Elasticsearch

2장 엘라스틱서치 기본 동작과 구조

강주영

2.1 색인

```
PUT [인덱스이름]/_doc/Id
POST [인덱스이름]/_doc
{
  문서내용
}

PUT my_index/_doc/2
POST my_index/_doc
{
  "title" : "hello world",
  "views" : 1234,
  "public" : true,
  "created" :
"2019-0117T14:05:01.234Z"
}
```

```
{
  "_index": "my_index",
  "_id": "2",
  "_version": 1,
  "result": "created",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 4,
  "_primary_term": 1
}
```

id값을 지정 안해도 _id 값이 자동생성되는데 그때는 PUT이 아니라 POST로 해야한다

2.1 조회

```
GET [인덱스이름]/_doc/id
GET [인덱스이름]/_doc/1
```


```
{
  "_index": "my_index",
  "_id": "1",
  "_version": 2,
  "_seq_no": 2,
  "_primary_term": 1,
  "found": true,
  "_source": {
    "title": "hello elasticsearch!",
    "views": 1234,
    "public": true,
    "created":
"2019-01-17T14:05:01.234Z"
  }
}
```

2.1 업데이트

```
POST [인덱스이름]/_update/id

POST my_index/_update/2
{
  "doc" : {
    "title" : "hello elasticsearch!"
  }
}
```

```
{
  "_index": "my_index",
  "_id": "2",
  "_version": 2,
  "result": "updated",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 6,
  "_primary_term": 1
}
```

2.1 검색

```
GET [인덱스이름]/_search
POST [인덱스이름]/_search

GET my_index/_search
{
  "query": {
    "match": {
      "title": "hello world"
    }
  }
}
```

```
{
  "took": 450,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 3,
      "relation": "eq"
    },
    "max_score": 0.6260078,
    "hits": [
      {
        "_index": "my_index",
        "_id": "1",
        "_score": 0.08701137, 유사도점수
        "_source": {
          "title": "hello elasticsearch!",
          "views": 1234,
          "public": true,
          "created": "2019-01-17T14:05:01.234Z"
        }
      },...
    ]
  }
}
```

엘라스틱은 전통적인 RDBMS가 아니라 전문검색엔진이다.
단순히 주어진 텍스트와 매칭되는 문서를 찾는 것이 아니라 문서를 분석해서 역색인을 만들어 두고
검색어를 분석해서 둘 사이의 유사도가 높은 문서를 찾는다.

2.1 삭제

DELETE [인덱스이름]/_doc/id

DELETE
my_index/_doc/QH0vS40BS0M06ur6nv1_


```
{
  "_index": "my_index",
  "_id": "QH0vS40BS0M06ur6nv1_",
  "_version": 2,
  "result": "deleted",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 7,
  "_primary_term": 1
}
```

2.2 엘라스틱 구조

문서
- 엘라스틱 서치가 저장하고 색인을 생성하는 JSON 문서

인덱스
- 문서를 모아놓은 단위, 클라이언트는 이단위로 검색을 요청하게된다

샤드
- 인덱스는 그 내용을 여러 샤드로 분리하여 분산 저장하고 엘라스틱서치는 고가용성을 제공하기위해 샤드내용을 복제하는데 원본역할을 담당하는 주 샤드(primary shard)라고 하고 복제본은 복제본 샤드(replication shard)라고 한다.

_id
- 인덱스 내 문서에 부여되는 고유한 구분자

타입
- 엘라스틱서치는 과거에 하나의 인덱스 안에 여러 문서를 묶어서 타입이라는 논리 단위로 나눴다. 예를 들어 상품 정보를 담는 하나의 인덱스안에 가전, 식품등 타입을 두어서 문서를 논리적인 단위로 구분하는데 사용했지만 이 개념은 폐기되었다.
타입이름이 들어 가야할 자리에는 기본값인 _doc이 들어간 API를 사용 해야 한다.

노드
- 엘라스틱서치 프로세스 하나가 노드 하나를 구성한다. 노드 하나는 여러 개의 샤드를 가지며 고가용성을 제공하기 위해 같은 종류의 샤드를 같은 노드에 배치하지 않는다.

데이터노드 - 샤드를 보유하고 샤드에 실제 읽기와 쓰기 작업을 수행하는 노드
마스터노드 - 클러스터를 관리하는 중요한 역할을 하는 노드 마스터, 후보 노드중에서 1대가 선출된다.
조정 노드 - 클라이언트의 요청을 받아서 데이터 노드에 요청을 분배하고 클라이언트에게 응답을 돌려주는 노드

클러스터
- 노드가 여러 개가 모여 하나의 클러스터를 구성한다.

2.3 엘라스틱서치 내부 구조와 루씬

루씬
–엘라스틱서치는 아파치 루씬을 코어 라이브러리로 사용하고 있다. 루씬은 문서를 색인하고 검색하는 라이브러리
–문서색인요청이 들어오면 루씬은 문서를 분석해서 역색인을 생성한다. 최초 생성 자체는 메모리 버퍼에 들어간다
색인,업데이트,삭제 등의 작업이 수행되면 루씬은 이러한 변경들을 메모리 버퍼에 들고 있다가 주기적으로 디스크에 flush한다.

루씬 refresh
– 색인이 완료된 문서만 검색 할 수 있는 데, 엘라스틱서치는 내부적으로 refresh 라는 작업을 한다.
– refresh 단계를 거쳐야 데이터가 검색 할 수 있는 데 refresh는 어느정도 비용이 있는 작업이기 때문에 적절한 간격마다 주기적으로 실행한다.
– refresh API를 사용하면 필요에 따라 refresh작업을 수행할 수도 있다.

루씬 commit
– 루씬의 flush는 시스템의 페이지 캐시에 데이터를 넘겨주는 것까지만 보장할 뿐 디스크에 파일이 실제로 안전하게 기록되는 것 까지는 보장하지 않는다.
따라서 루씬은fsync 시스템 콜을 통해 주기적으로 페이지 캐시 내용과 디스크 내용의 싱크를 맞추는작업을 수행하는데 이를 루씬 commit 이라고 한다.

세그먼트
– 디스크에 기록된 파일이 모이면 세그먼트라는 단위가 된다. 세그먼트 자체는 불변인 데이터로 구성되어 있다.
– 새로운 문서가 들어오면 새 세그먼트가 생성되고 삭제시 삭제 플래그만 표시해둔다.
– 업데이트시 삭제 플래그를 표시하고 새 세그먼트를 생성한다.
– 루씬의 검색은 모든 세그먼트 대상으로 수행하는데 불변인 세그먼트의 개수를 무작정 늘려갈수없기때문에 루씬은 중간중간 세그먼트 병합을 수행한다. 이때 삭제된 세그먼트는 실제로 삭제하는 작업을 병행한다. 병합은 비싼 작업 이지만 검색성능 향상을 기대할 수 있다.
Forcemerge api를 통해 병합을 수행할 수 있으나 더 이상 추가 데이터 색인이 없을 것이 보장될 때 수행해야 세그먼트가 누락 되는것을 방지할 수 있다.

2.3 엘라스틱서치 내부 구조와 루씬

루씬의 인덱스와 엘라스틱서치 인덱스

- 여러 세그먼트가 모이면 하나의 루씬 인덱스가 된다. 루씬은 이 인덱스 내에서만 검색이 가능하다.
  엘라스틱서치 샤드는 이 루씬 인덱스 하나를 wrap한 단위다.
  엘라스틱서치 샤드가 여러 개 모이면 엘라스틱서치 인덱스가 되고 엘라스틱서치 레벨에서는 여러 샤드의 있는 문서를 모두 검색할 수 있다.
  새 문서가 들 어오면 해당 내용을 라우팅하여 여러 샤드에 분산시켜 저장, 색인한다. 이후 클라이언트가 검색 요청하면 각 샤드를 대상으로 검색한 뒤 그 결과를 모아 병합하여 최종 응답을 만든다. 이런 구조를 통해 루씬 레벨에서는 불가능한 분산 검색을 엘라스틱서치 레벨에서는 가능하게 만들었다.

트랜스로그(translog)

- 루씬이 commit을 수행할 때 큰 비용이 드는 작업이다. 하지만 변경사항을 모아서 commit을 한다면 장애가 발생할 때 commit이 되지 않아 데이터 유실이 될수있다. 이런 문제를 해결하기 위해 엘라스틱서치 샤드는 모든 작업마다 translog 라는 이름의 작업 로그를 남긴다. translog는 색인, 삭제 작업이 루씬 인덱스에 수행된 직후에 기록된다. translog 기록까지 끝난이후에야 작업 요청이 성공으로 승인된다. 장애가 발생한 경우 샤드 복구 단계에서 translog를 읽어서 복구 가능하다.

그런데 translog가 너무 커지면 샤드 복구에 시간이 오래 걸리는데 이를 방지하기 위해 translog의 크기를 적절히 유지해줄 필요가 있다. 엘라스틱서치 flush는 루씬의 commit을 수행하고 새로운 translog를 만드는 작업을 한다. 엘라스틱서치 flush가 백그라운드에서 주기적으로 translog의 크기를 적절한 수준으로 유지한다.