2️⃣

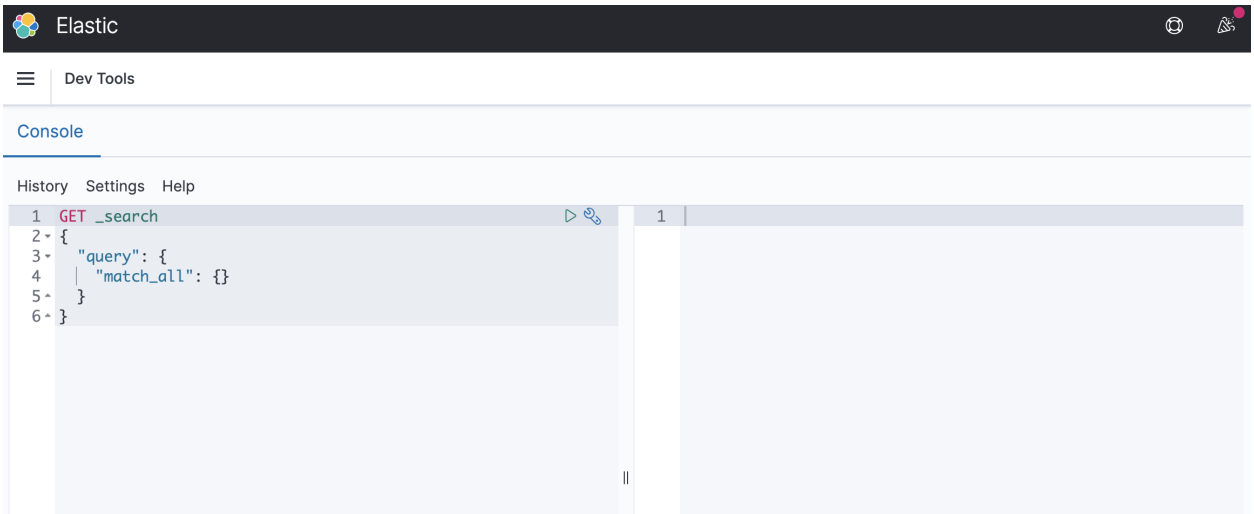
[2주차]3장 엘라스틱서치 기본

3.1 준비

3.1.1 엘라스틱서치 요청과 응답

- 모든 요청과 응답은 REST API 형태

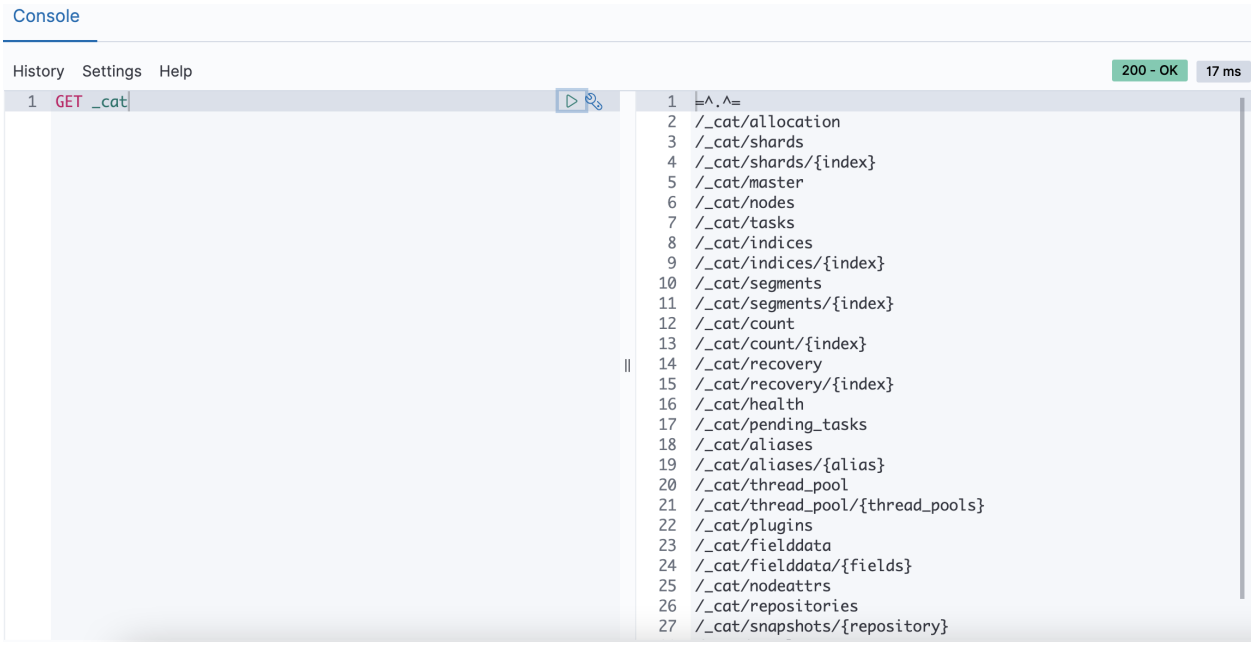
3.1.2 키바나 콘솔 사용법



Management > Dev Tools

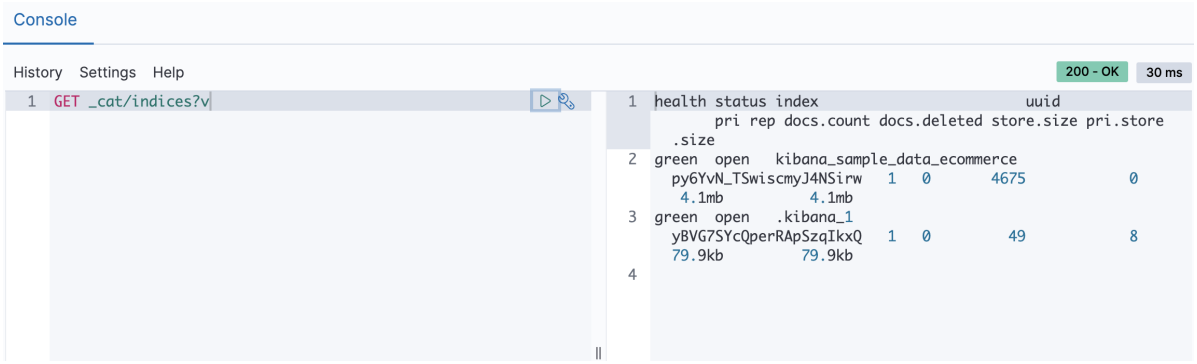
- Dev Tools 콘솔로 REST API 를 호출할 수 있다.

3.1.3 시스템 상태 확인



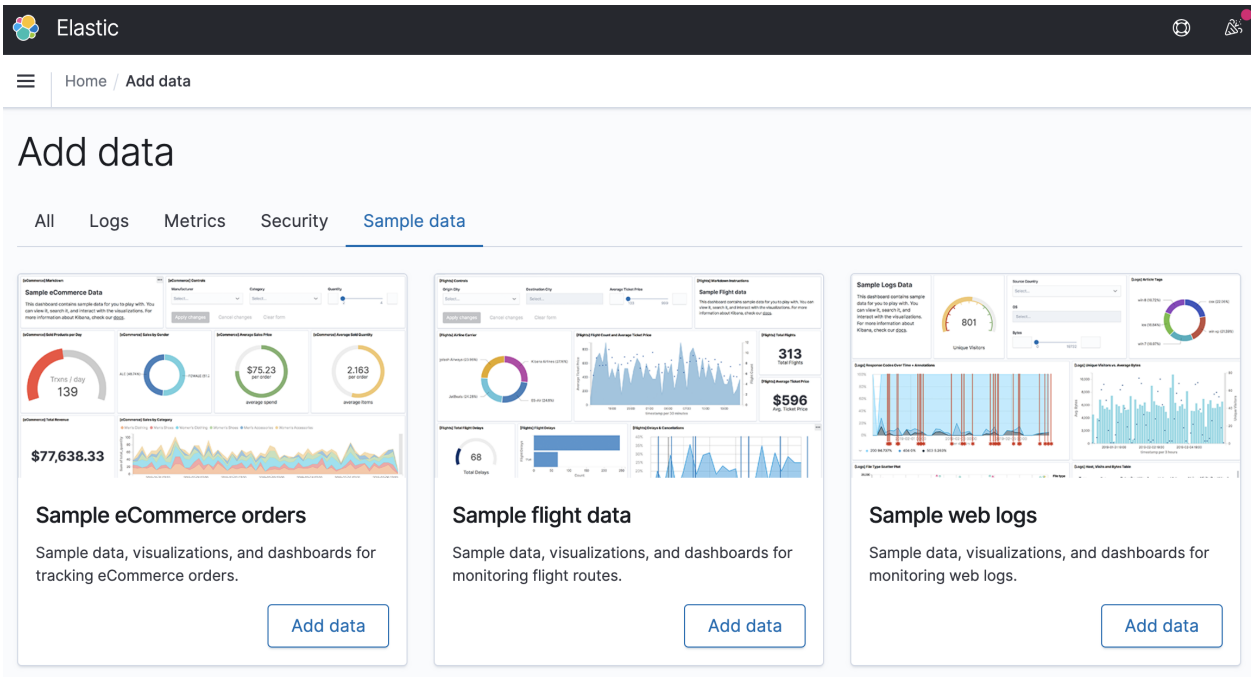
cat API가 지원하는 목록

- cat API를 통해 노드, 샤드, 템플릿 등의 상태 정보나 통계 정보를 확인할 수 있다.



클러스터 내부 인덱스 목록을 확인할 수 있다. (가독성이 좋다)

3.1.4 샘플 데이터 불러오기



3.2 인덱스와 도큐먼트

- 인덱스: 도큐먼트를 저장하는 논리적 구분자
- 도큐먼트: 실제 데이터를 저장하는 단위

3.2.1 도큐먼트

- 엘라스틱서치에서 데이터가 저장되는 기본 단위, JSON 형태
- 하나의 도큐먼트는 필드와 값을 갖는다.
- 엘라스틱서치 매핑으로 필드들의 데이터 타입을 지정할 수 있다.
- MySQL과 엘라스틱 서치 비교

MySQL	엘라스틱서치
테이블	인덱스
레코드	도큐먼트
컬럼	필드
스키마	매핑

  - 7.x 버젼에서는 타입이 사라지면서

`데이터베이스 → 인덱스`, `테이블 → 타입` 의 비교가 아니라

`테이블 → 인덱스` 비교가 맞다.

3.2.2 인덱스

- 도큐먼트를 저장하는 논리적 단위
- 관계형 데이터베이스의 테이블과 유사 개념이다.
- 하나의 인덱스에 다수의 도큐먼트가 포함되는 구조

< 스키마에 따른 그룹명 >

- 스키마에 따라 인덱스를 구분하는 것은 기본적이고 필수적인 사항이다.

< 관리 목적의 그룹핑 >

- 기본적으로 인덱스는 무한대의 도큐먼트를 포함할 수 있지만, 검색 성능을 위해 인덱스 용량에 제한을 두고 특정 용량을 넘어가면 인덱스를 분리한다.
  - 날짜/시간 단위로 인덱스를 분리하면 특정 날짜의 데이터를 쉽게 처리할 수 있다.
- 빅데이터 시스템에서 주로 사용
  - 데이터가 커서 디스크 용량이 부족해지거나
  - 시스템 가용성 차원에서 데이터 정제 작업을 많음

  ⇒효율적으로 처리하기 위해 관리 목적으로 그룹핑

3.3 도큐먼트 CRUD

3.3.1 인덱스 생성/확인/삭제

- 생성/수정: PUT(POST) index1

```
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "index1"
}
```

```
{
  "error" : "Incorrect HTTP method for uri [/index2?pretty
    =true] and method [POST], allowed: [GET, PUT, HEAD,
    DELETE]",
  "status" : 405
}
```


- 확인: GET index1

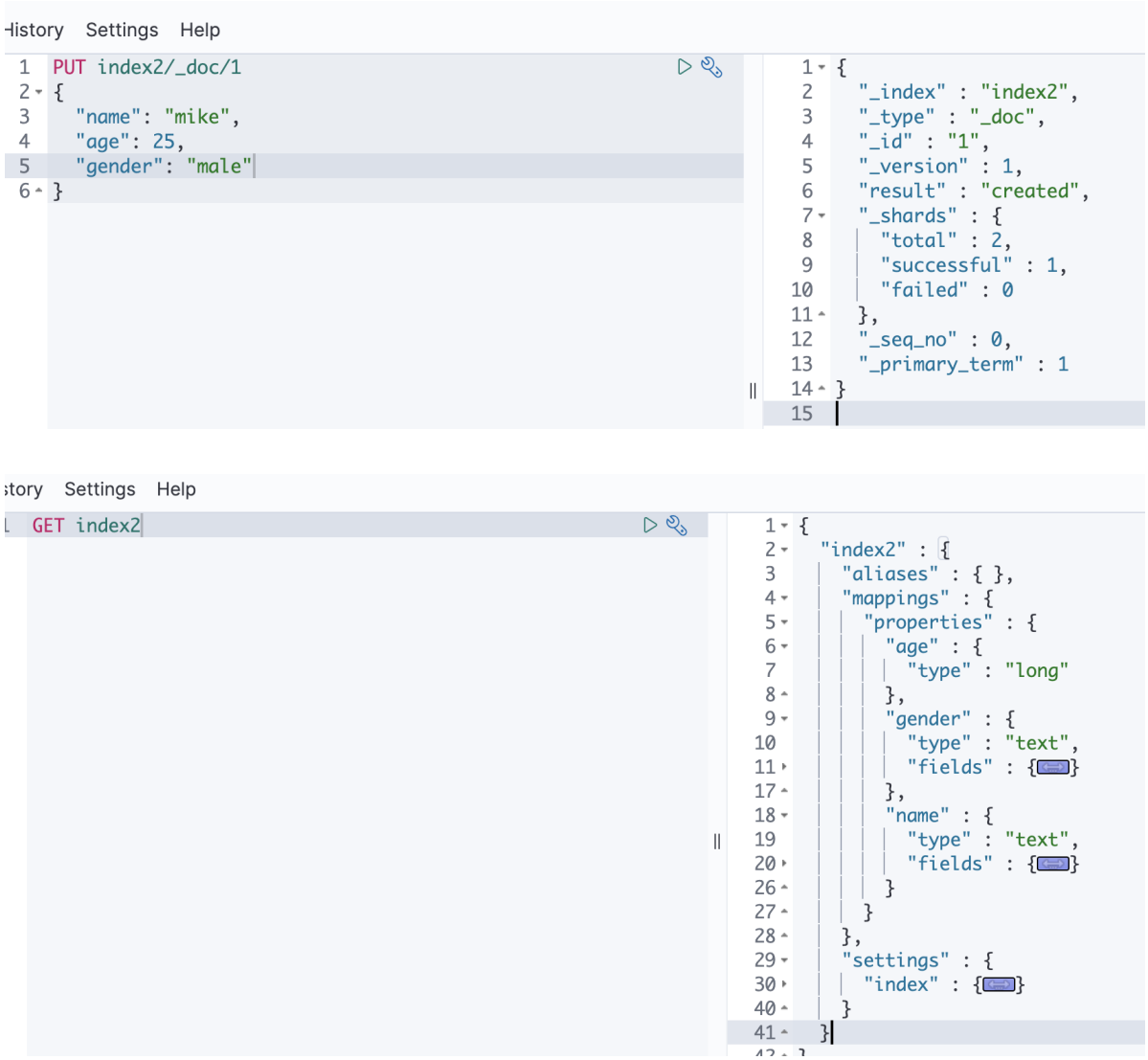

```
{
  "index1" : {
    "aliases" : { },
    "mappings" : { },
    "settings" : {
      "index" : {
        "creation_date" : "1645342272393",
        "number_of_shards" : "1",
        "number_of_replicas" : "1",
        "uuid" : "ALtNifCGS_W2aTTcPEPx5A",
        "version" : {
          "created" : "7100299"
        },
        "provided_name" : "index1"
      }
    }
  }
}
```


- 삭제: DELETE index1


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


3.3.2 도큐먼트 생성

- 인덱싱: 엘라스틱서치에서 도큐먼트를 인덱스에 포함시키는 것



age - long, gender,name-text 타입 자동 지정

- 데이터 타입을 지정하지 않아도 엘라스틱서치가 도큐먼트의 필트와 값을 보고 자동으로 지정한다. ⇒ 다이내믹 매핑

```
PUT index2/_doc/2
{
  "name": "jane",
  "country": "france"
}
```

```
{
  "_index" : "index2",
  "_type" : "_doc",
  "_id" : "2",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 1,
  "_primary_term" : 1
}
```

- 인덱스에 새로운 필드가 추가되어도 문제없다.

```
PUT index2/_doc/3
{
  "name": "kim",
  "age": "20",
  "gender": "female"
}
```

```
{
  "_index" : "index2",
  "_type" : "_doc",
  "_id" : "3",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 2,
  "_primary_term" : 1
}
```

- 인덱스에 데이터타입이 잘못되어도 엘라스틱서치가 타입을 변환해서 저장한다.
  - 숫자 필드에 문자열이 입력되면 숫자로 변환한다
  - 정수 필드에 소수가 입력되면 소수점 아래 자리를 무시한다.

3.3.3 도큐먼트 읽기

- 도큐먼트 아이디를 이용해 조회: 실제로 사용은 드물다.
- 쿼리 DSL 사용: search 라는 DSL 쿼리를 이용해 도큐먼트를 읽을 수 있다.

3.3.4 도큐먼트 수정

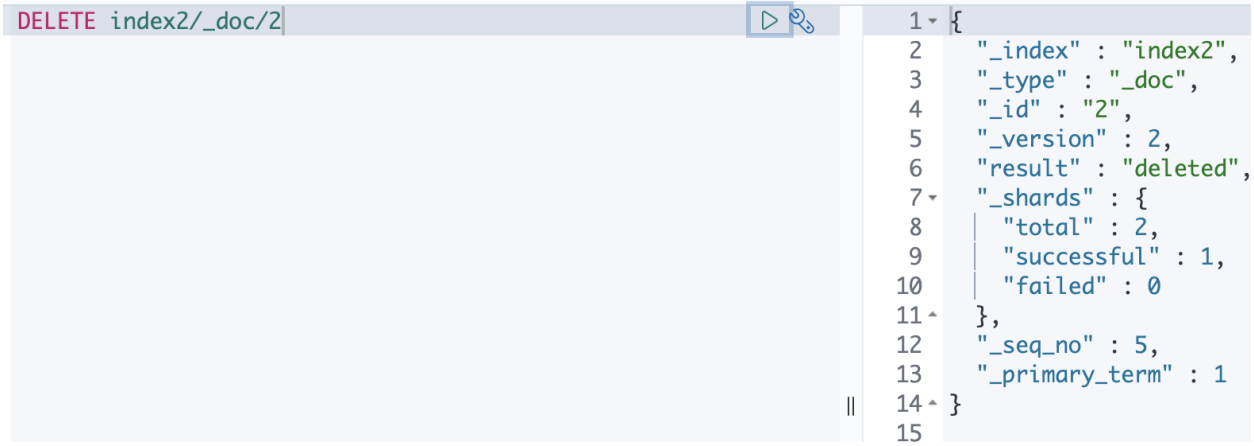
- 도큐먼트 수정 작업은 비용이 많이 들기 때문에 주의해야한다.

  도큐먼트 수정이 많은 작업이면 다른 데이터베이스를 쓰는것이 좋다.

_update 엔드포인트를 추가해 특정 필드의 값만 업데이트 할 수 있다.

3.3.5 도큐먼트 삭제

- 삭제도 비용이 많이 들어가는 작업이므로 주의해야 한다.



3.4 응답 메세지

코드	상태	해결방법
200, 201	정상적으로 수행함	
4xx	클라이언트 오류	클라이언트에서 문제점 수정
404	요청한 리소스가 없음	인덱스나 도큐먼트가 존재하는지 체크
405	요청 메소드(GET,POST)를 지원하지 않음	API 사용방법 다시 확인
429	요청 과부화	재전송, 노드 추가 같은 조치
5xx	서버 오류	엘라스틱서치 로그 확인 후 조치

3.5 벌크데이터

- 엘라스틱서치는 bulk API를 지원한다.
  - REST API 콜 횟수를 줄여 성능을 높인다.
  - 도큐먼트 생성/수정/삭제만 지원한다. (도큐먼트 읽기는 지원하지 않는다)
  - 복수의 JSON 구조를 줄바꿈 문자열로 구분 하는 NDJSON형태다.

```
POST _bulk
{"index": {"_index": "index2", "_id": "4"}}
{"name":"park", "age": 30, "gender": "female"}
{"index": {"_index": "index2", "_id": "5"}}
{"name":"jung", "age": 50, "gender": "male"}
```

  - 벌크 데이터를 파일로 만들어서 사용하는 것이 더 실용적이다.

    curl -H "Content-Type: application/x-ndjson" -XPOST loca
    lhost:9200/_bulk --data-binary "@./bulk_index2”

```
songi@MacBook-Pro-2 05 %  curl -H "Content-Type: application/x-ndjson" -XPOST lo
calhost:9200/_bulk --data-binary "@./bulk_index2"
{"took":44,"errors":false,"items":[{"index":{"_index":"index2","_type":"_doc","_
id":"6","_version":1,"result":"created","_shards":{"total":2,"successful":1,"fai
led":0},"_seq_no":8,"_primary_term":1,"status":201}},{"index":{"_index":"index2"
,"_type":"_doc","_id":"7","_version":1,"result":"created","_shards":{"total":2,"
successful":1,"failed":0},"_seq_no":9,"_primary_term":1,"status":201}}]}%
```

    - 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.

3.6 매핑

- 매핑: JSON 형태의 데이터를 루씬이 이해할 수 있도록 바꿔주는 작업
- 전문 검색과 대용량 데이터를 빠르게 실시간 검색할 수 있는 이유

3.6.1 다이내믹 매핑

- 엘라스틱서치가 자동으로 매핑
- 유연한 활용을 위해 인덱스 생성 시 매핑 정의를 강제하지 않는다.

원본 소스 데이터 타입	다이내믹 매핑으로 변환된 데이터 타입
null	필드를 추가하지 않음
boolean	boolean
float	float
integer	long
object	object
string	데이터 형태에 따라 date, text/keyword

< 주의 >

```
GET index2/_mapping
```


```
{
  "index2" : {
    "mappings" : {
      "properties" : {
        "age" : {
          "type" : "long"
        },
        "country" : {
          "type" : "text",
          "fields" : {...}
        },
        "gender" : {
          "type" : "text",
          "fields" : {...}
        },
        "name" : {
          "type" : "text",
          "fields" : {
            "keyword" : {...}
          }
        }
      }
```


age는 200살을 넘길 일이 없으므로 short 타입이 유리, gender나 country는 text 보다 집계나 정렬, 필터링을 위해 키워드 타입으로 지정되는게 좋다.

- 숫자 → 무조건 범위가 가장 넓은 long 으로 매핑, 불필요한 메모리를 차지할 수 있다.
- 문자열 → 검색과 정렬 등을 고려한 매핑이 제대로 되지 않는다.

3.6.2 명시적 매핑

```
PUT index3
{
  "mappings": {
    "properties": {
      "age": {"type": "short"},
      "name": {"type": "text"},
      "gender":{"type": "keyword"}
    }
  }
}
```

- 사용자가 직접 설정하는 매핑
- 저장할 데이터를 확실히 안다면, 인덱스를 생성할 때 직접 매핑하는 것이 좋다.
- 인덱스 매핑이 정해지면 이미 정의된 필드를 수정, 삭제할 수 없다.
    - 이름변경이나 데이터 타입 변경을 하려면 새로운 인덱스를 만들거나, reindex API를 써야한다.

- 추가할 필드명이 기존 필드와 중복되는 이름이면 오류가 발생한다.

3.6.3 매핑 타입

엘라스틱서치의 기본 데이터 타입

- 텍스트
  - text: 전문 검색이 필요한 데이터로 텍스트 분석기가 텍스트를 작은 단위로 분리한다
  - keyword: 정렬이나 집계에 사용되는 텍스트 데이터로 분석을 하지 않고 원문을 통째로 인덱싱한다.
- 날짜
  - date: 날짜/시간 데이터
- 정수
  - byte: 부호 있는 8비트 데이터
  - short: 부호 있는 16비트 데이터
  - integer: 부호 있는 32비트 데이터
  - long: 부호 있는 64비트 데이터
- 실수
  - scaled_float: float 데이터에 특정 값을 곱해서 정수형으로 바꾼 데이터. 정확도는 떨어지나 필요에 따라 집계 등에서 효율적으로 사용 가능하다.
  - half_float: 16비트 부동소수점 실수 데이터
  - double: 32비트 부동소수점 실수 데이터
  - float: 64비트 부동소수점 실수 데이터
- 불리언
  - boolean: true/false
- IP주소
  - ip: ipv4, ipv6 타입 IP 주소를 입력할 수 있다.
- 위치정보
  - geo-point: 위도, 경도 값을 갖는다.

    - geo-shape: 하나의 위치 포인트가 아닌 임의의 지형
- 범위값

integer_range, long_range	정수형 범위
float_range, double_range	실수형 범위
ip_range	IP 주소 범위
date_range	날짜/시간 데이터

    - 범위를 설정할 수 있는 데이터
    - 최솟값과 최댓값을 통해 범위를 입력한다.

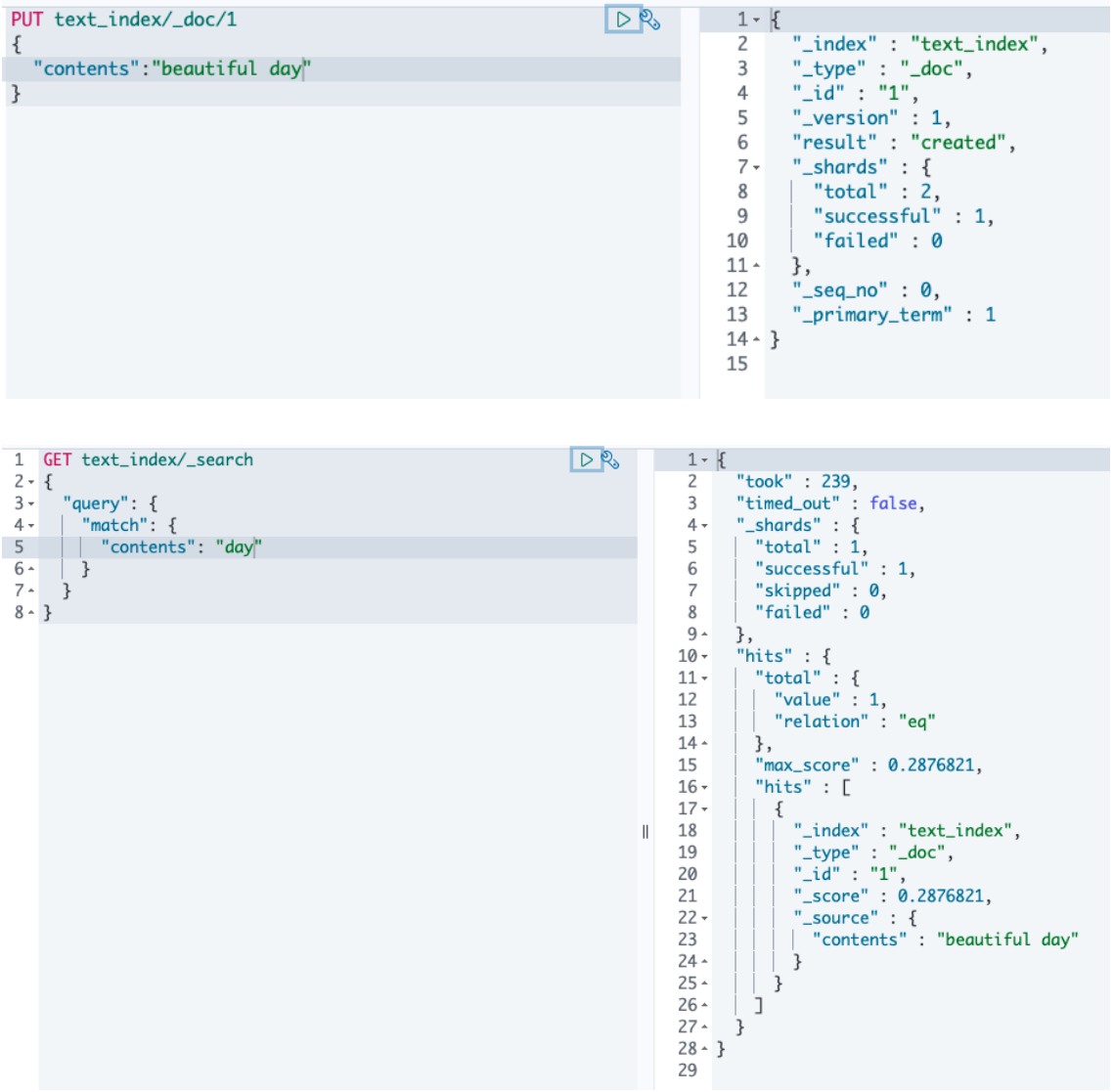
3.6.4 멀티 필드를 활용한 문자열 처리

- 텍스트 타입
    - 일반적으로 문장을 저장하는 매핑 타입
    - 역인덱싱: 텍스트 타입 문자열이 분석기에 의해 토큰으로 분리된다. → 분리된 토큰들은 인덱싱된다.
    - 기본적으로 집계나 정렬을 지원하지 않는다.
        - 매핑 파라미터로 집계나 정렬을 지원할 수는 있지만, 메모리를 많이 사용한다.
        - 텍스트 타입의 필드를 정렬하면 분해된 용어를 기준으로 정렬을 수행하므로 예상과 다른 결과가 나온다.
        - 집계나 정렬은 키워드 타입을 사용해야 한다.

    ▼ 실습

```
PUT text_index
{
  "mappings": {
    "properties": {
      "contents":{"type": "text"}
    }
  }
}
```

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



"match"는 전문 검색을 할 수 있는 쿼리다.

- 키워드 타입
  - 범주형 데이터에 주로 사용된다.
    - 범주형 데이터: 카테고리, 사람 이름, 브랜드 등 규칙성이 있거나, 유의미한 값들의 집합
    - 범주형 데이터는 용어를 분리할 필요가 없다.
    - 데이터 형태가 고정되어 문자열 집계나 정렬 작업에 활용가치가 높다.
  - 분석기를 거치지 않고 문자열 전체가 하나의 용어로 인덱싱된다.
    - 부분 일치 검색은 어렵지만 완전 일치 검색에 사용될 수 있다.

- 집계나 정렬에 사용할 수 있다.

▼ 실습

```
PUT keyword_index
{
  "mappings": {
    "properties": {
      "contents": {"type": "keyword"}
    }
  }
}
```

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

```
PUT keyword_index/_doc/1
{
  "contents": "beautiful day"
}
```

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

```
GET keyword_index/_search
{
  "query": {
    "match": {
      "contents": "beautiful"
    }
  }
}
```

```
{
  "took" : 239,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 0,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  }
}
```

```
GET keyword_index/_search
{
  "query": {
    "match": {
      "contents": "beautiful day"
    }
  }
}
```

```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 0.2876821,
    "hits" : [
      {
        "_index" : "keyword_index",
        "_type" : "_doc",
        "_id" : "1",
        "_score" : 0.2876821,
        "_source" : {
          "contents" : "beautiful day"
        }
      }
    ]
  }
}
```

키워드 타입은 문자열 전체를 하나의 용어로 보고 인덱싱하기 때문에 텍스트가 정확히 일치하는 경우에만 값을 가진다.

- 멀티 필드
  - 단일 필드 입력에 대해 여러 하위 필드를 정의하는 기능
  - “fields” 매핑 파라미터 사용
  - 전문 검색 + 정렬 ← 텍스트와 키워드 동시 지원

▼ 실습

```
PUT multifield_index
{
  "mappings": {
    "properties": {
      "message": {"type": "text"},
      "contents":{
        "type": "text",
        "fields": {
          "keyword": {"type": "keyword"}
        }
      }
    }
  }
}
```

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

```
PUT multifield_index/_doc/1
{
  "message" : "1 document",
  "contents": "beautiful day"
}
PUT multifield_index/_doc/2
{
  "message" : "2 document",
  "contents": "beautiful day"
}
PUT multifield_index/_doc/3
{
  "message" : "3 document",
  "contents": "wonderful day"
}
```

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

```
GET multifield_index/_search
{
  "query": {
    "match": {
      "contents": "day"
    }
  }
}
```

```
  "hits" : [
    {
      "_index" : "multifield_index",
      "_type" : "_doc",
      "_id" : "3",
      "_score" : 0.08701137,
      "_source" : {
        "message" : "3 document",
        "contents" : "wonderful day"
      }
    },
    {
      "_index" : "multifield_index",
      "_type" : "_doc",
      "_id" : "2",
      "_score" : 0.08701137,
      "_source" : {
        "message" : "2 document",
        "contents" : "beautiful day"
      }
    },
    {
      "_index" : "multifield_index",
      "_type" : "_doc",
      "_id" : "1",
      "_score" : 0.08701137,
      "_source" : {
        "message" : "1 document",
        "contents" : "beautiful day"
      }
    }
  ]
}
```

day로 전문 검색

```
GET multifield_index/_search
{
  "query": {
    "match": {
      "contents.keyword": "day"
    }
  }
}
```

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

day로 키워드 검색

```
GET multifield_index/_search
{
  "size": 0,
  "aggs": {
    "contents": {
      "terms": {
        "field": "contents.keyword"
      }
    }
  }
}
```

```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "contents" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "beautiful day",
          "doc_count" : 2
        },
        {
          "key" : "wonderful day",
          "doc_count" : 1
        }
      ]
    }
  }
}
```

agg: 집계 쿼리, 키워드로 같은 도큐먼트끼리 그룹핑되었다. (beautiful day/wonderful day)

3.7 인덱스 템플릿

- 설정이 동일한 복수의 인덱스를 만들 때 사용한다

3.7.1 템플릿 확인

> GET _index_template

3.7.2 템플릿 설정

- 인덱스 템플릿을 생성할 때 매핑과 설정 세팅을 주로 한다.
- 자주 사용하는 템플릿 파라미터
  - index_patterns: 새로 만들어지는 인덱스 중에 인덱스 이름이 인덱스 패턴과 매칭되는 경우 이 템플릿이 적용된다.
  - priority: 인덱스 생성 시 이름에 매칭되는 템플릿이 둘 이상일 때 템플릿이 적용되는 우선순위를 정할 수 있다. 숫자가 높은 템플릿이 먼저 적용된다.
  - template: 새로 생성되는 인덱스에 적용되는 settings, mappings 같은 인덱스 설정을 정의한다.
- 템플릿을 만든 이후에 만들어지는 인덱스들만 템플릿의 영향을 받는다.
- 똑같은 도큐먼트지만 템플릿이 적용되는지 여부에 따라 매핑이 달라진다.
- 템플릿 설정과 다른 타입의 도큐먼트는 400에러가 난다.

▼ 실습

```
PUT _index_template/test_template
{
  "index_patterns": ["test_*"],
  "priority": 1,
  "template": {
    "settings": {
      "number_of_shards": 3,
      "number_of_replicas": 1
    },
    "mappings": {
      "properties": {
        "name": {"type": "text"},
        "age": {"type": "short"},
        "gender": {"type": "keyword"}
      }
    }
  }
}
```

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

```
PUT test_index1/_doc/1
{
  "name":"kim",
  "age":10,
  "gender":"male"
}
```

```
{
  "_index" : "test_index1",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

```
GET test_index1/_mapping
```

```
{
  "test_index1" : {
    "mappings" : {
      "properties" : {
        "age" : {
          "type" : "short"
        },
        "gender" : {
          "type" : "keyword"
        },
        "name" : {
          "type" : "text"
        }
      }
    }
  }
}
```

test_template 이 적용된 test_index1

```
PUT train_index1/_doc/1
{
  "name":"kim",
  "age": 10,
  "gender": "male"
}
```

```
{
  "_index" : "train_index1",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

```
GET train_index1/_mapping
```

```
{
  "train_index1" : {
    "mappings" : {
      "properties" : {
        "age" : {
          "type" : "long"
        },
        "gender" : {
          "type" : "text",
          "fields" : {...}
        },
        "name" : {
          "type" : "text",
          "fields" : {
            "keyword" : {...}
          }
        }
      }
    }
  }
}
```

이름이 train이기 때문에 템플릿이 적용되지 않았고, 다이내믹 매핑이 적용되었다

```
PUT test_index2/_doc/1
{
  "name":"lee",
  "age": "19 years"
}
```

```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "mapper_parsing_exception",
        "reason" : "failed to parse field [age] of type [short] in document with id '1'. Preview of field's value: '19 years'"
      }
    ],
    "type" : "mapper_parsing_exception",
    "reason" : "failed to parse field [age] of type [short] in document with id '1'. Preview of field's value: '19 years'",
    "caused_by" : {
      "type" : "number_format_exception",
      "reason" : "For input string: \"19 years\""
    }
  },
  "status" : 400
}
```

도큐먼트 매핑타입과 템플릿 매핑 타입이 다르다. 숫자형 데이터에 문자형 데이터가 잘못 들어왔다는 400 에러를 보여준다.

- 템플릿 삭제
  - 기존 인덱스들은 영향을 받지 않는다.

```
DELETE _index_template/test_template
```

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

> DELETE _index_template/test_template

3.7.3 템플릿 우선순위

- 새로운 인덱스 템플릿은 우선순위가 높은 템플릿으로 덮어쓰기 된다.
- 숫자가 클수록 우선순위가 높다.

▼ 실습

```
PUT _index_template/multi_template1
{
  "index_patterns": "multi_*",
  "priority":1,
  "template":{
    "mappings": {
      "properties": {
        "age": {"type":"integer"},
        "name": {"type":"text"}
      }
    }
  }
}
```

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

```
PUT _index_template/multi_template2
{
  "index_patterns": "multi_data_*",
  "priority":2,
  "template":{
    "mappings": {
      "properties": {
        "name": {"type":"keyword"}
      }
    }
  }
}
```

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

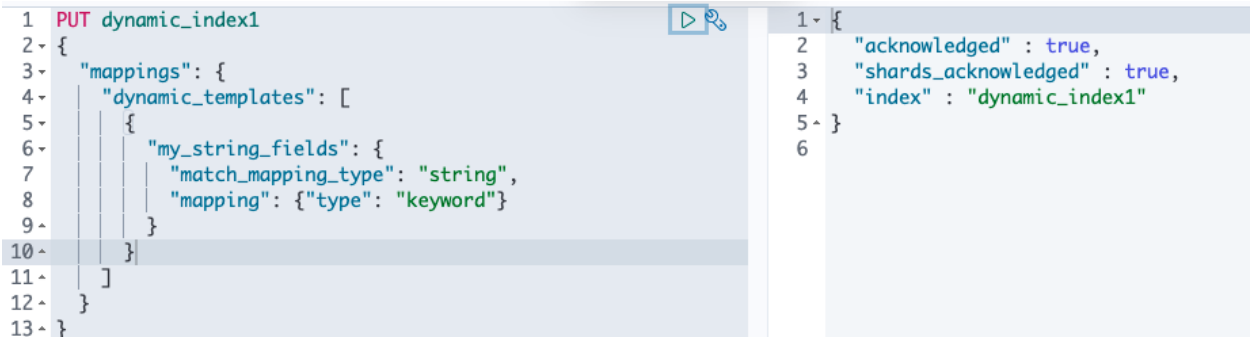
```
PUT multi_data_index


GET multi_data_index/_mapping
```

```
{
  "multi_data_index" : {
    "mappings" : {
      "properties" : {
        "name" : {
          "type" : "keyword"
        }
      }
    }
  }
}
```

tempate2로 적용되었다

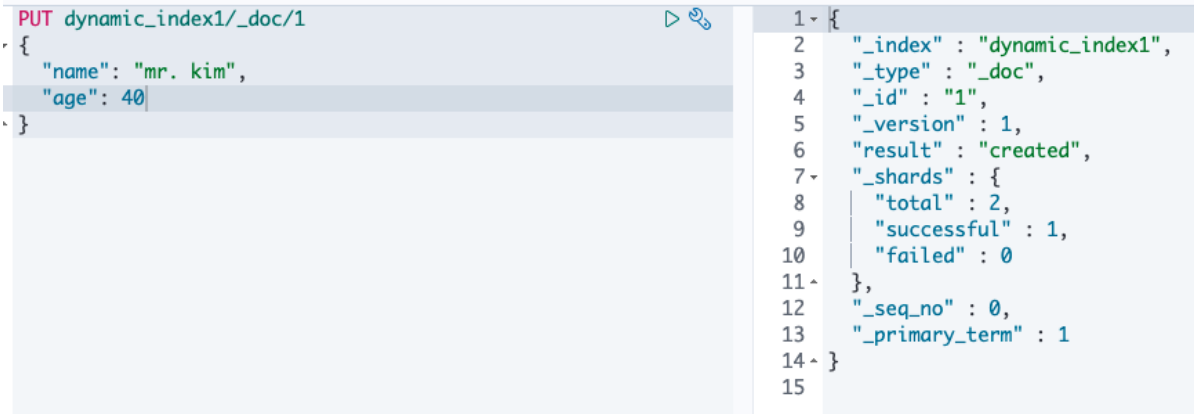
3.7.4 다이내믹 템플릿



my_string_fields: 다이내믹 템플릿 이름
match_mapping_type: 조건문 또는 매핑 트리거

- 매핑을 다이내믹하게 지정하는 템플릿 기술
- (로그시스템, 비정형화된 데이터를 인덱싱할 때처럼) 매핑을 정확하게 정할 수 없거나, 대략적인 데이터 구조만 알고 있을 때 사용한다.
- 다이내믹 템플릿 조건문
  - match_mapping_type: 데이터 타입을 확인하고 타입들 중 일부를 지정한 매핑 타입으로 변경한다.
  - match: 필드명이 패턴과 일치 하는 경우 매핑 타입으로 변경한다.
  - unmatch: match 패턴과 일치하는 경우 제외할 패턴을 설정할 수 있다.
  - match_pattern: match 패턴에서 사용할 수 있는 파라미터를 조정한다. 정규식이나 와일드 패턴등을 지정한다.
  - path_match, path_unmatch: match, unmatch와 비슷하지만 점이 들어가는 필드명에서 사용한다.

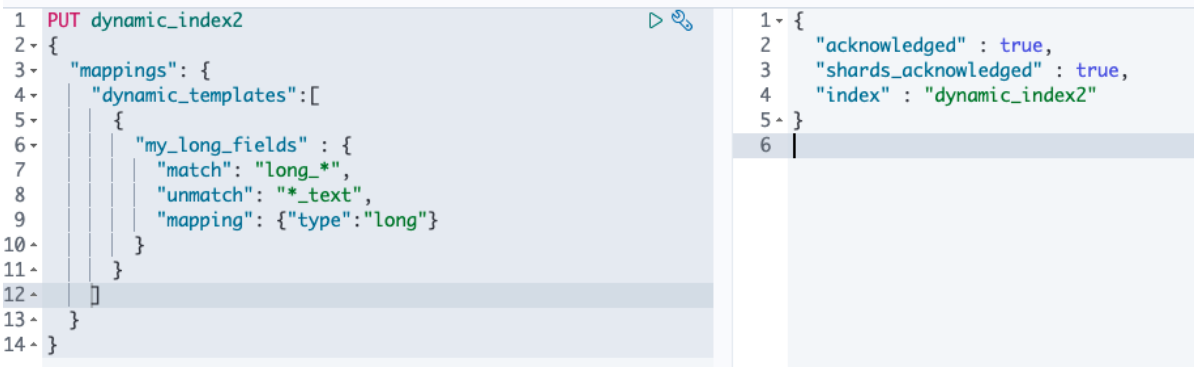
▼ 실습



```
GET dynamic_index1/_mapping
```

```
{
  "dynamic_index1" : {
    "mappings" : {
      "dynamic_templates" : [
        {
          "my_string_fields" : {
            "match_mapping_type" : "string
            "mapping" : {
              "type" : "keyword"
            }
          }
        }
      ],
      "properties" : {
        "age" : {
          "type" : "long"
        },
        "name" : {
          "type" : "keyword"
        }
      }
    }
  }
}
```

다이내믹 템플릿에 의해 문자열을 가진 데이터는 모두 키워드 타입으로 변경된다.
age필드는 다이내믹 템플릿 조건에 만족하지 않아 기존 다이내믹 매핑에 의해 long으로 매핑되었다.



```
PUT dynamic_index2
{
  "mappings": {
    "dynamic_templates":[
      {
        "my_long_fields" : {
          "match": "long_*",
          "unmatch": "*_text",
          "mapping": {"type":"long"}
        }
      }
    ]
  }
}
```

```
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "dynamic_index2"
}
```

match는 정규표현식을 이용해 필드명을 검사할 수 있다. 조건에 맞는 경우 mapping에 의해 필드들은 모두 숫자 타입을 갖는다.
unmatch는 조건에 맞는 경우 mapping에서 제외한다.

long_text는 match, unmatch모두 부합하기 때문에 다이내믹 템플릿에서 제외되어 다이내믹 패밍에 의해 텍스트/키워드를 갖는 멀티 필드 타입이 되었다.

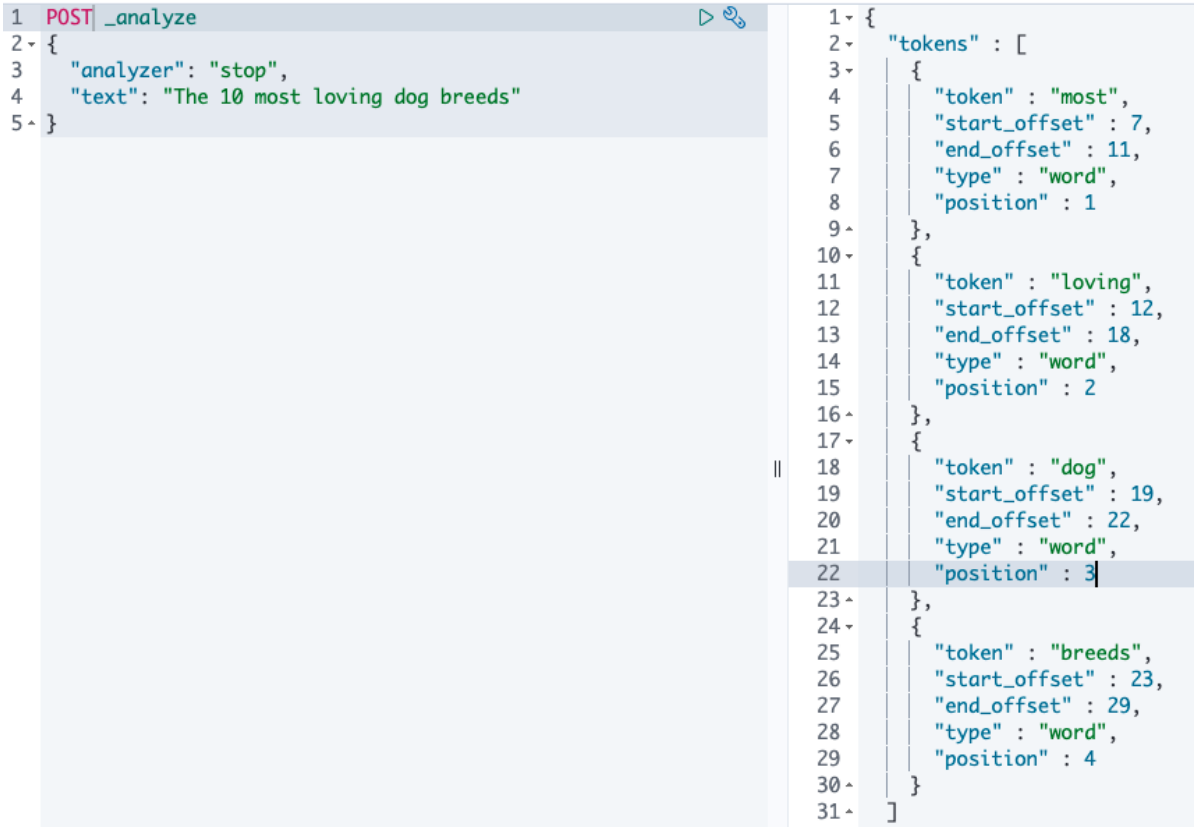
3.8 분석기

- 역인덱싱에서 문자열을 나누는 기준이 중요하다.
- 엘라스틱서치는 분석기 모듈(캐릭터 필터, 토크나이저, 토큰 필터) 를 갖는다.
  - 하나의 토크나이저는 반드시 포함돼야 한다.
  - 캐릭터 필터와 토큰 필터는 옵션이다.
  - 필터와 토크나이저 조합으로 커스텀 분석기를 만들 수 있다.
- 토큰: 토크나이저를 통해 필터링된 문자열이 잘린 단위
- 용어: 토큰 필터를 거쳐 정제 후 최종으로 역인덱스에 저장되는 상태의 토큰들

3.8.1 분석기 구성

- 구성요소

    - 캐릭터 필터: 입력받은 문자열을 변경하거나 불필요한 문자열을 제거한다.
    - 토크나이저: 문자열을 토큰으로 분리한다. 분리할 때 토큰의 순서/시작, 끝 위치도 기록한다.
    - 토큰 필터: 분리된 토큰들의 필터 작업을 한다. (대소문자 구분, 형태소 분석등의 작업)
- 역인덱싱
    - 역인덱스 테이블: 용어가 어떤 문서에 있는지 기록되어 문서를 찾기 쉽다.
- 분석기 API



the, 10 같은 것들은 **스톱 분석기**에 의해 토큰으로 지정되지 않기 때문에 most, loving, dog, breeds 4개의 토큰으로 분리되었다.

- 분석기 종류
    - standard: 특별한 설정이 없으면 엘라스틱서치가 기본적으로 사용하는 분석기다. 영문법을 기준으로 한 스탠다드 토크나이저와 소문자 변경 피럳, 스톱 필터가 포함되어 있다.

- simple: 문자만 토큰화한다. 공백, 숫자, 하이픈, 작은따옴표 같은 문자는 토큰화하지 않는다.
- whitespace: 공백을 기준으로 구분하여 토큰화한다.
- stop: simple 분석기와 비슷하지만 스톱 필터가 포함되어 the 가 제거되었다.

3.8.2 토크나이저

- 문자열을 분리해 토큰화하는 역할을 한다.
- 대표적인 토크나이저 종류
  - standard: 스탠다드 분석기가 사용하는 토크나이저로, 특별한 설정이 없으면 기본 토크나이저로 사용된다. 쉼표나 점 같은 기호를 제거하며 텍스트 기반으로 토큰화한다.
  - lowercase: 텍스트 기반으로 토큰화하며 모든 문자를 소문자로 변경해 토큰화한다.
  - ngram: 원문으로부터 N개의 연속된 글자 단위를 모두 토큰화한다.
  - uax_url_email: 스탠다드 분석기와 비슷하지만 URL이나 이메일을 토큰화하는 데 강점이 있다.

3.8.3 필터

- 옵션으로 하나 이상을 포함할 수 있다.
- 필터를 통해 더 세부적인 작업이 가능하다 → 엘라스틱에서 제공하는 분석기들은 하나 이상의 필터를 가진다.

uppercase 필터 적용

- 캐릭터 필터
  - 토크나이저 전에 위치하며 문자들을 전처리한다.
  - HTML문법을 제거/변경하거나 특정 문자가 왔을 때 다른 문자로 대체한다.
  - 캐릭터 필터를 사용하기 위해서는 커스텀 분석기를 만들어 사용하는 것이 좋다.
- 토큰 필터
  - 토크나이저에 의해 토큰화되어 있는 문자들에 필터를 적용한다.
  - 토큰들을 변경, 추가, 삭제 작업이 가능하다.
  - 자주 사용하는 토큰 필터
    - lowercase: 모든 문자를 소문자로 변환한다. 반대로 모든 문자를 대문자로 변환하는 uppercase 필터가 있다.
    - stemmer: 영어 문법을 분석하는 필터다. 언어마다 고유한 문법이 있어 필터 하나로 모든 언어에 대응하기는 힘들다. 한글의 경우 아리랑, 노리 같은 오픈소스가 있다.
    - stop: 기본 필터에서 제거하지 못하는 특정한 단어를 제거할 수 있다.

3.8.4 커스텀 분석기

- 사용자가 직접 토크나이저, 필터 등을 조합해 사용할 수 있는 분석기
- 필터들의 조합이나 순서에 따라 특별한 형태로 만들 수 있다.

```
PUT customer_analyzer
{
  "settings": {
    "analysis": {
      "filter": {
        "my_stopwords": {
          "type": "stop",
          "stopwords": ["lions"]
        }
      },
      "analyzer": {
        "my_analyzer": {
          "type": "custom",
          "char_filter": [],
          "tokenizer": "standard",
          "filter": ["lowercase","my_stopwords"]
        }
      }
    }
  }
}
GET customer_analyzer/_analyze
{
  "analyzer": "my_analyzer",
  "text": "Cats Lions Dogs"
}
```

```
{
  "tokens" : [
    {
      "token" : "cats",
      "start_offset" : 0,
      "end_offset" : 4,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    {
      "token" : "dogs",
      "start_offset" : 11,
      "end_offset" : 15,
      "type" : "<ALPHANUM>",
      "position" : 2
    }
  ]
}
```

- 필터 적용 순서
  - 필터 배열의 첫번째 순서부터 필터가 적용된다.

```
GET customer_analyzer/_analyze
{
  "tokenizer": "standard",
  "filter": ["lowercase", "my_stopwords"],
  "text": "Cats Lions Dogs"
}

GET customer_analyzer/_analyze
{
  "tokenizer": "standard",
  "filter": ["my_stopwords","lowercase"],
  "text": "Cats Lions Dogs"
}
```

```
{
  "tokens" : [
    {
      "token" : "cats",
      "start_offset" : 0,
      "end_offset" : 4,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    {
      "token" : "lions",
      "start_offset" : 5,
      "end_offset" : 10,
      "type" : "<ALPHANUM>",
      "position" : 1
    },
    {
      "token" : "dogs",
      "start_offset" : 11,
      "end_offset" : 15,
      "type" : "<ALPHANUM>",
      "position" : 2
    }
  ]
}
```