데이터 분산처리 끝내기
- Part3 Elasticsearch

이 수업에서 다루고자 하는 것

이론 & 실습

Chapter 1. ELK 스택이란?
Chapter 2. Elasticsearch의 기본 구성
Chapter 3. AWS EC2에 Elasticsearch 설치하기
Chapter 4. Elasticsearch 모니터링
Chapter 5. Elasticsearch TLS 적용하기
Chapter 6. AWS EC2에 Kibana 설치하기
Chapter 7. Elasticsearch API
Chapter 8. Elasticsearch Aggregation
Chapter 9. Elasticsearch Mapping API
Chapter 10 Filebeat와 Logstash로 데이터 전송하기
Chapter 11. Text Tokenizer, Nori

Chapter 1. Elasticsearch란?

Chapter 1.

01. Elasticsearch에 대한 이해

01. Elasticsearch?

Elasticsearch

- 텍스트, 숫자, Geospatial, 정형 및 비정형 데이터 등의 데이터에 대한 검색, 분석이 가능한 **검색엔진 또는 Information Retrieval 라이브러리**
- **Apache Lucene**을 기반으로 구축됨
- Elasticsearch N.V.(현재 명칭 Elastic)가 2010년에 최초로 출시
- REST API, 분산처리, 속도, 확장성의 특징을 갖고있음
- Elastic Stack의 핵심 구성 요소(ELK)
- 보통 **ELK Stack**(Elasticsearch, Logstash, Kibana)에 Beats가 포함되어 있음

01. Elasticsearch?

*Apache Lucene

- 루씬은 확장 가능한 고성능 정보검색(IR, Information Retrieval) **라이브러리**이다. 루씬은 소프트웨어 프로그램에 색인과 검색 기능을 간단하게 추가할 수 있도록 지원
- 하둡 개발자인 더그 커팅이 개발 (Doug Cuting)
- 2001년 Apache Software Foundation에 참여된 뒤,
- 2005년 Apache top-level 프로젝트로 승격

- 이 라이브러리를 이용해 Elasticsearch, SolR이 실행됨
- Full Text 검색에 유용함
  *전문(Fulltext) 검색이란 긴 장문의 문자열 속에서 일부를 검색해도 나오는 것을 말함
- 긴 문장에 대해 파싱 하고 Analyzer가 분석해 검색에 필요한 인덱스를 생성한다

01. Elasticsearch?

검색 엔진

- Wikipedia: “검색 엔진(search engine)은 컴퓨터 시스템에 저장된 정보를 찾아주거나 웹 검색(web search query)을 도와주도록 설계된 정보 검색 시스템 또는 컴퓨터 프로그램이다. 이러한 검색 결과는 목록으로 표시되는 것이 보통이다. 검색 엔진을 사용하면 정보를 찾는데 필요한 시간을 최소화할 수 있다.”

1. **문서 수집** : 검색하려는 대상 수집하기
2. **문서 정제** : 수집된 비정형 문서를 정형 데이터로 정제하고 가공
3. **문서 데이터 색인** : 정제된 문서를 빠르게 검색 가능한 구조로 저장
4. **문서 검색** : 문서 정보에서 검색어 찾기

01. Elasticsearch?

정보검색 (Information Retrieval)

- 사용자가 입력한 키워드에 대해서 적절한 문서를 찾는 것을 의미
- 정보들은 대량의 문서 형태로 되어있음
- 검색 대상이 비정형/반정형 데이터
- SQL을 사용하지 않고 자연어를 사용함
- 정답보다는 관련성을 찾아줌

01. Elasticsearch?

검색엔진(IR) VS RDBMS

- 색인을 통해 검색에 활용한다는 공통점
- 인덱싱의 차이
  - RDBMS : 컬럼 기준, id 같은 값을 활용해 인덱스를 생성
  - 검색엔진 : 필드 값을 분해하여 Term형태로 만들어 인덱스 생성
- 검색 방법
  - RDBMS : 특정 단어 검색 시, 전체 컬럼 정보를 하나씩 비교하며 조회
  - 검색엔진 : 미리 정의된 색인에서 조회
- 검색 대상
  - RDBMS : 구조화된 대상
  - 검색엔진 : 구조화되지 않은 텍스트에 대해서도 검색이 가능

01. Elasticsearch?

Elasticsearch(Elastic) 왜 필요할까

- Elastic는 단순 검색 엔진이 아니다
- 검색과 분석을 위한 플랫폼을 표방

  수집 – 처리 – 분석
- 각 단계에 맞는 스택들을 개발
- 데이터 수집 **Logstash**

  (Beats)
- 데이터 처리 **Elasticsearch**
- 데이터 분석 **Kibana**

Chapter 2.

02. ELK 스택이란? -1

02. ELK 스택

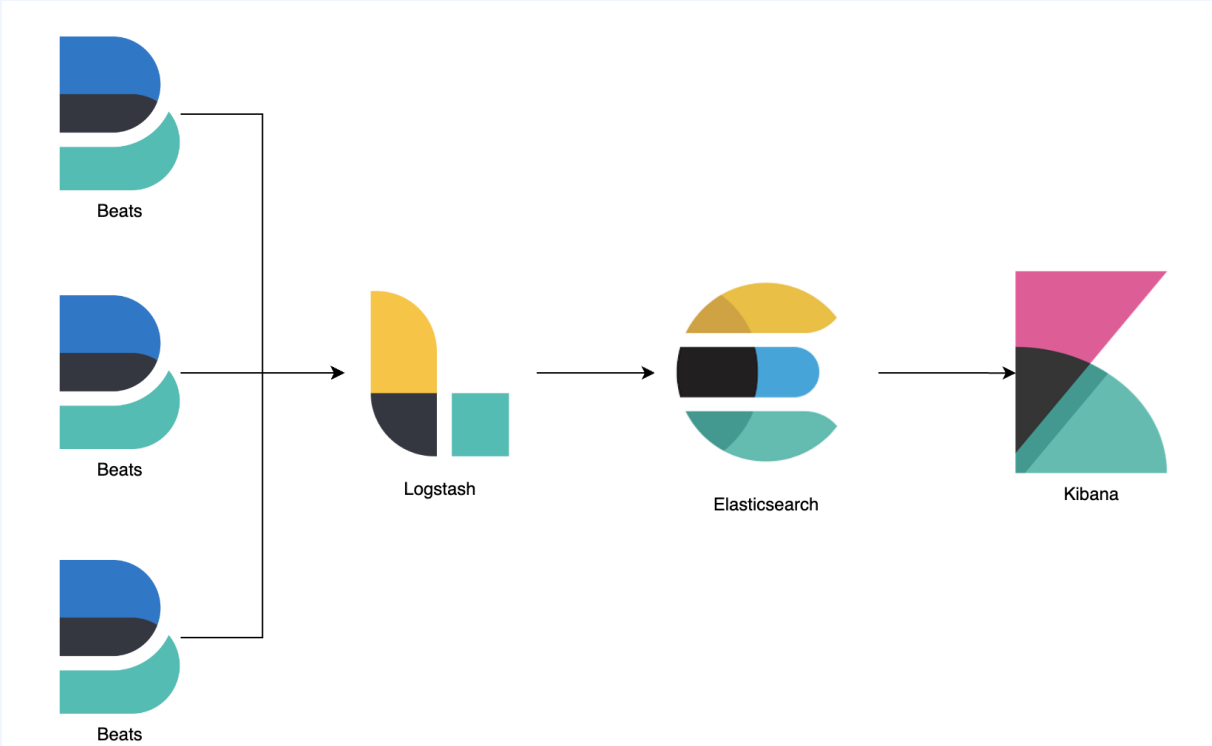
ELK

- 검색과 분석을 위한 플랫폼을 표방

  수집 – 처리 – 분석
- 각 단계에 맞는 스택들을 개발
- 데이터 수집 **Logstash (Beats)**
- 데이터 처리 **Elasticsearch**
- 데이터 분석 **Kibana**

02. ELK 스택

ELK 수집 – 처리 – 분석



Elasticsearch

엘라스틱서치와 RDB와의 비교

엘라스틱 서치	RDB
인덱스	데이터베이스
샤드	파티션
*타입	테이블
문서	행
필드	열
매핑	스키마
Query DSL	SQL

02. ELK 스택

Elasticsearch

엘라스틱서치와 RDB와의 비교

- Elasticsearch는 검색에 특화
- RDBMS는 SQL을 통해 원하는 정보의 검색은 가능하지만, 단순한 검색에 해당됨 Full Text 검색이 아님
- Elasticsearch는 비정형 데이터를 Indexing해서 검색할 수 있고, 형태소 분석이 가능하며 역색인을 통해 빠른 검색이 가능하다.
- 둘의 인덱스를 혼동하지 말자
  RDBMS에서나 보통 개발 쪽 단어로 인덱스는 쿼리와 join을 빠르게 해주는 도구, 엘라스틱서치에서는 인덱스는 데이터의 단위(DB)이다.

02. ELK 스택

Elasticsearch

- 2억 5천만번 이상 다운됨
- Full Text Search (전문검색)

  text 내용 전체를 색인해서 특정 단어가 포함된 문서를 검색하는 것을 말함

  다양한 플러그인을 조합해 빠르게 검색 가능
- Inverted Index

  역색인을 통해 빠르게 검색이 가능함

  검색을 원하는 해당 단어가 포함된 모든 문서의 위치를 알 수 있음
- 확장성 가용성

  분산 구성이 가능한 구조

  샤드를 통해 데이터를 분산해서 빠르게 처리가 가능함

Elasticsearch 단점

- **실시간은 아님**

  인덱싱된 데이터는 1초 뒤에 검색이 가능함

  Near Real Time

- **트랜잭션, 롤백 불가**

  분산시스템으로 구성되어 있음

  리소스 소모가 큰 롤백이나 트랜잭션은 지원하지 않아, 데이터 손실의 위험이 있음

- **데이터 업데이트 불가**

  업데이트는 기존 문서 삭제하고 변경내용으로 새 문서를 생성하는, reindexing 방식

  비용이 큰 편

Chapter 1.

03. ELK 스택이란? -2

03. ELK 스택

Logstash

- 다양한 소스에서 데이터를 수집하여 변환한 후 자주 사용하는 저장소로 전달하는 기능
- 플러그인 기반

  파이프라인을 구성하는 각 요소들은 전부 플러그인 형태

  기본 플러그 외에도 수 많은 커뮤니티 플러그인이 있음

  플러그인 개발을 위한 프레임워크와 플러그인을 관리할 수 있는 기능도 제공
- 모든 형태의 데이터 처리

  플러그인들의 조합으로 다양한 형태의 데이터를 입력받아 가공한 다음에 저장이 가능

03. ELK 스택

Logstash

- 성능

  자체적 내장 메모리, 파일 기반의 큐를 통해 안정성이 높고 처리속도가 빠름.

  벌크 인덱싱 및 파이프라인 배치 크기 조정을 통한 병목현상을 방지, 성능 최적화 가능

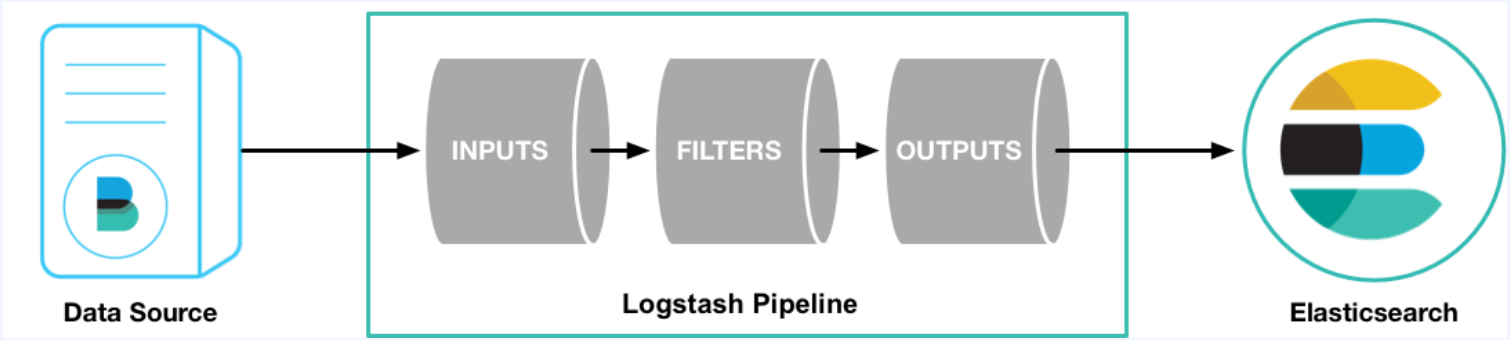
- 안정성

  엘라스틱 서치의 장애 상황에 대응하기 위한 재시도 로직이나 오류가 발생한 도큐먼트를 따로 보관하는 Dead-Letter-Queue를 내장하고 있다.

Logstash

Pipeline

- 데이터를 입력받고 실시간으로 변경하고 변경한 데이터를 다른 시스템에 전달한다
- Input, Output이 필수 구성요소
- Filter는 Optional



https://www.elastic.co/guide/kr/logstash/current/first-event.html

Logstash

Input

- 외부에서 데이터를 받아오는 영역
- 자주 사용하는 플러그인
  - file : 경로 지정한 파일을 읽는 방식
  - syslog : 리눅스 시스템에서 시스템로그를 읽음 (port 514)
  - redis : redis 에 있는 데이터를 읽는 방식
  - beats : Beats에서 데이터를 읽는 방식

Logstash

Filter(Optional)

- 입력 받은 데이터를 가공, 조건에 대한 가공
- 자주 사용하는 플러그인
  - grok : grok 패턴(정규식 비슷)을 사용해 메시지를 구조화된 형태로 분석
    대부분의 텍스트 포맷을 처리할 수 있음
  - mutate : 필드 이름 변경, 삭제, 추가
    - gsub : 가장 상단에서 grok에 보낼 메시지를 미리 전처리할 작업이 있을 때 사용
  - date : 문자열을 지정한 패턴의 날짜형으로 변경
    실제 로그 파일에 찍힌/적힌 시간으로 엘라스틱에 적재하기 위해 원하는 filed로 재 설정

Logstash

Output

- 입력과 필터를 거친 데이터를 Target 대상으로 보내는 단계
- 자주 사용하는 플러그인
  - elasticsearch : Elasticsearch에 인덱싱을 수행
    시계열 데이터 세트(예: 로그, 이벤트 및 지표), 비시계열 데이터 모두 전송 가능
  - file : 파일에 output 데이터를 저장
  - Kafka : Kafka Topic에 데이터를 전송

Kibana

- Kibana는 Elasticsearch에 있는 데이터를 시각화하고 Elastic Stack을 탐색
- 저장된 데이터를 분석, 시각화하는데 주로 사용
- 데이터의 패턴과 동향을 파악하고, 인사이트를 얻을 수 있다
- Kibana 대시보드는 여러 그래프와 차트를 한눈에 볼 수 있는 화면을 제공
  사용자는 데이터의 상태와 성능을 실시간으로 모니터링

03. ELK 스택

Kibana

- Discover : 데이터 확인
- Visualize : 데이터 시각화
- Dashboard : 시각화 차트 모아보기
- Setting : 인덱스 등록, 설정

Chapter 2. Elasticsearch의 기본 구성

Chapter 2. Elasticsearch의 기본 구성

01. Elasticsearch 아키텍쳐

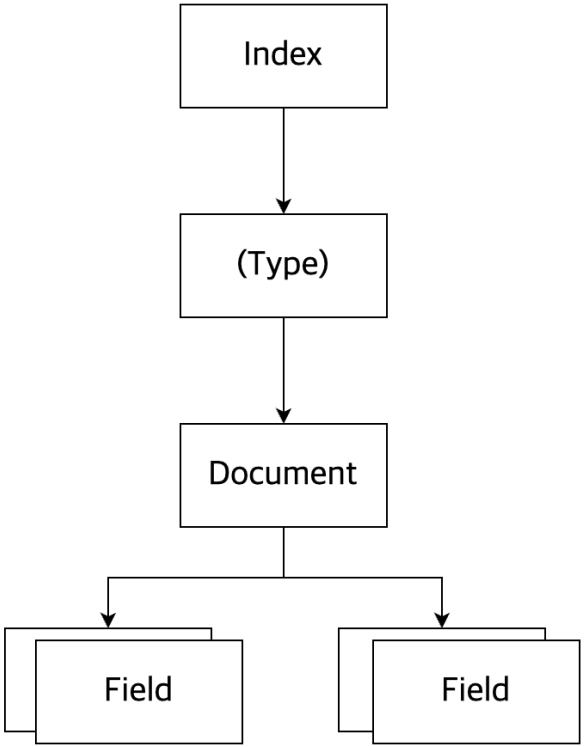
01. Elasticsearch 아키텍쳐

Elasticsearch 구성

엘라스틱 서치	**RDB**
인덱스	데이터베이스
샤드	파티션
*타입	테이블
문서	행
필드	열
매핑	스키마
Query DSL	SQL

01. Elasticsearch 아키텍쳐

Elasticsearch 구성



01. Elasticsearch 아키텍쳐

인덱스

- 단일 데이터 단위가 Document, Document의 집합이 인덱스
- ES에 저장하는 것을 인덱싱
- 인덱스 명으로 문서를 검색
- 분산환경에서 인덱스는 여러 노드에 분산되어 저장됨

샤드

- 샤드를 통해 데이터의 안전성을 높인다
- 인덱스의 저장 단위, 노드에 분산되어 저장

01. Elasticsearch 아키텍쳐

타입

- 인덱스의 논리적 구조
- 7.0버전부터는 인덱스 당 하나의 타입을 제공
- Index == Type

문서

- 문서는 데이터가 저장되는 최소 단위
- 한 문서는 다수의 필드로 구성됨
- nested 구조를 지원함

01. Elasticsearch 아키텍쳐

필드

- 문서를 구성하는 속성
- Column과 비슷한 개념
- 한 필드는 여러 데이터 타입을 가질 수 있다

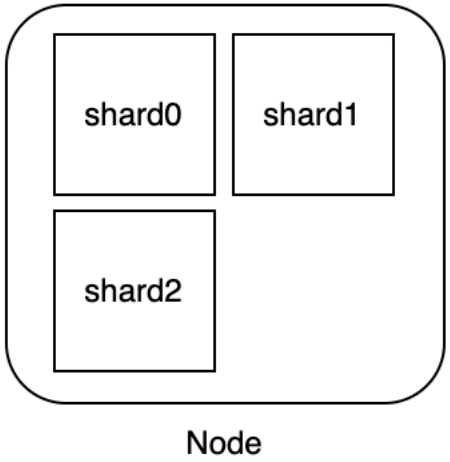
  ex) keyword, text

매핑

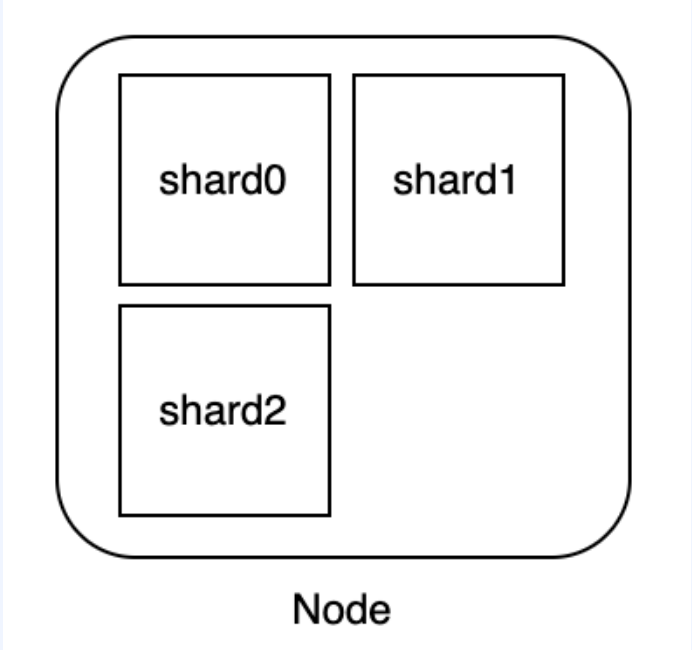
- 데이터의 저장 형태와 검색을 위해 데이터를 어떻게 접근하고 처리할지
- 문서의 필드와 필드의 속성을 정의, 인덱싱 방법을 정의
- 인덱스 매핑에서는 여러 데이터 타입이 지정 가능하지만 필드명은 중복사용이 불가

샤드

- 인덱스가 저장되는 단위
- 인덱스는 하나 이상의 샤드로 저장
- 인덱스는 샤드 단위로 분리, 데이터를 분산해 저장

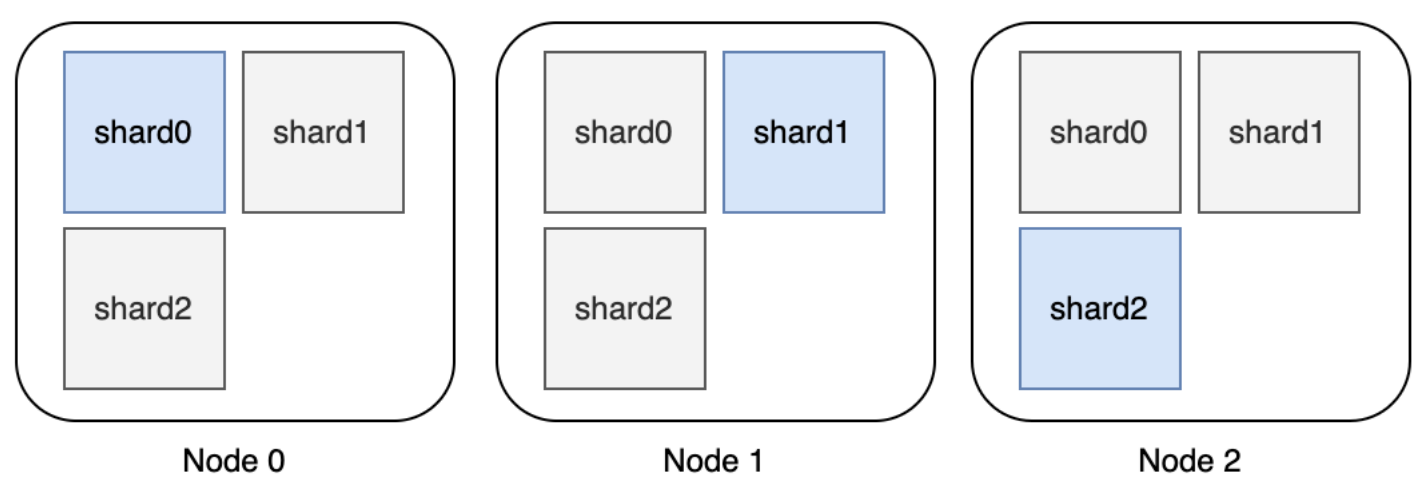


샤드



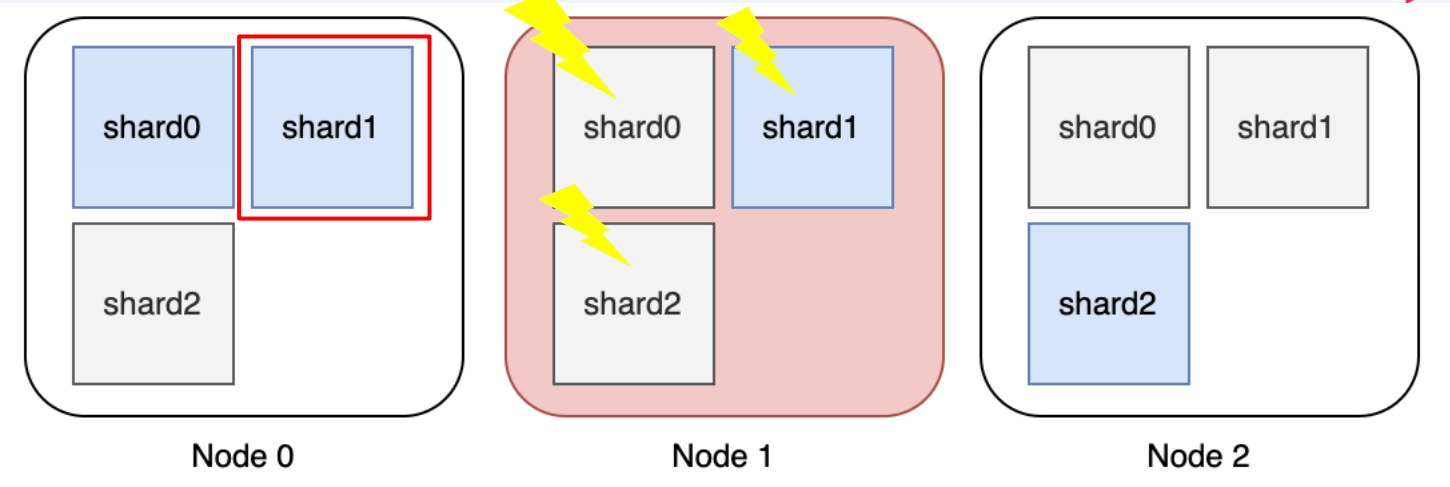
- 인덱스가 저장되는 단위
- 인덱스는 샤드 단위로 분리, 데이터를 분산해 저장

샤드



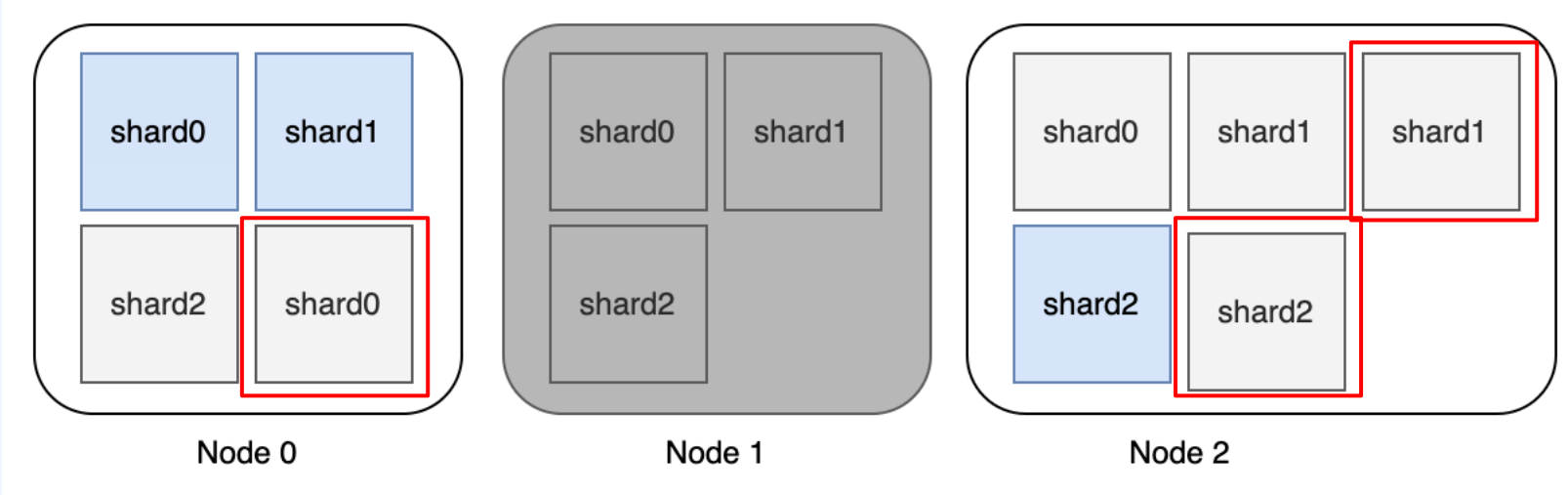
- 각 노드의 샤드에는 프라이머리 샤드와 레플리카 샤드가 존재
- 장애 상황에 대비를 위해 샤드의 복제본을 갖고 있음

샤드



- 노드 1번에 장애가 난다면, 노드1에 있는 프라이머리 샤드1을 사용할 수 없음
- Node0에서 기존에 갖고있던 레플리카 샤드1을 프라이머리로 사용한다

샤드

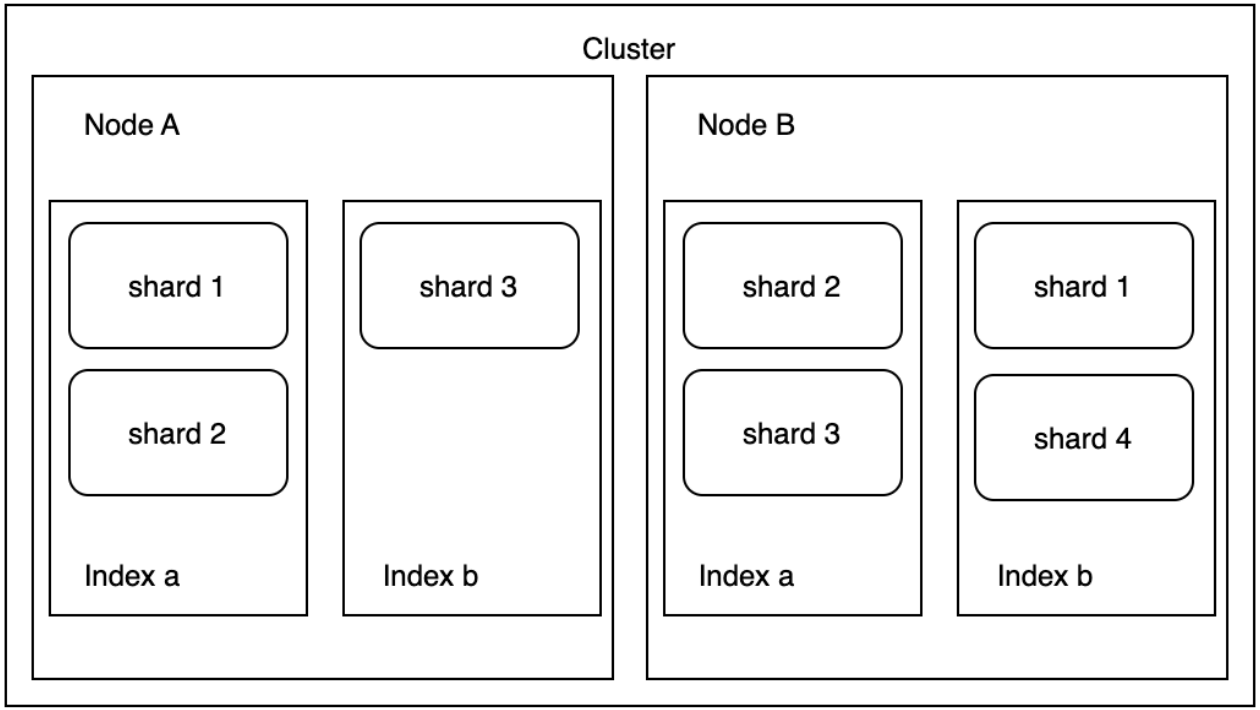


- 노드 1번이 복구되기를 기다리지만 복구가 되지 않는 경우
  설정된 레플리카 샤드 수대로 각 샤드의 복제를 수행한다
- 운영 중에 노드가 다운되어도 데이터 유실없이 가용성을 유지

Chapter 2. Elasticsearch의 기본 구성

02. ElasticSearch 클러스터, 노드

클러스터, 노드, 샤드



02. Elasticsearch 클러스터, 노드

클러스터

- 노드의 집합
- 모든 노드를 포괄하는 통합 색인화 및 검색 기능을 제공
- 고유 이름을 가짐
- 클러스터에 하나의 노드만 있는 것도 가능
- 고유한 클러스터를 여러 개 둘 수도 있음

02. Elasticsearch 클러스터, 노드

노드

- Elasticsearch를 구성하는 서버
- 데이터를 저장하고 클러스터의 Indexing 및 검색 기능
- 노드에도 고유한 이름이 존재(기본은 UUID)
- 클러스터 이름을 통해 해당 클러스터에 속할 수 있음
- 노드의 종류

02. Elasticsearch 클러스터, 노드

노드의 종류

마스터 노드

- 클러스터 상태(Cluster Status) 정보를 관리
- 지연이 없고 네트워크 가용성이 좋은 노드여야 함
- 마스터 노드는 하나, 최소 마스터 노드 대수 설정 필요 (Split Brain)

데이터 노드

- 문서가 저장되는 노드, 샤드가 배치된다
- 인덱싱 작업에 리소스를 많이 소모하므로 리소스 모니터링이 필수적

  마스터 노드와 분리하는 것을 추천,인덱싱할 문서가 적다면 괜찮음

*마스터 후보 노드들은

node.data: false 로 설정하여 마스터 노드 역할만 하고 데이터는 저장하지 않도록 설정 가능

노드의 종류

마스터 노드

- 클러스터가 1개의 노드로 이뤄지면, 1개의 노드가 마스터 노드가 됨
- 기존 마스터 노드가 다운되면, 다른 마스터 후보 노드 중 하나가 마스터 노드로 선출이 되어 마스터 노드의 역할을 대신 수행
- 마스터 후보 노드(master eligible)들은 처음부터 마스터 노드의 정보들을 공유하고 있기 때문에 즉시 마스터 역할의 수행이 가능
- 노드와 샤드의 개수가 많은 상황이면, 일부 노드만 마스터 노드 옵션을 true, 나머지는 false로 설정 부하를 줄이도록 하는게 좋음

노드의 종류

코디네이팅 노드

- 들어온 요청을 분산시켜주는 노드(Round-Robin)

인제스트 노드

- 색인 전에 데이터를 전처리 한다
- 데이터 포맷 변경을 위해서 스크립트로 파이프라인을 구성할 수 있다

Split Brain

- 마스터 후보 노드간 네트워크가 단절되었을 때
  각각의 마스터 후보 노드가 마스터로 승격, 두 개의 클러스터로 나뉘어 독립적으로 동작하는 현상
- 양쪽 클러스터에서 각각 데이터 쓰기, 업데이트가 이루어지면 네트워크가 복구되어도 마스터가 따로 존재하기 때문에 데이터 비동기 문제로 데이터의 문제가 발생

Split Brain

- discovery.zen.minimum_master_nodes 설정
  : (master_eligible_nodes / 2) + 1
- 마스터 노드 후보 개수를 홀수로 설정
- Split brain이 발생 후 네트워크가 복구된다면,
  투표를 통해 두 클러스터 중 하나를 선택하고 선택된 클러스터 기준으로 데이터를 맞춤
  짝수 노드인 경우 반반으로 나뉘어 결과가 도출되지 않음
  n/2+1 조건을 통해 조건을 넘지 못하는 분리된 클러스터는 동작하지 못하게 하고
  나머지 클러스터는 운영되도록 함
  짝수개로 설정해놨다면 둘 다 중지될 것

Chapter 2. Elasticsearch의 기본 구성

03. Elasticsearch 역색인, Inverted Index

03. Elasticsearch 역색인, Inverted Index

Inverted Index

- Elasticsearch의 검색이 빠른 이유는 Inverted Index 때문
- DB와의 차이점

  DB는 모든 문서의 내용을 읽어서 검색하는 키워드가 있는지를 확인 => 상당한 연산,느림
- Inverted Index는 특정 Term을 포함하고 있는 문서에 대한 인덱스 테이블을 만듬
- 이 인덱스 테이블에서 검색을 하기 때문에 빠른 것

Inverted Index

ID	Text
doc 1	The Kafka is great
doc 2	The ES is wonderful
doc 3	How about Airflow
doc 4	Kubernetes is good

Inverted Index

Where is ES?

ID	Text
doc 1	The Kafka is great
doc 2	The ES is wonderful
doc 3	How about Airflow
doc 4	Kubernetes is good

Inverted Index

DB 방식

Where is ES?

	ID	Text	
ES? →	doc 1	The Kafka is great	제외
ES? →	doc 2	The ES is wonderful	doc 2 선택
ES? →	doc 3	How about Airflow	제외
ES? →	doc 4	Kubernetes is good	제외

Inverted Index

역색인 생성

ID	Text
doc 1	The Kafka is great
doc 2	The ES is wonderful
doc 3	How about Airflow
doc 4	Kubernetes is good

Inverted Index

ID	Text
doc 1	The Kafka is great
doc 2	The ES is wonderful
doc 3	How about Airflow
doc 4	Kubernetes is good

Term	ID	Term	ID
The	doc1, doc2	wonderful	doc2
Kafka	doc1	How	doc3
ES	doc2	about	doc3
is	doc1,doc2,doc3	Airflow	doc3
great	doc1	good	doc4
wonderful	doc2		

Inverted Index

ID	Text
doc 1	The Kafka is great
doc 2	The ES is wonderful
doc 3	How about Airflow
doc 4	Kubernetes is good

Where is ES?

Term	ID	Term	ID
The	doc1, doc2	wonderful	doc2
Kafka	doc1	How	doc3
ES	doc2	about	doc3
is	doc1,doc2,doc3	Airflow	doc3
great	doc1	good	doc4
wonderful	doc2		

Inverted Index

ID	Text
doc 1	The Kafka is great
doc 2	The ES is wonderful
doc 3	How about Airflow
doc 4	Kubernetes is good

Where is ES?

Term	ID	Term	ID
The	doc1, doc2	wonderful	doc2
Kafka	doc1	How	doc3
ES	**doc2**	about	doc3
is	doc1,doc2,doc3	Airflow	doc3
great	doc1	good	doc4
wonderful	doc2		

Inverted Index

- 문서가 수천만이 넘더라도 빠르게 검색이 가능 -> 검색시스템에 유리한 구조
- 자주 등장하는 단어(a, the, and 등등) 는 오히려 느려지게 됨
  Stop Words 설정을 통해 인덱스에서 제거, 검색어에 등장해도 무시 가능

- 업데이트, 삭제 및 삽입에 대한 큰 스토리지 오버헤드 및 높은 유지 관리 비용

Chapter 3. AWS EC2에 ElasticSearch 설치하기

Chapter 3. AWS EC2에 ElasticSearch 설치하기

01. Elasticsearch와 Opensearch 그리고 버전

01. Elasticsearch와 Opensearch 그리고 버전

Opensearch

Amazon OpenSearch Service가 Elasticsearch는 아닙니다

Amazon OpenSearch Service(예전의 Amazon Elasticsearch Service)는 Elasticsearch가 아니며 Elasticsearch의 개발사인 Elastic과의 파트너십도 아닙니다. Amazon의 OpenSearch Service는 Elasticsearch의 오래된 포크 버전을 기반으로 하며 Elastic에서 직접 사용할 수 있는 기능, 선택 및 지원의 일부를 제공합니다.

Elastic의 공식 제품인 Elasticsearch 서비스를 선택하세요. 이 공식 제품은 AWS 내에서 Elasticsearch, Kibana 및 완전한 Elastic 환경을 제공합니다. Elastic의 검색 전문성이 보안 업데이트, 솔루션 및 지원을 통해 여러분의 배포를 지원하고 있으므로 안심하세요.

https://www.elastic.co/kr/amazon-opensearch-service

01. Elasticsearch와 Opensearch 그리고 버전

Opensearch

- AWS는 지난 Apache License, Version 2.0 (ALv2) 버전의 Elasticsearch 및 Kibana에서 포크를 만들어 유지하기로 결정했습니다.
- 이 포크를 OpenSearch라고 하며 ALv2 하에서 사용 가능합니다
- Opensearch는 Elasticsearch와 Kibana의 ALv2 오픈소스를 따와서 만든 것.
- Elasticsearch의 대부분의 기능을 사용 가능

01. Elasticsearch와 Opensearch 그리고 버전

Opensearch

- Elasticsearch는 지난 2021년 1월 Elastic은 소프트웨어 라이선스 전략을 변경
- Apache 라이선스 버전 2.0 (ALv2)에 따라 Elasticsearch 및 Kibana의 새 버전을 출시하지 않을 것이라고 발표
- 새 버전의 Elasticsearch는 오픈 소스가 아님
- 설치해서 사용할 수 있지만 추가 기능을 사용하기 위해서는 라이센스 구매가 필수적

01. Elasticsearch와 Opensearch 그리고 버전

Opensearch

- Opensearch는 완전한 오픈 소스임을 밝히고 있다.
- “OpenSearch는 Apache 2.0 라이선스 하에 제공되는 분산형 커뮤니티 기반 100% 오픈 소스 검색 및 분석 제품군으로, 실시간 애플리케이션 모니터링, 로그 분석 및 웹 사이트 검색과 같이 다양한 사용 사례에 사용됩니다.” (https://aws.amazon.com/ko/what-is/opensearch/)
- Elasticsearch의 모든 업데이트를 따라갈 수는 없다
- Opensearch의 자체 개발 및 업데이트

01. Elasticsearch와 Opensearch 그리고 버전

Opensearch

- “7.10 이후의 Elasticsearch 버전은 오픈 소스가 아니며 허용적 ALv2 라이선스 하에서 릴리스되지 않습니다.”
- “Amazon OpenSearch Service는 오픈 소스 Elasticsearch 버전 7.10까지 계속해서 지원할 것이지만 앞으로는 오픈 소스 OpenSearch의 새 버전을 활용하여 Amazon OpenSearch Service 사용 고객에게 로그 분석, 검색 및 관측성 등 다양한 사용 사례에 적합한 새로운 기능과 혁신을 제공할 것입니다.”

01. Elasticsearch와 Opensearch 그리고 버전

AWS OpenSearch? Opensearch?

- AWS OpenSearch는 AWS에서 제공되는 매니지드 Opensearch이다
- AWS OpenSearch는 클러스터 관리를 알아서 해준다는 점에서 훌륭하지만 내장된 플러그인만 사용가능하다
- Opensearch는 es에 등록되지 않은 플러그인들을 설치 할 수 있다.
- https://aws.amazon.com/ko/opensearch-service/getting-started/

Chapter 8. ElasticSearch Aggregation

Chapter 8. ElasticSearch Aggregation

01. 메트릭 집계, 버킷집계

01. 메트릭 집계, 버킷 집계

Aggregation

- v5.0이후로 집계기능이 대폭강화
- ES는 많은 양의 데이터를 잘라서 관리
  문서의 수가 늘어나도 좀 더 실시간에 가깝게 처리가능
- Elasticsearch에서 여러 연산이 가능한 기능
- 데이터를 그룹화하고 통계값을 얻을 수 있음

01. 메트릭 집계, 버킷 집계

Aggregation

- Aggregation은 Cost가 큼 (MEM)
- Master 노드가 노드의 데이터를 aggregate해서 결과를 전달 함
- 결과는 Cache에 저장
- 이미 계산된 결과는 cache를 통해 빠르게 가져올 수 있음
- Aggregation의 종류

  Metrics

  Bucket

01. 메트릭 집계, 버킷 집계

Cache

Node Query Cache

- Query 에 따른 결과를 Cache
- Node 레벨로 동작
- 캐시가 가득 차면 최근에 가장 적게 사용된 쿼리 결과가 제거 됨 (LRU, Least Recently Used)
- 계산이 필요한 query, aggregation등은 Node Query Cache가 적용되지 않음
- index.queries.cache.enabled : 활성화 유무
- indices.queries.cache.size : 보통 heap 메모리의 1% 사용
- filter query만 캐시

01. 메트릭 집계, 버킷 집계

Cache

Shard Request Cache

- 검색 요청이 인덱스 또는 여러 인덱스에 대해 실행될 때
- 관련된 각 샤드는 검색을 로컬환경에서 실행 후 결과를 coordinating node로 반환
- coordinating node는 각각의 결과를 수집하여 global 결과 집합으로 모음
- 모은 후, 각 샤드에서 로컬 결과를 캐시
- size: 0 일 때만 cache, 문서를 리턴받지 않고 집계 데이터만 사용하는 것
- shard 단위로 캐싱
- shard 내의 문서에 변화가 생기면 모든 cache 는 무효화
  문서의 변화가 거의 없는 정적인 shard 에서 사용하는 것을 권장

01. 메트릭 집계, 버킷 집계

Cache

Field Data Cache

- 검색 결과가 포함된 문서를 빠르게 접근하기 위한 캐시
- 집계 연산 수행할 때 사용됨
- 모든 필드의 값을 메모리에 로드, Cost가 많이 듬 => 모니터링 필수적
- indices.fielddata.cache.size 캐시 사용공간 설정
- 일단 먼저 메모리에 데이터를 올리는 방식이기에 OOM 발생가능
- **Circuit breaker** 사용 필요
  메모리에 올리기 전에 데이터의 사이즈를 측정해서 OOM 을 막는 방식
- indices.breaker.fielddata.limit 로 제한 (default 40%)
- indices.breaker.fielddata.overhead: 메모리 오버헤드를 허용 (default: 1.03)

01. 메트릭 집계, 버킷 집계

Metrics, Bucket Aggregation

Metric Aggregation

- 숫자 연산 할 수 있는 값들에 대한 agg
- Bucket 단위로 특정 연산을 수행
- sum, avg, max, min : 합, 평균, 최대, 최소
- stats : 기초 통계
- cardinality : 특정 field의 unique한 수 (모든 문서에 대한 중복 집계는 성능에 영향, 근사치)
- percentiles : 숫자 값의 백분위 수나 랭킹 구하기(문서 크기 커질 수록 오차 커짐, 근사치)
- Geo Bound, Geo Centroid : 지리 정보

01. 메트릭 집계, 버킷 집계

Metrics, Bucket Aggregation

Bucket Aggregation

- 범위나 keyword 값 등을 가지고 문서를 그룹화
- 조건으로 버킷들을 만들고 버킷 안의 문서를 모아 그룹으로 구분
- range : 숫자 필드 값으로 범위를 지정, 범위에 해당하는 버킷을 생성
- histogram : 주어진 간격 크기대로 버킷 생성
- terms : keyword 필드의 문자열 별 버킷 생성
- Bucket Agg를 사용해서 만든 Bucket 안에 다시 agg 또는 Metrics Agg를 만들 수 있음
  => 데이터 set을 메모리에 저장하는 것, 중첩할수록 메모리 사용이 증가함
- search.max_bucket 으로 최대 버킷 수 정의 가능

Chapter 9. ElasticSearch Mapping API

Chapter 9. ElasticSearch Mapping API

01. Mapping

01. Mapping

Mapping

- Indexing 할 때 데이터 유형에 따른 데이터 타입 지정이 필요 => 맵핑
- 일종의 데이터 모델링 (스키마 지정)
- 맵핑을 설정하면 지정된 데이터 타입으로 Indexing됨
- 맵핑 지정하지 않으면 동적으로 Mapping이 됨 => 의도치 않은 문제 발생 가능
- 잘못된 타입 지정되면 수정할 수 없음

01. Mapping

Mapping

- Document의 필드 속성을 정의 – 데이터 타입
- 필드가 추가될 때마다 인덱스가 자동으로 업데이트
- 기존에 정의된 필드에 각각 다른 타입의 데이터가 입력되면 이후의 데이터 색인 생성 실패
  Reindex, 인덱스 삭제 후 재생성
- Dynamic Mapping 가능하지만, 실무에서 사용하지 않는 이유

01. Mapping

Setting

- 인덱스의 정보 단위 setting, mapping
- setting

  number_of_shards

  number_of_replicas
- 자연어 처리에 사용되는 analyzer, tokenizer, filter

01. Mapping

Mapping 파라미터

Indexing할 데이터를 어떻게 저장할지에 대한 설정들

- analyzer : 지정한 분석기로 형태소 분석을 수행, text타입은 기본적으로 사용해야 함
  default는 Standard Analyzer
- coerce : 자동 변환 허용 여부, “35”가 integer에 들어온다면?
- fielddata : 힙 공간에 생성하는 메모리 캐시, 메모리 부족 현상 발생 가능
  text 타입을 aggregation하고 싶은 경우에 사용. => Tag Cloud
- doc_values : 기본 캐시, 파일 시스템 캐시를 통해 디스크 데이터에 빠르게 접근

01. Mapping

Metadata Field

Indexing할 데이터를 어떻게 저장할지에 대한 설정들

- _index : 인덱스의 이름
- _type : Document가 속한 맵핑의 타입 정보
- _id : Document를 식별하는 키 값
- _source : Document의 데이터, json
- _field_names : null이 아닌 값을 포함하는 문서의 모든 필드
- _ routing : 특정 문서를 특정 샤드에 저장하기 위해 사용

Chapter 9. ElasticSearch Mapping API

02. 다양한 데이터 타입 소개

데이터 타입

Keyword

- 키워드 형태로 사용할 데이터에 적합
- 분석기를 거치지 않고 원문 그대로 인덱싱
- 검색 시 필터링, 정렬 필요, 집계
- keyword -> “elastic search” elastic, search (X)

데이터 타입

Text

- 인덱싱 시 지정한 Analyzer가 데이터를 문자열로 인식하고 분석을 시작
- 기본은 Standard Analyzer
- 문장 형태의 데이터에 적합함
- Full Text Search가 가능함 -> 특정 단어 검색이 가능
- 정렬, 집계가 필요한 경우 Text와 Keyword 멀티 필드로 지정 => 목적에 맞게 활용 가능
- fielddata

02. 다양한 데이터 타입 소개

데이터 타입

Array

- 데이터가 2차원 형식이라면 Array 데이터 타입 사용
- Array 안의 데이터는 모두 같은 타입으로 구성해야 한다
- 인덱스 필드에 배열 값이 입력되면 자동으로 Array로 저장
- “phone_type” : [“iphone”, “galaxy”]

Boolean

- true, false
- 문자로 들어와도 해석해서 저장됨

02. 다양한 데이터 타입 소개

데이터 타입

Date

"2023-07-07"
"2023-07-07T17:15:20"
"2023-07-07T17:15:20+09:00"
"2023-07-07T17:15:20.428Z”

- 위의 형식대로 오면(ISO8601) 자동으로 Date로 인식됨
- 다른 형식으로 오면 text, keyword로 저장됨
- epoch_mills, epoch_second 도 가능
- 원하는 형식으로 저장하려면 format 옵션 사용
- "format": "yyyy-MM-dd HH:mm:ss||yyyy/MM/dd||epoch_millis”
- 2023-07-07 13:15:30, 2023/07/07 형식과 epoch_mills로 받는 예

02. 다양한 데이터 타입 소개

데이터 타입

IP

- IP 주소 데이터를 저장하는데 사용
- IPv4, IPv6 모두 지정가능

Range

- 숫자, 날짜, IP 등 범위 지정가능한 2차원의 형태로 저장
- **integer_range**, **float_range**, **long_range**, **double_range**, **date_range**, **ip_range**
- 데이터의 범위는 다음과 같이 gt, gte, lt, lte 를 사용해서 지정

02. 다양한 데이터 타입 소개

데이터 타입

Object

- JSON 에서는 내부 객체를 계층적으로 포함 가능
- 한 필드 안에 하위 필드 넣는 객체 타입 사용 가능
- 한 요소가 여러 하위 정보를 갖고 있는 경우에 사용함
- 문서 계층 구조에 따라 데이터를 입력

Nested

- Object 객체 배열을 독립적으로 색인하고 쿼리하는 형태의 데이터 타입
- 객체 내 배열의 형태로 저장 가능