카카오 인프라 플랫폼 도구 교육

Indexing & Search

엘라스틱서치의 모든 것

2019.09.06

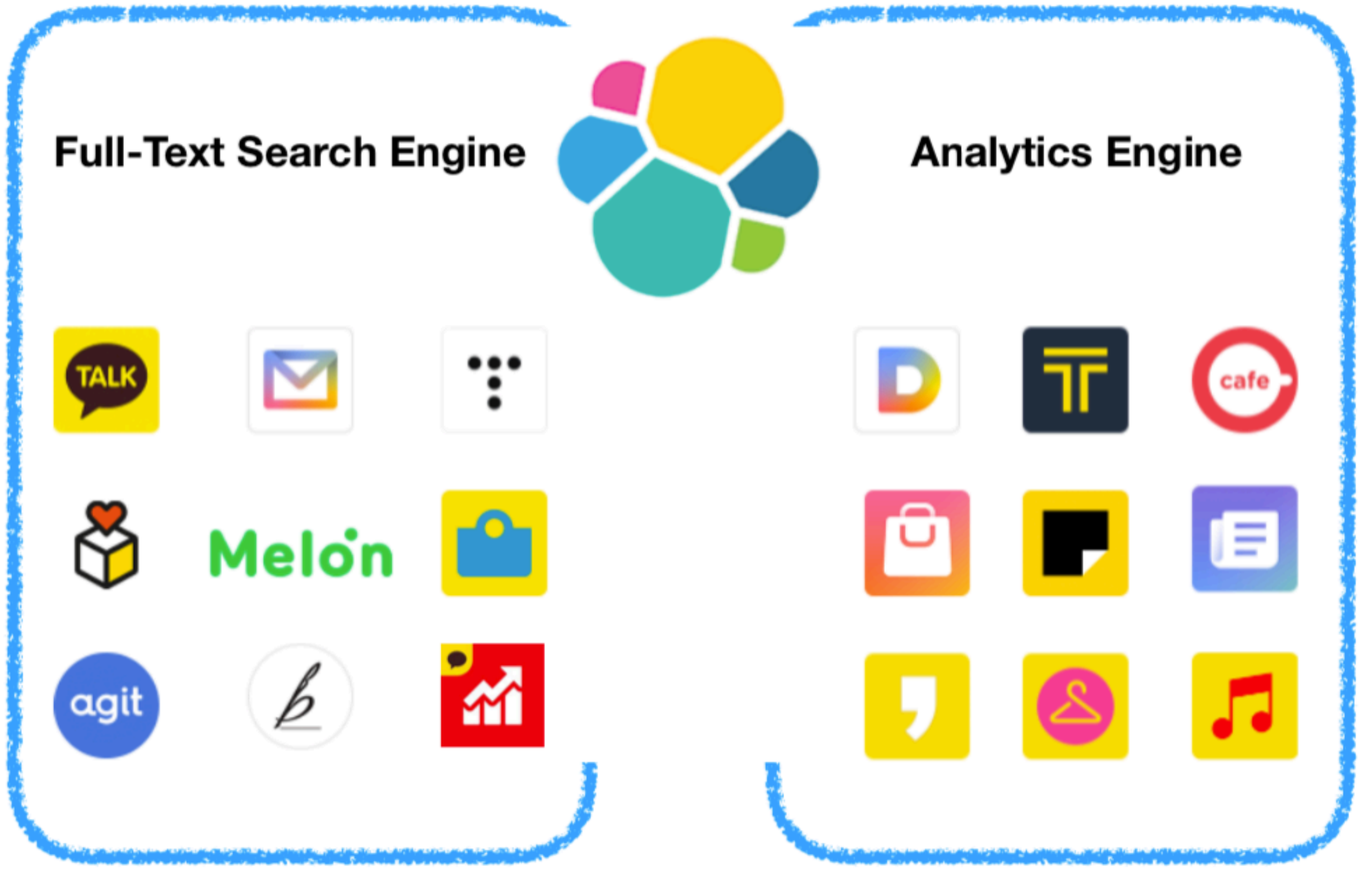
시스템엔지니어링파트/Leo.woo

kakao

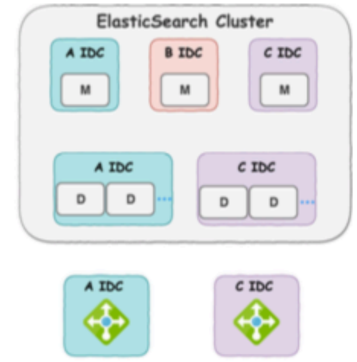
About Me

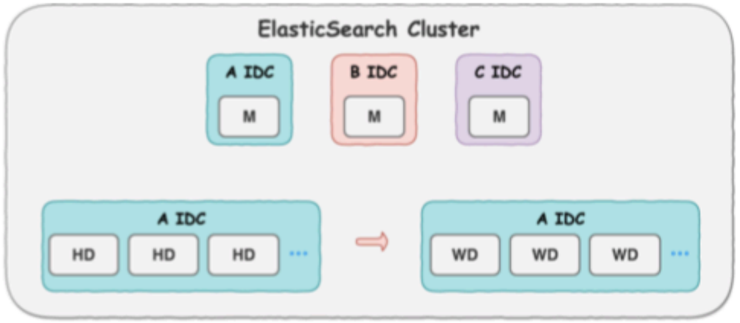
```
POST /employee/_doc
{
 "name": "Leo.Woo",
 "tag" : "@@esse",
 "depart" : "시스템엔지니어링파트",
 "task" : "Elasticsearch 구축/운영 및 가이드 제공"
}
```

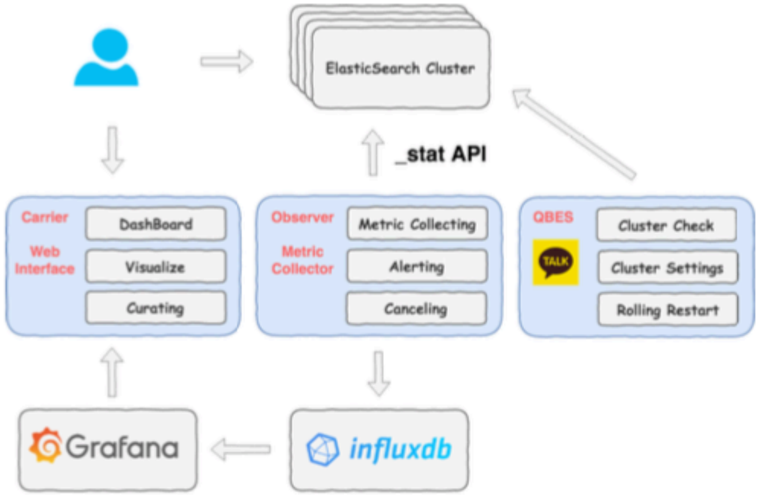
Elasticsearch in Kakao

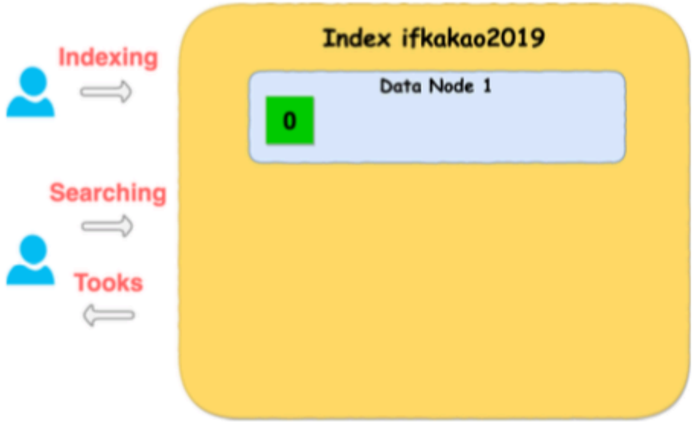


Elasticsearch in Kakao









INDEXING & SEARCH

INDEXING - Basic

Indexing

- Indexing(색인)은 Document를 Index에 저장하는 것
- 색인할 때 Index가 사전에 생성되어 있지 않으면 Elasticsearch가 자동으로 Index 생성

```
POST /exmple/_doc
{
 "name": "john",
 "street" : "판교역로",
 "phone" : "000-0000"
}
```

Index

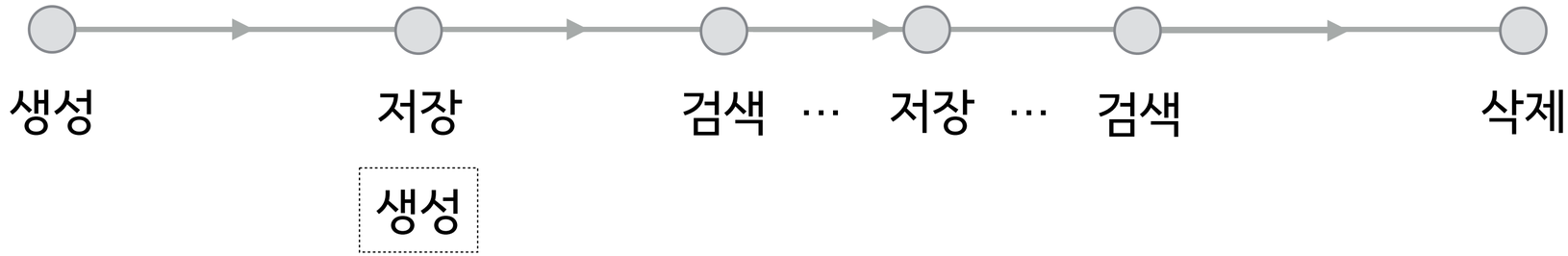
- Index는 Document의 논리적인 집합
- 동일한 목적과 속성을 가진 Document는 하나의 Index에 저장
- 하나의 Index에 있는 Document는 같은 설정을 가지게 됨

```
PUT /exmple

{ "acknowledged": true }
```

Index Life Cycle

Index Life Cycle



- **생성** : Document를 저장하기 위해서 Index를 생성
- **저장** : 사용자는 Index에 Document를 색인하여 데이터를 저장
- **검색** : Index에 저장된 Document를 검색
- **삭제** : 더이상 데이터 저장과 검색에 필요없는 Index는 삭제

Document

- Document(문서)는 Index에 데이터를 색인하는 기본 단위
- JSON 형태로 표현 *JSON(Javascript Object Notation)
- Document를 구분하는 ID는 사용자가 임의로 지정하거나 색인시 자동으로 생성

```
PUT /exmple/_doc/1
{
  "name": "leo",
  "street" : "판교역로",
  "phone" : "000-0000"
}
```

```
POST /exmple/_doc
{
  "name": "john",
  "street" : "판교역로",
  "phone" : "000-0000"
}
```

Inverted Index

- Inverted Index는 Full-text 검색을 위해 고안된 자료구조
- Index안에 있는 각 Field별로 모든 Document에 등장하는 개별 term이 어느 문서에 있는지 저장

```
PUT /exmple/_doc/1
{ "comment" : "leo love pineapple pizza"}
```

```
PUT /exmple/_doc/1
{ "comment" : "wife love pineapple"}
```

Document 색인

Term	doc_1	doc_2
leo	O	
wife		O
love	O	O
pineapple	O	O
pizza	O	

Inverted Index

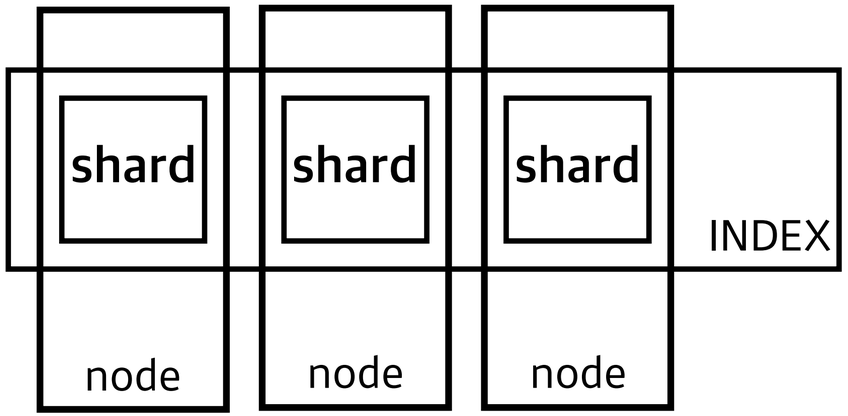
Bulk API

- Bulk API를 사용하면 Elasticsearch에 빠르게 데이터를 색인 가능

```
POST _bulk
{ "index" : { "_index" : "test", "_id" : "1" } }
{ "field1" : "value1" }
{ "delete" : { "_index" : "test", "_id" : "2" } }
{ "create" : { "_index" : "test", "_id" : "3" } }
{ "field1" : "value3" }
{ "update" : {"_id" : "1", "_index" : "test"} }
{ "doc" : {"field2" : "value2"} }
```

Shard

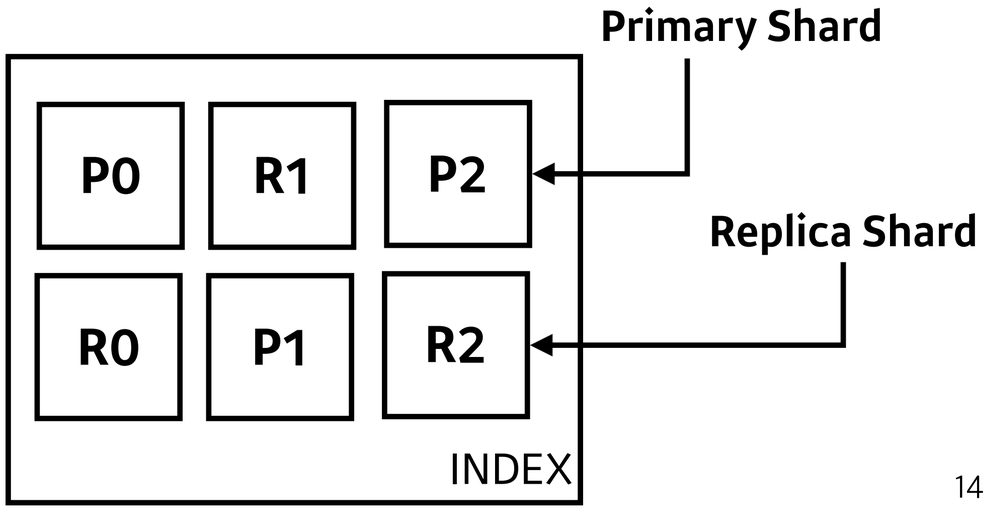
- Shard는 Index의 데이터를 Elasticsearch Cluster에 분산하여 저장하는 단위



node : Elasticsearch cluster를 구성하는 서버

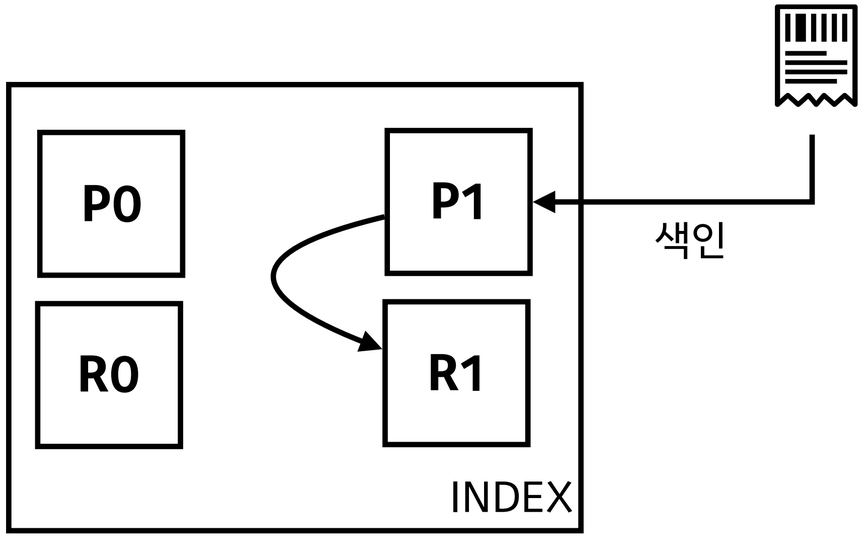
Shard

- Shard는 Primary Shard와 Replica Shard로 나뉨
  - Primary Shard : 색인된 Document의 원본을 저장하는 Shard
  - Replica Shard : 색인된 Document의 복제본을 저장하는 Shard



Shard

- Index에 색인시 Primary Shard에 먼저 색인 후 Replica에 색인



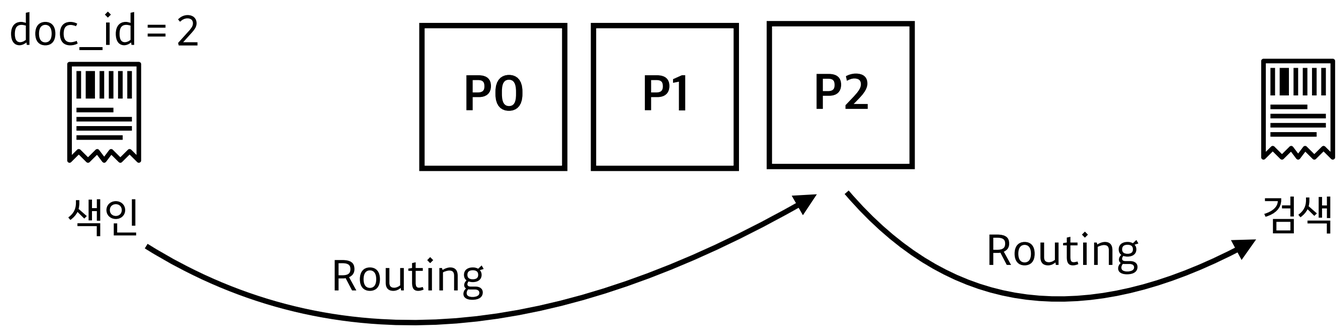
Settings

- Index의 속성과 관련된 설정은 Settings 값을 통해 정의 가능
- Static settings은 Index를 생성하는 순간에만 설정
- Dynamic settings은 Index를 사용하는 중에도 변경 가능

Static settings

- number_of_shards : 대표적인 Static settings이며 Primary Shard의 개수를 정의하는 값.
- Document가 어느 Shard에 저장될지 결정하는 Routing 공식에 사용되는 변수이기 때문에 Index 생성 이후에는 변경 불가

```
Routing  = hash(id | key) % (num_of_shards)
```



Dynamic settings

- number_of_replicas : Replica Shard의 개수를 정의
- refresh_interval : 색인한 데이터를 Refresh 하는 주기를 결정
- blocks.read_only : Index를 읽기 전용으로 변경하는 설정
- blocks.read_only_allow_delete : Index를 읽기와 삭제만 가능하도록 변경하는 설정

Template

- Template 기능을 통해 Index의 Settings, Mappings, Analyzer 설정을 Index 생성시 자동으로 적용 가능

template name : user
index name : **user-YYYY.MM.DD**
primary shard : **5개**
Field : category**(keyword)**, name**(keyword)**, comment**(text)**
Analyzer : comment 필드에 **english analyzer**


```
PUT _template/user
{
 "index_patterns": ["user-*"],
 "settings": {"number_of_shards": 5},
 "mappings": {
  "properties": {
   "category": {
    "type": "keyword"
   },
   "name": {
    "type": "keyword"
   },
   "comment": {
    "type": "text",
    "analyzer": "english"
   }
}}}
```

Refresh

- Refresh는 색인시 메모리에 생성된 Index와 관련된 자료구조를 디스크에 쓰는 과정

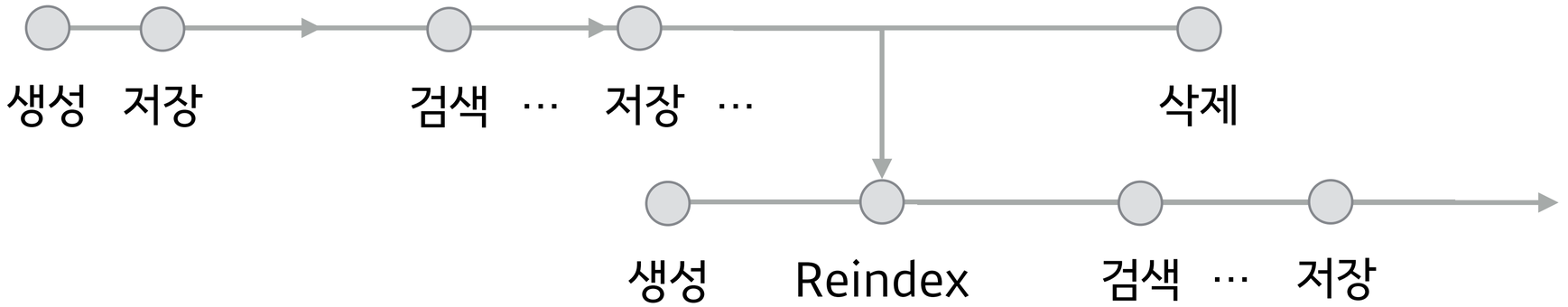
  데이터가 색인될 때 생성되는 inverted index와 같은 자료구조를 우선 메모리에 저장하고
  Refresh과정에서 디스크에 Segment라는 파일로 만들어서 자료구조를 저장

- Refresh 과정을 거쳐야 Document 검색이 가능 (Searchable)

- 디스크에 데이터를 쓰는 과정은 작업 부하가 크기 때문에 refresh_interval을
  너무 짧게 설정하는 것은 좋지 않음

Reindex

- Reindex API를 사용하여 기존 Index의 데이터를 새로운 Index로 이동 가능



Reindex

- 새로운 Index의 이름은 기존 Index의 이름과 같을 수 없음
- Index운영 중에 Reindex를 하면 부하가 발생하기 때문에 적절한 Reindex Size 설정이 필요
- 기존 Index에 Alias가 적용되어 있어야 Reindex 이후에도 문제없이 사용 가능

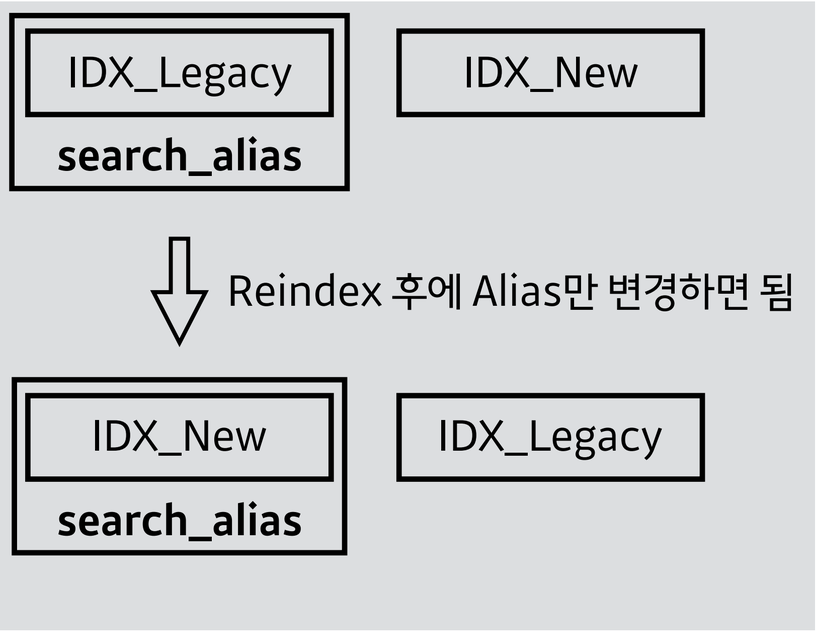
Alias

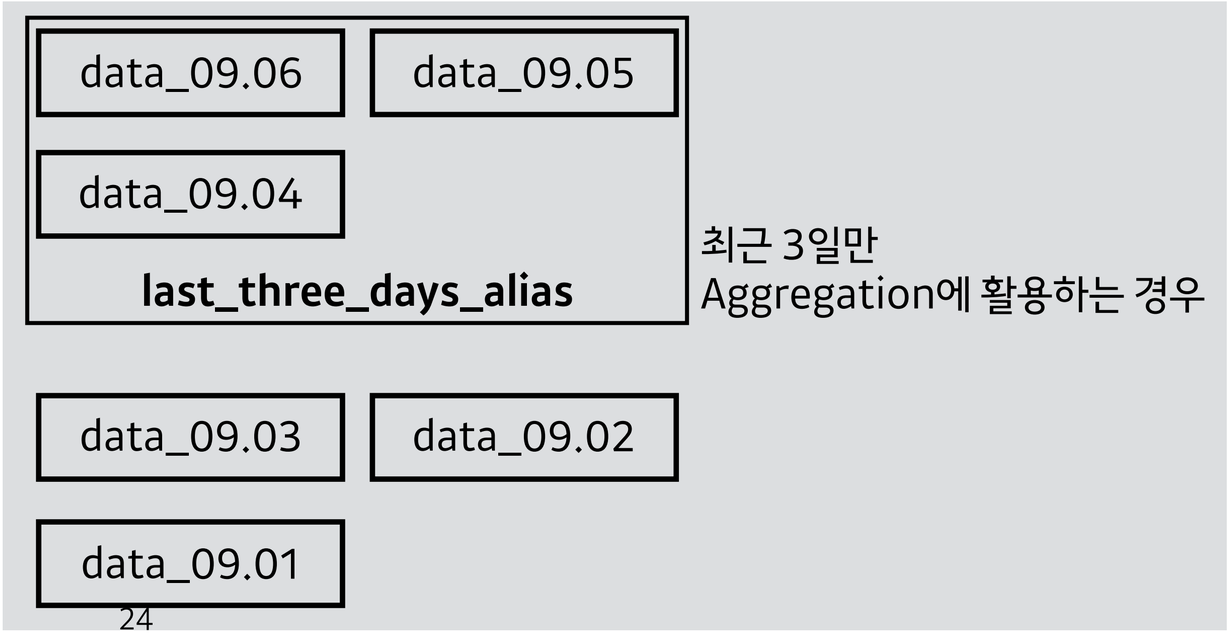
- Alias는 Index의 별명을 지어주는 것
- Alias는 Index Name과 다르게 유연하게 생성/변경/삭제가 가능

```
POST /_aliases
{
  "actions" : [
    { "add" : { "index" : "example", "alias" : "example_alias" } }
  ]
}
POST /_aliases
{
  "actions" : [
    { "remove" : { "index" : "example", "alias" : "example_alias" } },
    { "add" : { "index" : "new_example", "alias" : "example_alias" } }
  ]
}
```

Alias

- Alias를 사용하면 검색에 사용하는 Index가 변경되는 경우에도 쉽게 대응 가능
- 데이터 분석 대상 Index를 지정하고 싶은 경우에 Alias를 활용 가능





Delete Index

- Index는 DELETE 명령으로 삭제가 가능
- 운영환경에서는 Elasticsearch 관리 도구를 사용
- 서비스 운영시 Nginx를 사용하여 DELETE 명령 제한 가능

```
DELETE /example

{ "acknowledged": true }
```

INDEXING - Advanced

Mappings

- Mappings는 Index에 색인되는 Document의 Field와 Field datatype을 정의
- 사전에 정의되지 않은 Mapping 정보는 Elasticsearch가 자동으로 정의
  - Static Mapping : Document 색인 전에 정적으로 정의한 Mapping
  - Dynamic Mapping : Document 색인시 동적으로 정의된 Mapping

Field

- Field는 관계형 데이터베이스의 열(columns)에 해당
- 사용자는 Document의 Field에 원하는 데이터를 저장하고 Field를 이용하여 검색

```
POST /exmple/_doc
{
  "name": "leo",
  "street" : "판교역로",
  "phone" : "000-0000"
}
```

```
GET /exmple/_search
{
  "query": {
    "match" : {
      "name" :  "leo"
    }
  }
}
```

Field datatype

- Field datatype은 Field에 저장될 데이터의 타입을 결정하며 변경 불가
- Dynamic Mapping의 Field datatype을 결정할 때 가장 넓은 형태의 datatype을 지정

```
POST /exmple/_doc
{
  "name": "phone",
  "price" : "10000"
}
```

```
  "mappings":{
    "properties": {
      "name": {
          "type":  "text",
            "fields": {
              "keyword": {
                  "type": "keyword", ...}
              }

        },
        "price" : {
            "type": "long"...
```

Field datatype

- String
  - text : Full-text 데이터를 저장. Analyzer를 통해 분석을 거쳐서 저장됨
  - keyword : 분석하지 않고 문자열을 그대로 저장
- Numeric
  - long, integer, short, byte : 정수형 데이터 저장
  - double, float, half float, scaled float : 소수 데이터 저장

Field datatype

- Date : 문자열, 정수형으로 입력을 받고 내부적으로 long type으로 변환하여 저장
- Boolean : True/False
- Binary : 1/0
- Range
  - date_range, integer_range, float_range, long_range, double_range : 범위 데이터를 저장

Multi Field

- Multi Field를 사용하여 하나의 Field를 여러 datatype으로 저장하는 것도 가능
- 하나의 Field를 여러 datatype으로 저장하고 용도에 맞게 활용 가능

```
PUT example
{
 "mappings": {
  "properties": {
   "city": {
    "type": "text",
    "fields": {
     "raw": {
      "type":  "keyword"
     }
…
```

```
GET example/_search
{
 "query": {
  "match": {
   "city": "york"
  }
 },
 "sort" : { "city.raw": "asc" }
}
```

Dynamic Mapping

- Dynamic Mapping은 사전에 Index에 Mapping이 정의되지 않았거나 처음 색인된 Field가 존재하는 경우에 생성됨
- Field datatype에 가장 넓은 범위의 datatype이 지정 됨
- 최적의 색인, 검색 성능을 위해서 Static Mapping을 사전에 정의하는 것을 지향

Text vs Keyword

- 문자열 데이터를 저장할 때 Text와 Keyword datatype을 용도에 맞게 구분해야함

특징	Text	Keyword
문자열 분석	색인시 문자열 분석	문자열 분석 없음
Full-Text 검색 지원	가능	불가능
색인 속도	느림	빠름

Analysis

- Analysis(분석)는 Elasticsearch의 full text 검색을 가능하게 하는 필수 요소
- 분석 과정을 통해 문자열 데이터를 token(term)으로 분리
- Analyzer를 통해 분석이 이루어지며 기본적으로 제공되는 Analyzer를 활용하거나 사용자가 임의로 Analyzer를 정의할 수 있음
- Index의 Settings의 analysis에 Analyzer를 정의하거나 Mapping의 개별 Field에 Analyzer를 따로 지정할 수 있음
- 한번 Setting과 Mapping에 지정된 Analyzer는 변경 불가

Analyzer

- Analyzer는 Character filter, Tokenizer, Token filter로 구성
  - Character filter : 문자열 데이터를 분석하기 좋은 형태로 가공
  - Tokenizer : 문자열 데이터를 의미를 가진 단위인 token으로 분리
  - Token filter : 분리된 token을 검색에 좋은 형태로 변환

Analyzer

- Standard Analyzer 예시

Standard Analyer 구성
-Standard Tokenizer
-Lower case Token Filter

{"analyzer": "standard",
"text": "The QUICK Brown-Foxes jumped over the lazy dog's bone."}

↓ **Standard Tokenizer**

[The, QUICK, Brown, Foxes, jumped, over, the, lazy, dog's, bone]

↓ **Lower case Token Filter**

[the, quick, brown, foxes, jumped, over, the, lazy, dog's, bone]

Analyzer

- Snowball Analyzer 예시

Snowball Analyer 구성
-Whitespace Tokenizer
-Lower case Token Filter
-Stop Token Filter
-Snowball Token Filter

{"analyzer": "snowball",
"text": "The QUICK Brown-Foxes jumped over the lazy dog's bone."}

↓ **Whitespace Tokenizer**

[The, QUICK, Brown, Foxes, jumped, over, the, lazy, dog's, bone]

↓ **Lower case, stop, snowball Token Filter**

[quick, brown, fox, jump, over, lazi, dog, bone]

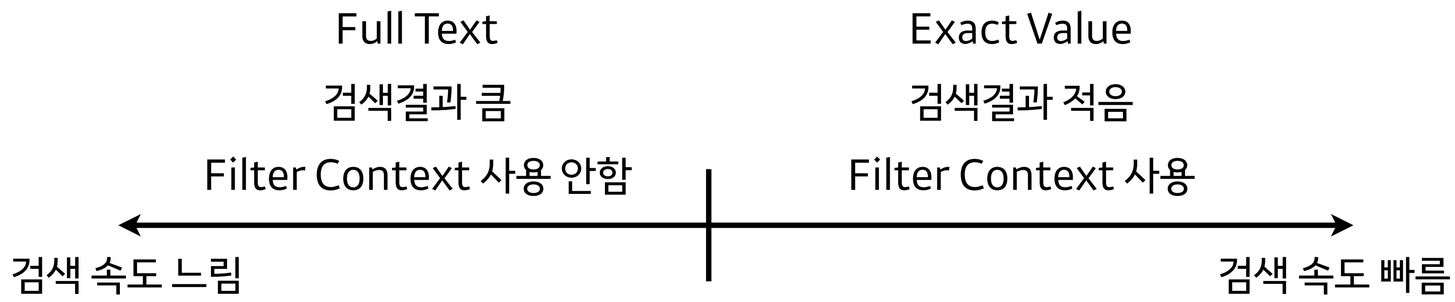
SEARCH

Search

- Search(검색)는 Index에 저장된 Document 중 검색 조건에 부합하는 Document를 찾는 것
- Elasticsearch에서 제공하는 Query DSL을 통해 다양한 검색 조건을 만들 수 있음

Search Performance

- 목적에 부합하는 최적의 Search query를 작성해야 검색 성능을 높일 수 있음
- 검색 성능에 영향을 끼치는 것
  - Full Text / Exact Value
  - 검색 결과 크기(Size)
  - Filter Context 사용 유무



Full Text Query

- Full Text query는 Elasticsearch의 Full Text Search 기능을 사용하는 것
- 색인시 분석된(Analyzed) 필드를 검색할 때 사용함
- Document가 검색 조건에 부합하는 정도인 Score를 계산하여 높은 순서대로 반환

Match

- Match query는 대표적인 Full Text query
- Document의 분석된 문자열 필드를 대상으로 검색 조건에 있는 문자열과 얼마나 일치하는지 계산하여 Score가 높은 순서대로 Document를 반환
- Match query를 수행하는 과정에서 문자열 분석과 점수 계산이 필요함

Match

- 분석된 문자열 Field를 대상으로 검색 조건과 일치하는 정도를 계산(Scoring)하여 반환

```
GET example/_search
{
  "query": {
    "match" : {
      "comment" : {
        "query" : "I like this cheese burger"
      }
    }
  }
}
```

[return]
"comment" : leo **like cheese burger**
"comment" : leo hate ham **burger**
…

Score

- Score는 full text 검색에서 문서가 검색 문자열과 얼마나 일치하는지 측정한 것

```
GET example/_search
{
  "query": {
    "match" : {
      "comment" : {
        "query" : "I like this cheese burger"
      }
    }
  }
}
```

검색 문자열 Analyze

I, like, this, cheese, burger

검색

Term	doc_1	doc_2
leo	O	O
like	**O**	
hate		O
cheese	**O**	
burger	**O**	**O**
ham		O

doc_1 : 3개 doc_2 : 1개

[return]
"comment" : leo **like cheese burger**
"comment" : leo hate ham **burger**
…

Score

- Score는 TF, IDF, Field Length를 이용하여 계산됨
- TF(Term Frequency) : Term이 해당 Document에 등장하는 빈도
- IDF(Inverse Document Frequency) : Term이 전체 Index에서 등장하는 빈도
- Field Length : Term이 포함된 Field의 길이

Score↑ : [TF↑ · IDF↓ · Field Length↓]

Term

- Term query는 검색 조건과 정확하게 일치하는 Field를 찾을 때 사용
- 점수를 계산하는 과정이 없기 때문에 검색 부하가 작음
- Text Field 보다는 분석되지 않은 Keyword Field를 검색할 때 적합함

```
GET example/_search
{
  "query": {
    "term" : {
      "name" : {
        "value" : "jemin"
      }
    }
  }
}
```

[return]
"name": "jemin"
…
이름이 정확하는 문서만 반환됨

Size

- Size는 Search query에 의해 반환되는 Document의 최대 개수를 결정
- 반환되는 Document의 수가 많으면 검색 시간이 증가
- 검색 결과의 offset을 from으로 설정 가능

```
GET example/_search
{
  "from" : 10, "size" : 100,
  "query" : {
    "match" : { "comment" : "I love pizza" }
  }
}
```

from + size는 10000을 넘을 수 없으며
from+size 만큼의 문서를 우선 검색 후 offset 적용

Bool

- Bool query를 이용하여 다양한 검색 조건을 만들어 낼 수 있음
- Bool query에서 지원하는 조건문
  - must : 반드시 충족해야 하는 조건. Score 계산 과정에 포함됨
  - filter : 반드시 충족해야 하는 조건. Score 계산 과정에 포함 안됨. Cache에 결과 저장.
  - should : 충족하면 Score계산 과정에 포함되는 조건
  - must_not : 충족하면 안되는 조건. Score 계산 과정에 포함 안됨. Cache에 결과 저장.

Bool

- Bool query의 조건문은 크게 2가지로 나눌 수 있음
- Scoring에 기여하는 조건문
  - must, should
- Scoring에 기여하지 않고 Caching되는 조건문
  - filter, must_not
- 검색 성능을 높이기 위해서 chache에 저장되는 filter와 must_not을 사용

Filter

- Filter를 사용하여 Scoring 과정에 포함되는 Document 수를 줄일 수 있음
- Filter의 결과는 메모리의 Cache에 저장되기 때문에 훨씬 빠르게 결과가 반환됨
- must_not도 같은 용도로 활용

```
GET example/_search
{
  "query" : {
    "bool" : {
      "should" :  [ { "match" : {"comment" : "love pizza" } } ],
      "filter" : [ { "term": {"company" : "kakao" }}]
    }
  }
}
```

Cache

doc id	company : kakao
1	1
2	0
3	0
4	1

Aggregation

- Aggregation은 Search를 기반으로 수집된 데이터를 분석하는 것
- 대표적으로 Aggregation 방식에는 Bucket과 Metric이 있음

Bucket Aggregation

- Bucket Aggregation은 검색 조건에 부합하는 Document들을 Grouping 하는 것
- 주로 사용되는 Bucket은 Range, Histogram, Terms가 있음

Range Bucket

- Range Bucket은 숫자 필드의 범위를 기준으로 Document bucket을 생성하는 것

```
GET example/_search
{
  "aggs" : {
    "price_ranges" : {
      "range" : {
        "field" : "price",
        "ranges" : [
          { "to" : 100.0 },
          { "from" : 100.0, "to" : 200.0 },
          { "from" : 200.0 }
        ]
      }
    }
  }
}
```

```
"buckets" : {
  "key": "*-100.0",
  "to": 100.0,
  "doc_count": 2
},
{
  "key": "100.0-200.0",
  "from": 100.0,
  "to": 200.0,
  "doc_count": 2
},
{
  "key": "200.0-*",
  "from": 200.0,
  "doc_count": 3
}
```

Histogram Bucket

- Histogram Bucket은 숫자 필드를 일정 간격으로 나눠서 Document bucket을 생성

```
GET example/_search
{
  "aggs" : {
    "connection_ranges" : {
      "histogram" : {
        "field" : "connection",
        "interval" : 500
        ]
      }
    }
  }
}
```

```
"buckets" : {
  "key": 0,
  "doc_count": 1
},
{
  "key": 500,
  "doc_count": 24
},
{
  "key": 1000,
  "doc_count": 102
}
...
```

Terms Bucket

- Terms Bucket은 keyword 필드의 문자열을 기준으로 Document Bucket을 생성

```
GET example/_search
{
  "aggs" : {
    "es_version" : {
      "terms": { "field" : "es_version.keyword" }
    }
  }
}
```

```
"buckets" : {
  "key": "6.1",
  "doc_count": 13
},
{
  "key": "6.2",
  "doc_count": 5
},
...
{
  "key": "7.3"
  "doc_count": 10
}
...
```

Metric Aggregation

- Metric Aggregation은 Document Field의 min, max, sum, avg 값을 계산

```
POST /example/_search
{
  "aggs" : {
    "avg_score" : { "avg" : { "field" : "score" } }
  }
}
```

```
{
  ...
  "aggregations": {
    "avg_score": {
      "value": 75.0
    }
  }
}
```

Deep Aggregation

- Aggregation은 여러개를 중첩할 수 있음
- Elasticsearch Cluster 성능을 고려하여 여러 단계의 Aggregation을 중첩하는 것은 지양해야 함

```
"aggregations":{
      "A":{
         "size":100…
       },
       "aggregations":{
         "B":{
            "size":70…
          },
          "aggregations":{
            "C":{
               "size":50…
            }}}
```

→ 최대 100 x 70 x 50 개 Bucket 생성

마치며

INDEXING & SEARCH

Index Life Cycle

Search Performance

Support by @@esse

마치며

- 인프라요청 아지트 : Elasticsearch 관련 업무 요청
  (https://kakao.agit.in/g/238059/wall)

- Elasticsearch 연구실 : Elasticsearch 관련 정보 업데이트
  (https://kakao.agit.in/g/300012335/wall)

- 자료 출처
  Elasticsearch - The Definitive Guide
  Elasticsearch Reference Page
  Jongmin's Blog (Elasticsearch 김종민 Engineer)
  @@esse (Benjamin.butn, James.js, Homi.e)

Q & A

오늘 교육 어떠셨나요?

설문조사
부탁드립니다.

kakao