Elasticsearch

3장 인덱스 설계

3.1 인덱스 설정

GET [인덱스이름]/_settings

number_of_shards
이 인덱스가 데이터를 몇 개의 샤드로 쪼갤 것인지 지정하는 값, 한번 지정하면 reindex 같은 동작을 통해 인덱스를 통째로 재 색인하는 등 특별한 작업을 수행하지 않는 한 바꿀수 없으므로 신중하게 설계해야함
샤드 하나마다 루씬 인덱스가 하나씩 더 생성되고, 주 샤드 하나당 복제본 샤드도 늘어나므로 주의하자.
7버전 이상은 기본값 1, 7버전 이하는 기본값 5

number_of_replicas
주 샤드 하나당 복제본 샤드를 몇 개 둘것인지 지정하는 값
이 값은 인덱스 생성후에도 동적으로 변경 가능

PUT [인덱스 이름]/_settings
{
"index.number_of_replicas" : 0 // 복제본 샤드 생성 x , 대용량의 초기 데이터를 마이그레이션하는 등의 상황에 쓰기 성능을 위해 주로 사용
}

refresh_interval
Refresh를 얼마나 자주 수행할 것인지 설정
PUT [인덱스이름]/_settings
{
"index.refresh_interval" : "1s"
"index.refresh_interval" : null
}
이 값을 명시적으로 설정하지 않으면 매초마다 refresh 실행하고 30초 이상 검색 쿼리가 들어오지 않는것을 확인하면 다음 첫 검색 쿼리 들어올때까지 Refresh 수행 안한다 이 설정은 "index.search_idle.after" 에서 변경 가능

삭제
DELETE [인덱스이름]

3.1 인덱스 설정

기본값 말고 인덱스 설정

PUT [인덱스이름]
```
{
        “settings” : {
          “number_of_shards” : 2,
          “number_of_replicas” : 2
        }
}
```

동적매핑
엘라스틱 서치가 자동으로 생성

명시적매핑
사용자가 직접 매핑을 지정

매핑 설정은 한번 지정되면 변경이 쉽지 않기때문에 운영환경에서는 명시적매핑 사용하는게 좋음.
PUT mapping_test/_mapping // 신규 필드 추가
```
{
        “properties” : {
                “longValue” : {
                        “type” : “long”
                }
        }
}
```

3.2 필드타입

심플 타입
Text, keyword, date, long, double, Boolean, ip 등

계층구조를 지원하는 타입
Object, nested 등

특수타입
geo_point, geo_shape, binary, long_range, date_range, ip_range, completion등

Date 타입에 format 종류
epoch_milis, epoch_second, date_time, strict_date_time, strict_date_optional_time

배열을 표현하는 타입 구분은 없이 배열 넣을수 있음

Object vs nested 간단 비교

타입	object	Nested
용도	일반적인 계층구조	배열 내 각 객체를 독립적으로 취급해야 하는 상황
성능	가벼움	무거움
검색	일반적인 쿼리	전용 nested 쿼리로 감싸서 사용

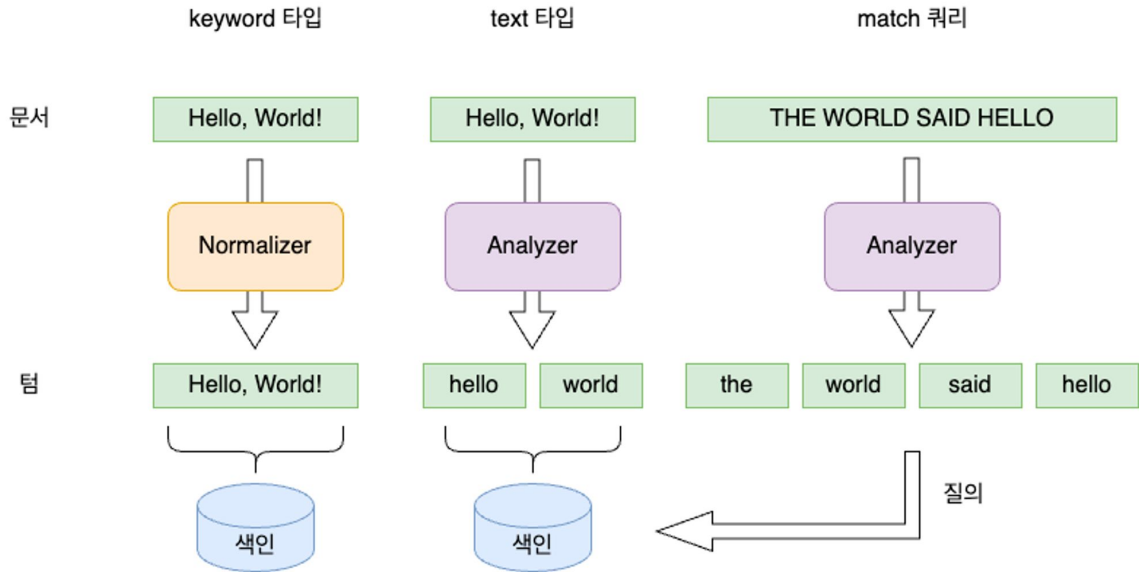
3.2 필드타입

Text 타입과 keyword 타입 비교
Text 는 string을 토큰작업 후 최종적으로 역색인 들어가는 형태(term) Hello, World! 일시 hello 만 검색해도 검색가능
Keyword는 정확히 입력해줘야 검색가능

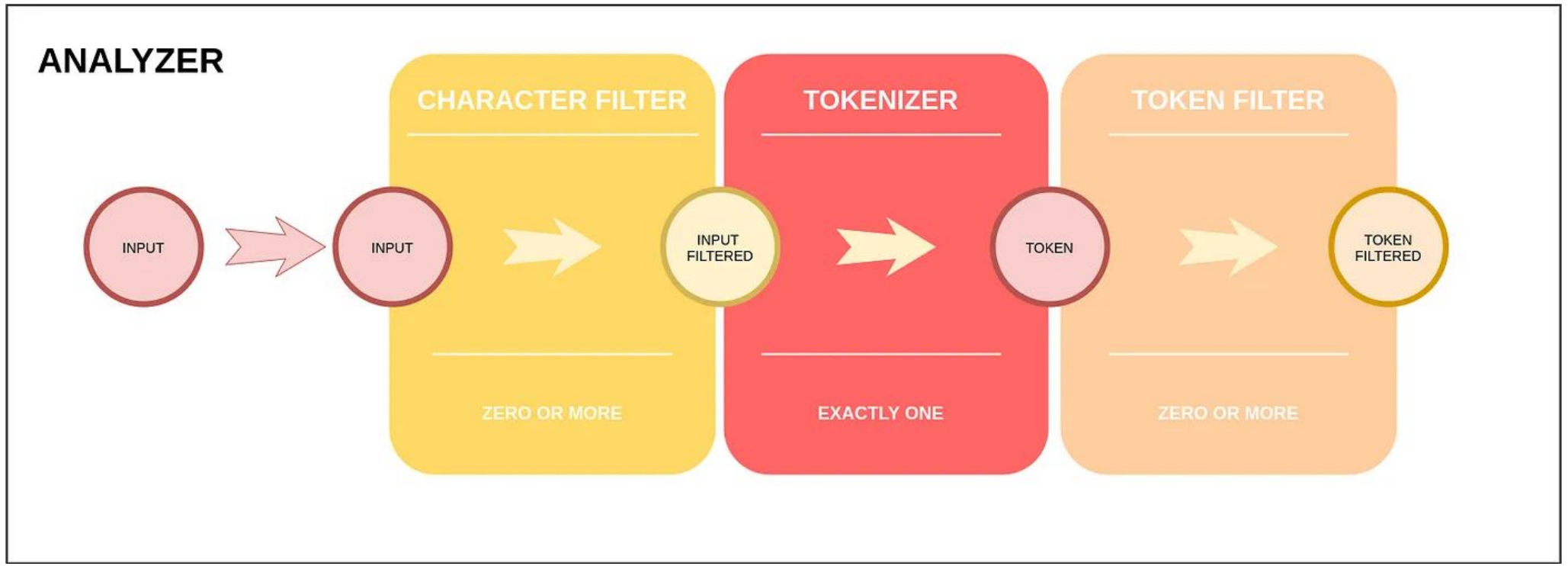
Text 타입은 주로 전문 검색에 적합하고 keyword 타입은 일치 검색에 적합함.

Doc_values 는 디스크를 기반으로 한 자료 구조로 파일 시스템 캐시를 통해 효율적으로 정렬, 집계, 스크립트 작업 수행
정렬, 집계, 스크립트 작업을 할 일이 없는 필드는 false 로 지정한다.

Text타입은 doc_values를 사용할 수 없는데 fielddata를 사용해 정렬이나 집계등의 작업시에 역색인 전페를 힙 메모리에 올림
힙을 순식간에 차지하므로 oom 등의 문제를 발생할 수 있어서 주의해야함



3.3 애널라이저와 토크나이저



Character Filter
인풋으로 들어온 텍스트를 character stream으로 받아서 특정한 문자를 추가, 변경, 삭제한다.
여러개의 character filter가 지정되었다면 순서대로 수행된다.

Tokenizer
character stream을 받아서 여러 token으로 쪼개어 token stream을 만든다. 하나의 analyzer에는 한 개의 tokenizer만 지정 가능하다.

Token Filter
token stream을 받아서 token을 추가, 변경, 삭제한다. 여러개의 필터가 있는 경우 순차 적용된다.

3.4 인덱스 템플릿

index의 template을 지정하지 않을 경우 인덱스를 생성할 때마다 shard 및 field type을 계속해서 지정을 해줘야 하는 귀찮은 문제가 발생합니다.
template를 사용하게 되면 특정 패턴의 인덱스가 생성될때 template으로 지정된 설정이 자동으로 적용되어 인덱스가 생성되기 때문에 shard, refresh, field 타입이 자동으로 적용됩니다.

템플릿 이름이 test_template인 인덱스 템플릿 생성

```
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"}
      }
  }
  }
}
```

```
PUT test_index1/_doc/1
{
  "name": "kang",
  "age": 20,
  "gender": "male"
}
```

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

3.4 컴포넌트 템플릿

다수의 템플릿을 관리하는데에는 여러 문제가 있을 수 있다.

- 인덱스 템플릿 간의 중복이 존재하는 경우 클러스터 상태가 커질 수 있다.
- 모든 인덱스 템플릿을 변경하려면 각 템플릿을 수동으로 변경해야 한다.
- 인덱스가 여러 템플릿과 일치하는 경우, Opensearch 자체적으로 병합을 시도하면서 예기치 않은 방식으로 병합이 될 수 있다.

여기서 Component 템플릿은 공통 설정, 매핑 및 별칭을 재사용 가능한 빌딩 블록으로 추상화하여 위 문제들을 해결 할 수 있다.

** 인덱스 생성 요청(create index)에서 직접 지정하는 설정 및 매핑은 인덱스 템플릿 및 해당 컴포넌트 템플릿에서 지정된 모든 설정 또는 매핑을 재정의한다.

```
// 컴포넌트 템플릿 예시
// 컴포넌트 템플릿 1
PUT _component_template/component_templ
{
  "template": {
    "mappings": {
      "properties": {
        "@timestamp": {
          "type": "date"
        }
      }
    }
  }
}


// 컴포넌트 템플릿 2
PUT _component_template/component_templ
{
  "template": {
    "mappings": {
      "properties": {
        "ip_address": {
          "type": "ip"
        }
      }
    }
  }
}
```

```
// 인덱스 템플릿 내 컴포넌트 템플릿 적용
// composed_of에 컴포넌트 템플릿을 추가해야 된다.
PUT _index_template/daily_logs // 예시
{
  "index_patterns": [
    "logs-2022-01-*"
  ],
  "template": {
    "aliases": {
      "my_logs": {}
    },
    "settings": {
      "number_of_shards": 2,
      "number_of_replicas": 1
    },
    "mappings": {
      "properties": {
        "timestamp": {
          "type": "date",
          "format": "yyyy-MM-dd HH:mm:ss||yyyy-MM-dd||epoch_millis"
        },
        "value": {
          "type": "double"
        }
      }
    }
  },
  "priority": 200,
  "composed_of": [
    "component_template_1",
    "component_template_2"
  ],
  "version": 3,
  "_meta": {
    "description": "using component templates"
  }
}
```

3.4 동적 템플릿

인덱스에 새로 들어온 필드의 매핑을 사전에 정의한대로 동적 생성한다.

```
PUT my-index-000001
{
  "mappings": {
    "dynamic_templates": [
      {
        "ip_fields": {
          "match":   ["ip_*", "*_ip"],
          "unmatch": ["one*", "*two"],
          "mapping": {
            "type": "ip"
          }
        }
      }
    ]
  }
}

PUT my-index/_doc/1
{
  "one_ip":   "will not match", 1
  "ip_two":   "will not match", 2
  "three_ip": "12.12.12.12", 3
  "ip_four":  "13.13.13.13" 4
}
```

3.5 라우팅

_routing은 쉽게는 문서에 대한 그룹이나 분류에 활용
즉, 문서를 색인 할 때 같은 분류에 속하는 문서들을 특정 shard로 색인하고 검색 시에도 모든 shard를 대상으로 하지 않기때문에 성능적으로 유용

이 간단한 예시에서 학생은 'document', 반은 routing 기준이 되는 field, 교실은 'shard'에 비유할 수 있을 것이다.

직접 예제를 수행해 보자.

일단 아래와 같은 bulk 데이터를 준비하자

```
{"index": {"_index": "school", "_type": "students", "_id": 1}}
{"class": "A", "name": "albert"}
{"index": {"_index": "school", "_type": "students", "_id": 2}}
{"class": "A", "name": "david"}
{"index": {"_index": "school", "_type": "students", "_id": 3}}
{"class": "A", "name": "brown"}
{"index": {"_index": "school", "_type": "students", "_id": 4}}
{"class": "A", "name": "jimmy"}
{"index": {"_index": "school", "_type": "students", "_id": 5}}
{"class": "A", "name": "jennifer"}
{"index": {"_index": "school", "_type": "students", "_id": 6}}
{"class": "B", "name": "hong"}
{"index": {"_index": "school", "_type": "students", "_id": 7}}
{"class": "B", "name": "kim"}
{"index": {"_index": "school", "_type": "students", "_id": 8}}
{"class": "B", "name": "john"}
{"index": {"_index": "school", "_type": "students", "_id": 9}}
{"class": "C", "name": "tomas"}
{"index": {"_index": "school", "_type": "students", "_id": 10}}
{"class": "C", "name": "jamie"}
```

```
$ curl -XPUT 'localhost:9200/school?pretty' -d '{
    "mappings": {
        "students": {
            "_routing": {
                "required": true,
                "path": "class"
            },
            "properties": {
                "class": {
                    "type": "string",
                    "index": "not_analyzed"
                },
                "name": {
                    "type": "string"
                }
            }
        }
    }
}'
```

```
localhost:9200/school/students/_search?pretty&routing=B'
```