elastic

엘라스틱서치
검색 및 샤드 동작

순천향대학교 컴퓨터공학과
이 상 정

학습 내용

1. 분산 검색 실행
   - 분산 검색 질의
   - 질의 단계, 페치 단계
2. 샤드 내부 동작
   - 샤드의 구성과 동작
   - 역색인 동작
   - 실시간에 가까운 검색
   - 갱신의 지속성 유지

1. 분산 검색 실행
(Distributed Search Execution)

분산 환경 검색 동작

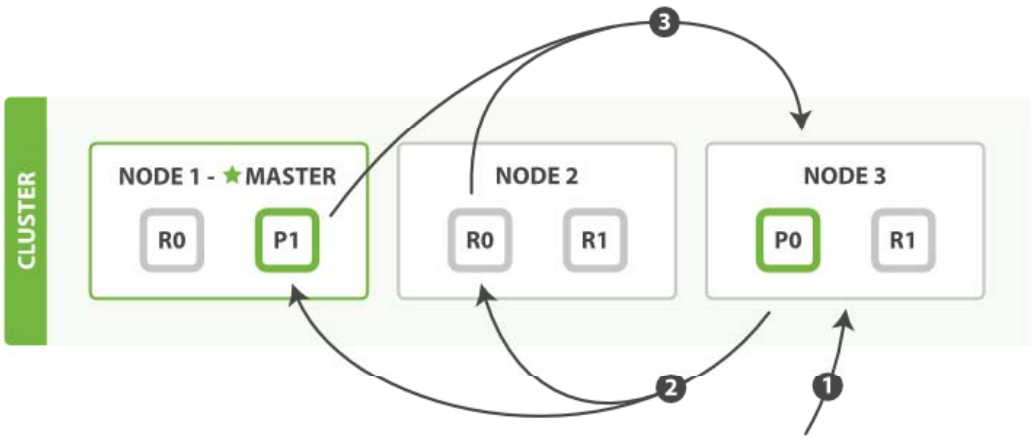
- 분산 환경의 검색은 앞에서 소개한 다큐먼트 데이터 처리 (CRUD) 과정 보다 복잡
  - CRUD 동작은 _index, _type, 라우팅 값이 결합된 단일의 다큐먼트에서의 동작
  - 검색은 질의 매치되는 클러스터 내의 여러 샤드에 있는 다큐먼트들에 대한 동작
    - 다 수의 샤드들에서 매치된 결과들을 통합도 필요
- 검색 동작은 다음 2 단계로 구분
  - 질의 (query)
  - 페치 (fetch)

질의 단계 (Query Phase)

- 검색이 시작되면 인덱스 내의 모든 샤드에 질의가 브로드캐스트
  - 처음 요청을 받은 노드(조정 노드)가 라우드-로빈 방식으로 모든 샤드들에게 브로드캐스트
- 각 샤드는 로컬에서 검색을 실행한 후, 매칭된 다큐먼트에 대한 우선순위 큐(priority queue)를 생성
  - 우선순위는 큐는 n개의 매치된 다큐먼트들을 저장하는 정렬된 리스트
  - 우선순위 큐는 from과 size 범위 지정(pagination) 파라미터에 따라 결정
    - 다음 예의 경우 100개의 다큐먼트를 저장하는 우선순위 큐 생성

```
GET /_search
{
    "from": 90,
    "size": 10
}
```

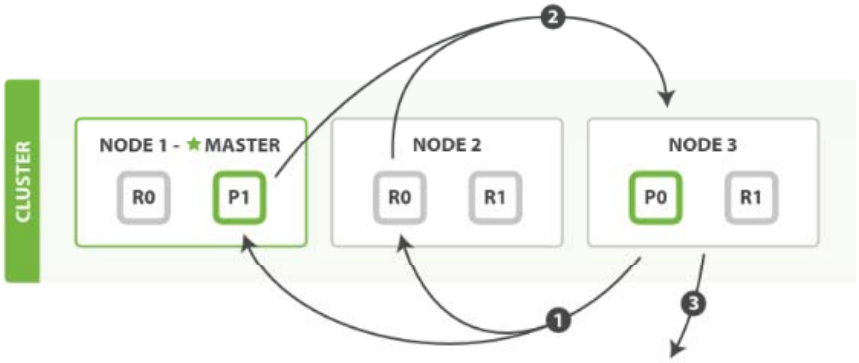

분산 검색 질의



1. 클라이언트가 노드 3(조정 노드)에 검색 요청을 하면 노드 3는 from+size 크기의 빈 우선순위 큐를 생성
2. 노드 3는 검색 요청을 인덱스의 최초 또는 복사본 샤드들에 전달. 각 샤드는 로컬로 질의를 수행하고 그 결과를 from_size 크기의 로컬 우선순위 큐에 추가.
3. 각 샤드는 다큐먼트 ID들과 우선순위 큐에 있는 다큐먼트들의 정렬된 값들을 조정 노드인 노드 3에 리턴. 노드 3는 이들 값들을 통합하여 우선순위 큐에 글로벌 정렬 리스트를 생성.

페치 단계 (Fetch Phase)

- 질의 단계는 검색 요청 만족하는 다큐먼트만을 찾고, 다큐먼트 자체를 응답하지 않고, 패치 단계에서 다큐먼트 가져옴



1. 조정 노드 3가 페치할 다큐먼트의 샤드들에 다 수의 GET 요청을 이슈
2. 요청을 받은 각 샤드는 다큐먼트를 적재하여 조정 노드에 응답
3. 모든 다큐먼트들이 조정 노드에 페치되면 다큐먼트 결과를 클라이언트에 응답

2. 샤드 내부 동작 (Inside a Shard)

샤드의 구성과 동작 과정 이해

- 엘라스틱서치에서 샤드(shard)는 저수준의 동작 단위 (low-level worker unit)

- 샤드는 무엇이며 어떻게 다음 동작을 수행하는가?
  - 어떻게 실시간에 가까운(near real-time) 검색이 가능한가?
  - 어떻게 실시간으로 다큐먼트의 CRUD 동작이 가능한가?
  - 변경된 내용이 지속되고, 전원이 꺼지더라도 손실되지 않는 이유는?
  - 다큐먼트를 삭제하더라도 바로 디스크에서 삭제되지 않는 이유는?
  - refresh, flush, optimize API는 무엇이고, 언제 사용하는가?

검색 가능한 텍스트 만들기

- 엘라스틱서치는 텍스트가 검색 가능하도록 앞에서 소개한 역 색인(inverted index)을 구성
  - 역 색인은 다큐먼트에 나타난 모든 텀들의 각 다큐먼트 포함 여부를 나타내는 정렬된 리스트
  - 역 색인에는 텀들의 빈도, 순서, 길이 등의 정보도 포함

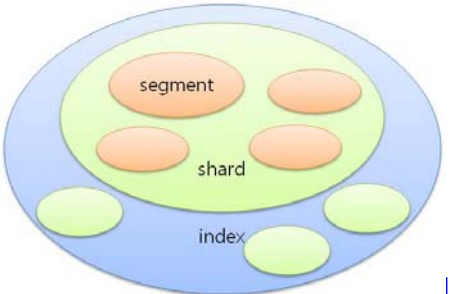
Term	Doc 1	Doc 2	Doc 3	...
brown	X		X	...
fox	X	X	X	...
quick	X	X		...
the	X		X	...

역 색인 불변성 (Immutability)

- 디스크에 저장되는 역 색인은수정이 불가능 (immutable)
- 역 색인 불변성의 이점
  - 수정이 불가능하므로 동시성(concurrency) 제어를 위한 락킹 (locking) 등이 필요 없음
  - 전체 역 색인이 파일 시스템 캐시에 적재된다면, 수정 변경이 없으므로 항상 적재된 상태가 됨. 또 검색은 파일 입출력이 아닌 메모리 상에서 동작되므로 성능이 향상
  - 하나의 커다란 역 색인의 경우 압축을 하여 디스크 입출력을 줄이고, 인덱스의 캐싱에 필요한 RAM 용량을 줄일 수 있음
- 수정 갱신은?
  - 새로운 다큐먼트가 추가되고 검색 가능하게 하려면 전체 역 색인을 새로 구축해야 함.
    - 단일 역 색인 데이터의 크기 제약
    - 역 색인 수정 갱신(다큐먼트 추가) 빈도 제약

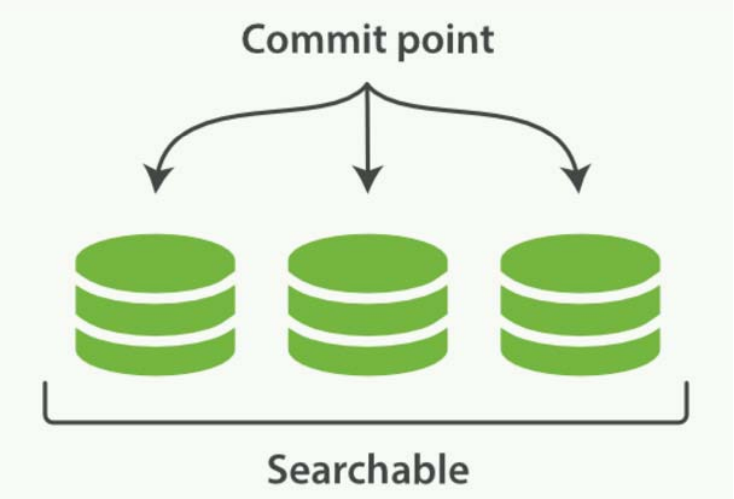
역 색인의 동적 수정

- 역 색인의 불변성 이점은 살리면서 역 색인을 수정 갱신하는 방법은?
  - 하나 이상의 여러 개 역색인들을 사용
    - 새로운 다큐먼트 추가 시 기존의 큰 역 색인은 그대로 유지
    - 새로운 다큐먼트를 반영하는 새로운 역 색인을 생성
    - 검색 요청 시 여러 개의 역색인을 차례로 검색한 후 결과를 병합하여 리턴
- 엘라스틱서치의 기반이되는 루씬(Lucene) 자바 라이브러리는 세그먼트 단위 검색(per-segment search) 개념을 도입
  - 세그먼트는 여러 역 색인 중 하나
  - 루씬 인덱스 = 세그먼트들 + 커밋 포인트 (commit point)
    - 커밋 포인트는 현재 세그먼트들의 목록 파일
    - 루씬의 인덱스는 엘라스틱서치의 샤드에 해당



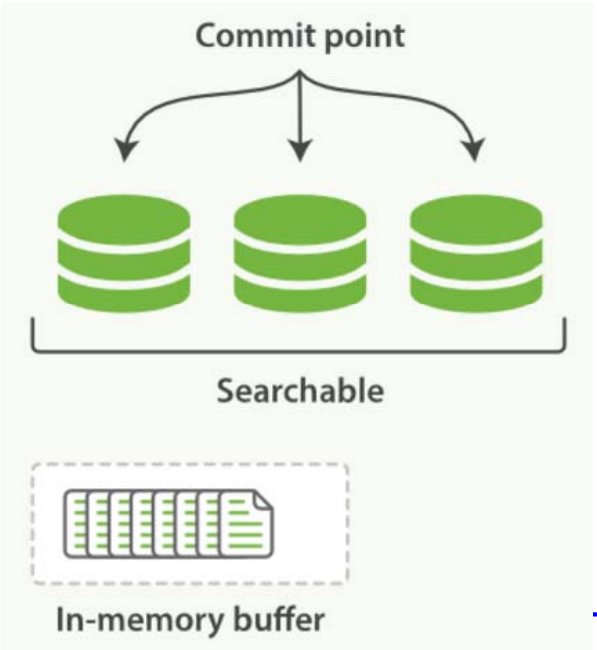
3개의 세그먼트를 갖는 루씬 인덱스 예

- 루씬 인덱스 (샤드)
  - 3개의 세그먼트(역 색인), 커밋 포인트



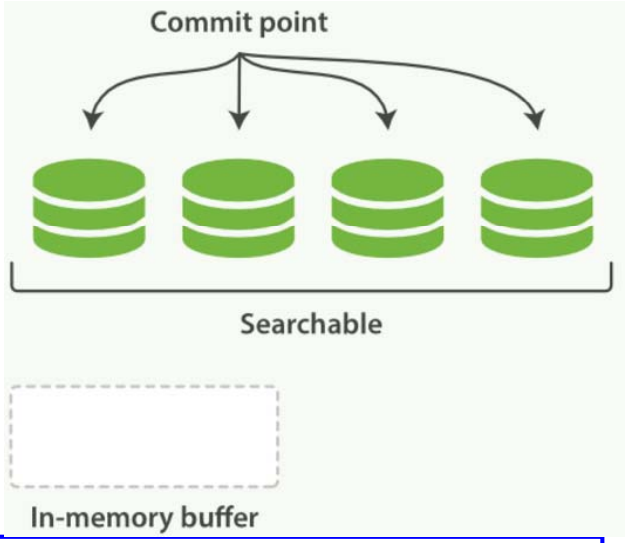
루씬의 세그먼트 단위 검색 동작 (1)

- 새로운 다큐먼트가 추가되면 메모리 색인 버퍼에 저장
  - 현재는 검색 불가능



루씬의 세그먼트 단위 검색 동작 (2)

- 주기적으로 메모리 색인 버퍼가 커밋(commit)됨
  - 새로운 세그먼트(역 색인)가 디스크에 저장
  - 새로운 커밋 포인트가 디스크에 저장
  - 디스크가 fsync 됨
    - 파일시스템 캐시에 있는 모든 데이터가 물리적으로 디스크에 플러시(flush) 됨
  - 새로운 다큐먼트가 검색 가능해짐

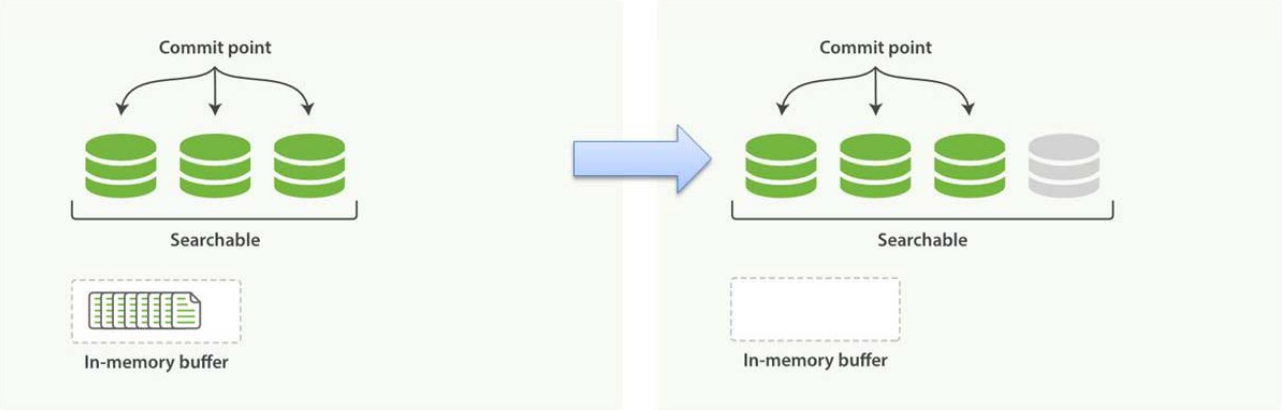


다큐먼트 삭제와 갱신 동작

- 세그먼트(역 색인)는 수정할 수 없으므로(immutable) 다큐먼트 삭제 시 세그먼트에서 제거되지 않음
  - 제거하는 대신 .del 파일에 삭제될 다큐먼트를 “deleted”로 표시
  - “delete”로 표시된 다큐먼트는 검색 질의 시 매칭되지만 클라이언트로 응답 리턴하지 않음
- 갱신 동작도 삭제와 유사하게 동작
  - 다큐먼트 갱신 시 이전 버전 다큐먼트는 “deleted”로 표시
  - 다큐먼트의 새 갱신 버전이 새 세그먼트로 색인
- 세그먼트 통합 시 “deleted”로 표시된 다큐먼트들이 삭제

실시간에 가까운 검색 (Near Real-Time Search)

- 세그먼트 단위 검색 시 디스크 접근으로 다큐먼트의 색인 시점에서 검색 가능한 시점까지 지연이 발생
  - 다큐먼트 색인은 메모리 색인 버퍼에서 수행되지만 파일 시스템 캐시 생성 후 디스크에 fsync 해야 하는 디스크 입출력 지연 발생
- 해결책으로 파일 시스템 캐시에서 세그먼트가 생성되면 검색 가능하도록 하여 지연을 줄임
  - fsync는 나중에 주기적으로 실행
  - 새로운 세그먼트의 검색은 가능하지만 커밋 포인트가 (fsync) 없는 상태



refresh API (1)

- refresh는 fsync를 포함하는 완전한 커밋을 하지 않으면서 새로운 다큐먼트를 작성하고 오픈하는 동작
- 엘라스틱서치는 주기적으로 refresh를 실행
  - 디폴트는 1초 간격으로 refresh하여 1초 이내에 다큐먼트의 검색이 가능
  - refresh 간격은 인덱스 단위로 refresh_interval 속성으로 지정 가능
  - my_logs 인덱스의 refresh 간격을 1초 단위로 설정하는 예

```
PUT /my_logs
{
  "settings": {
    "refresh_interval": "30s"
  }
}
```

refresh API (2)

- 자동이 아닌 refresh API를 사용하여 수동 refresh도 가능

```
POST /_refresh ❶
POST /blogs/_refresh ❷
```

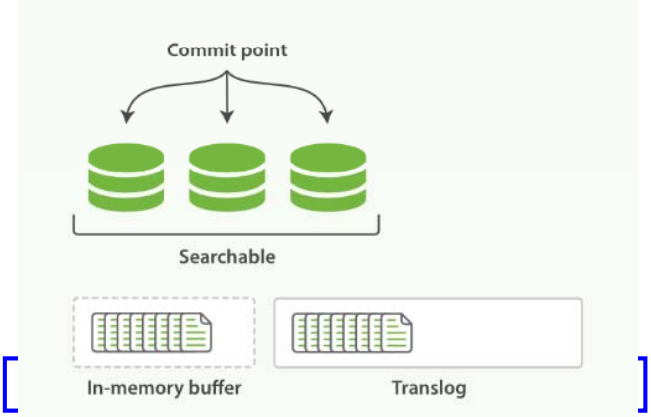
① 모든 인덱스들을 refresh
② blogs 인덱스만 refresh

- refresh_interval 값을 -1로 설정하면 자동 refresh를 중지

```
PUT /my_logs/_settings
{ "refresh_interval": -1 }
```

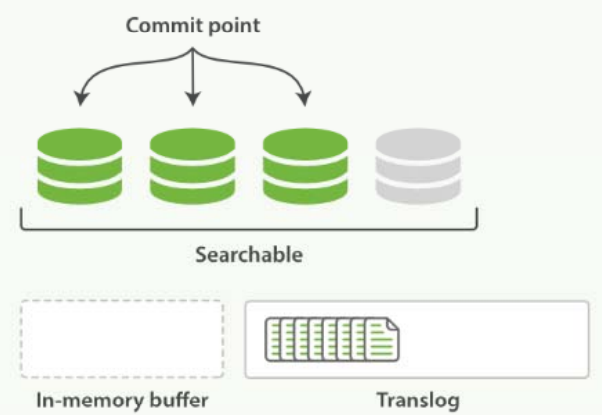

갱신의 지속성 유지 (1) (Making Changes Persistent)

- 파일 시스템 캐시를 디스크에 내보기(flush)는 커밋 포인트가 없는(fsync가 없는) 상태에서 시스템 오류나 응용 종료하면 변경된 데이터가 디스크에 저장되지 않고 손실되어 지속성 유지 못함
- 엘라스틱서치는 위 문제 해결을 위해 모든 동작을 기록하는 translog (transaction log)를 도입
  - 다큐먼트의 색인 시 메모리 버퍼에 저장하고 translog에 기록



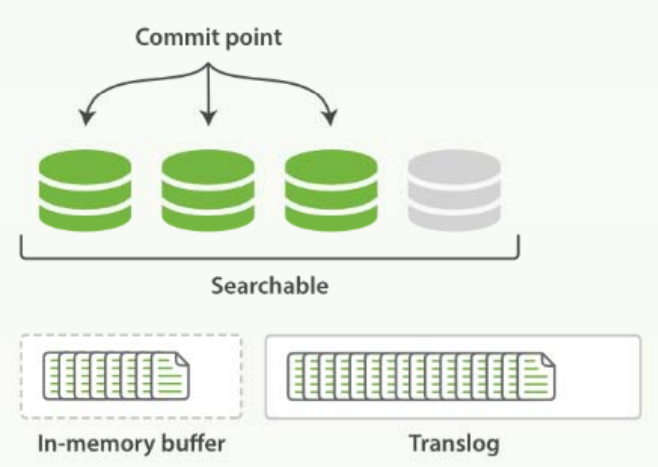
갱신의 지속성 유지 (2)

- 주기적으로 refresh가 발생하면
  - 메모리 버퍼의 다큐먼트는 커밋(fsync) 없이 새로운 세그먼트로 저장(파일 시스템 캐시) 되고 검색 가능해짐
  - 메모리 버퍼는 비워짐



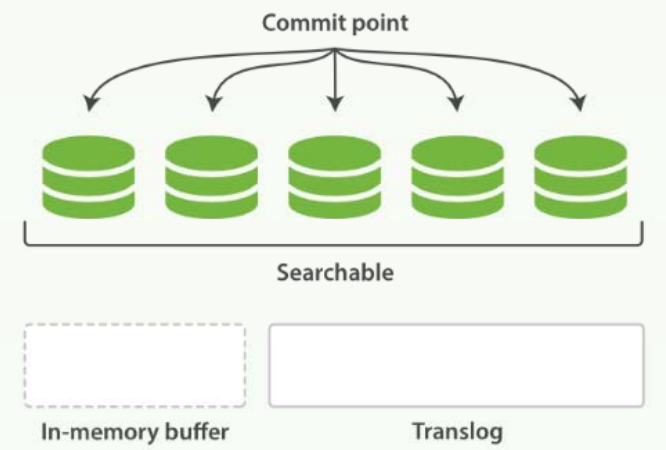
갱신의 지속성 유지 (3)

- 계속 새로운 다큐먼트가 색인되면
  - 메모리 버퍼에 저장되고 translog에 기록
  - translog는 이전의 기록 위에 새로운 기록이 추가(append)



갱신의 지속성 유지 (4)

- translog가 충분히 커지면 커밋이 수행되어
  - 메모리 버퍼의 새로운 다큐먼트는 새 세그먼트로 생성되고, 검색 가능해지며 메모리 버퍼는 비워짐
  - 새로운 커밋 포인트가 기록되고, 파일 시스템 캐시는 fsync를 통해 physical 디스크로 내보내기(flush) 됨
  - translog는 비워짐



트랜잭션 로그

- translog는 디스크에 커밋(저장)되지 않은 모든 동작을 기록
  - 엘라스틱서치 실행 시 마지막 커밋 포인트를 사용하여 디스크로부터 세그먼트들을 복구하고,
  - Translog를 사용하여 모든 동작을 재 수행하여 마지막 커밋 후 발생한 변경을 추가

- translog를 사용하여 CRUD의 실시간 처리가 가능
  - RUD의 경우 해당 세그먼트의 다큐먼트를 가져오기 전에 다큐먼트 ID를 사용하여 translog를 조사하여 최근의 변경 사항을 조사
  - 즉, 실시간으로 다큐먼트의 가장 최근의 버전을 접근

flush API

- flush는 커밋을 수행하고 translog를 비우는 동작
  - 디폴트로 30분 간격으로 자동 수행
  - flush API를 사용하여 자동이 아닌 수동으로도 수행

```
POST /blogs/_flush ❶

POST /_flush?wait_for_ongoing ❷
```

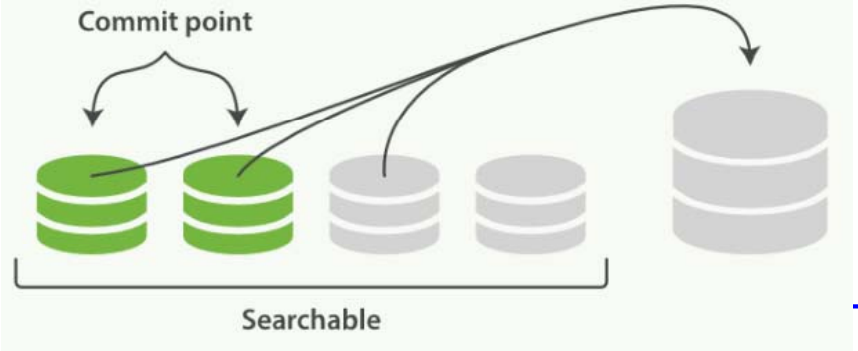
① Blogs 인덱스 flush

② 모든 인덱스를 flush하고, 모든 flush가 완료될 때까지 기다림

  - 노드를 재 시작할 때 수동을 flush하여 translog를 비우는게 성능에 바람직
- translog의 안전성
  - translog 자체는 5초 주기로 디스크에 flush
    - index.translog.interval로 설정 가능
  - 최악의 경우 5초 분량의 데이터는 소실될 가능성은 있음

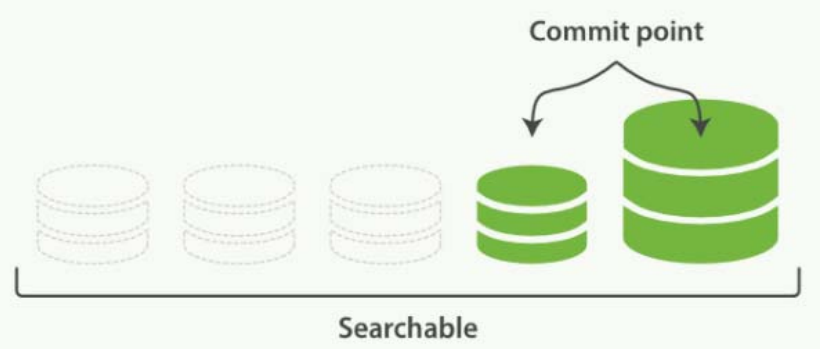
세그먼트 병합 (Segment Merging) (1)

- refresh를 수행하여 실시간에 가까운 검색이 가능하지만 자동으로 refresh되는 1초 단위로 새로운 세그먼트가 생성되어 세그먼트의 수가 기하급수적으로 증가
- 엘라스틱서치는 백그라운드로 세그먼트 병합을 수행
  - merge 프로세스가 비슷한 크기의 세그먼트를 병합하여 좀 더 큰 새로운 세그먼트를 생성
  - "deleted"로 표시된 다큐먼트는 세그먼트 병합 시 포함되지 않고 완전히 삭제



세그먼트 병합 (2)

- 세그먼트 병합 수행 후
  - 새로운 병합 세그먼트가 디스크에 flush
  - 병합된 이전의 세그먼트들 삭제되고 병합된 세그먼트가 포함된 새로운 커밋 포인트를 기록
  - 새 병합 세그먼트가 오픈되어 검색 가능해짐
  - 이전의 세그먼트들은 삭제



세그먼트 병합 – 성능 이슈

- 세그먼트 병합은 디스크 입출력과 CPU를 크게 소모하여 병합 중에 검색 성능을 저하
  - 병합 억제(merge throttling) 기능을 사용하여 병합을 제약

- optimize API
  - 샤드에서 지정된 max_num_segments 개수로 강제적으로 병합 수행
  - 로그와 같이 날짜 단위로 인덱스를 생성하는 구조라면, 이전 날짜의 인덱스에는 새로운 데이터가 추가되지 않으므로 직접 optimize API를 사용하면 효율적

```
POST /logstash-2014-10/_optimize?max_num_segments=1
```

  - optimize API를 직접 사용하게 되면 병합 억제되지 않고 노드의 모든 I/O와 CPU를 사용하게 됨으로 주의해서 사용해야 함

참고 자료

- Elasticsearch: The Definitive Guide, Getting Started
  - Distributed Search Execution
    - https://www.elastic.co/guide/en/elasticsearch/guide/current/distributed-search.html
  - Inside a Shard
    - https://www.elastic.co/guide/en/elasticsearch/guide/current/inside-a-shard.html