# Elasticsearch 검색엔진 활용(5회차)

# 목차

- 검색엔진으로 Elasticsearch 활용하기
- Q & A

# 검색엔진으로 Elasticsearch 활용하기

인덱스 생성 과정

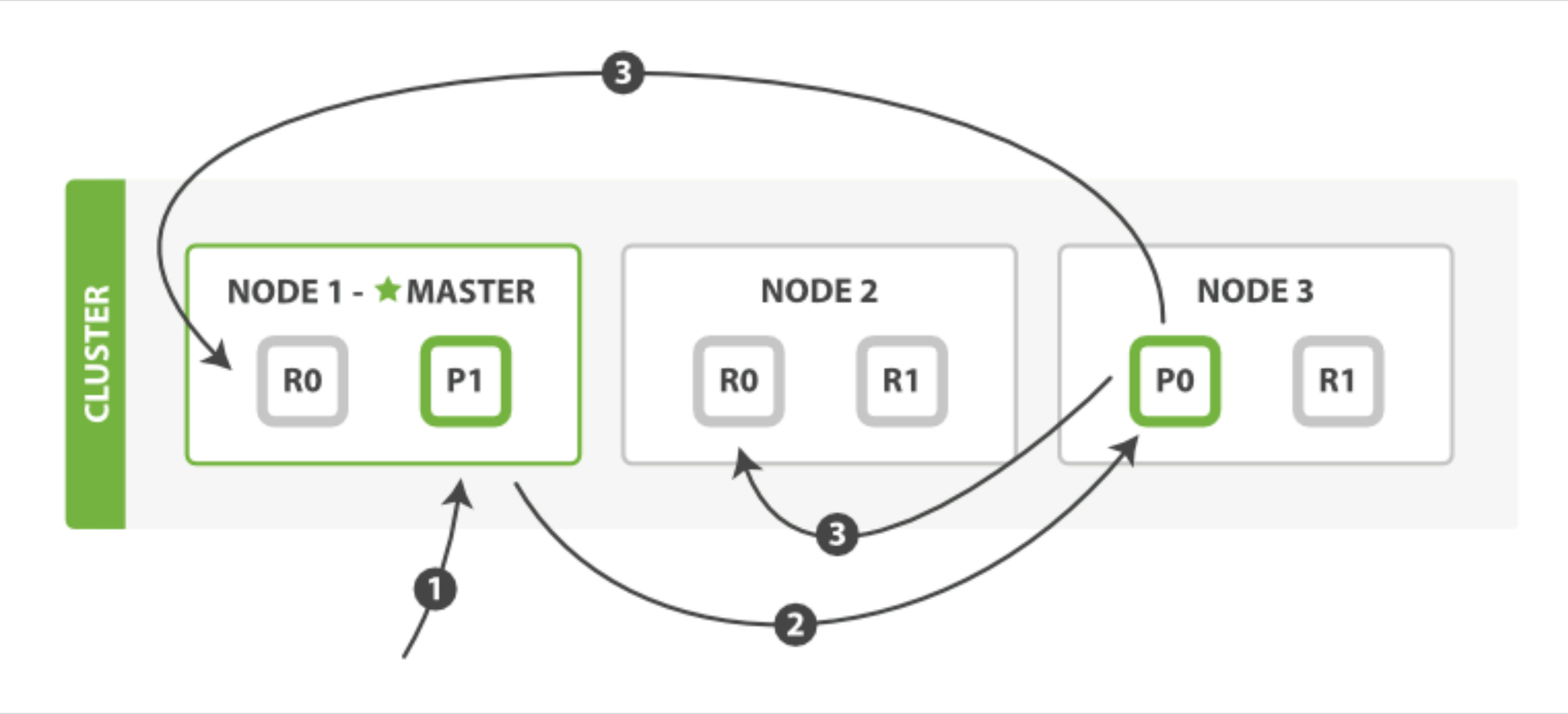
분석기 변경 방법

쿼리 생성

# Elasticsearch 인덱스 생성 과정

## 색인의 필수 조건

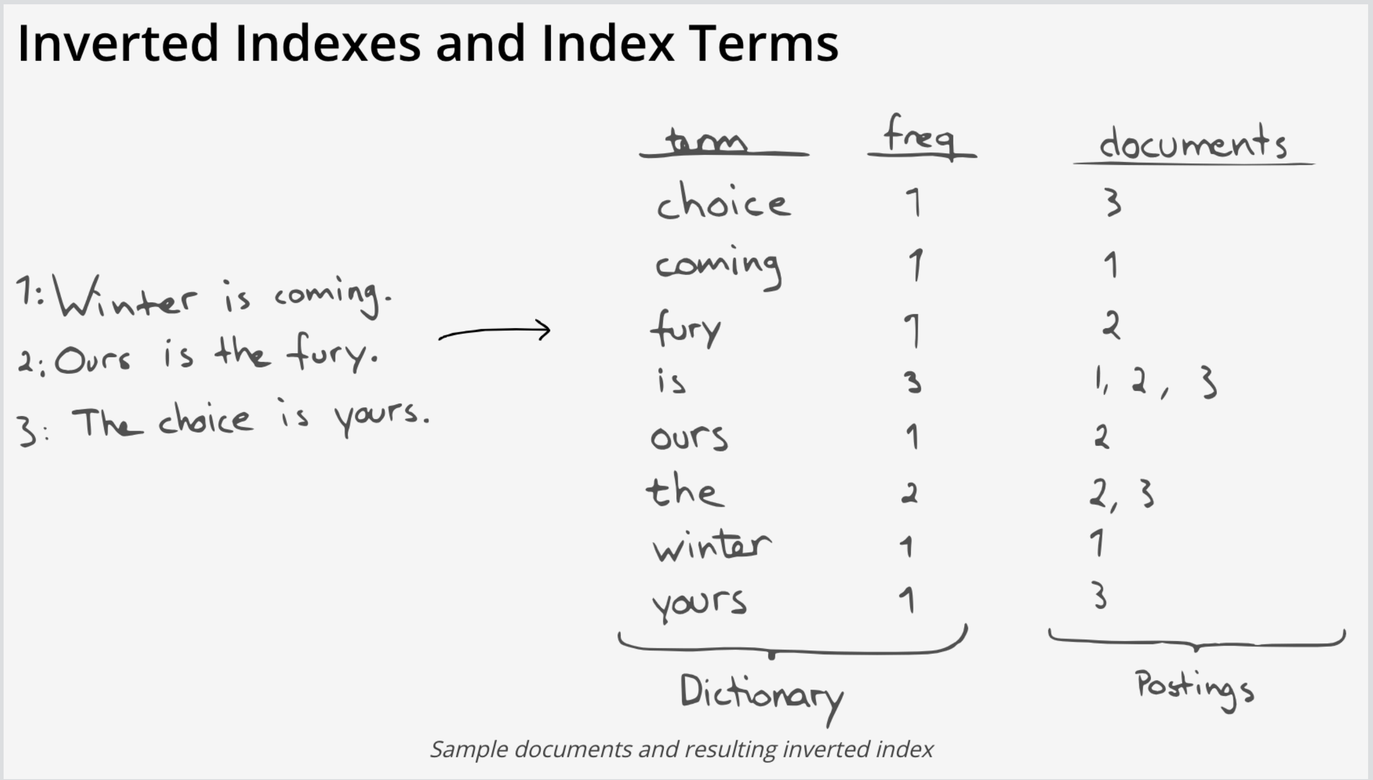
1) 프라이머리 샤드가 항상 제일 먼저 writing 되어야 한다
2) 프라이머리 샤드가 writing 이 전부 완료된 이후에 리플리카 샤드로 복제를 한다

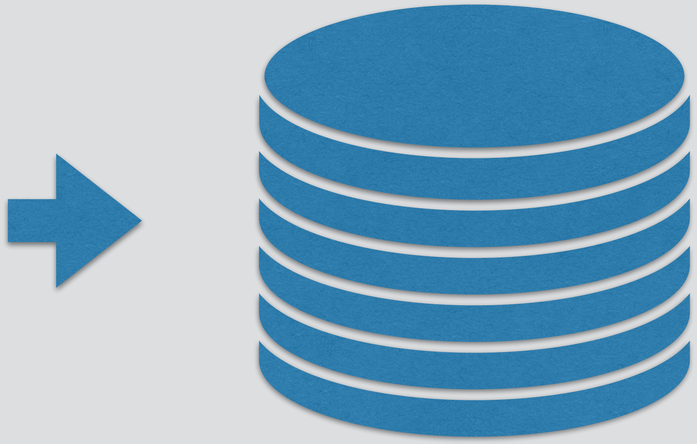


# Elasticsearch 인덱스 생성 과정

## Inverted Index

- 색인되어 들어온 문서는 Inverted Index 형태로 세그멘트에 저장
- 정의된 Analyzer 에 의해 토크나이징 된 단어를 기준으로 색인
- 별도로 정의하지 않으면 기본 standard analyzer 가 적용

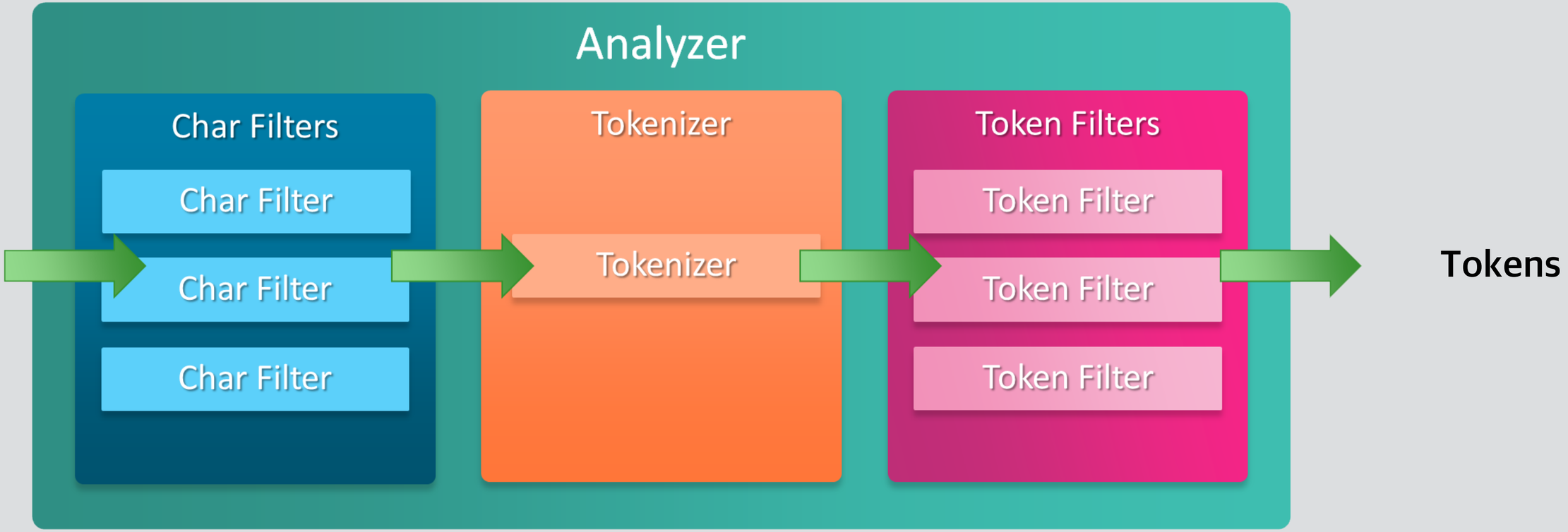


Segment

# Elasticsearch Analysis

Analysis
- 메일 본문같은 full-text 에 대해 검색이 될 수 있도록 분석하는 과정



# Elasticsearch Analysis

Analyzer 의 구성요소

1) Character filters

- 원본 text 를 사전에 가공하는 과정
- 설정하지 않거나 다중으로 필터 설정 가능

ex) html 태그 제거, 패턴매칭(123-456-789 -> 123_456_789),
Hindu-Arabic 숫자 (٠١٢٣٤٥٦٧٨٩) 를 아라비아 숫자로 매핑 (0123456789)

2) Tokenizer

- 어떤 방식으로 원본 text 를 토크나이징 할지를 결정
- 토크나이징 된 term 은 token 이라 부름
- 하나의 Tokenizer 만 설정 가능

ex) 공백을 기준으로 토크나이징, You are a boy!!! -> You / are / a / boy!!!

3) Token filters

- Tokenizer 에 의해 결정된 token 들에 대해 가공
- 설정하지 않거나 다중으로 필터 설정 가능

ex) stopwords 로 지정된 단어 제거, You / are / a / boy -> you / boy

# Elasticsearch Analysis

Analyzer

- 기본은 standard analyzer, 변경 가능, 사용자 정의 analyzer 설정 가능
- _analyzer API 를 통해 분석되는 토큰 확인 가능(테스트 목적)
- 실제 인덱스 analysis 를 정의하여 해당 인덱스를 대상으로 _analyze API 실행

```
POST _analyze
{
  "text": "Winter is Coming!!!"
}

POST _analyze
{
  "tokenizer": "standard",
  "filter": [ "lowercase", "asciifolding" ],
  "text": "Is this déja vu?"
}
```


```
{
  "tokens": [
    {
      "token": "winter",
      "start_offset": 0,
      "end_offset": 6,
      "type": "<ALPHANUM>",
      "position": 0
    },
    {
      "token": "is",
      "start_offset": 7,
      "end_offset": 9,
      "type": "<ALPHANUM>",
      "position": 1
    },
    {
      "token": "coming",
      "start_offset": 10,
      "end_offset": 16,
      "type": "<ALPHANUM>",
      "position": 2
    }
  ]
}
```

# Elasticsearch Analysis

standard Analyzer
- character filter 미사용
- Standard Tokenizer 사용, 공백기준, 문자열기준 의미있는 토큰 생성
- Standard Token Filter, Lower Case Token Filter, Stop Token Filter 사용

```
POST _analyze
{
   "analyzer": "standard",
   "text": "Winter is coming!!!"
}

POST _analyze
{
  "tokenizer": "standard",
  "filter": [ "lowercase", "asciifolding" ],
  "text": "Is this déja vu?"
}
```

# Elasticsearch Analysis

whitespace Analyzer
- character filter 미사용
- Whitespace Tokenizer 사용, space 기준으로 토큰 생성
- token filter 미사용

```
POST _analyze
{
   "analyzer": "whitespace",
   "text": "Winter is coming!!!"
}

POST _analyze
{
  "char_filter": [ "html_strip" ],
  "tokenizer": "whitespace",
  "filter": [ "uppercase" ],
  "text": "<B>This is mixed analyzer</B>"
}
```

# Elasticsearch Analysis

## Analyzer 의 설정 방법

1) 이미 정의되어 있는 analyzer 를 그대로 가져다가 사용하는 방식
- analyzer 자체에서 제공되는 parameter 설정만 사용하여 토큰 생성

2) 이미 정의되어 있는 tokenizer 에 character filter, token filter 를 조합하여 사용하는 방식
- tokenizer 를 선택하고 tokenizer parameter 추가
- character filter 와 token filter 를 사용자 정의하여 추가로 설정하여 토큰 생성

3) plugin 형태로 제공되는 외부 analyzer 를 하여 사용하는 방식
- ES 에서 기본적으로 제공되지 않는 외부 애널라이저를 플러그인으로 설치하여 토큰 생성

# Elasticsearch Analysis

## 이미 정의되어 있는 analyzer 를 그대로 가져다가 사용하는 방식

```
PUT index_analyzer_settings1
{
  "settings": {
    "analysis": {
      "analyzer": {
        "my_analyzer" : {
          "type": "standard",
          "max_token_length": 5,
          "stopwords": "_english_"
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "comment": {
        "type": "text",
        "analyzer": "my_analyzer"
      }
    }
  }
}
```

# Elasticsearch Analysis

```
POST index_analyzer_settings1/_analyze
{
  "analyzer": "my_analyzer",
  "text": "<B>This is Standard Analyzer</B>"
}

POST index_analyzer_settings1/_doc
{
  "comment": "<B>This is Standard Analyzer</B>"
}

GET index_analyzer_settings1/_search
{
  "query": {
    "match": {
        "comment": "standard"
    }
  }
}
```

# Elasticsearch Analysis

## 이미 정의되어 있는 tokenizer 에 character filter, token filter 를 조합하여 사용하는 방식

```
PUT index_analyzer_settings2
{
  "settings": {
    "analysis": {
      "analyzer": {
        "my_analyzer" : {
          "type": "custom",
          "char_filter": [ "html_strip" ],
          "tokenizer": "standard",
          "filter": [ "uppercase" ]
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "comment": {
        "type": "text",
        "analyzer": "my_analyzer"
      }
    }
  }
}
```

# Elasticsearch Analysis

```
POST index_analyzer_settings2/_analyze
{
  "analyzer": "my_analyzer",
  "text": "<B>This is Standard Analyzer</B>"
}

POST index_analyzer_settings2/_doc
{
  "comment": "<B>This is Standard Analyzer</B>"
}

GET index_analyzer_settings2/_search
{
  "query": {
    "match": {
        "comment": "standard"
    }
  }
}
```

# Elasticsearch Analysis

```
PUT mixed_analyzer
{
   "settings": {
      "analysis": {
         "char_filter": {
            "my_char_filter": {
               "type": "mapping",
               "mappings": [ ":) => _happy_", ":( => _sad_" ]
            }
         },
         "tokenizer": {
            "my_tokenizer": {
               "type": "standard",
               "max_token_length": 20
            }
         },
        "filter": {
            "my_stop": {
               "type": "stop",
               "stopwords": ["and", "is", "the", "this"]
            }
         },
         "analyzer": {
            "my_analyzer": {
               "type": "custom",
               "char_filter": [ "html_strip", "my_char_filter" ],
               "tokenizer": "my_tokenizer",
               "filter": [ "lowercase", "my_stop" ]
            }
         }
      }
   }
}
```

# Elasticsearch Analysis

## plugin 형태로 제공되는 외부 analyzer 를 하여 사용하는 방식

```
PUT nori_analyzer
{
  "settings": {
    "analysis": {
      "tokenizer": {
        "nori_user_dict": {
          "type": "nori_tokenizer",
          "decompound_mode": "mixed",
          "user_dictionary": "userdict_ko.txt"
        }
      },
      "analyzer": {
        "my_analyzer": {
          "type": "custom",
          "tokenizer": "nori_user_dict"
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "comment": {
        "type": "text",
        "analyzer": "my_analyzer"
      }
    }
  }
}
```

# Elasticsearch Analysis

Nori Analyzer

- decompound_mode 별로 복합어에 대한 다른 토큰을 가져가는 설정

1) none : 단어를 분리하지 않고 그대로 제공
   ex) 가곡역 => 가곡역
2) discard(default) : 복합어는 버리고 복합어를 나눈 토큰으로 설정
   ex) 가곡역 => 가곡, 역
3) mixed : 복합어와 복합어를 나눈 토큰으로 설정
   ex) 가곡역 => 가곡역, 가곡, 역

- mecab-ko-dic dictionary 를 기본 사전으로 사용

- 사전 파일을 <token> [<token 1> ... <token n>] 로 정의

- index close/open 후 적용

# Elasticsearch Analysis

Analyzer 가 바뀌면 토큰도 변경

검색할 단어가 바뀌므로 중간에 Analyzer 를 변경하면 의도와 다른 결과를 가져오게 됨

특정 text type field 에 대해 Analyzer 를 변경하려면 반드시 _reindex 필요

# Elasticsearch Analysis

token 이나 token 의 frequency 를 확인하고 싶을 때는 Terms Vector 를 활용

```
POST /bank/account/25/_termvectors
{
    "fields": ["address"],
    "offsets" : true,
    "payloads" : true,
    "positions" : true,
    "term_statistics" : true,
    "field_statistics" : true
}
```

# Elasticsearch 분석기 변경 방법

Analyzer More..
https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html

Character filters More..
https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-charfilters.html

Tokenizer More..
https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-tokenizers.html

Token filters More..
https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-tokenfilters.html

# 실습

Nori Analyzer
https://www.elastic.co/guide/en/elasticsearch/plugins/current/analysis-nori-analyzer.html

https://github.com/benjamin-btn/ES7-Tutorial/tree/master/ES-Tutorial-5

# Elasticsearch 쿼리 생성

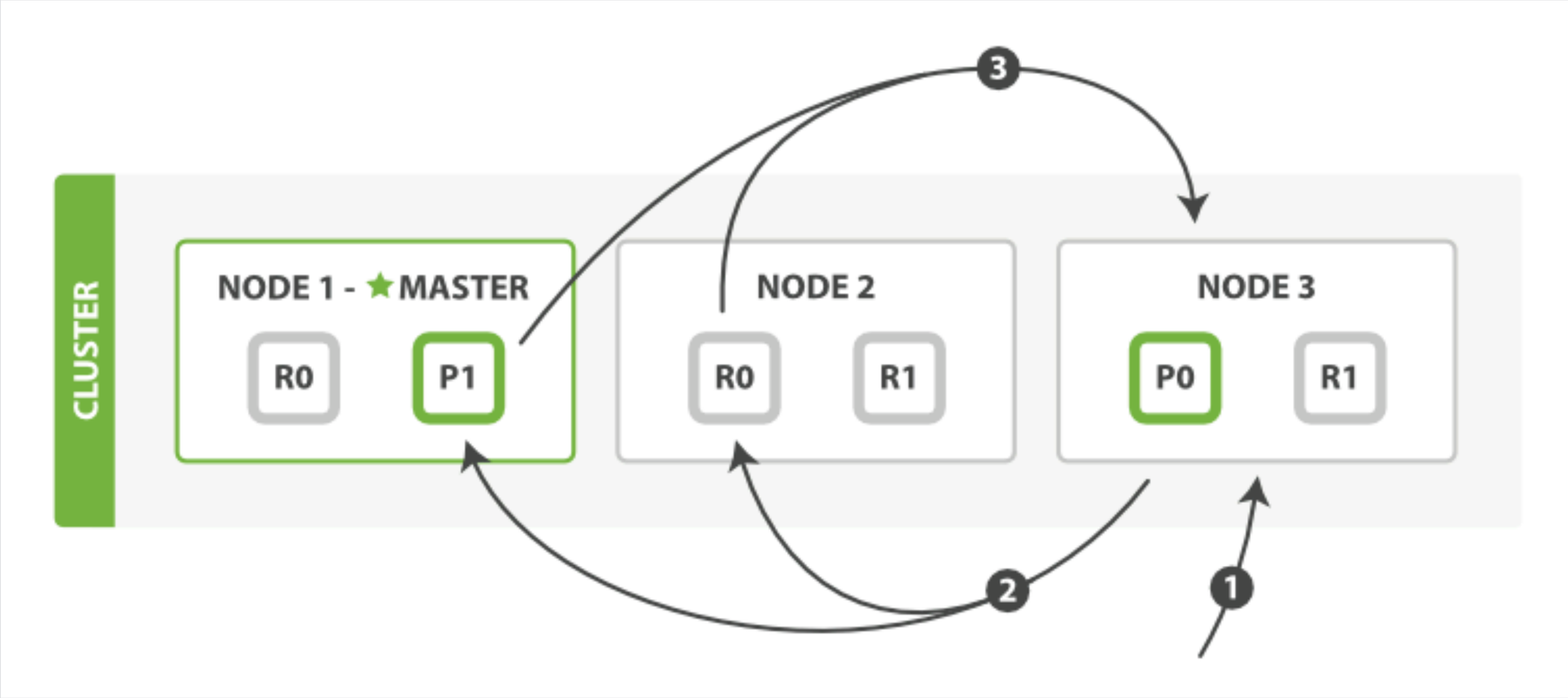
## 검색하기

- _search API 를 통해 원하는 도큐멘트를 검색
- URI 혹은 HTTP request body 를 통해 쿼리 작성
- 문서의 위치를 찾는 Query 와 문서를 가져오는 Fetch 과정으로 구성

# Elasticsearch 쿼리 생성

Query Phase

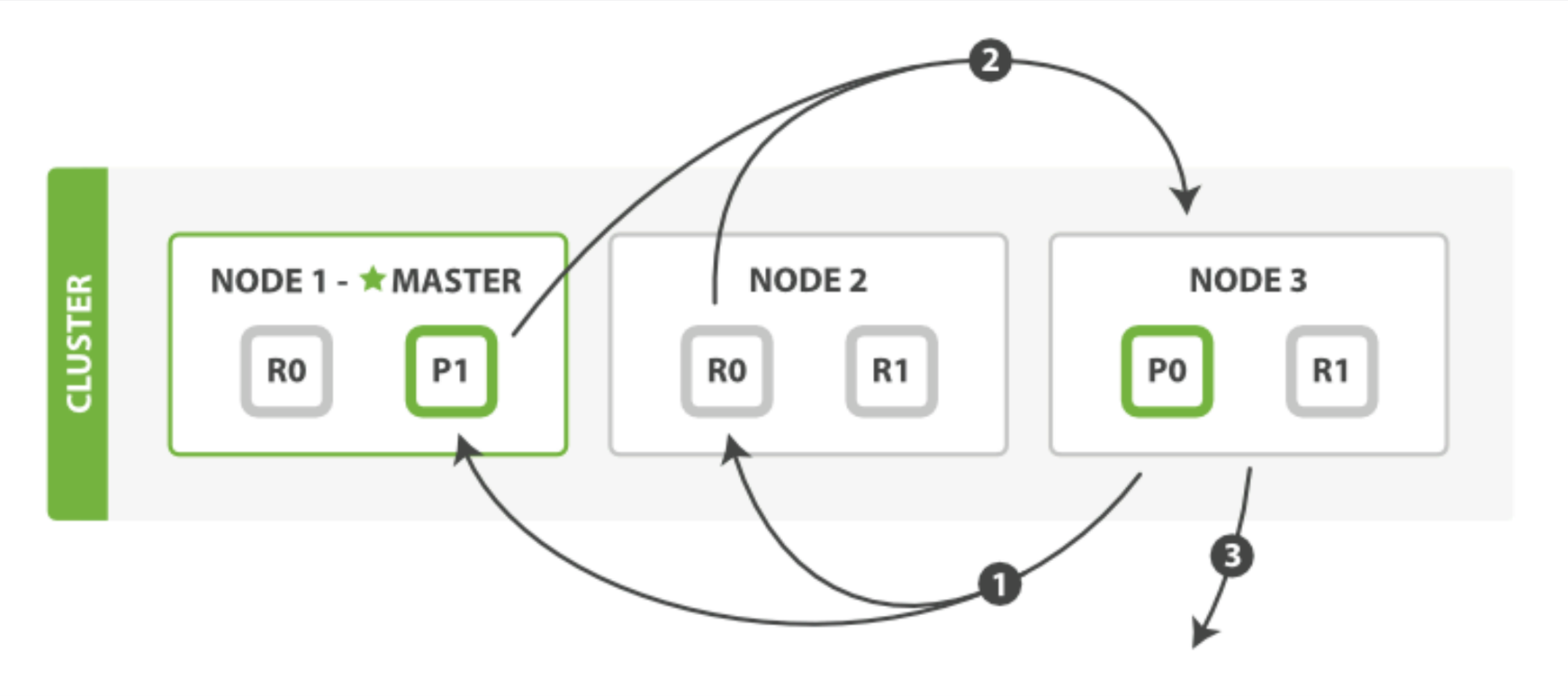
- 쿼리를 받아 문서가 어느 노드 어느 샤드에 있는지를 찾는 과정
- from, size 를 계산하여 빈 queue 를 생성
- 전체 노드, 샤드에 문서가 있는지를 확인, 노드들도 로컬에 queue 를 생성
- queue 에 검색된 문서의 id 를 넣고 결과를 score 기준으로 sorting 후 리턴



# Elasticsearch 쿼리 생성

Fetch Phase

- 리턴받은 doc id 를 기준으로 multi get 요청
- 노드별로 리턴받은 문서를 클라이언트에게 리턴



# 실습

검색엔진 실습 인덱스 생성
https://github.com/benjamin-btn/ES7-Tutorial/tree/master/ES-Tutorial-5/tools/bulkapi

# Elasticsearch 쿼리 생성

## URI Search

- URI 에 request parmeters 를 통해 검색 질의
- 한정된 검색 옵션만 사용 가능(Quick Test)

```
GET bank/_search?from=0&size=100&q=address:Fleet&sort=age:asc
```

# Elasticsearch 쿼리 생성

Request Body Search
- Query DSL 을 이용해 HTTP Body 정의 후 질의
- query 구문을 사용하여 질의

```
GET bank/_search
{
   "query" : {
      "term" : {
         "city.keyword": "Mulino"
      }
   }
}
```

# Elasticsearch 쿼리 생성

- from, size parameter 로 pagination
- 기본은 from:0, size:10
- from + size 가 기본으로는 10000 까지만 허용
- 더 필요하면 index.max_result_window 조정 필요

```
GET bank/_search
{
   "from":0, "size": 2,
   "query" : {
      "match" : {
         "address": "Fleet"
      }
   }
}

PUT bank/_settings
{
  "index.max_result_window": 10001
}
```

# Elasticsearch 쿼리 생성

- key field 기준으로 sort 가능
- 오름차순(asc), 내림차순(desc) 정렬

```
GET bank/_search
{
   "sort": {
      "age": "desc"
   }
}
```

# Elasticsearch 쿼리 생성

- score 가 계산되는 과정을 확인하는 explain

```
GET bank/_search
{
   "explain": true,
   "from":0, "size": 2,
   "query" : {
      "match" : {
         "address": "Fleet"
      }
   }
}
```

- 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
- TF 가 높을수록, IDF 가 낮을수록, Field Length 가 낮을수록 스코어가 높아짐

# Elasticsearch 쿼리 생성

- _source filtering 으로 원하는 데이터만 필터링

```
GET bank/_search
{
    "_source": false,
    "sort": { "age": "desc" }
}
GET bank/_search
{
    "_source": [ "age","gender" ],
    "sort": { "age": "desc" }
}
GET bank/_search
{
    "_source": [ "*ge*" ],
    "sort": { "age": "desc" }
}
```

# Elasticsearch 쿼리 생성

- highlight로 검색결과 하이라이팅

```
GET bank/_search
{
  "_source": ["account_number", "firstname", "lastname"],
  "query": {
    "query_string": {
      "query": "Fleet"
    }
  },
  "highlight": {
    "fields": {
      "address": {}
    }
  }
}
```

# Elasticsearch 쿼리 생성

Request Body Search more..
https://www.elastic.co/guide/en/elasticsearch/reference/current/search-request-body.html

# Elasticsearch 쿼리 생성

Query DSL (Domain Specific Language)
- JSON 기반의 Elasticsearch 쿼리를 정의하는 언어

1) Leaf query clause
- 자체적으로 쿼리를 할 수 있는 완성된 검색 쿼리 절
- match, term, range 등

2) Compound query clause
- Leaf query 혹은 Compound query 를 혼합해주는 검색 쿼리 절
- bool, boosting 등

# Elasticsearch 쿼리 생성

Full Text Query(Query Context query)

- 쿼리문을 analyze 하여 검색, 스코어가 가장 높은 문서순으로 노출

- 검색 쿼리 절이 얼마나 문서에 잘 매치되는지 유사성을 확인

- 검색 쿼리와의 매칭율에 따라 _score 를 부여

ex) elastic search 검색 -> elastic 포함된 문서, search 포함된 문서 검색

- match, match_phrase, match_phrase_prefix, query_string 등

- 각 쿼리는 key:value 형태로 바로 검색하거나 쿼리 종류별로 제공하는 parameter 는 사용할 수 있음

쿼리 종류 - https://www.elastic.co/guide/en/elasticsearch/reference/current/full-text-queries.html
쿼리 별 parameter - https://opendistro.github.io/for-elasticsearch-docs/docs/elasticsearch/full-text/#options

# Elasticsearch 쿼리 생성

match Query
- 쿼리문의 analyze 된 토큰으로 유사한 문서 검색
- key:value 형태 검색

```
GET bank/_search
{
   "query": {
      "match": {
         "address": "345 Fleet"
      }
   }
}
```

# Elasticsearch 쿼리 생성

match Query
- boost parameter 로 검색 score 가중치 적용 가능

```
GET bank/_search
{
   "query": {
      "match": {
         "address": {
            "query": "345 Fleet",
            "boost": 2.0
         }
      }
   }
}
```

# Elasticsearch 쿼리 생성

match_phrase Query

- analyze 된 쿼리의 토큰 중 해당 value 를 순서대로 합쳐서 쿼리 구문을 만들어 검색

```
GET bank/_search
{
    "query": {
        "match_phrase": {
            "address": "Fleet Walk"
        }
    }
}
```

# Elasticsearch 쿼리 생성

match_phrase_prefix Query
- match_phrase 와 동일한 방식으로 문서 검색
- 마지막 문자를 와일드 카드로 검색

```
GET bank/_search
{
   "query": {
      "match_phrase_prefix": {
         "address": "425 Fleet W"
      }
   }
}
```

# Elasticsearch 쿼리 생성

multi_match Query
- fields parameter 에 정의된 field 들에서 검색
- match 쿼리를 여러개의 필드들에 대해서 검색

```
GET multi_match_index/_search
{
   "query": {
      "multi_match": {
         "query": "ks",
         "fields": [ "first", "comment"]
      }
   }
}
```

# Elasticsearch 쿼리 생성

query_string Query
- match, multi_match, match_prase_prefix 의 기능을 할 수 있는 쿼리
- match_prase_prefix 기능은 와일드카드(*) 추가로 가능

```
GET bank/_search
{
  "query": {
      "query_string" : {
         "query" : "Walk Flee*",
         "fields": [ "address", "employer"]
      }
   }
}
```

# Elasticsearch 쿼리 생성

Term Level Query(Filter Context 에 주로 쓰이는 쿼리)

- 정확히 일치하는 용어만 검색, analyze 되지 않음, 범위 검색

- 정확히 일치하는 문서만 검색되기 때문에 _score 무의미하지만 스코어링이 되지 않는 것은 아님

- 쿼리로 사용하는 전체 string 을 기준으로 정확히 일치하는 token 이 있는 경우에만 검색 가능

- keyword field 기반으로 검색

ex) date 필드의 값이 2015년과 2018년 사이에 속해있는지, 이름이 정확히 benjamin 인 문서 검색
status 필드의 값이 'ok' 인지

- 각 쿼리는 key:value 형태로 바로 검색하거나 쿼리 종류별로 제공하는 parameter 는 사용할 수 있음

- term, terms, range 등

쿼리 종류 - https://www.elastic.co/guide/en/elasticsearch/reference/current/term-level-queries.html
쿼리 별 parameter - Term Level Query 는 parameter 가 다양하지 않아 각 쿼리문에서 확인

# Elasticsearch 쿼리 생성

term Query
- 역색인 된 토큰 중에 term 이 정확히 일치할때만 결과 리턴
- keyword 필드에 대해서만 쿼리 가능
- 아래 예제는 dynamic indexing 된 sub field 인 {field}.keyword 를 사용하는 것을 전제로 함
- mapping 시 keyword 로 미리 설정이 되었다면 "gender": "M" 형태로 검색
- key:value 형태 검색

```
GET bank/_search
{
   "query": {
      "term" : {
         "gender.keyword": "M"
      }
   }
}
```

# Elasticsearch 쿼리 생성

term Query
- boost parameter 로 검색 score 가중치 적용 가능

```
GET bank/_search
{
   "query": {
      "term" : {
         "gender.keyword": {
            "value": "M",
            "boost": 2.0
         }
      }
   }
}
```

# Elasticsearch 쿼리 생성

terms Query
- 여러개의 용어에 대해 검색 가능
- or 검색과 유사한 기능

```
GET bank/_search
{
   "query": {
      "terms" : {
         "gender.keyword": ["F","M"]
      }
   }
}
```

# Elasticsearch 쿼리 생성

range Query
- numeric, date, geo field 에 대해서만 가능
- gte(greater than || equal), gt, lte(less than || equal), lt parameter 사용

```
GET bank/_search
{
   "query": {
      "range" : {
         "age": {
            "gte": 25,
            "lte": 30
         }
      }
   }
}
```

# Elasticsearch 쿼리 생성

wildcard Query
- term level 쿼리 중 와일드카드를 쓸 수 있는 쿼리
- analyze 안됨

```
GET bank/_search
{
   "query": {
      "wildcard" : {
         "lastname.keyword": "D*e"
      }
   }
}

GET bank/_search
{
   "query": {
      "wildcard" : {
         "lastname.keyword": "*e*"
      }
   }
}
```

# Elasticsearch 쿼리 생성

Query DSL (Domain Specific Language)
- JSON 기반의 Elasticsearch 쿼리를 정의하는 언어

1) Leaf query clause
- 자체적으로 쿼리를 할 수 있는 완성된 검색 쿼리 절
- match, term, range 등

2) Compound query clause
- Leaf query 혹은 Compound query 를 혼합해주는 검색 쿼리 절
- bool, boosting 등

# Elasticsearch 쿼리 생성

## bool 쿼리

```
GET bank/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "address": {
              "query": "Fleet"
            }
          }
        }
      ],
      "filter": [
        {
          "term":  {
            "gender.keyword": "F"
          }
        },
        {
          "range": {
            "age": {
              "gte": "30"
            }
          }
        }
      ]
    }
  }
}
```

# Elasticsearch 쿼리 생성

bool 쿼리
- 하나 이상의 boolean clause 가 사용됨
- must, filter, should, must_not 항목으로 구성

1) must - 문서에 일치하는 항목, 스코어 계산
2) filter - 문서에 일치하는 항목, 스코어 0, 보통 filter context 실행
3) should - 문서에 일치하는 항목,
   must 나 filter 항목이 없으면 적어도 하나의 쿼리절과 일치되는 결과 리턴
4) must_not - 문서에 일치하지 않는 항목, 보통 filter context 실행

자주 사용되는 filter, must_not 절은 캐싱의 대상이 됨

# Elasticsearch 쿼리 생성

must
- 문서에 일치하는 항목, 스코어 계산

```
GET bank/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "address": {
              "query": "Fleet"
            }
          }
        }
      ]
    }
  }
}
```

# Elasticsearch 쿼리 생성

filter

- 문서에 일치하는 항목, 스코어 0, term level query 권고

```
GET bank/_search
{
  "query": {
    "bool": {
      "filter": [
        {
          "match": {
            "address": {
              "query": "Fleet"
            }
          }
        }
      ]
    }
  }
}
```

# Elasticsearch 쿼리 생성

should
- 문서에 일치하는 항목
- must 나 filter 항목이 없으면 적어도 하나의 쿼리절과 일치되는 결과 리턴
- minimum_should_match 가 없으면 default 로 1 이 적용됨
- must, filter 항목이 있으면 default 는 0 이 되므로 필요할 때에는 반드시 옵션 적용 필요
- minimum_should_match 옵션은 반드시 should 절 밑에 바로 설정
- must 나 filter 항목이 없는데 minimum_should_match 를 작성하지 않은 경우는 스코어링에만 기여

```
GET bank/_search
{
  "query": {
    "bool": {
      "should": [
        {
          "match": {
            "state": {
              "query": "MI",
              "boost": 2
            }
          }
        },
        {
          "term": {
            "gender.keyword": {
              "value": "M"
            }
          }
        }
      ],
      "minimum_should_match" : 1
    }
  }
}
```

# Elasticsearch 쿼리 생성

must_not

- 문서에 일치하지 않는 항목, 스코어 1 세팅, term level query 권고

```
GET bank/_search
{
  "query": {
    "bool": {
      "must_not": [
        {
          "match": {
            "address": {
              "query": "Fleet"
            }
          }
        }
      ]
    }
  }
}
```

# Elasticsearch 쿼리 생성

```
GET bank/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "term":  {
            "gender.keyword": "F"
          }
        }
      ],
      "filter": [
        {
          "range": {
            "age": {
              "lte": "30"
            }
          }
        }
      ],
      "must_not": [
        {
          "match": {
            "address": "Hope"
          }
        }
      ],
      "should": [
        {
          "match": {
            "state": {
              "query": "MI"
            }
          }

        },
        {
          "match": {
            "city": {
              "query": "Nogal"
            }
          }
        }
      ],
      "minimum_should_match" : 1
    }
  }
}
```

# Elasticsearch 쿼리 생성

Query DSL more..
https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html

# Q & A