하세요 Elasticsearch"
```

```
  }
}
```

삭제(DELETE)

DELETE 메서드를 이용해서 도큐먼트 또는 인덱스 단위의 삭제가 가능하다. 두 경우에 차이가 있는데 먼저 `DELETE my_index/_doc/1` 명령으로 하나의 도큐먼트를 삭제하면 다음과 같이 도큐먼트가 삭제되었다는 `"result" : "deleted"` 결과가 리턴된다.

request

```
DELETE my_index/_doc/1
```

response

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

도큐먼트는 삭제되었지만 인덱스는 남아있는 경우 삭제된 도큐먼트를 GET 해서 가져오려고 하면 아래와 같이 my_index/_doc/1 도큐먼트를 못 찾았다는 `"found" : false` 응답을 받는다. 인덱스는 있으나 입력되지 않은 조회할 때도 마찬가지다.

request

```
GET my_index/_doc/1
```

response

```
{
  "_index" : "my_index",
  "_type" : "_doc",
  "_id" : "1",
  "found" : false
}
```

이제 `DELETE my_index` 으로 전체 인덱스를 삭제하면 다음과 같이 `"acknowledged" : true` 응답만 리턴된다.

request

```
DELETE my_index
```

response

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

삭제된 인덱스 또는 처음부터 없는 인덱스의 도큐먼트를 조회하려고 하면 도큐먼트를 못 찾았다는 `"found" : false` 응답이 아니라 다음과 같이 `"type" : "index_not_found_exception"` , `"status" : 404` 오류가 리턴된다.

request

```
GET my_index/_doc/1
```

response

```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "index_not_found_exception",
        "reason" : "no such index [my_index]",
```

```
      "resource.type" : "index_expression",
      "resource.id" : "my_index",
      "index_uuid" : "_na_",
      "index" : "my_index"
    }
  ],
  "type" : "index_not_found_exception",
  "reason" : "no such index [my_index]",
  "resource.type" : "index_expression",
  "resource.id" : "my_index",
  "index_uuid" : "_na_",
  "index" : "my_index"
 },
 "status" : 404
}
```

수정(POST)

POST 메서드는 PUT 메서드와 유사하게 데이터 입력에 사용이 가능하다. 도큐먼트를 입력할 때 POST 메서드로 `<인덱스>/_doc` 까지만 입력하게 되면 자동으로 임의의 도큐먼트id 가 생성된다. 도큐먼트id의 자동 생성은 PUT 메서드로는 동작하지 않는다.

도큐먼트id가 중복이 되게 되면 덮어씌워지게 될 위험이 있기 때문에, 보통은 여러 개의 데이터들을 겹치지 않게 저장할 때는 POST 명령으로 도큐먼트id 를 자동으로 생성되게 하는 것이 좋을 수도 있다.

request

```
POST my_index/_doc
{
 "name":"Suhwan Lee",
 "message":"안녕하세요 Elasticsearch"
}
```

response

```
{
  "_index" : "my_index",
  "_type" : "_doc",
  "_id" : "ZuFv12wBspWtEG13dOut",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

_update

입력된 도큐먼트를 수정하기 위해서는 기존 도큐먼트의 URL에 변경될 내용의 도큐먼트 내용을 다시 PUT 하는 것으로 대치가 가능하다. 하지만 필드가 여럿 있는 도큐먼트에서 필드 하나만 바꾸기 위해 전체 도큐먼트 내용을 매번 다시 입력하는 것은 번거로운 작업일 것이다. 이 때는 `POST <인덱스>/_update/<도큐먼트 id>` 명령을 이용해 원하는 필드의 내용만 업데이트가 가능하다. 업데이트 할 내용에 "doc" 이라는 지정자를 사용한다.

_update API를 이용해서 `my_index/_doc/1` 도큐먼트의 "message" 필드 값을 *"안녕하세요 Kibana"* 로 업데이트를 한 뒤 도큐먼트 내용을 확인 해 보겠다. `my_index/_doc/1` 도큐먼트를 삭제 하였다면 위의 입력 (PUT) 내용을 참고해서 새로 입력 한 뒤 아래 명령을 실행한다.

request

```
POST my_index/_update/1
{
  "doc": {
    "message":"안녕하세요 Kibana"
  }
}
```

response

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


이제 다시 GET 명령으로 `my_index/_doc/1` 도큐먼트를 조회 해 보면 message 필드가 "안녕하세요 Kibana" 로 변경 된 것을 확인할 수 있다.

request

```
GET /my_index/_doc/1
```

response


```
{
  "_index" : "my_index",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 2,
  "_seq_no" : 1,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "name" : "Suhwan Lee",
    "message" : "안녕하세요 Kibana"
  }
}
```

위 결과를 보면 `"_version" : 2`로 버전이 증가한 것을 확인할 수 있다. _update API 를 사용해서 단일 필드만 수정하는 경우에도 실제로 내부에서는 도큐먼트 전체 내용을 가져와서 _doc에서 지정한 내용을 변경한 새 도큐먼트를 만든 뒤 전체 내용을 다시 PUT 으로 입력하는 작업을 진행한다.

bulk API

여러 명령을 배치로 수행하기 위해서 **_bulk API**의 사용이 가능하다. _bulk API로 **index, create, update, delete**의 동작이 가능하며 delete를 제외하고는 명령문과 데이터문을 한 줄씩 순서대로 입해야 한다. delete는 내용 입력이 필요 없기 때문에 명령문만 있다.

> 💡 _bulk 의 명령문과 데이터문은 반드시 한 줄 안에 입력이 되어야 하며 줄바꿈을 허용하지 않는다.

다음은 _bulk 명령을 실행한 예제이다. 각 명령의 결과가 items에 배열로 리턴된다.

request

```
POST _bulk
{"index":{"_index":"test", "_id":"1"}} # 명령
{"field":"value one"} # 데이터
{"index":{"_index":"test", "_id":"2"}}
{"field":"value two"}
{"delete":{"_index":"test", "_id":"2"}}
{"create":{"_index":"test", "_id":"3"}}
{"field":"value three"}
{"update":{"_index":"test", "_id":"1"}}
{"doc":{"field":"value two"}}
```

response

```
{
  "took" : 440,
  "errors" : false,
  "items" : [
```

```
    {
      "index" : {
        "_index" : "test",
        "_type" : "_doc",
        "_id" : "1",
        "_version" : 1,
        "result" : "created",
        "_shards" : {
          "total" : 2,
          "successful" : 1,
          "failed" : 0
        },
        "_seq_no" : 0,
        "_primary_term" : 1,
        "status" : 201
      }
    },
    {
      "index" : {
        "_index" : "test",
        "_type" : "_doc",
        "_id" : "2",
        "_version" : 1,
        "result" : "created",
        "_shards" : {
          "total" : 2,
          "successful" : 1,
          "failed" : 0
        },
        "_seq_no" : 1,
        "_primary_term" : 1,
        "status" : 201
      }
    },
...
```

위 명령이 실행하는 동작들은 다음과 같다.

모든 명령이 동일한 인덱스에서 수행되는 경우에는 아래와 같이 `<인덱스명>/_bulk` 형식으로도 사용이 가능하다.

```
POST test/_bulk
{"index":{"_id":"1"}}
{"field":"value one"}
{"index":{"_id":"2"}}
{"field":"value two"}
{"delete":{"_id":"2"}}
{"create":{"_id":"3"}}
{"field":"value three"}
{"update":{"_id":"1"}}
{"doc":{"field":"value two"}}
```

인덱스 단위로 _bulk 사용

벌크 동작은 따로따로 수행하는 것 보다 속도가 훨씬 빠르다. 특히 대량의 데이터를 입력 할 때는 반드시 _bulk API를 사용해야 불필요한 오버헤드가 없습니다. **Logstash** 와 **Beats** 그리고 Elastic 웹페이지에서 제공하는 대부분의 언어별 클라이언트에서는 데이터를 입력할 때 _bulk를 사용하도록 개발되어 있다.

Elasticsearch 에는 커밋이나 롤백 등의 트랜잭션 개념이 없다. _bulk 작업 중 연결이 끊어지거나 시스템이 다운되는 등의 이유로 동작이 중단 된 경우에는 어느 동작까지 실행되었는지 확인이 불가능하다. 보통 이런 경우 전체 인덱스를 삭제하고 처음부터 다시 하는 것이 안전하다.

따로 수행하는 거에 비해 bulk api는 많게는 10배까지도 속도 차이가 난다고 한다.

파일에 저장내용 실행

벌크 명령을 파일로 저장하고 curl 명령으로 실행시킬 수 있다. 저장한 명령 파일을 `--data-binary` 로 지정하면 저장된 파일로 부터 입력할 명령과 데이터를 읽어올 수 있다. 다음 내용을 **bulk.json** 이라는 이름의 파일로 먼저 저장 해 보겠다.

bulk.json

```
{"index":{"_index":"test","_id":"1"}}
{"field":"value one"}
{"index":{"_index":"test","_id":"2"}}
{"field":"value two"}
{"delete":{"_index":"test","_id":"2"}}
{"create":{"_index":"test","_id":"3"}}
{"field":"value three"}
{"update":{"_index":"test","_id":"1"}}
{"doc":{"field":"value two"}}
```

다음 명령으로 **bulk.json** 파일에 있는 내용들을 _bulk 명령으로 실행 가능하다. 파일 이름 앞에는 `@`문자를 입력한다.

bulk.json 파일 내용을 _bulk로 실행

```
$ curl -XPOST "http://localhost:9200/_bulk" -H 'Content-Type: application/json' --data-binary @bulk.json
```

검색 API - _search API

지금까지는 도큐먼트 단위의 입력, 수정, 삭제, 조회 하는 방법을 알아보았다. 하지만 Elasticsearch의 진가는 쿼리를 통한 검색 기능에 있다. 검색은 인덱스 단위로 이루어진다. `GET <인덱스명>/_search` 형식으로 사용하며 쿼리를 입력하지 않으면 전체 도큐먼트를 찾는 **match_all** 검색을 한다.

URI 검색

_search 뒤에 `q` 파라메터를 사용해서 검색어를 입력할 수 있다. 이렇게 요청 주소에 검색어를 넣어 검색하는 방식을 **URI 검색**이라고 한다.

앞에서 만든 test 인덱스에서 **"value"** 라는 값을 검색하기 위해서는 다음과 같이 입력한다.

request

```
GET test/_search?q=value
```

response


```
{
  "took" : 3,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 2,
      "relation" : "eq"
    },
    "max_score" : 0.105360515,
    "hits" : [
      {
        "_index" : "test",
        "_type" : "_doc",
        "_id" : "3",
        "_score" : 0.105360515,
        "_source" : {
          "field" : "value three"
        }
      },
      {
```

```
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "1",
      "_score" : 0.105360515,
      "_source" : {
        "field" : "value two"
      }
    }
  ]
 }
}
```

결과를 보면 `hits.total.value` 부분에 검색 결과 전체에 해당되는 문서의 개수가 표시되고 다시 그 안의 `hits:[ ]` 구문 안에 배열로 가장 정확도가 높은 문서 10개가 나타난다. 이 정확도를 **relevancy**(렐러번시 라고 읽는다) 라고 하며 뒤에서 다시 설명하도록 하겠다.

두 개의 검색어 **"value"** 그리고 **"three"** 를 **AND** 조건으로 검색 하려면 다음과 같이 입력한다. URI 쿼리에서는 `AND`, `OR`, `NOT` 의 사용이 가능하며 반드시 모두 대문자로 입력해야 한다.

request

```
GET test/_search?q=value AND three
```

response

```
{
 "took" : 3,
 "timed_out" : false,
 "_shards" : {
  "total" : 1,
  "successful" : 1,
  "skipped" : 0,
  "failed" : 0
 },
 "hits" : {
  "total" : {
   "value" : 1,
   "relation" : "eq"
```

```
    },
    "max_score" : 0.87546873,
    "hits" : [
      {
        "_index" : "test",
        "_type" : "_doc",
        "_id" : "3",
        "_score" : 0.87546873,
        "_source" : {
          "field" : "value three"
        }
      }
    ]
  }
}
```

value 와 **three** 를 모두 포함한 `test/_doc/3` 도큐먼트 만이 결과로 리턴되었다. 검색어 value 을 field 필드에서 찾고 싶으면 다음과 같이 `<필드명>:<검색어>` 형태로 입력한다. 검색은 항상 필드를 지정해서 하는 것이 좋다.

request

```
GET test/_search?q=field:value
```

response

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 2,
```

```
      "relation" : "eq"
    },
    "max_score" : 0.18232156,
    "hits" : [
      {
        "_index" : "test",
        "_type" : "_doc",
        "_id" : "3",
        "_score" : 0.18232156,
        "_source" : {
          "field" : "value three"
        }
      },
      {
        "_index" : "test",
        "_type" : "_doc",
        "_id" : "1",
        "_score" : 0.18232156,
        "_source" : {
          "field" : "value two"
        }
      }
    ]
  }
}
```

URI 검색은 루씬의 기본 쿼리 문법을 사용하며 손쉽게 다룰 수 있다. 또한 웹 브라우저 주소창 등에서도 사용 가능하기 때문에 빠르게 쓰긴 쉬우나 좀 더 복잡한 검색을 위해서는 다음에 설명하는 데이터 본문(data body) 검색을 이용해야 한다.

데이터 본문(Data Body) 검색

많이 사용하는 방법. 권한 때문?

데이터 본문(data body) 검색은 검색 쿼리를 데이터 본문으로 입력하는 방식이다. Elasticsearch의 QueryDSL을 사용하며 쿼리 또한 Json 형식으로 되어 있다. 처음 익힐때는 다소 복잡 해 보일 수 있으나 주로 사용하는 쿼리 몇가지들 부터 차근 차근 익혀나가면 크게 어렵지 않게 사용이 가능하다.

가장 쉽고 많이 사용되는 것은 **match** 쿼리다. 여기서는 문법만 살펴보고 다음 검색 장에서 더 많은 쿼리들에 대해 자세히 다뤄보도록 하겠다. 데이터 본문 검색으로 field 필드값이 value 인 도큐먼트를 검색하기 위해서는 다음 명령을 실행한다.

request

```
GET test/_search
{
  "query": {
    "match": {
      "field": "value"
    }
  }
}
```

response

```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 2,
      "relation" : "eq"
    },
    "max_score" : 0.105360515,
    "hits" : [
      {
        "_index" : "test",
        "_type" : "_doc",
        "_id" : "3",
        "_score" : 0.105360515,
```

```
      "_source" : {
        "field" : "value three"
      }
    },
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "1",
      "_score" : 0.105360515,
      "_source" : {
        "field" : "value two"
      }
    }
  ]
 }
}
```

쿼리 입력은 항상 `query` 지정자로 시작한다. 그 다음 레벨에서 쿼리의 종류를 지정하는데 위에서는 **match** 쿼리를 지정했다. 그 다음은 사용할 쿼리 별로 문법이 상이할 수 있는데 match 쿼리는 `<필드명>:<검색어>` 방식으로 입력한다.

멀티테넌시(Multitenancy)

Elasticsearch는 여러 개의 인덱스를 한꺼번에 묶어서 검색할 수 있는 **멀티테넌시**를 지원한다. `logs-2018-01`, `logs-2018-02` ... 와 같이 날짜별로 저장된 인덱스들이 있다면 이 인덱스들을 모두 `logs-*/_search` 명령으로 한꺼번에 검색이 가능하다. 특히 시간순으로 따라 쌓이는 로그 데이터를 다룰 때는 인덱스를 일단위 등으로 구분하는것이 좋다. 나중에 필드 구조가 변경되거나 크기가 커져서 샤드 설정을 변경하거나 할 때 더욱 용이하다.

여러 인덱스를 검색할때는 쉼표`,` 로 나열하거나 와일드카드 `*` 문자로 묶을 수 있다.

쉼표로 나열해서 여러 인덱스 검색

```
GET logs-2018-01,2018-02,2018-03/_search
```

와일드카드 * 를 이용해서 여러 인덱스 검색

```
GET logs-2018-*/_search
```

인덱스명 대신 `_all` 지정자를 사용하여 `GET _all/_search` 와 같이 실행하면 클러스터에 있는 모든 인덱스를 대상으로 검색이 가능하다. 하지만 `_all`은 시스템 사용을 위한 인덱스 같은 곳의 데이터까지 접근하여 불필요한 작업 부하를 초래하므로 `_all` 은 되도록 사용하지 않도록 한다.