

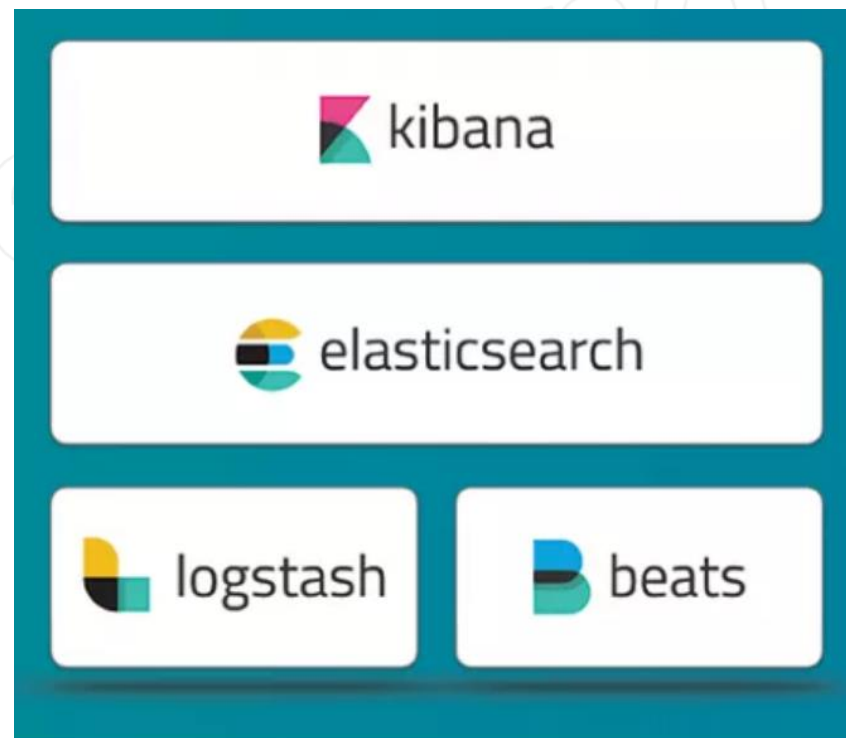
# Elastic Stack

Kim Hye Kyung  
topickim@naver.com

# | Elastic Stack

# Elastic Stack이란?

- ELK(ELK Stack)
  - Elasticsearch + Logstash + Kibana를 같이 묶어 서비스명으로 제공
  - 데이터 분석
  - 데이터 시각화
  - 훌륭한 검색 엔진기능
- Elastic Stack history
  - 5.0.0 버전부터 Beats가 포함
  - Elasticsearch + Logstash + Kibana

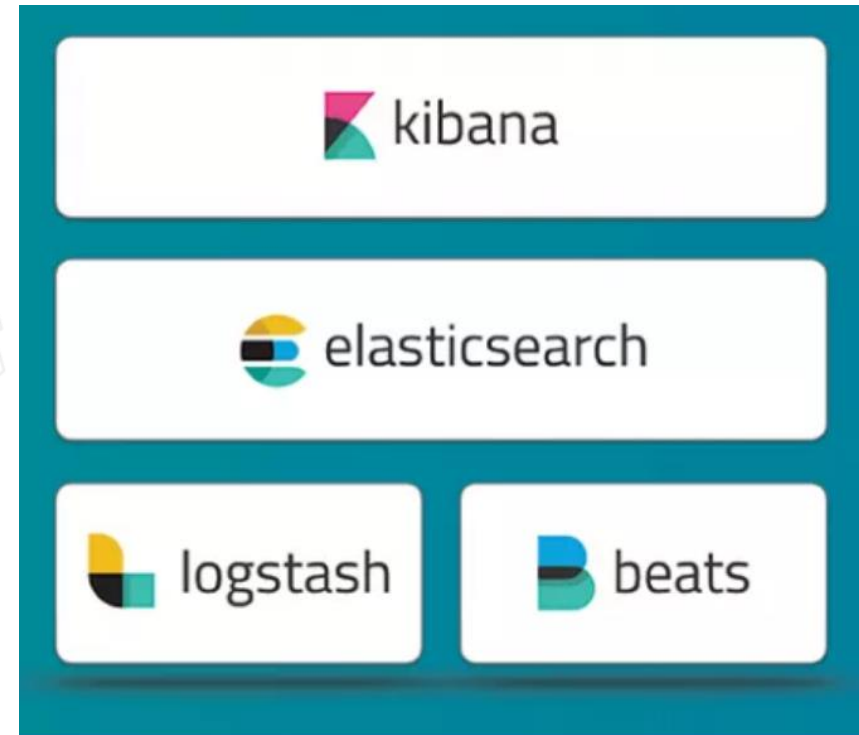


Elastic Stack

# Elastic Stack 주요 기능

명 칭	설 명
ElasticSearch	NoSQL 의 하나 데이터 저장소
Beats	데이터 수집
Logstash	데이터 가공 후 ElasticSearch에 저장
Kibana	데이터 시각화

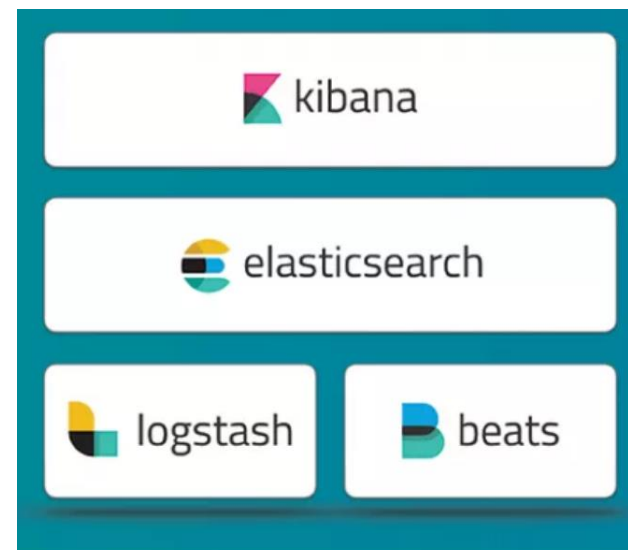
- 데이터를 시각화해주는 도구



Elastic Stack

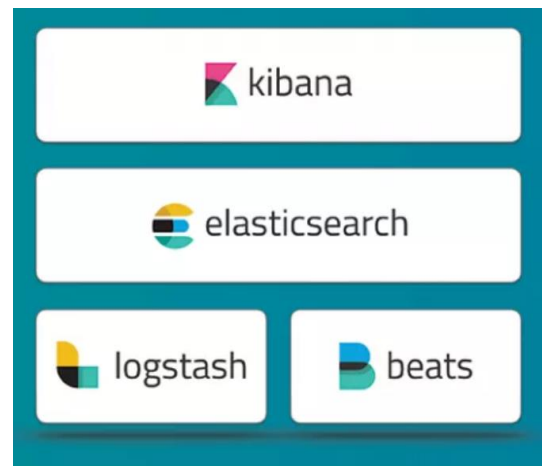


- 루씬 기반의 Full Text로 검색이 가능한 오픈소스 분석엔진
  - 텍스트 기반 검색 엔진
- 주로 REST API를 이용해 처리
- 대량의 데이터를 신속하고 거의 실시간으로 저장, 검색 및 분석 할 수 있음



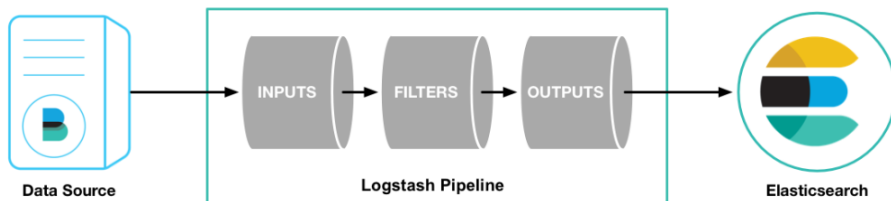
Elastic Stack

- 다양한 플러그인을 이용하여 데이터 집계 및 보관, 서버 데이터 처리
- 파이프라인으로 데이터를 수집하여 필터를 통해 변환 후 Elastic Search로 전송



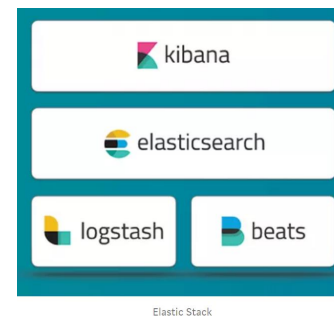
Elastic Stack

- 파이프라인으로 데이터를 수집하여 필터를 통해 변환 후 Elastic Search로 전송
  - 입력 : Beats, Cloudwatch, Eventlog 등의 다양한 입력을 지원하여 데이터 수집
  - 필터 :
    - 형식이나 복잡성에 상관없이 설정을 통해 데이터를 동적으로 변환
    - 수집된 데이터는 필터의 각 플러그인에 의해 가공
    - 가공 목적에 따라 여러 플러그인에서 사용 가능
    - 가령 grok 패턴을 사용해 비정형 데이터를 수집했다면 데이터 구조를 구성하거나 IP 주소로부터 지시적 위치 좌표를 생성하는 일이 가능
  - 출력 : Elastic Search, Email, ECS, Kafka 등 원하는 저장소에 데이터를 전송

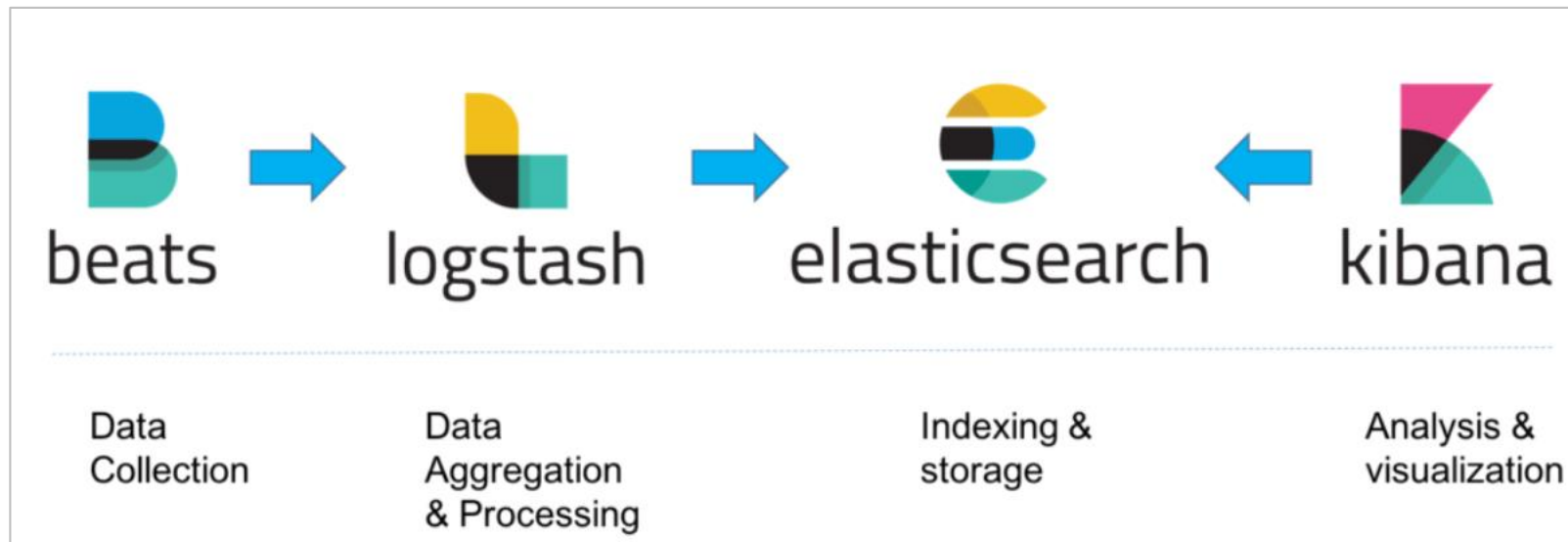




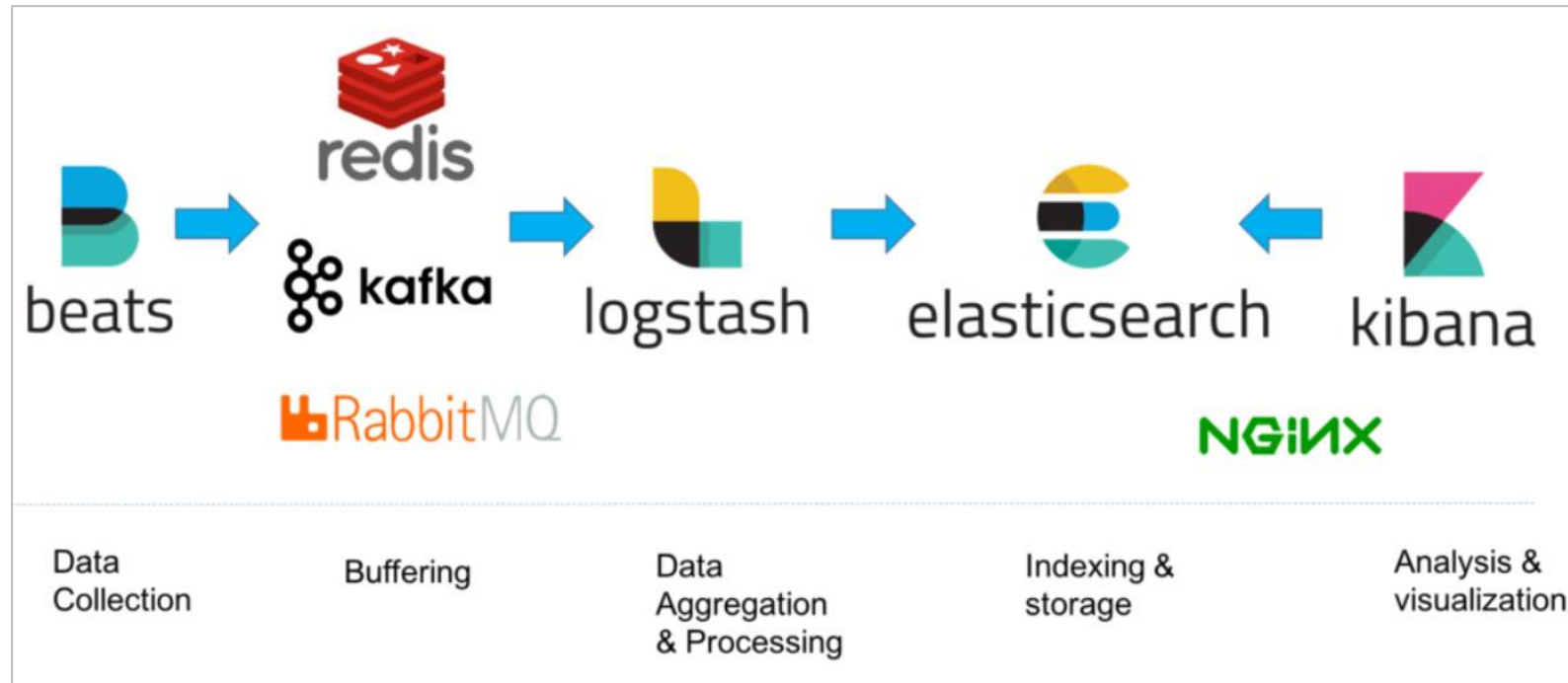
- 경량 에이전트로 설치되어 데이터를 Logstash 또는 Elastic Search로 전송하는 도구
- Logstash보다 경량화되어 있는 서비스
- Filebeat, Metricbeat, Packetbeat, Winlogbeat, Heartbeat 등이 있음
  - Packetbeat은 응용 프로그램 서버간에 교환되는 트랜잭션에 대한 정보를 제공하는 네트워크 패킷 분석기
  - Filebeat는 서버에서 로그 파일을 제공
  - Metricbeat은 서버에서 실행중인 운영 체제 및 서비스에서 메트릭을 주기적으로 수집하는 서버 모니터링 에이전트
  - Winlogbeat는 Windows 이벤트 로그를 제공



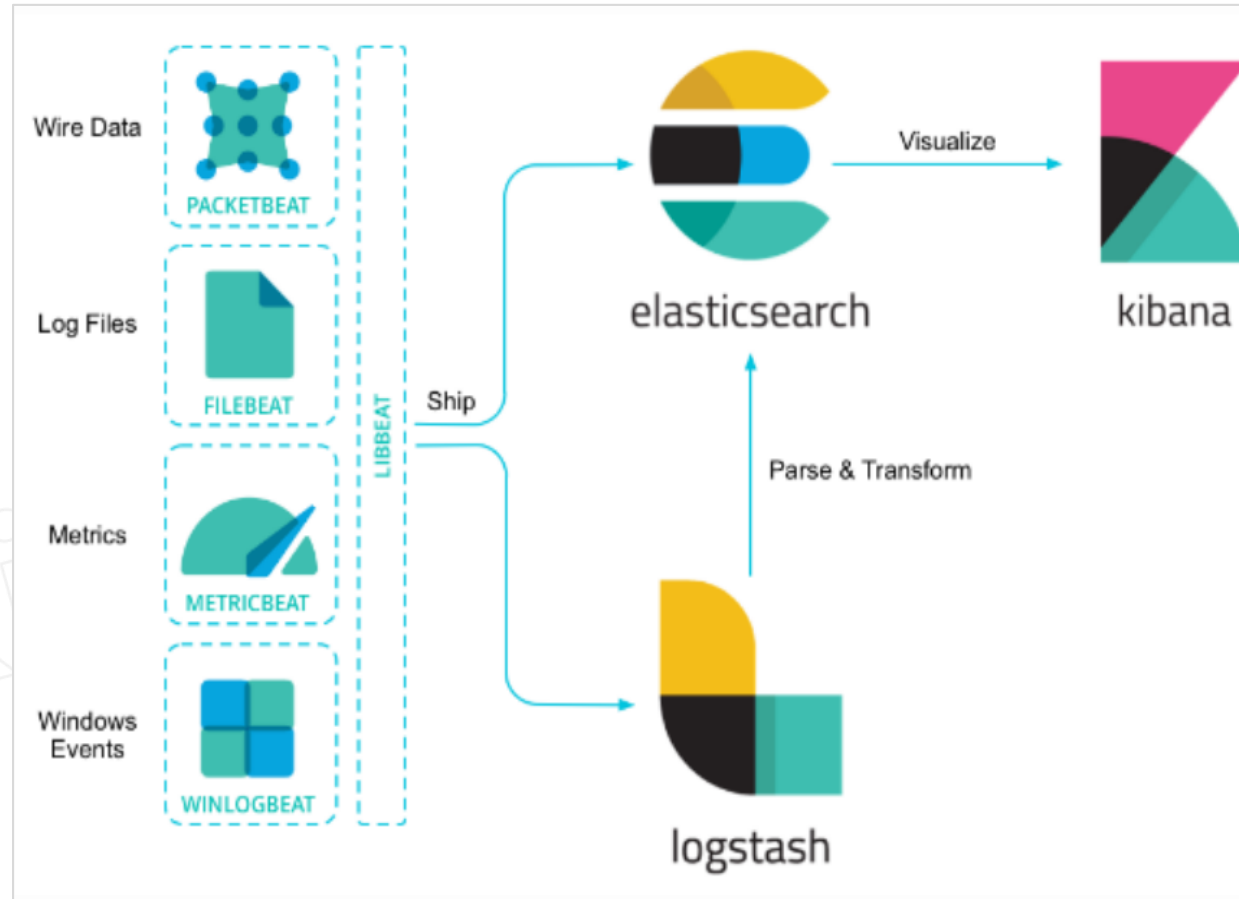
# Elastic Stack의 Flow



# Elastic Stack의 Flow



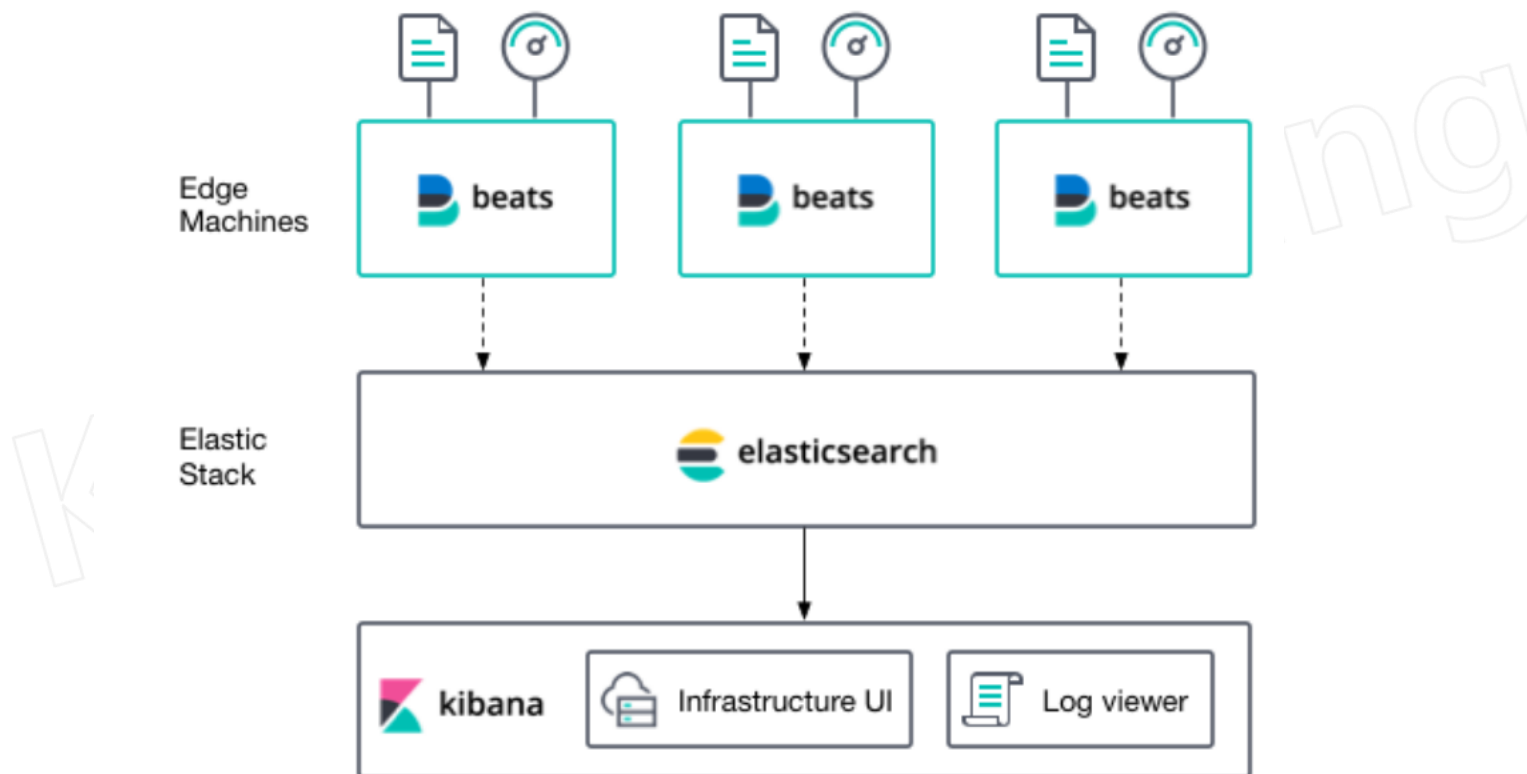
# Elastic Stack의 Flow



Elastic Stack의 Flow

# Elastic Stack의 Flow

## Infrastructure monitoring components



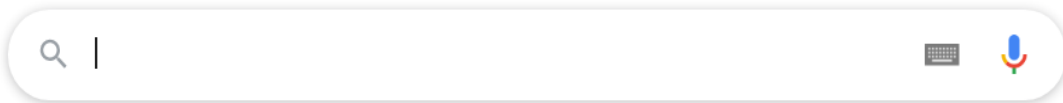
# | 검색 시스템

# 검색 시스템 이해하기

- 검색 엔진?
- 검색 시스템?
- 검색 서비스?



# 검색 시스템 이해하기



- 구글 & 네이버의 대표 서비스
  - 검색 서비스
- 검색 시스템이란?
  - 사용자가 원하는 검색어에 대한 결과를 제공



# 검색 시스템 이해하기

## 검색 엔진(Search Engine)

- 광활한 웹에서 정보를 수집해 검색 결과를 제공하는 프로그램
- 검색 결과로 제공되는 데이터의 특성에 따라 구현 형태가 각각 달라짐
- 예
  - Yahoo
    - 검색 역사에 한 획을 그음
    - 디렉터리 기반의 검색 결과를 세계 최초로 제공
      - 영향력
        - 뉴스, 블로그, 카페 등 대범주에 따른 카테고리별 검색 결과를 대부분의 검색 업체에서 제공

# 검색 시스템 이해하기

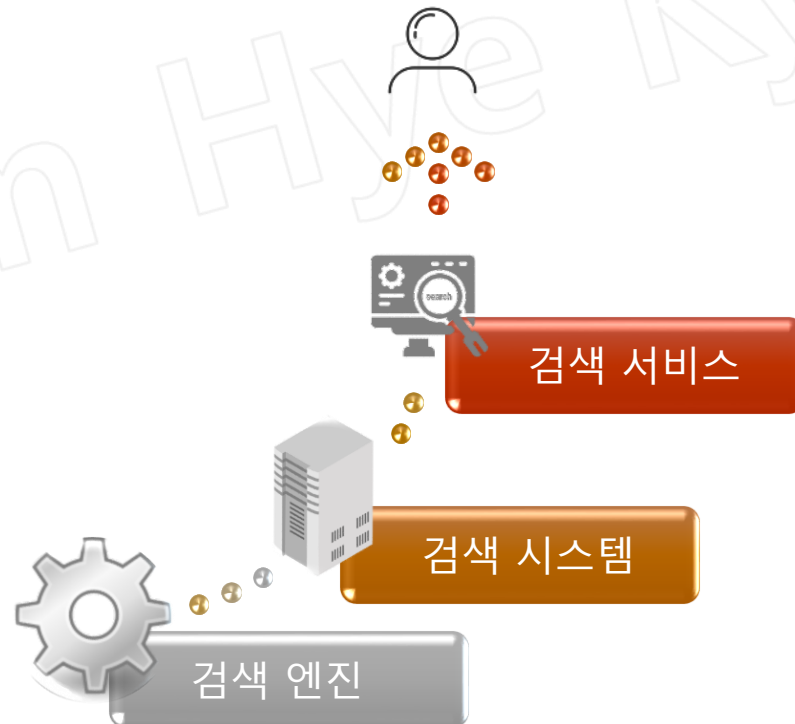
## 검색 시스템(Search System)

- 대용량 데이터를 기반으로 신뢰성 있는 검색 결과를 제공하기 위해 검색 엔진을 기반으로 구축된 시스템을 통칭하는 용어
- 수집기를 이용해 방대한 데이터 수집 -> 다수의 검색 엔진을 이용해 색인 -> 검색 결과를 UI로 제공
- 시스템 내부의 정책에 따라 가능한 작업
  - 관련도가 높은 문서를 검색 결과의 상위에 배치
  - 특정 필드나 문서에 가중치를 뒀서 검색의 정확도를 높일 수 있음

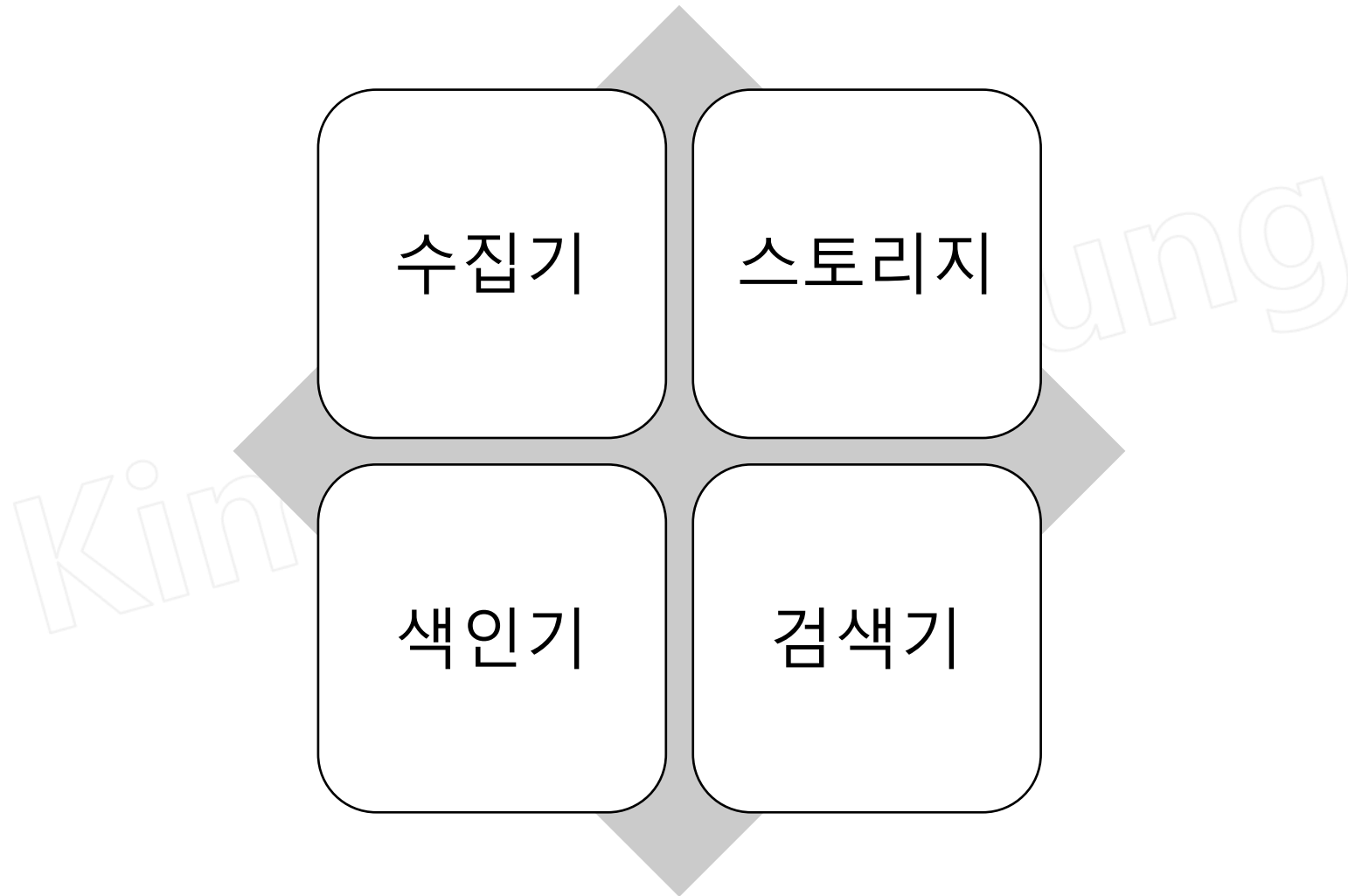
# 검색 시스템 이해하기

## 검색 서비스(Search Service)

- 검색 엔진을 기반으로 구축한 검색 시스템을 활용해 검색 결과를 서비스로 제공

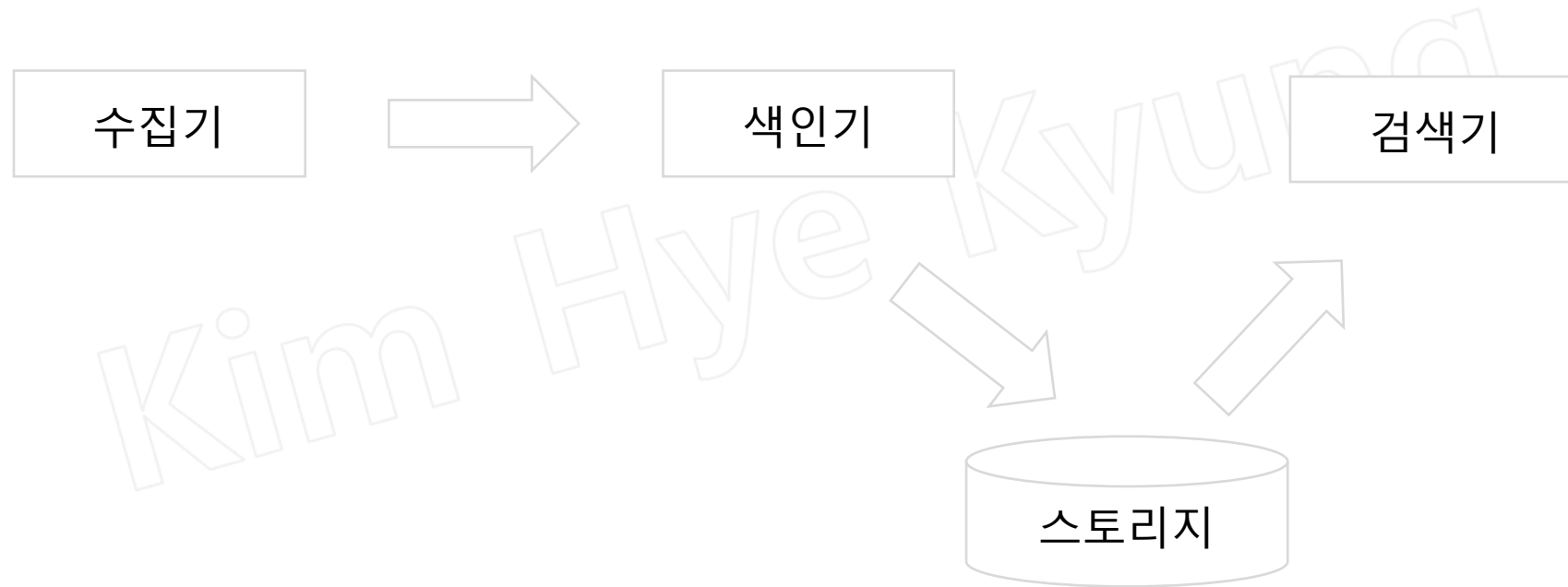


# 검색 시스템 구성 요소



# 검색 시스템 구성 요소

- 검색 시스템의 구성



# 색인 한다는 의미

---

- 역색인 파일을 만드는 것
  - 원문 자체를 변경한다는 의미는 아님
  - 색인 파일에 들어갈 토큰만 변경되어 저장되고 실제 문서의 내용은 변함없이 저장
  - 색인시 특정한 규칙과 흐름에 의해 텍스트를 변경하는 과정을 분석(Analyze)이라 함

# 역색인(inbverted index)

- 역색인(inbverted index) 이란?

- 종이책의 마지막 페이지에서 제공하는 색인 페이지와 비슷하게 제공되는 특수한 데이터 구조

- 예시

- '검색엔진' 이란 단어가 포함된 모든 문서를 찾아라?

- 일반적인 구조

- 처음부터 끝까지 모든 문서를 읽어야만 원하는 결과를 얻을 수 있음

- 역색인 구조

- 해당 단어만 찾으면 단어가 포함된 모든 문서의 위치를 알수 있음
      - 빠른 검색이 가능

단어	문서 번호
엘라스틱서치	1
문서	4,7
데이터베이스	2,8
역색인	2
검색엔진	2,10

## 역색인 구조

---

- 모든 문서가 가지는 단어의 고유 단어 목록
- 해당 단어가 어떤 문서에 속해 있는지에 대한 정보
- 전체 문서에 각 단어가 몇 개 들어 있는지에 대한 정보
- 하나의 문서에 단어가 몇 번씩 출현했는지에 대한 빈도



## 역색인 구조 예시

- text 보유한 2개의 문서가 있는 경우

문서 1

The autumn sky is blue.

문서 2

The Autumn sky is high.

- 역색인 단계

1단계 : 문서를 토큰화

2단계 : 토큰화된 단어에 대해 문서 상의 위치와 출현 빈도 등의 정보를 확인

# 역색인 구조 예시

- 역색인 단계

1단계 : 문서를 토큰화된 정보

문서 1

The autumn sky is blue.

문서 2

The Autumn sky is high.

토큰
The
autumn
Autumn
sky
is
blue
high

# 역색인 구조 예시

- 역색인 단계

2단계 : 토큰화된 단어에 대해 문서 상의 위치와 출현 빈도 등의 정보를 확인

문서 1

The autumn sky is blue.

문서 2

The Autumn sky is high.

토큰	문서 번호	Term의 위치	Term의 빈도
The	문서1, 문서2	1,1	2
autumn	문서1	2	1
Autumn	문서2	2	1
sky	문서1, 문서2	3,3	2
is	문서1, 문서2	4,4	2
blue	문서1	5	1
high	문서 2	5	1

# 역색인 구조 예시

토큰	문서 번호	Term의 위치	Term의 빈도
The	문서1, 문서2	1,1	2
autumn	문서1	2	1
Autumn	문서2	2	1
sky	문서1, 문서2	3,3	2
is	문서1, 문서2	4,4	2
blue	문서1	5	1
high	문서 2	5	1

- 알수 있는 정보
  - 토큰이 어떤 문서의 어디에 위치
  - 몇번 나왔는지(빈도)
- 문제
  - **ElasticSearch는 대소문자를 구분!!**
  - autumn을 검색어로 지정할 경우?
    - 문서1, 문서2에서 검색되어야 함
    - 문제점
      - 정확하게 일치하는 데이터만 출력
      - 따라서 문서1만 검색됨
  - 해결책
    - 색인 전에 전체를 소문자로 변환한 다음 색인

## 역색인(inverted index)의 장점

- ElasticSearch가 빠른 사유
  - 역색인(inverted index)

index
<ul style="list-style-type: none"><li>• 색인</li><li>• 책 목차 개념</li></ul>

inverted index
<ul style="list-style-type: none"><li>• 역색인</li><li>• 책 맨 뒷부분의 키워드로 검색하는 개념</li></ul>

# 검색 시스템 구성 요소

## 수집기

- 정보 수집

## 스토리지

- 수집한 데이터를 저장

## 색인기

- 수집한 데이터를 검색에 적절한 형태로 변환

## 검색기

- 색인된 데이터에서 일치하는 문서를 찾는 검색기

# 검색 시스템 구성 요소

## 수집기

- 정보 수집
- 웹사이트, 블로그, 카페 등 웹에서 필요한 정보를 수집하는 프로그램
- 크롤러(Crawler), 스파이더(Spider), 웜(Worms), 웹로봇(Web Robot)등으로 불리
- 수집 대상 : 파일, 데이터베이스, 웹페이지 등 웹상의 대부분의 정보가 수집 대상
- 파일의 경우 : 수집기가 파일명, 파일 내용, 파일 경로 등의 정보를 수집하고 저장하면 검색 엔진이 저장된 정보를 검색하고, 사용자 질의에 답함

## 스토리지

- 수집한 데이터를 저장
- 데이터베이스에서 데이터를 저장하는 물리적인 저장소
- 검색 엔진은 색인한 데이터를 스토리지에 보관

# 검색 시스템 구성 요소

## 색인기

- 수집한 데이터를 검색에 적절한 형태로 변환
  - 검색 엔진이 수집한 정보에서 사용자 질의와 일치하는 정보를 찾으려면 수집된 데이터를 검색 가능한 구조로 가공 및 저장해야 함
- 다양한 형태소 분석기를 조합해 정보에서 의미 있는 용어를 추출 및 검색에 유리한 역색인 구조로 데이터 저장

## 검색기

- 색인된 데이터에서 일치하는 문서를 찾는 검색기
- 사용자 질의를 입력받아 색인기에서 저장한 역색인 구조에서 일치하는 문서를 찾아 결과로 반환
- 질의와 문서가 일치하는지는 유사도 기반의 검색 순위 알고리즘으로 판단
- 색인기와 마찬가지로 형태소 분석기를 이용해 사용자 질의에서 유의미한 용어를 추출해 검색
- 사용하는 형태로 분석기에 따라 검색 품질이 달라짐





## 쉬어가기 – 용어 이해하기

- index
  - RDB에서의 index
    - where 절의 쿼리와 join을 빠르게 만들기 위한 보조 데이터 도구

Kim Hye Kyung

# 관계형 데이터베이스와의 차이점 이해하기

## 관계형 데이터베이스

- 데이터베이스는 데이터를 통합 관리하는 데이터의 집합
- 저장되는 구조
  - 중복을 제거
  - 정형 데이터 구조화
  - 행과 열로 구성
  - 테이블 구조로 저장
- sql 문을 이용해 원하는 정보의 검색이 가능
- 텍스트 매칭을 통한 단순한 검색만 가능
- 텍스트를 여러 단어로 변형하거나 여러 개의 동의어나 유의어를 활용한 검색은 불가능

## 검색 엔진

- 데이터베이스에서는 불가능한 비정형 데이터를 색인하고 검색 할 수 있음
- 형태소 분석을 통해 사람이 구사하는 자연어 처리가 가능해짐
- 역색인 구조를 바탕으로 빠른 검색 속도 보장

# 관계형 데이터베이스와의 차이점 이해하기

- CRUD 작업의 차이점
  - ElasticSearch
    - HTTP를 통해 JSON 형식의 RESTful API 사용

Elastic Search	Relation DB	CRUD 기능
GET	데이터 조회(select)	데이터 조회
PUT	데이터 생성, 수정(insert, update)	데이터 생성, 수정
POST	데이터 생성, 수정, 검색(update, select)	인덱스 생성, 수정, 조회
DELETE	데이터 삭제(delete)	데이터 삭제
HEAD	-	인덱스 정보 확인

# 기본 명령어

Elastic Search	Relation DB	CRUD
GET	Select	Read
PUT	Update, Insert	Update, Insert
POST	Insert, Update	Create, Read, Update
DELETE	Delete	Delete

# REST API[Representational State Transfer]

- 직관적인 의미 : 대표적인 상태 전달

REST란?

웹에 존재하는 모든 자원(이미지, 동영상, DB 자원)에 고유한 URI를 부여해 활용하는 것으로,  
자원을 정의하고 자원에 대한 주소를 지정하는 방법론을 의미

Restful API란?

REST 특징을 지키면서 API를 제공하는 것을 의미

**HTTP 헤더(header)와 URL만 사용해 다양한 형태의 요청을 할 수 있는 HTTP 프로토콜을**

**최대한 활용하도록 고안된 아키텍처**

# ElasticSearch의 RESTful API

- API 요청 구조

<http://host:port/인덱스/타입/문서id>

- RDBMS의 sql과 비교

- 예시 : customer의 주소(address)에 "서초"가 포함된 고객의 모든 데이터 검색을 요할 경우

- RDBMS

select \* from customer from address like '%서초%' -> table 형태로 검색

- ElasticSearch

GET [http://ip:port/customer/\\_search?q=address:서초](http://ip:port/customer/_search?q=address:서초) -> json 형태로 검색

# | 검색 시스템 & Elasticsearch

# ElasticSearch 특징

---

- Shay Banon이 Lucene을 바탕으로 개발한 오픈소스 분산 검색엔진
- 2010년 2월 첫 버전이 공개
- 설치와 서버 확장이 매우 편리
- 실시간 검색 (Near Real Time)
- 자체적으로 클러스터를 탐색하고 관리



# 검색 엔진 관점에서의 Elasticsearch 특징

- 대용량 데이터를 빠르게 검색하기 위해 NoSQL(No Structured Query Language) 많이 사용
- 분류가 가능하고 분산 처리를 통해 실시간에 준하는 빠른 검색이 가능
- 기존 데이터베이스로 처리하기 힘겨운 대량의 비정형 데이터 검색 가능
- 전문 검색(full text)과 구조 검색 모두 지원
  - 전문 검색이란?
    - 내용 전체를 색인해서 특정 단어가 포함된 문서를 검색하는 것 의미
    - RDB – 전문 검색에 부적합
    - Elasticsearch – 다양한 기능별, 언어별 플러그인을 조합해 빠른 검색이 가능
- MongoDB나 Hbase처럼 대용량 스토리지로도 활용 가능

# 검색 엔진 관점에서의 Elasticsearch 특징

- JSON 기반의 스키마 없는 저장소
  - 검색 엔진이지만, NoSQL처럼 사용할 수 있음
  - 데이터 모델을 JSON으로 사용하고 있어서, 요청과 응답을 모두 JSON 문서로 주고받음
  - 소스 저장도 JSON 형태로 저장
  - 스키마를 미리 정의하지 않아도, JSON 문서를 넘겨주면 자동으로 인덱싱
  - 숫자나 날짜 등의 타입은 자동으로 매핑
  - 구조화된 JSON으로 검색 자체적으로 통계 정보 반환

# 검색 엔진 관점에서의 Elasticsearch 특징

- 통계 분석
  - 비정형 로그 데이터 수집 후 통계 분석 가능
  - 키바나 연계시 실시간으로 로그 데이터 시각화 가능
- 스키마리스
  - RDB와 달리 다양한 형태의 문서도 자동으로 색인 및 검색 가능
- RESTful API
  - HTTP 기반의 RESTful API 로 요청 및 JSON 형태의 응답으로 개발언어, 운영체제, 시스템에 비 종속적
  - 이기종 플랫폼에서 이용 가능
- 멀티테넌시(Multi-tenancy)
  - 서로 상이한 인덱스일지라도 검색할 필드명만 동일하다면 여러 개의 인덱스를 한번에 조회 가능

# 검색 엔진 관점에서의 Elasticsearch 특징

- Document-Oriented
  - 여러 계층의 데이터를 JSON 형식의 구조화된 문서로 인덱스에 저장 가능
  - 계층 구조로 문서도 한번의 쿼리로 쉽게 조회 가능
- 역색인(inverted index)
  - NoSQL지원하는 MongoDB or Cassandra와 달리 역색인 지원
- 확장성과 가용성
  - 대용량 데이터 저장 및 색인시 막대한 비용과 시간 소비되나 ElasticSearch를 분산 구성해서 확장한다면 대량의 문서를 좀 더 효율적으로 처리 가능
  - 분산 환경에서 샤드(shard)라는 작은 단위로 나뉘어 제공, 인덱스를 만들 때마다 샤드의 수를 조절해서 데이터의 종류와 성격에 따라 분산해서 빠르게 처리 가능

# 검색 엔진 관점에서의 Elasticsearch 특징

- Multi-tenancy
  - 하나의 elasticsearch 서버에 여러 인덱스를 저장하고, 여러 인덱스의 데이터를 하나의 쿼리로 검색할 수 있음

예제 1 Multi-tenancy 예제 쿼리

```
# log-2012-12-26 인덱스에 로그 저장
curl -XPUT http://localhost:9200/log-2012-12-26/hadoop/1 -d '{
  "projectName" : "hadoop",
  "logType": "hadoop-log",
  "logSource": "namenode",
  "logTime": "2012-12-26T14:12:12",
  "host": "host1",
  "body": "org.apache.hadoop.hdfs.StateChange: DIR* NameSystem.completeFile"
}'

# log-2012-12-27 인덱스에 로그 저장
curl -XPUT http://localhost:9200/log-2012-12-27/hadoop/1 -d '{
  "projectName" : "hadoop",
  "logType": "hadoop-log",
  "logSource": "namenode",
  "logTime": "2012-12-27T02:02:02",
  "host": "host2",
  "body": "org.apache.hadoop.hdfs.server.namenode.FSNamesystem"
}'

# log-2012-12-26, log-2012-12-27 인덱스에 한번에 검색 요청
curl -XGET http://localhost:9200/log-2012-12-26, log-2012-12-27/_search
```

# ElasticSearch의 query 한글 문서

---

- <https://www.elastic.co/guide/kr/elasticsearch/reference/current/gs-executing-aggregations.html>

Kim Hye Kyung

# | Elasticsearch

# Elasticsearch 구성 이해하기

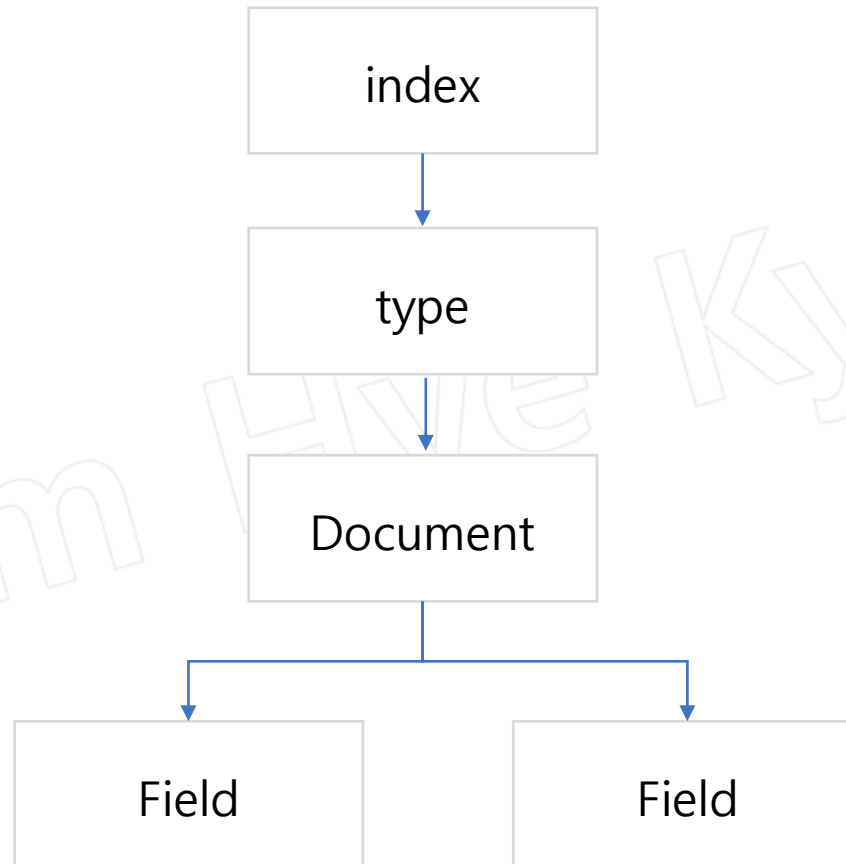
- 관계형 데이터베이스와 elasticsearch 용어 비교

관계형 데이터베이스	elasticsearch
Database	Index
Table	Type
Row	Document
Column	Field
Schema	Mapping
Index	Everything is indexed
SQL	Query DSL



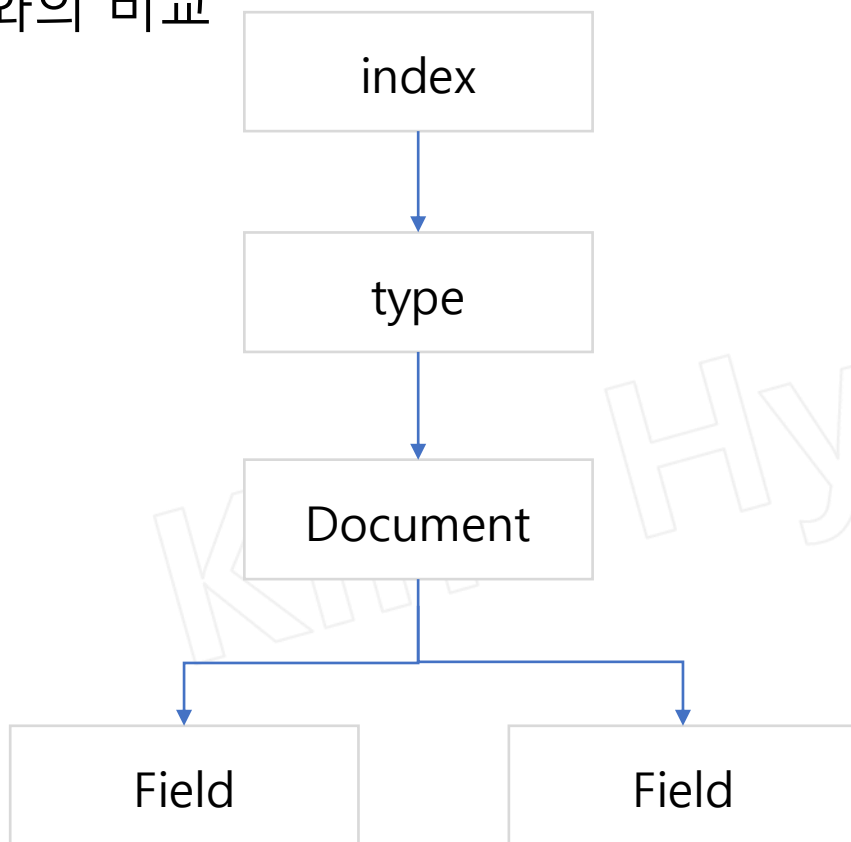
# Elasticsearch 구성 이해하기

- 구성

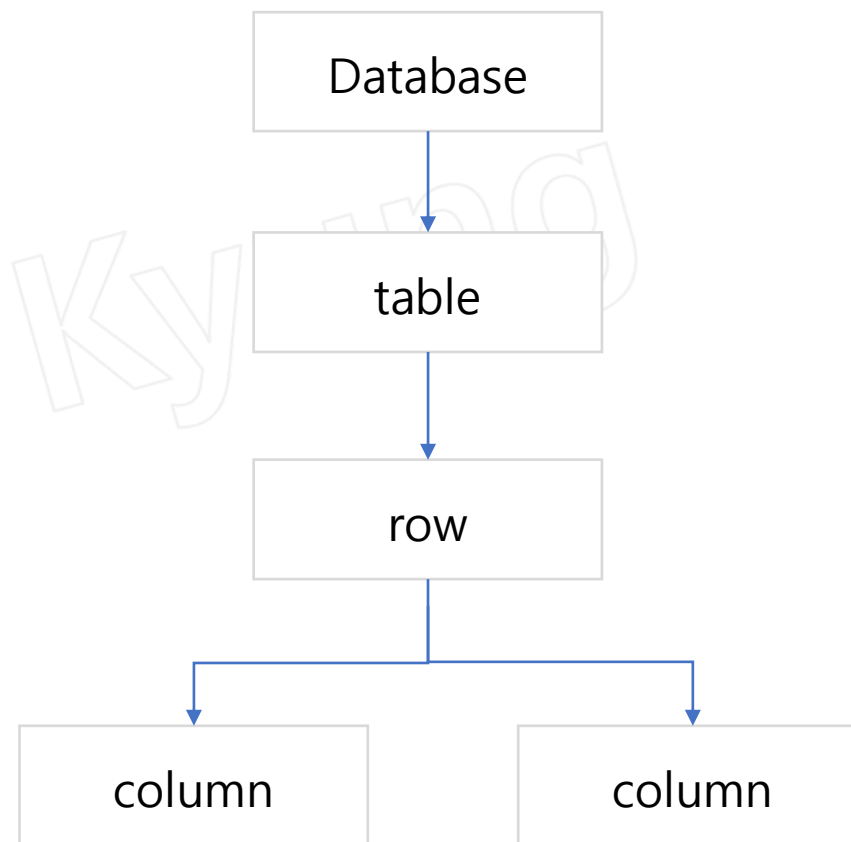


# Elasticsearch 구성 이해하기

- RDB와의 비교

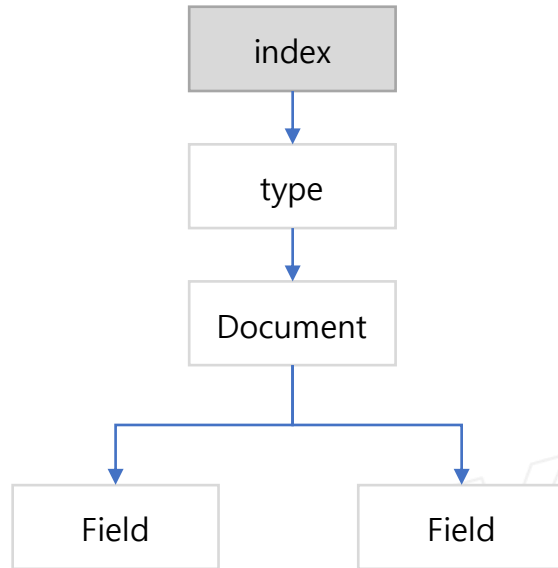


elasticsearch



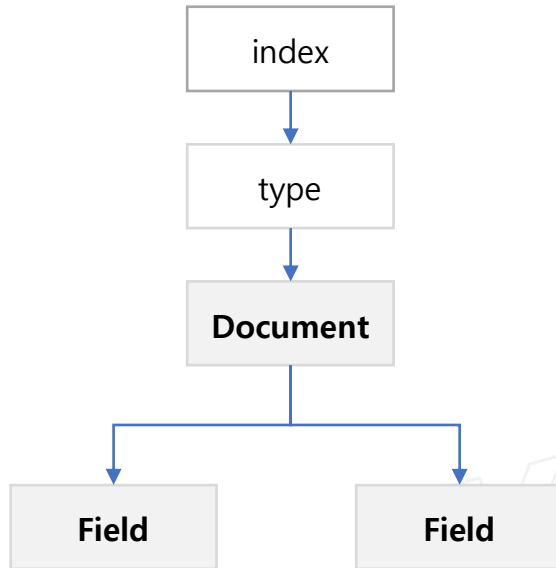
RDB

# Elasticsearch 구성 이해하기



용 어	설 명
index	데이터 저장 공간 이름은 소문자 여야만 함 추가, 수정, 삭제, 검색은 RESTFul API로 수행
샤드	인덱스 내부에 색인된 데이터는 물리적인 공간에 여러 개의 파티션으로 나뉘어 구성 이 파티션을 샤드라고 함 다수의 샤드로 문서를 분산 저장하고 있기 때문에 데이터 손실 위험이 최소화 하나의 엘라스틱서치에는 2개의 물리적인 노드 존재 인덱스 문서 조회시 2개의 노드 모두 조회하고 각 데이터를 취합한 후 하나로 합쳐서 제공
type	인덱스의 논리적 구조 인덱스 속성에 따라 분류 v6.1 이상 부터는 인덱스당 하나의 타입만 사용

# Elasticsearch 구성 이해하기



용 어	설 명
문서	하나의 행을 문서라 부름(테이블의 row과 흡사한 개념) 데이터가 저장되는 최소 단위 JSON 포맷으로 데이터가 저장 다수의 field로 구성
필드	문서를 구성하기 위한 속성 데이터의 형태에 따라 용도에 맞는 데이터 타입을 정의해야 함 RDBMS의 column과 흡사한 개념
매핑	필드의 구조와 제약조건에 대한 명세 RDB의 개념 관점에서 스키마라 함 문서의 필드와 필드의 속성을 정의하고 그에 따른 색인 방법을 정의하는 프로세스

# Elasticsearch 구성 이해하기

- 문서를 색인화 한다는 의미
  - Rest API를 통해 index에 document를 추가
- Rest API로 document 추가 및 조회
  - -d 옵션
    - 추가할 데이터를 json 포맷으로 전달합니다.
  - -H 옵션
    - 헤더를 명시
    - json으로 전달하기 위해서 application/json으로 작성
  - ?pretty
    - 결과를 예쁘게 보여주도록 요청

ver. window

```
C:\Users\Kimhyekyung>curl -H "Content-Type:application/json" -XPUT "localhost:9200/user/_doc/1?pretty" -d '{"name":"khk", "age":20}'
{
  "_index" : "user",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

```
curl -XPUT 'localhost:9200/user/1?pretty' -d '{"name" : "khk", "age" : 20}' -H 'Content-Type: application/json'
```

# Elasticsearch 구성 이해하기

- Rest API로 document 추가 및 조회
  - 조회

```
curl -XGET "localhost:9200/user/_doc/1?pretty"
```

```
C:\Users\Kimhyekyung>curl -XGET "localhost:9200/user/_doc/1?pretty"
{
  "_index" : "user",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "name" : "khk",
    "age" : 20
  }
}
```

# | Elasticsearch Architecture



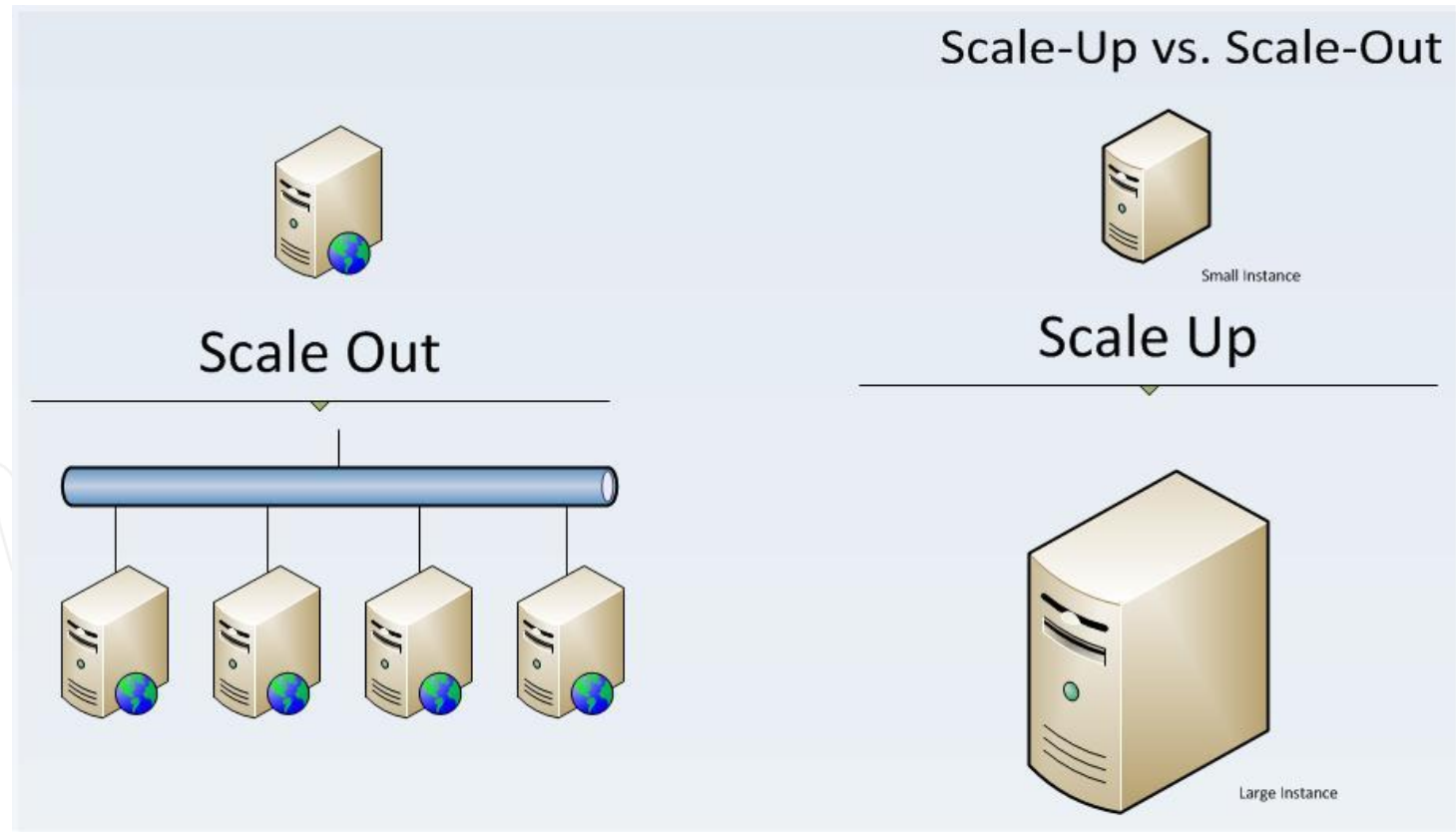
## 쉬어가기 - 용어정리

- 클러스터란?
  - 컴퓨팅 파워를 증가시키기 위한 방법
  - 컴퓨터의 집합을 의미
  - 여러 대의 일반 워크스테이션(노드)을 네트워크로 연결하여 하나의 PC 처럼 작동하게 하는 기술
  - 워크스테이션의 CPU 성능이 좋아지고 네트워크 속도 또한 엄청나게 발달하여 클러스터의 실 적용이 가능해짐
  - 클러스터 PC들의 OS는 오픈소스로 인해 자유롭게 튜닝이 가능한 리눅스 사용



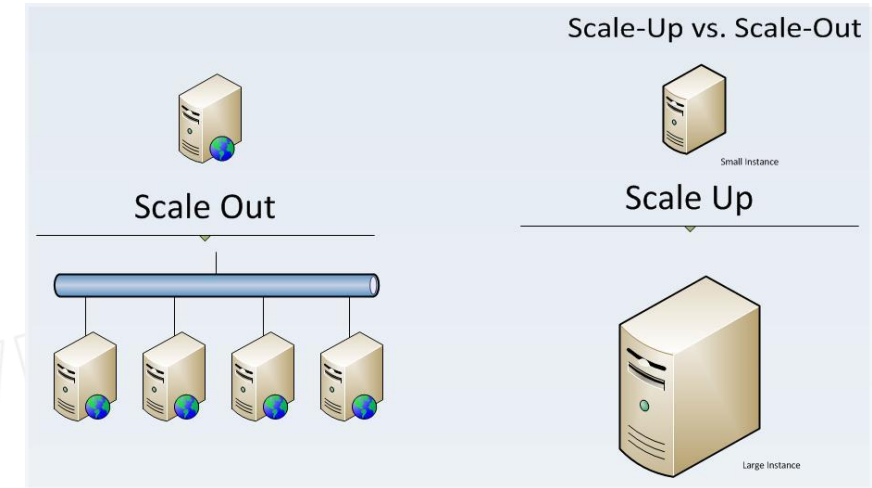
# 쉬어가기 – ScaleUp/ScaleOut

- Scale up : 사양추가
- Scale out : 장비추가



# 쉬어가기 – ScaleUp/ScaleOut

- Scale up : 사양추가
  - 수직 확장 개념
  - 복잡한 계산이 많을 경우 사양(CPU, RAM, DISK 구성등) 고가의 장비로 대체하여 처리 속도 향상
- Scale out : 장비추가
  - 수평 확장 개념
  - 분산처리, 병렬처리등으로 불리우기도 함
  - 단순 데이터 처리가 많은 환경에 적합
  - 단, 병렬 구조된 서버들에 데이터를 어떻게 동기화 해야 하며 세션은 어떤식으로 공유 해야 할지에 대한 기술적인 한계가 있음
  - 데이터의 정합성(데이터가 서로 모순 없이 일관되게 일치해야 하는 경우) 유지에 대한 요건이 별로 어렵지 않을 경우 적합



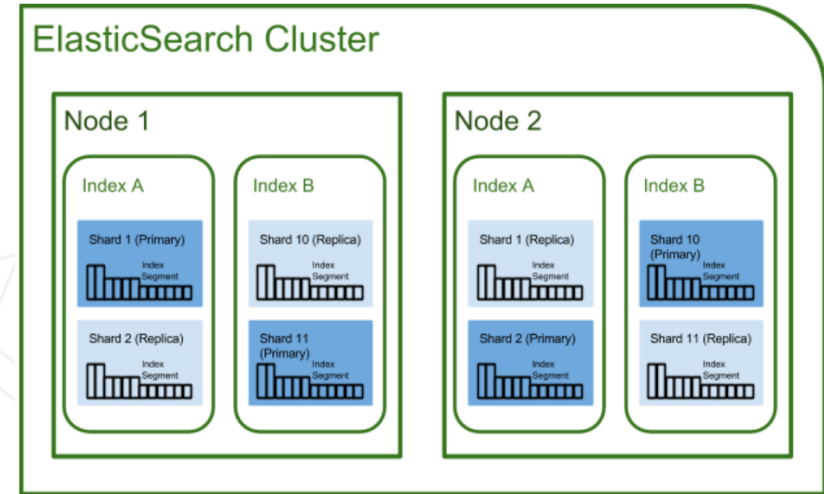
# Scale out과 Scale up 비교

	Scale out	Scale up
확장성	하나의 장비에서 처리하던 일을 여러 장비로 나눠서 처리(수평확장) 지속적 확장이 가능	더 빠른 속도의 CPU로 변경 또는 더 많은 RAM 추가 등의 HW 장비 성능 향상(수직 확장) 성능 확장에 한계가 있음
서버 비용	비교적 저렴한 서버 사용 일반적으로 비용 부담이 적음	성능 증가에 따른 비용 증가폭이 큼 일반적으로 비용 부담이 큼
운영 비용	대수가 늘어날수록 관리 편의성이 떨어짐 서버의 상면 비용을 포함한 운영 비용 증가	관리 편의성, 운영 비용은 스케일 업에 따라 큰 변화 없음
장애	읽기/쓰기가 여러대의 서버에 분산되어 처리됨으로 장애 시 전면 장애의 가능성이 적음	한대의 서버에 부하가 집중 장애시 장애 영향도가 큼
주요 기술	Sharding, NoSQL, In Memory Cache등	고성능 CPU, Memory 확장, SSD
주요 용도 장/ 단점	분산처리 시스템, 점진적 증가 가능, Scale up 보다 저렴 단점 : 설계, 구축, 관리 비용 증가	고성능 Legacy Application 구축이 쉽고 관리 용이 단점 : 단계적 증가가 어렵고 근본적인 해결이 안 될수도 있음

# ElasticSearch Architecture

- ElasticSearch Cluster

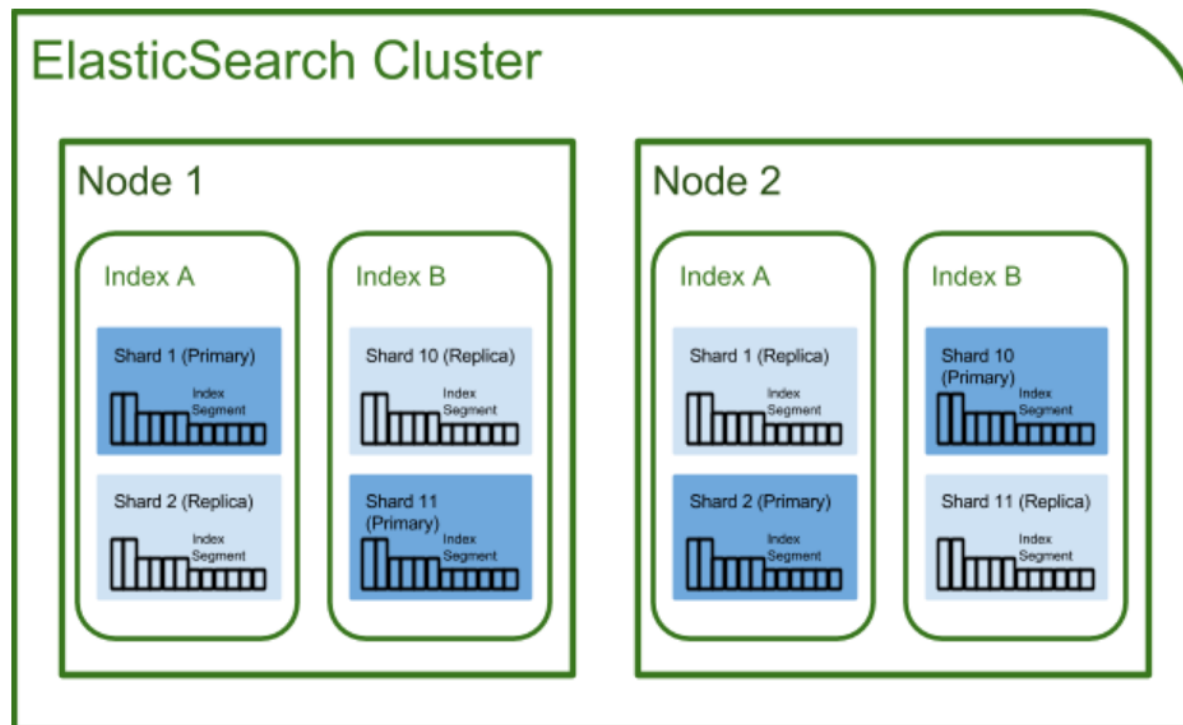
- 가장 큰 시스템 단위
- 하나 이상의 노드로 구성
- 여러대의 서버가 하나의 클러스터 구성 가능
- 한 서버에 여러 개의 클러스터 존재 가능



- 서로 다른 클러스터는 데이터의 접근, 교환 불가능한 독립적인 시스템으로 유지
- 클러스터에 있는 노드는 실시간으로 추가, 제거 가능 따라서 가용성이나 확장성 측면에서 매우 유연

# ElasticSearch Architecture

- 노드(node)
  - ElasticSearch를 구성하는 하나의 단위 프로세스
  - 역할에 따라 Master node, Data node등으로 구분



# ElasticSearch Architecture

## Master Node

- 클러스터를 제어하는 마스터
- 네트워크 속도가 빠르고 지연이 없는 노드를 선정해야 함
  - 인덱스 생성, 삭제
  - 클러스터 노드들의 추적 및 관리
  - 데이터 입력시에 어느 샤드에 할당할 것인지 지정

## Coordination Node

- 요청을 단순히 라운드로빈 방식으로 분산시켜주는 도구

## Data Node

- 문서가 실제로 저장되는 노드

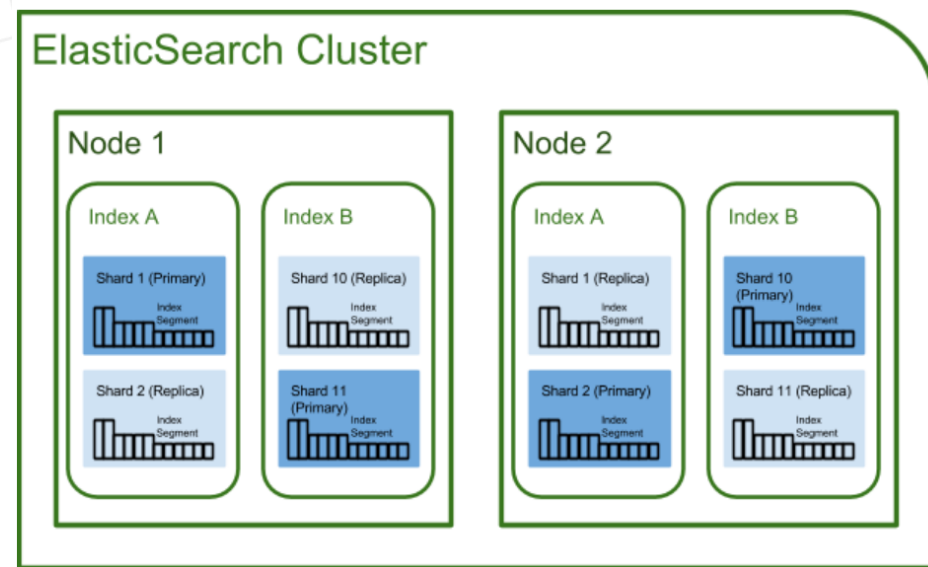
- 데이터와 관련된 CRUD
- 데이터가 실제

## Ingest Node

- 색인에 앞서 데이터를 전처리 하기 위한 node
- 데이터 포맷을 변경해야 할 경우 사전 처리 파이프라인을 실행하는 역할

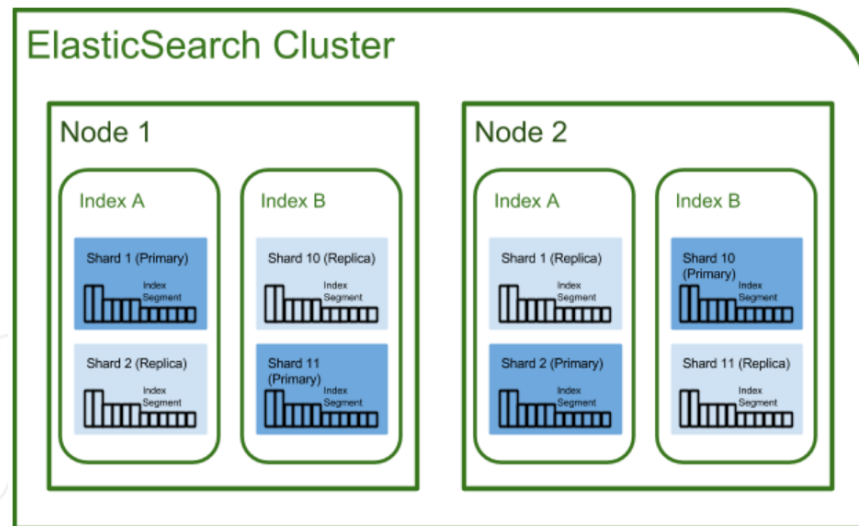
# ElasticSearch Architecture

- 하나의 ES 클러스터에 총 2개의 물리적인 노드 존재
- ES cluster
  - 인덱스의 문서 조회시 master node를 통해 2개의 node 모두 조회해서 각 데이터를 취합한 후 결과값을 하나로 합쳐서 제공
- Scale out
  - 샤드를 통해 규모가 수평적으로 늘어날 수 있음
- 고가용성
  - Replica를 통해 데이터의 안정성을 보장



# ElasticSearch 클러스터란?

- 클러스터란?
  - ES에서 가장 큰 시스템 단위
  - 물리적인 노드 인스턴스들의 집합
  - 모든 노드의 검색과 색인 작업 관장하는 논리적인 개념
- RDB : ElasticSearch 비교
  - RDB
    - 모든 요청을 서버 하나에서 처리해서 결과 제공
  - ElasticSearch
    - 다수의 서버로 분산해서 처리하는 것이 가능하기 때문에 대용량 데이터를 처리할 수 있음
    - Scale out
      - 샤드를 통해 규모가 수평적으로 늘어날 수 있음
    - 고가용성
      - Replica를 통해 데이터의 안정성을 보장
  - 분산 처리를 위해서는 다양한 형태의 노드들을 조합해서 클러스터 구성을 해야 함





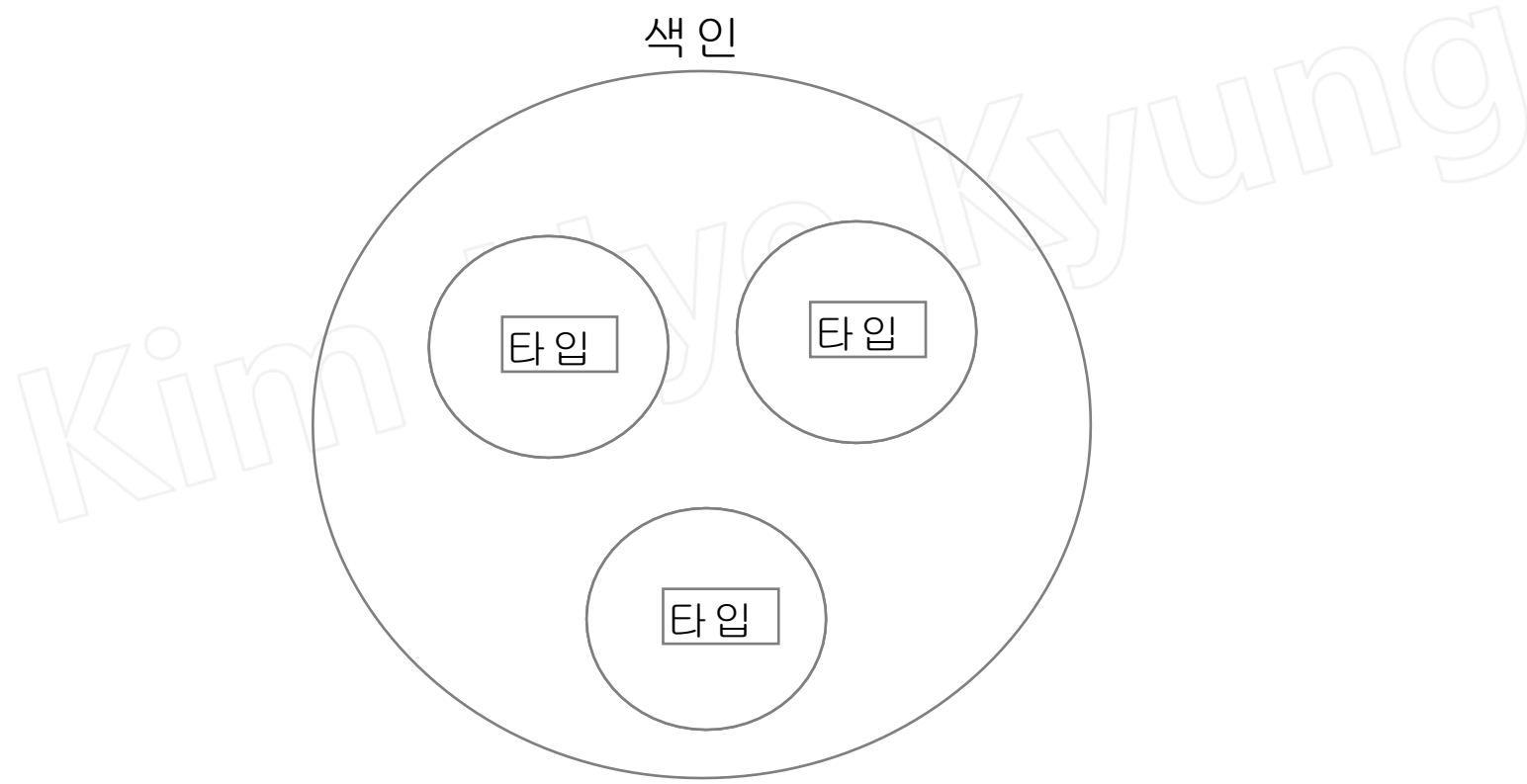
# ElasticSearch 클러스터 상태

- 클러스터 상태
  - ES를 설치하고 클러스터 상태를 보니, yellow 상태
  - yellow 상태는 모든 데이터의 읽기/쓰기가 가능한 상태이지만 일부 replica shard가 아직 배정되지 않은 상태를 의미
  - 즉, 정상 작동중이지만 replica shard가 없기 때문에 검색 성능에 영향이 있을 수 있음

## || 용어 이해하기

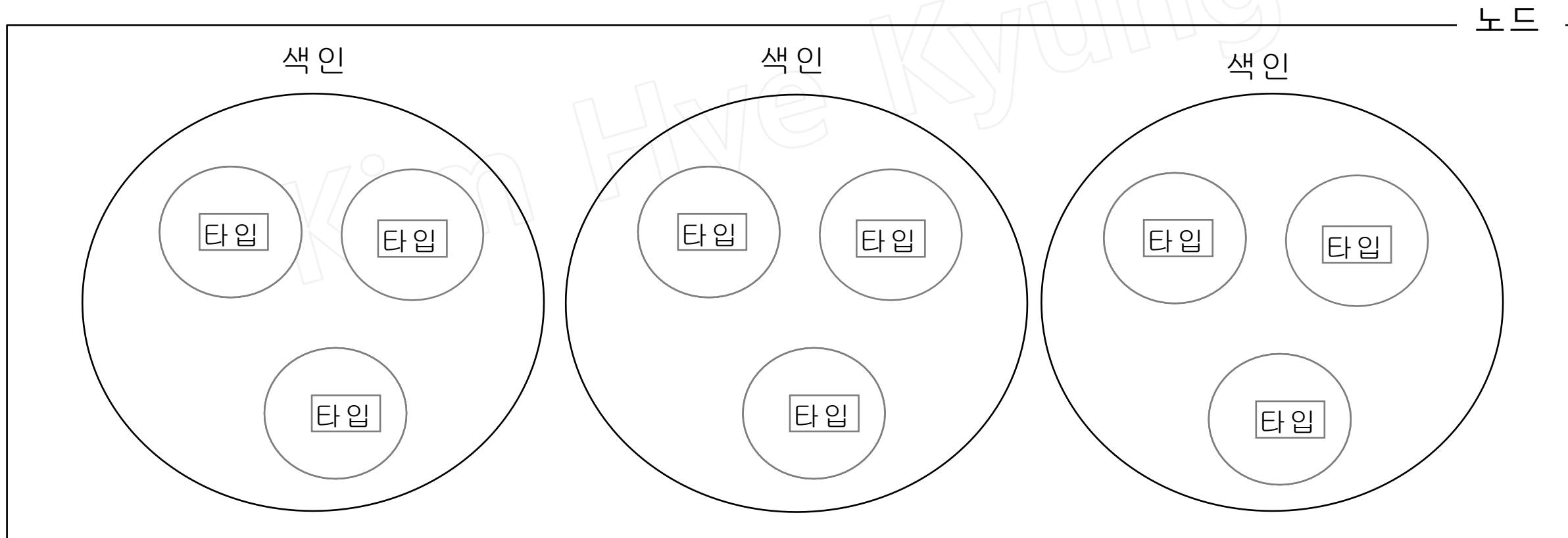
# 색인(index)이란?

- 검색엔진의 논리적 저장 단위



## 노드란?

- 실행중인 Elasticsearch 인스턴스 하나를 지칭
- 통상 서버 1대를 노드로 봄



## 샤드(shard) & 복제(replica)

---

- shard와 replica는 Elasticsearch에만 존재하는 개념은 아님
- 분산 데이터베이스 시스템에도 존재하는 개념

Kim Hye Kyung

# 관계형 데이터베이스와의 차이점 이해하기

## 샤딩( sharding )

- 데이터를 분산해서 여러 그룹으로 쪼개어 이 그룹들을 각기 다른 머신에 저장하는 처리 방법
- 파티셔닝(Partitioning)이라고도 함
  - Elasticsearch에서 Scale Out을 위해 index를 여러 shard로 쪼갠 것
- 기본적으로 1개가 존재하며, 검색 성능 향상을 위해 클러스터의 샤드 갯수를 조정하는 튜닝을 하기도 함

## replica

- 또 다른 형태의 shard
- 노드를 손실했을 경우 데이터의 신뢰성을 위해 샤드들을 복제
- replica는 서로 다른 노드에 존재할 것을 권장

# 샤드와 리플리카(복제본)

분산 검색엔진에서 **shard**란 일종의 파티션과 같은 의미이며, 데이터를 저장할 때 나누어진 하나의 조각에 대한 단위

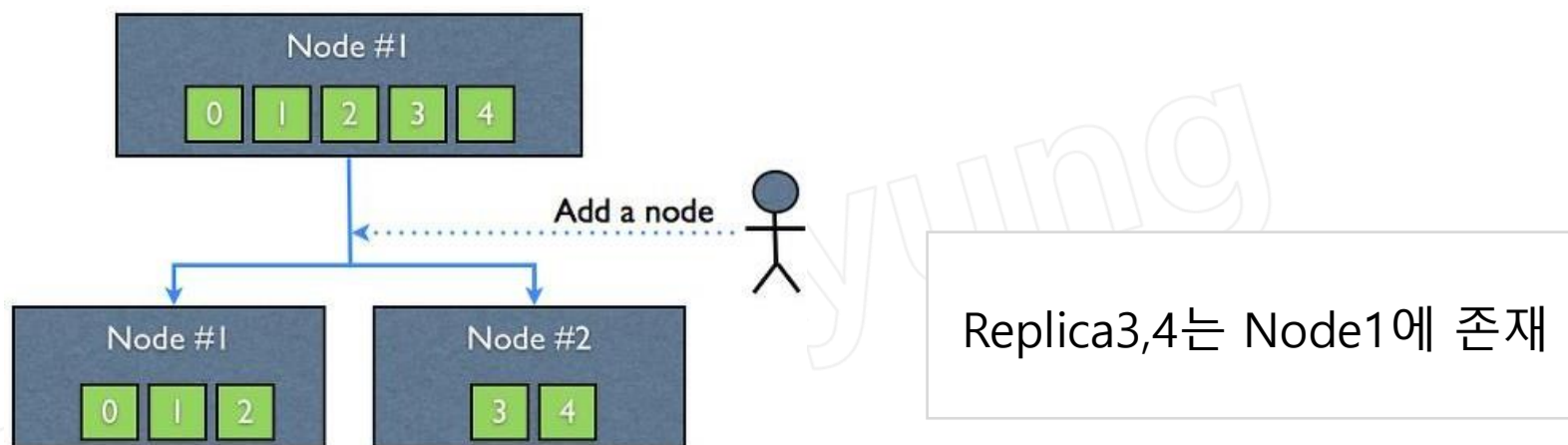
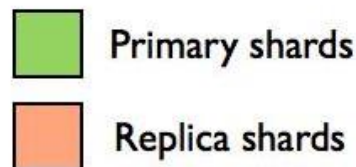
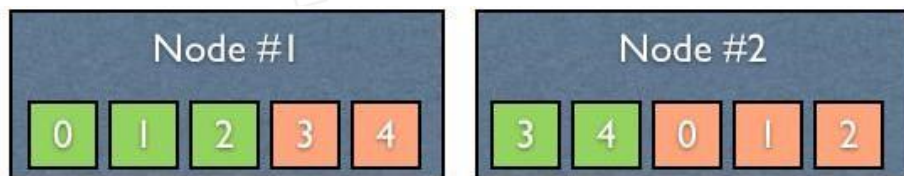


Figure 2. **Shard** relocating



리플리카란 샤드의 복제본

리플리카는 운영중에 그 수를 변경할 수 있으나  
샤드는 수를 조정 할 수 없음

# 샤딩(Sharding)을 적용하기에 앞서

- 샤딩(Sharding)을 적용한다는것은?
  - 프로그래밍, 운영적인 복잡도는 더 높아지는 단점이 있음
- 가능하면 Sharding을 피하거나 지연시킬 수 있는 방법을 찾는 것이 우선되어야 함
  - Scale-in
    - Hardware Spec이 더 좋은 컴퓨터를 사용
  - Read 부하가 크다면?
    - Cache나 Database의 Replication을 적용하는 것도 하나의 방법
  - Table의 일부 컬럼만 자주 사용한다면?
    - Vertically Partition도 하나의 방법
    - Data를 Hot, Warm, Cold Data로 분리하는 것



## ElasticSearch status

# ElasticSearch status

- Status란?
  - 현재 클러스터의 상태 표시
  - cat API를 사용하여 손쉽게 클러스터 상태 확인 가능
  - green & yellow
    - 모든 index의 읽기, 쓰기에는 이상이 없는 상태

## cat health

health is a terse, one-line representation of the same information from `/_cluster/health`.

```
GET /_cat/health?v
```

[Copy as cURL](#) [View in Console](#)

epoch	timestamp	cluster	status	node.total	node.data	shards	pri	relo
1475871424	16:17:04	elasticsearch	green	1	1	1	1	0

It has one option `ts` to disable the timestamping:

```
GET /_cat/health?v&ts=false
```

[Copy as cURL](#) [View in Console](#)

which looks like:

cluster	status	node.total	node.data	shards	pri	relo	init	unassign	pending_t
elasticsearch	green	1	1	1	1	0	0	0	

<https://www.elastic.co/guide/en/elasticsearch/reference/7.1/cat-health.html>

# ElasticSearch status

- status 값 3가지

값	설 명
green	모든 shard가 정상적으로 동작하는 상태 모든 인덱스에 쓰기, 읽기가 정상적으로 동작
yellow	일부 혹은 모든 인덱스의 replicas 샤드가 정상적으로 동작하고 있지 않은 상태, 모든 인덱스에 쓰기/읽기가 정상적으로 동작하지만, 일부 인덱스의 경우 replicas가 없어서 primary 샤드에 문제가 생기면 데이터 유실이 발생할 가능성이 있음
red	일부 혹은 모든 인덱스의 primary와 replicas 샤드가 정상적으로 동작하고 있지 않은 상태, 일부 혹은 모든 인덱스에 쓰기, 읽기가 정상적으로 동작하지 않으며, 데이터의 유실이 발생 할 가능성이 있음

# | Elasticsearch 환경 구축

# ElasticSearch 실습 환경 구축

- 자바 언어로 개발된 프로그램

- 자바 실행 환경 필요

- 설치 방법

- 다운로드

- 압축해제

- 실행



elasticsearch



elasticsearch.bat

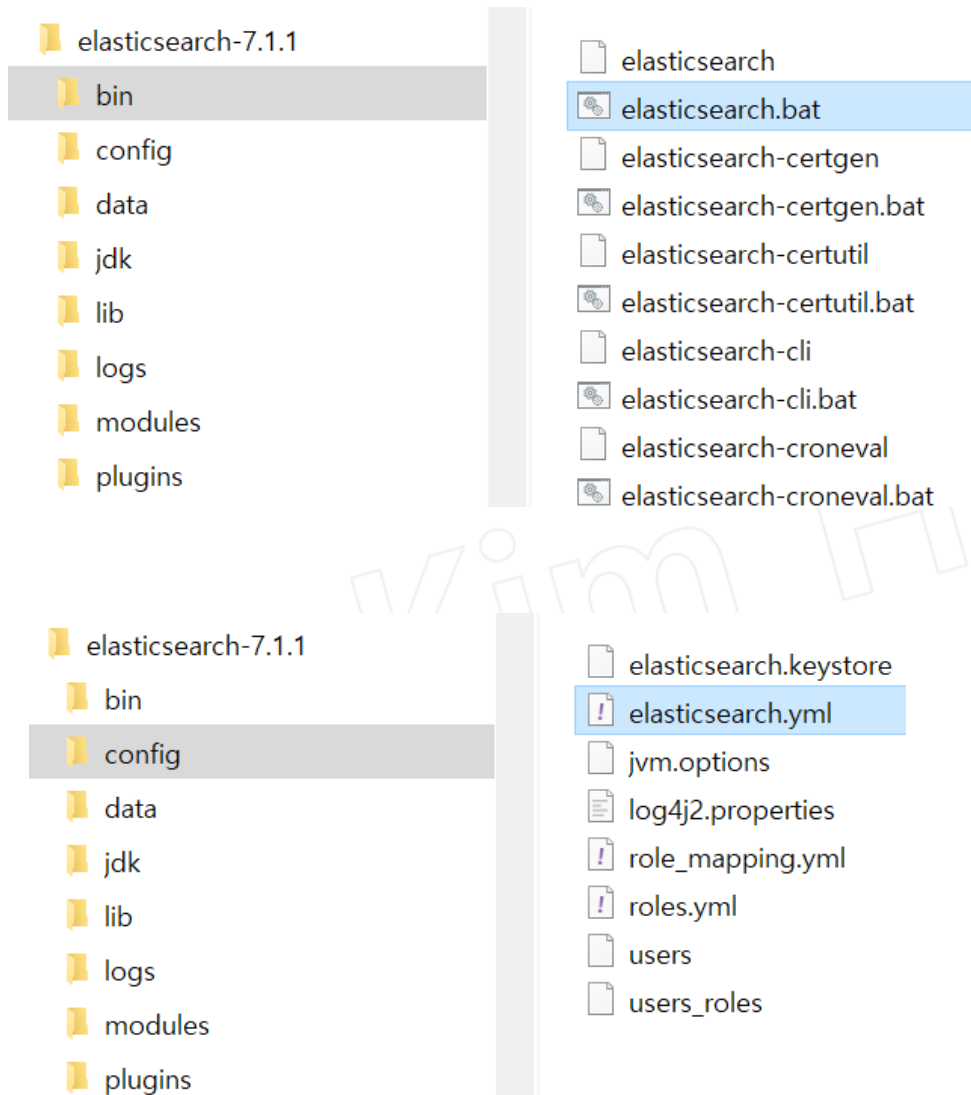


elasticsearch-certgen

The screenshot shows the Elastic.co website with a blue header. The main content area has a large blue banner with the text "다운로드 할 준비가 모두 되었나요? 이제 시작합니다!" (Are you ready to download? Now start!). Below the banner, there's a link to <https://www.elastic.co/kr/downloads/>. The page features six cards for different Elastic products: Elasticsearch, Kibana, Beats, Logstash, APM, and App Search. Each card includes the product logo, name, a brief description, and buttons for "클라우드로 시작" (Start with cloud) and "다운로드" (Download).

Product	Description	Buttons
Elasticsearch	RESTful 검색 및 분석 분산 시스템.	클라우드로 시작, 다운로드
Kibana	데이터의 시각화. 전체 스택 탐색.	클라우드로 시작, 다운로드
Beats	데이터의 가벼운 수집, 파싱 그리고 전송.	다운로드
Logstash	데이터의 수집, 변환, 확장 및 출력.	다운로드
APM	Application에서 성능의 병목 지점 탐지.	클라우드로 시작, 다운로드
앱 검색	직관적인 인터페이스로 여러분의 앱에 강력한 검색 기능을 손쉽게 추가가 가능합니다	클라우드로 시작, 다운로드

# ElasticSearch 설치 디렉토리



bin : 실행 파일 디렉토리

elasticsearch.bat : 실행 파일

elasticsearch.yml : 주 설정 파일

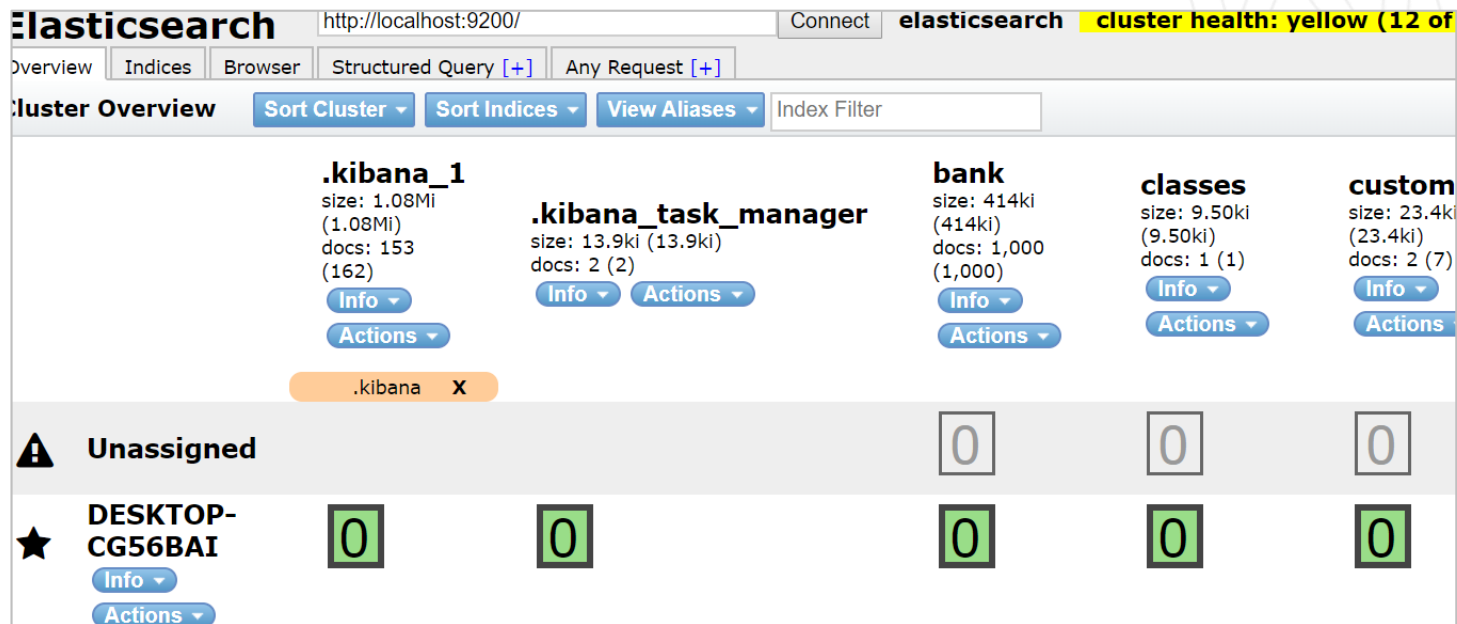
jvm.options : java 설정 파일

# ElasticSearch plug in

- Head

# ElasticSearch Head

- <http://mobz.github.io/elasticsearch-head/>
- head plugin 을 이용해서 cluster 상태, index 정보, 간단한 쿼리 수행 등의 기능을 편리하게 사용할 수 있다.



bank의 Actions 버튼에서 삭제 가능  
post맨 통해서 push하면 다시 생성됨



# ElasticSearch Head

- Browser tab을 이용한 bank index 상세 보기

**Elasticsearch**   **elasticsearch** **cluster health: yellow (15 of 25)**

**Browser**

All Indices

**INDICES**

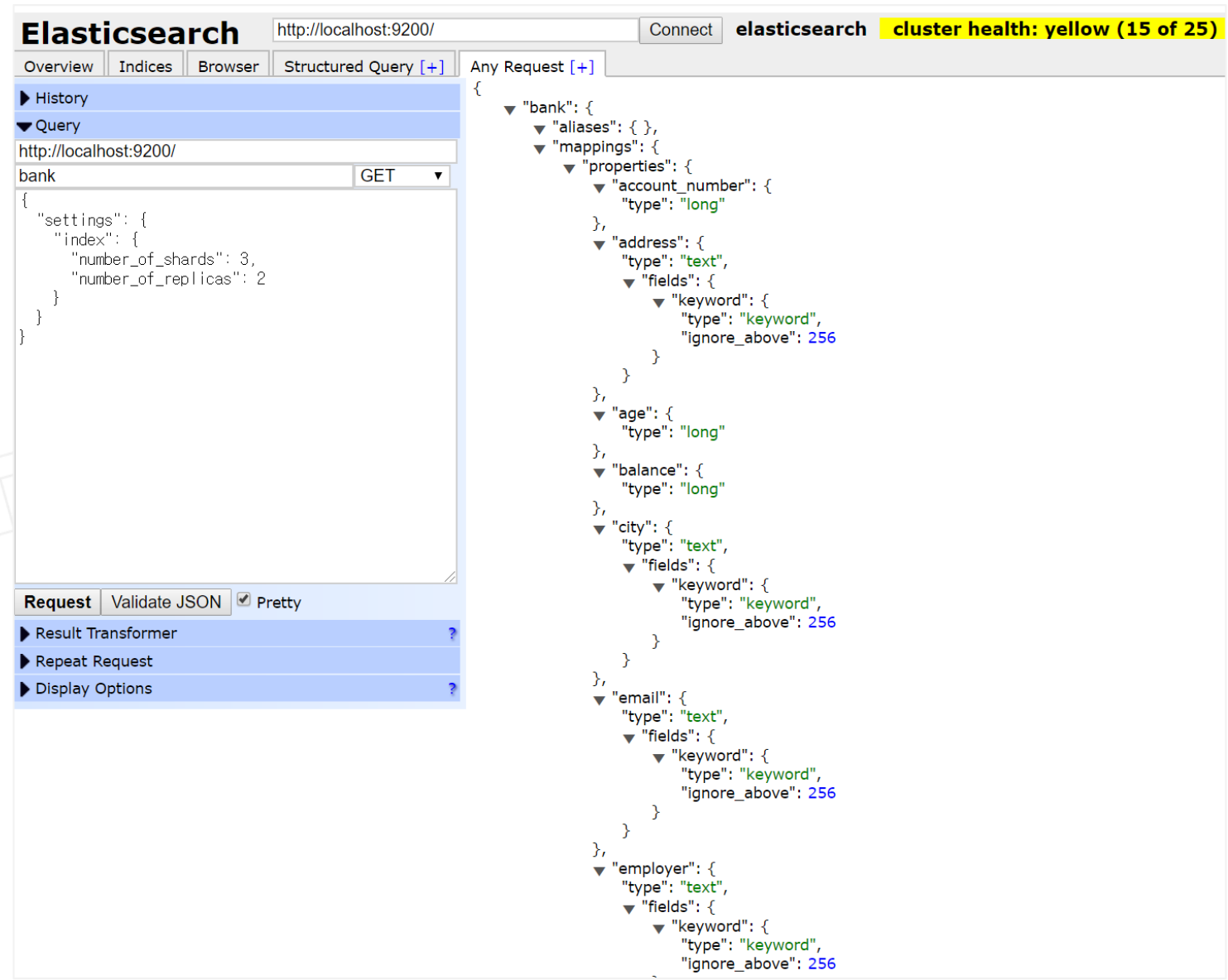
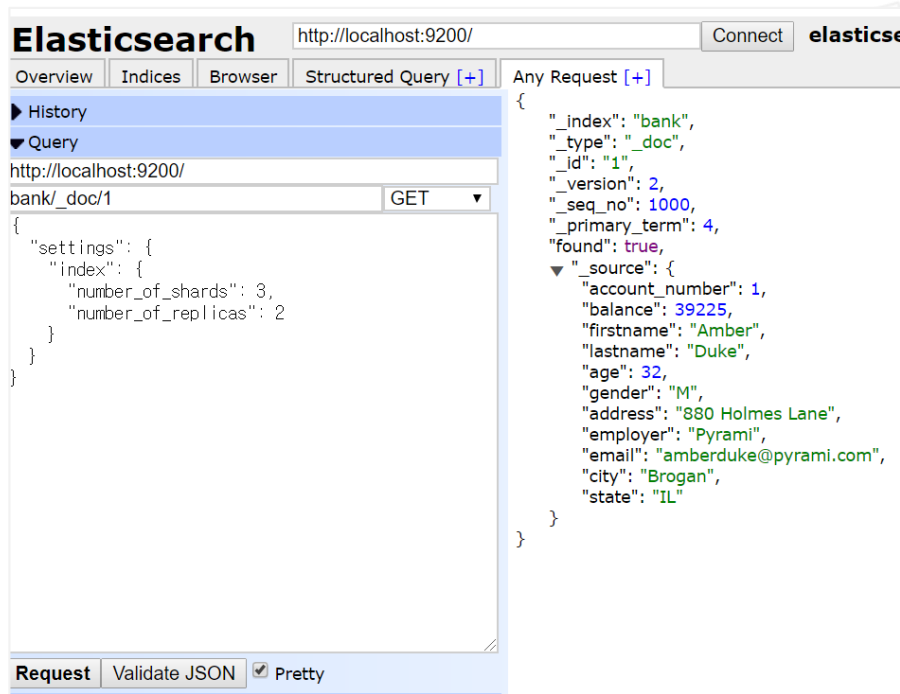
- .kibana\_1
- .kibana\_task\_manager
- bank**
- chapter
- classes
- emails
- kibana\_sample\_data\_ecommerce
- kibana\_sample\_data\_flights
- kibana\_sample\_data\_logs
- logstash-2015.05.18
- logstash-2015.05.19
- logstash-2015.05.20
- shakespeare

Searched 1 of 1 shards. 1000 hits. 0.002 seconds

_index	_type	_id	_score	account_number	balance	firstname	lastname	age	gender	address	employer	email
bank	_doc	1	1	1	39225	Amber	Duke	32	M	880 Holmes Lane	Pyrami	amberduke@pyrami.com
bank	_doc	6	1	6	5686	Hattie	Bond	36	M	671 Bristol Street	Netagy	hattiebond@netagy.com
bank	_doc	13	1	13	32838	Nanette	Bates	28	F	789 Madison Street	Quility	nanettebates@quility.com
bank	_doc	18	1	18	4180	Dale	Adams	33	M	467 Hutchinson Court	Boink	daleadams@boink.com
bank	_doc	20	1	20	16418	Elinor	Ratliff	36	M	282 Kings Place	Scentric	elinorratliff@scentric.com
bank	_doc	25	1	25	40540	Virginia	Ayala	39	F	171 Putnam Avenue	Filodyne	virginiaayala@filodyne.com
bank	_doc	32	1	32	48086	Dillard	Mcpherson	34	F	702 Quentin Street	Quailcom	dillardmcpherson@quailcom.com
bank	_doc	37	1	37	18612	Mcgee	Mooney	39	M	826 Fillmore Place	Reversus	mcgeemooney@reversus.com
bank	_doc	44	1	44	34487	Aurelia	Harding	37	M	502 Baycliff Terrace	Orbalix	aureliaharding@orbalix.com
bank	_doc	49	1	49	29104	Fulton	Holt	23	F	451 Humboldt Street	Anocha	fultonholt@anocha.com
bank	_doc	51	1	51	14097	Burton	Meyers	31	F	334 River Street	Bezal	burtonmeyers@bezal.com
bank	_doc	56	1	56	14992	Josie	Nelson	32	M	857 Tabor Court	Emtrac	josienelson@emtrac.com
bank	_doc	63	1	63	6077	Hughes	Owens	30	F	510 Sedgwick Street	Valpreal	hughesowens@valpreal.com

# ElasticSearch Head

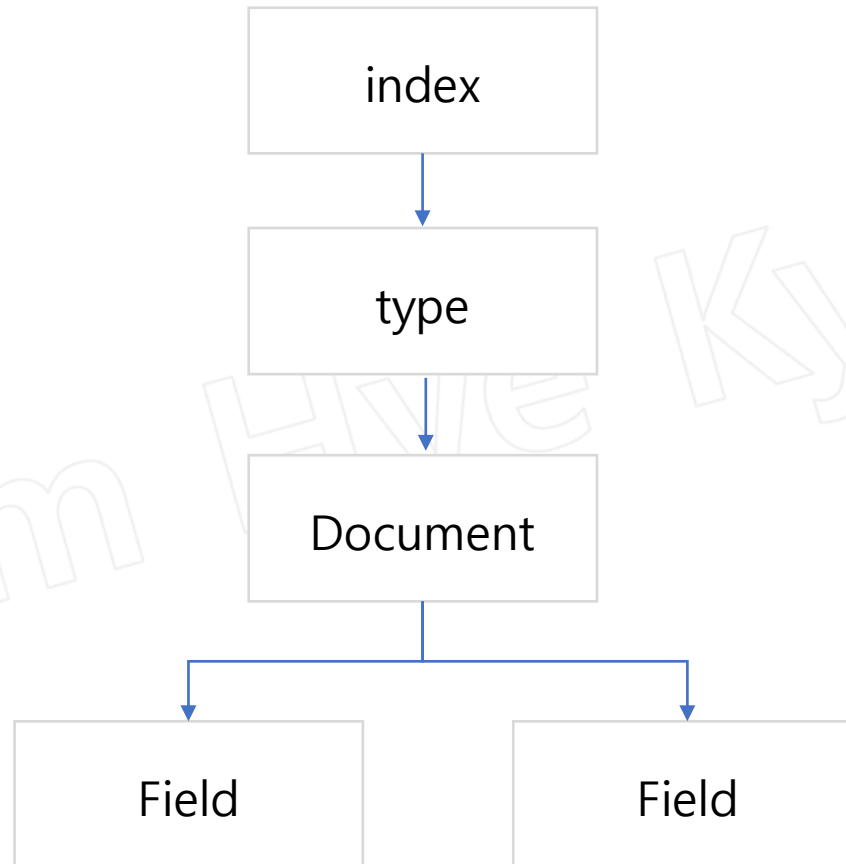
- Any Request(+) tab을 이용한  
bank index 정보 보기



# | Elasticsearch API

# Elasticsearch 구성

- 구성



# Elasticsearch 구성

---

- Cluster – node의 집합
- Node – shard로 구성
- data – shard로 분산되어 저장
- 문서를 색인화 한다는 의미
  - Rest API를 통해 index에 document를 추가



## 잠깐 쉬어가기 : 용어

no	용 어	의 미
1	index	색인 데이터
2	indexing	색인하는 과정
3	indices	매핑 정보를 저장하는 논리적인 데이터 공간

Kim Hye Kyung

# Elasticsearch 주요 API

---

- RESTful 방식의 API를 제공
- JSON 기반으로 통신
- 주요 API
  1. 인덱스 관리 API – Indices API
  2. 문서 관리 API : 문서의 추가/수정/삭제 – Document API
  3. 검색 API : 문서 조회 – Search API
  4. 집계 API : 문서 통계 – Aggregation API

# Elasticsearch 주요 API

---

## 1. 인덱스 관리 API – Indices API

### 1. 인덱스를 관리하기 위한 API

#### 1. 인덱스 생성

1. 매핑이라는 세부 설정 이용
2. 매핑은 문서에 포함된 필드, 필드 타입등을 세세하게 지정하는 것이 가능한 설정 방식
3. 한번 생성된 매핑 정보는 변경 불가

#### 2. 인덱스 삭제



# Elasticsearch 주요 API

## 2. 문서 관리 API : 문서의 추가/수정/삭제 – Document API

1. 실제 문서를 색인하고 조회, 수정, 삭제를 지원하는 API

2. Elasticsearch는 기본적으로 검색엔진 따라서 검색을 위한 다양한 검색 패턴 지원하는 Search API 제공

### 3. 종류

1. Single Document API : 한 건의 문서를 처리하기 위한 기능 제공

2. Multi Document API : cluster 운영시 다수의 문서 처리

### 4. 세부 기능

#### Single Document API

1. Index API : 한 건의 문서를 색인
2. Get API : 한 건의 문서 조회
3. Delete API : 한 건의 문서 삭제
4. Update API : 한 건의 문서 업데이트

#### Multi Document API

1. Multi Get API : 다수의 문서 조회
2. Bulk API : 대량의 문서를 색인
3. Delete By Query API : 다수의 문서를 삭제
4. Update By Query API : 다수의 문서를 업데이트
5. Reindex API : 인덱스의 문서를 다시 색인

# Elasticsearch 주요 API

## 3. 검색 API : 문서 조회 – Search API

### 1. 사용 방식

1. HTTP URI(Uniform Resource Identifier) 형태의 파라미터를 URI에 추가해 검색하는 방법

1. 문서 ID인 \_id 값을 사용해 문서를 조회하는 방식

2. url에 parameter를 붙여 조회하는 방식

```
GET twitter/_search?q=user:kimchy
```

2. RESTful API 방식인 QueryDSL을 사용해 요청 본문(request body)에 질의 내용 <https://www.elastic.co/guide/en/elasticsearch/reference/7.1/search-uri-request.html>

1. 현업에서 더 선호 하는 방식

2. 쿼리의 조건이 복잡하고 길어질 경우에 적합

1. json 방식으로 질의

3. Query DSL (Domain Specific Language)

1. 가독성이 높음

2. JSON 형식으로 다양한 표현이 가능

3. Query의 조건을 여러 개 만들거나,  
통계를 위한 집계(Aggregation) 쿼리 등 복잡한 쿼리 작성에 권장

POST /{index명}/\_search

```
{  
  
  JSON 쿼리 구문  
}
```

```
GET /_search  
{  
  "query": { ①  
    "bool": { ②  
      "must": [  
        { "match": { "title": "Search" }}, ③  
        { "match": { "content": "Elasticsearch" }} ④  
      ],  
      "filter": [ ⑤  
        { "term": { "status": "published" }}, ⑥  
        { "range": { "publish_date": { "gte": "2015-01-01" }}} ⑦  
      ]  
    }  
  }  
}
```

<https://www.elastic.co/guide/en/elasticsearch/reference/7.1/query-filter-context.html>

# Elasticsearch 주요 API

## 4. 집계 API : 문서 통계 – Aggregation API

1. 메모리 기반으로 동작하기 때문에 대용량의 데이터 통계 작업이 가능
2. 각종 통계 데이터를 실시간으로 제공할 수 있는 강력한 기능

**Elasticsearch**   **elasticsearch** **cluster health: yellow (15 of 25)**

Overview

Indices

Browser

Structured Query [\[+\]](#)

Any Request [\[+\]](#)

**Browser**

All Indices ▼  
**INDICES**  
.kibana\_1  
.kibana\_task\_manager  
bank  
chapter  
classes  
emails  
kibana\_sample\_data\_ecommerce  
kibana\_sample\_data\_flights  
kibana\_sample\_data\_logs

Searched 1 of 1 shards. 1000 hits. 0.004 seconds  

_index	_type	_id	_score	account_number ▼	balance	firstname	lastname	age
bank	_doc	999	999		6087	Dorothy	Barron	22
bank	_doc	998	998		16869	Letha	Baker	40
bank	_doc	997	997		25311	Combs	Frederick	20
bank	_doc	996	996		17541	Andrews	Herrera	30
bank	_doc	995	995		21153	Phelps	Parrish	25
bank	_doc	994	994		33298	Madge	Holcomb	31
bank	_doc	993	993		26487	Campos	Olsen	37
bank	_doc	992	992		11413	Kristie	Kennedy	33
bank	_doc	991	991		4239	Connie	Berry	28

# Elasticsearch 주요 API

## 4. 집계 API : 문서 통계 – Aggregation API

예시 : age field의 평균 구하기

The screenshot shows the Elasticsearch Kibana interface. At the top, the URL is `http://localhost:9200/` and the cluster health is **yellow (15 of 25)**. The 'Indices' tab is selected, and the 'bank' index is chosen from the 'All Indices' list.

The search results table shows 1000 hits. The columns are `_index`, `_type`, `_id`, `_score`, and `account_number`. The results are sorted by `account_number` in descending order.

The aggregation results are shown in the right sidebar. The query is a `GET /bank/_search?size=0` with an aggregation named `avg_grade` of type `avg` on the `age` field. The result shows a total of 1000 hits and an average value of 30.171.

```
1 GET /bank/_search?size=0
2 {
3   "aggs" : {
4     "avg_grade" : { "avg" : { "field" : "age" } }
5   }
6 }
```

_index	_type	_id	_score	account_number	age
bank	_doc	999	999	999	22
bank	_doc	998	998	998	40
bank	_doc	997	997	997	20
bank	_doc	996	996	996	30
bank	_doc	995	995	995	25
bank	_doc	994	994	994	31
bank	_doc	993	993	993	37
bank	_doc	992	992	992	33
bank	_doc	991	991	991	28

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1000,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "avg_grade" : {
      "value" : 30.171
    }
  }
}
```

# | Elasticsearch API 실습

## 인덱스 관리 API – Indices API

# Response 분석 - 예시

```
GET /twitter/_search
{
  "query" : {
    "term" : { "user" : "kimchy" }
  }
}
```

```
1 {
2   "took" : 3,
3   "timed_out" : false,
4   "_shards" : {
5     "total" : 1,
6     "successful" : 1,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : {
12      "value" : 4,
13      "relation" : "eq"
14    },
15    "max_score" : 0.105360515,
16    "hits" : [
17      {
18        "_index" : "twitter",
19        "_type" : "_doc",
20        "_id" : "NyzsAG0BrvdLXMEXgSPp",
21        "_score" : 0.105360515,
22        "_routing" : "kimchy",
23        "_source" : {
24          "user" : "kimchy",
25          "post_date" : "2009-11-15T14:12:12",
26          "message" : "trying out Elasticsearch"
27        }
28      }
29    ]
30  }
31 }
```

- took : 검색 소요 시간(단위 : ms)
- timed\_out : 검색시 시간 초과 여부
- \_shards : 검색한 shard 수 및 검색에 성공 또는 실패한 shard 수
- hits : 검색 결과
  - total : 검색 조건과 일치하는 문서의 총 개수
  - max\_score : 검색 조건과 거로가 일치 수준의 최댓값
  - hits : 검색 결과에 해당하는 실제 데이터들 ( 기본 값으로 10개가 설정되며, size를 통해 조절 가능 )
    - \_score :
      - 해당 document가 지정된 검색 쿼리와 얼마나 일치하는지를 상대적으로 나타내는 숫자 값
      - 높을수록 관련성이 높음

## 실습 단계

---

모든 index 조회

index 추가

document 추가

document 조회

document 수정 & 조회

document 삭제 & 조회

index 삭제 & 조회

## 실습 단계

모든 index 조회	curl -XGET "localhost:9200/_cat/indices?v"	주의사항 : window에선 " "표기
index 추가	curl -XPUT 'localhost:9200/user?pretty'	
document 추가	curl -H "Content-Type:application/json" -XPUT "localhost:9200/user/_doc/1?pretty" -d "{₩"name₩":₩"khk₩", ₩"age₩":20}"	주의사항 : window에선 ₩ 사용
document 조회	curl -XGET "localhost:9200/user/_doc/1?pretty"	
document 수정 & 조회	curl -H "Content-Type:application/json" -XPOST "localhost:9200/user/_doc/1?pretty" -d "{₩"name₩":₩"kim₩", ₩"age₩":20}"	
document 삭제 & 조회	curl -XDELETE "localhost:9200/user/_doc/1?pretty"	
index 삭제 & 조회	curl -XDELETE "localhost:9200/user"	



# 모든 index 조회

- # 모든 index 조회
- curl -XGET "localhost:9200/\_cat/indices?v"

```
C:\Users\Kimhyekyung>curl -XGET "localhost:9200/_cat/indices?v"
```

health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
green	open	kibana_sample_data_logs	liapGVrwQyyfQnTbNHehNg	1	0	14005	0	11.4mb	11.4mb
yellow	open	twitter	3_dLqxd0Ro6qdICvdfu9Qg	1	1	4	0	10.3kb	10.3kb
yellow	open	customer	Jce_xn_HQAqufwc-lJiinA	1	1	2	3	9.9kb	9.9kb
yellow	open	logstash-2015.05.19	T6ZfOLwBSe25b7lxAtbWjw	1	1	4624	0	14.3mb	14.3mb
yellow	open	test	pCOaGiKHT6GYy7s7AXvjA	1	1	0	0	283b	283b
yellow	open	chapter	ikPkajCdSfWYKompFOAtHw	1	1	2	0	8.7kb	8.7kb
yellow	open	school	4gBGtU29TP2_r19tS_1qsQ	1	1	2	0	9.1kb	9.1kb
green	open	.kibana_1	RfamjdOyTNmrbHvDXMx3hg	1	0	153	10	1mb	1mb
green	open	.kibana_task_manager	tTJpvOnBSliXG9l8vw866g	1	0	2	0	13.8kb	13.8kb
yellow	open	shakespeare	ludEUGWHS_WiKYTHKW3eIQ	1	1	111396	0	19.5mb	19.5mb
yellow	open	classes	KoITTGFnTGSU8vtNDj1aQ	1	1	1	0	9.4kb	9.4kb
yellow	open	user	8C9wGo68ToiFziUxEnBe2w	1	1	1	0	3.7kb	3.7kb
yellow	open	logstash-2015.05.18	ySHtI3MKTHG6YAE8oehqfw	1	1	4631	0	14.2mb	14.2mb
yellow	open	logstash-2015.05.20	w_vXyFIVT3yIDMuYvo7jFQ	1	1	4750	0	14.6mb	14.6mb
green	open	kibana_sample_data_ecommerce	zO_c8S3YSjm1rRLjBtu12A	1	0	4675	0	4.8mb	4.8mb
green	open	kibana_sample_data_flights	WnWSM9e1QtS1RlvNEIkLvw	1	0	13059	0	6.4mb	6.4mb
yellow	open	bank	c2k9zi6bQPWlfp5M59mLJQ	1	1	1000	0	414.3kb	414.3kb
yellow	open	emails	lQUdwGSNRWqc4qHd6mQeJg	1	1	3	0	3.7kb	3.7kb

## document 추가(insert)

- # user에 document 추가
- `curl -H "Content-Type:application/json" -XPUT "localhost:9200/user/_doc/1?pretty" -d '{"name":"khk", "age":20}'`

```
C:\Users\Kimhyekyung>curl -H "Content-Type:application/json" -XPUT "localhost:9200/user/_doc/1?pretty"
-d '{"name":"khk", "age":20}'
{
  "_index" : "user",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

## document 조회

- # user index의 id가 1인 document 조회
- `curl -XGET "localhost:9200/user/_doc/1?pretty"`

```
C:\Users\Kimhyekyung>curl -XGET "localhost:9200/user/_doc/1?pretty"
{
  "_index" : "user",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "name" : "khk",
    "age" : 20
  }
}
```

## 특정 index의 모든 document 조회

- # user index의 모든 document 조회
- `curl -XGET "localhost:9200/user/_search?pretty"`

```
C:\Users\Kimhyekyung>curl -XGET "localhost:9200/user/_search?pretty"
{
  "took" : 29,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "user",
        "_type" : "_doc",
        "_id" : "1",
        "_score" : 1.0,
        "_source" : {
          "name" : "khk",
          "age" : 20
        }
      }
    ]
  }
}
```

## document의 source만 조회

- # user index의 id가 1인 document의 source만 조회
- `curl -XGET "localhost:9200/user/_doc/1?pretty&filter_path=_source"`

```
C:\Users\Kimhyekyung>curl -XGET "localhost:9200/user/_doc/1?pretty&filter_path=_source"
{
  "_source" : {
    "name" : "khk",
    "age" : 20
  }
}
```

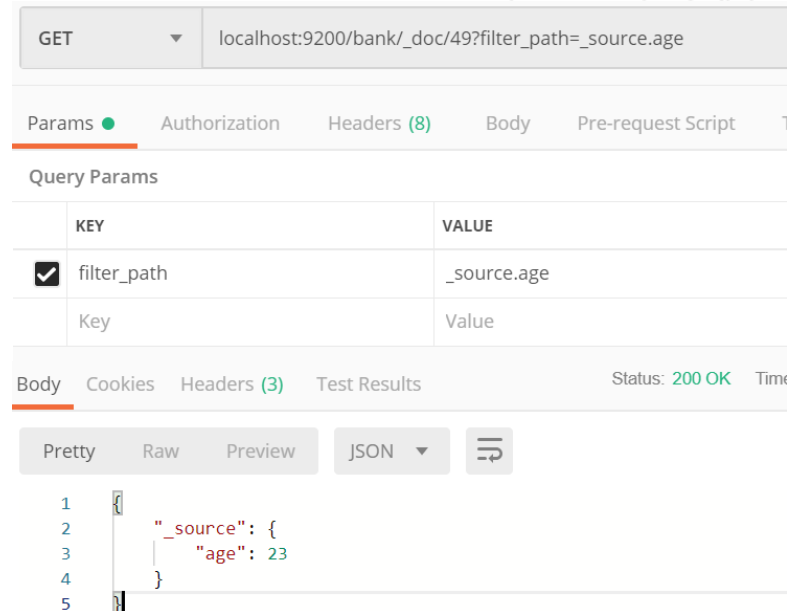
# document의 source의 name field만 조회

- user index의 id가 1인 document의 source의 name field만 조회
- `curl -XGET "localhost:9200/user/_doc/1?pretty&filter_path=_source.name"`

**command  
line**

```
C:\Users\Kimhyekyung>curl -XGET "localhost:9200/user/_doc/1?pretty&filter_path=_source.name"
{
  "_source" : {
    "name" : "khk"
  }
}
```

**post man**



## document 수정 및 조회

- # user의 id가 1인 document의 name field값 수정
- `curl -H "Content-Type:application/json" -XPOST "localhost:9200/user/_doc/1?pretty" -d '{"name":"kim", "age":20}'`

```
C:\Users\Kimhyekyung>curl -H "Content-Type:application/json" -XPOST "localhost:9200/user/_doc/1?pretty"
-d '{"name":"kim", "age":20}'
{
  "_index" : "user",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 2,
  "result" : "updated",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 1,
  "_primary_term" : 1
}
```

```
C:\Users\Kimhyekyung>curl -XGET "localhost:9200/user/_doc/1?pretty"
{
  "_index" : "user",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 2,
  "_seq_no" : 1,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "name" : "kim",
    "age" : 20
  }
}
```

## document의 삭제

- 특정 id로 document 삭제
- `curl -XDELETE "localhost:9200/user/_doc/1?pretty"`

```
C:\Users\Kimhyekyung>curl -XDELETE "localhost:9200/user/_doc/1?pretty"
{
  "_index" : "user",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 3,
  "result" : "deleted",
  "shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 2,
  "_primary_term" : 1
}
```

```
C:\Users\Kimhyekyung>curl -XGET "localhost:9200/user/_doc/1?pretty"
{
  "_index" : "user",
  "_type" : "_doc",
  "_id" : "1",
  "found" : false
}
```



# index 삭제

- # index 삭제
- curl -XDELETE "localhost:9200/user"

```
C:\Users\Kimhyekyung>curl -XDELETE "localhost:9200/user"  
{ "acknowledged": true }
```

```
C:\Users\Kimhyekyung>curl -XDELETE "localhost:9200/user?pretty"  
{  
  "error" : {  
    "root_cause" : [  
      {  
        "type" : "index_not_found_exception",  
        "reason" : "no such index [user]",  
        "resource.type" : "index_or_alias",  
        "resource.id" : "user",  
        "index_uuid" : "_na_",  
        "index" : "user"  
      }  
    ],  
    "type" : "index_not_found_exception",  
    "reason" : "no such index [user]",  
    "resource.type" : "index_or_alias",  
    "resource.id" : "user",  
    "index_uuid" : "_na_",  
    "index" : "user"  
  },  
  "status" : 404  
}
```

# ElasticSearch Tutorial

- Window 기반의 실습 일부

"size" : 0

응답에서 집계 결과만 보고 싶기 때문에 검색  
적중을 표시하지 않도록 `size=0`을 설정했습  
니다

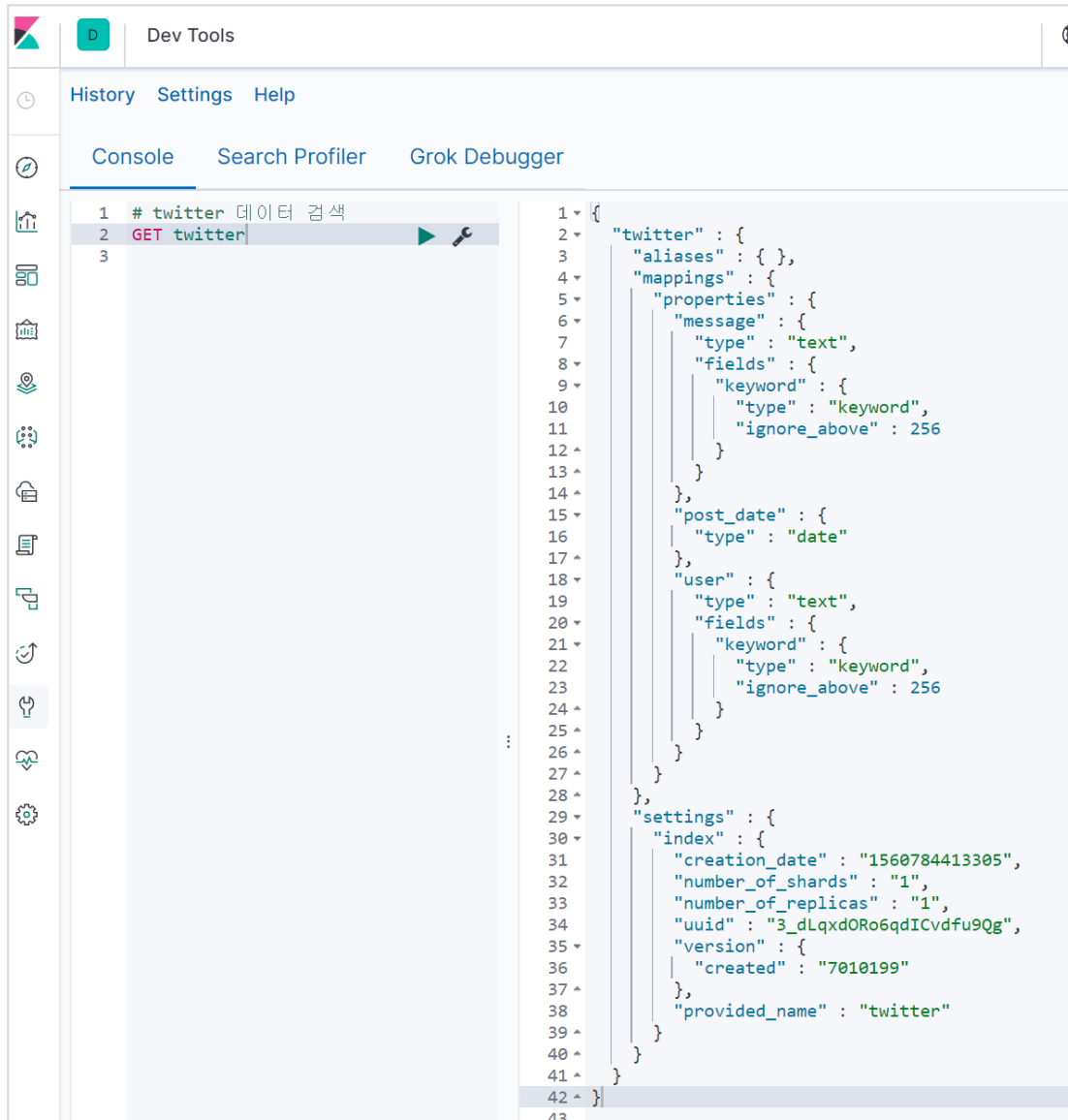
The screenshot shows a REST client interface with the following details:

- URL:** http://localhost:9200/bank/\_search
- Method:** GET
- Body (JSON):**

```
1 {  
2   "size": 0,  
3   "aggs": {  
4     "group_by_state": {  
5       "terms": {  
6         "field": "state.keyword"  
7       }  
8     }  
9   }  
10 }
```
- Status:** 200 OK, Time: 19 ms, Size: 390 B
- Response Body (JSON):**

```
1 {  
2   "took": 7,  
3   "timed_out": false,  
4   "_shards": {  
5     "total": 1,  
6     "successful": 1,  
7     "skipped": 0,  
8     "failed": 0  
9   },  
10  "hits": {  
11    "total": {  
12      "value": 1000,  
13      "relation": "eq"  
14    },  
15    "max_score": null,  
16    "hits": []  
17  },  
18  "aggregations": {  
19    "group_by_state": {  
20      "doc_count_error_upper_bound": 0,  
21      "sum_other_doc_count": 743,  
22      "buckets": [  
23        {  
24          "key": "TX",  
25          "doc_count": 30  
26        },  
27        {  
28          "key": "MD",  
29          "doc_count": 28
```

# Kibana & PostMan을 활용한 검색



The screenshot shows the Kibana Dev Tools console. The left sidebar has a search icon selected. The console shows a GET request to the twitter index. The response is a JSON object representing the index settings.

```
1 # twitter 데이터 검색
2 GET twitter
3
4 {
5   "twitter": {
6     "aliases": {},
7     "mappings": {
8       "properties": {
9         "message": {
10          "type": "text",
11          "fields": {
12            "keyword": {
13              "type": "keyword",
14              "ignore_above": 256
15            }
16          }
17        },
18        "post_date": {
19          "type": "date"
20        },
21        "user": {
22          "type": "text",
23          "fields": {
24            "keyword": {
25              "type": "keyword",
26              "ignore_above": 256
27            }
28          }
29        }
30      }
31    },
32    "settings": {
33      "index": {
34        "creation_date": "1560784413305",
35        "number_of_shards": "1",
36        "number_of_replicas": "1",
37        "uuid": "3_dLqxdORo6qdICvdfu9Qg",
38        "version": {
39          "created": "7010199"
40        },
41        "provided_name": "twitter"
42      }
43    }
44  }
45 }
```



The screenshot shows the Postman interface. The request is a GET to localhost:9200/twitter. The response is a JSON object representing the index settings.

```
1 {
2   "twitter": {
3     "aliases": {},
4     "mappings": {
5       "properties": {
6         "message": {
7           "type": "text",
8           "fields": {
9             "keyword": {
10              "type": "keyword",
11              "ignore_above": 256
12            }
13          }
14        },
15        "post_date": {
16          "type": "date"
17        },
18        "user": {
19          "type": "text",
20          "fields": {
21            "keyword": {
22              "type": "keyword",
23              "ignore_above": 256
24            }
25          }
26        }
27      }
28    },
29    "settings": {
30      "index": {
31        "creation_date": "1560784413305",
32        "number_of_shards": "1",
33        "number_of_replicas": "1",
34        "uuid": "3_dLqxdORo6qdICvdfu9Qg",
35        "version": {
36          "created": "7010199"
37        },
38        "provided_name": "twitter"
39      }
40    }
41  }
42 }
```

```
1 {
2   "twitter": {
3     "aliases": {},
4     "mappings": {
5       "properties": {
6         "message": {
7           "type": "text",
8           "fields": {
9             "keyword": {
10              "type": "keyword",
11              "ignore_above": 256
12            }
13          }
14        },
15        "post_date": {
16          "type": "date"
17        },
18        "user": {
19          "type": "text",
20          "fields": {
21            "keyword": {
22              "type": "keyword",
23              "ignore_above": 256
24            }
25          }
26        }
27      }
28    },
29    "settings": {
30      "index": {
31        "creation_date": "1560784413305",
32        "number_of_shards": "1",
33        "number_of_replicas": "1",
34        "uuid": "3_dLqxdORo6qdICvdfu9Qg",
35        "version": {
36          "created": "7010199"
37        },
38        "provided_name": "twitter"
39      }
40    }
41  }
42 }
```

김혜경 [topickim@naver.com] 107

# Kibana & PostMan을 활용한 검색

- 문서 검색

GET twitter/\_doc/1

데이터가 없을 경우

```
{
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "1",
  "found" : false
}
```

데이터가 있을 경우

```
{
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 15,
  "_primary_term" : 3,
  "found" : true,
  "_source" : {
    "user" : "kimchy",
    "post_date" : "2009-11-15T14:12:12",
    "message" : "trying out Elasticsearch"
  }
}
```

# Kibana & PostMan을 활용한 검색

- 문서 생성

```
PUT twitter/_doc/1
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

```
{
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 15,
  "_primary_term" : 3
}
```

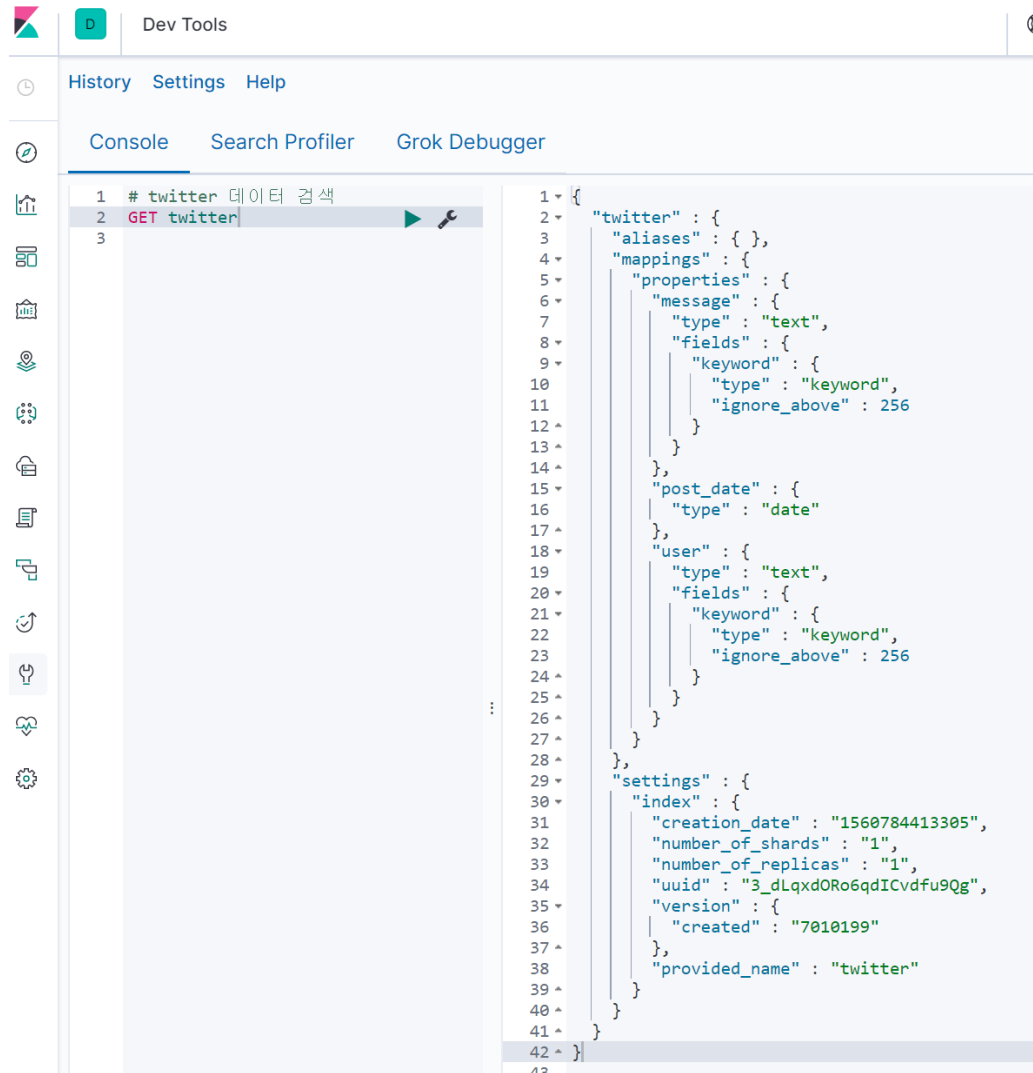
# Kibana & PostMan을 활용한 검색

- 문서 삭제

```
DELETE twitter/_doc/1
```

```
{
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 15,
  "result" : "deleted",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 14,
  "_primary_term" : 3
}
```

# Kiban & PostMan을 활용한 검색



The screenshot shows the Kibana Dev Tools console. The top bar includes 'History', 'Settings', and 'Help'. Below it are 'Console', 'Search Profiler', and 'Grok Debugger'. The console shows a GET request to 'localhost:9200/twitter' and its JSON response. The response is a JSON object with the following structure:

```
1 # twitter 데이터 검색
2 GET twitter
3
4 {
5   "twitter": {
6     "aliases": {},
7     "mappings": {
8       "properties": {
9         "message": {
10          "type": "text",
11          "fields": {
12            "keyword": {
13              "type": "keyword",
14              "ignore_above": 256
15            }
16          }
17        },
18        "post_date": {
19          "type": "date"
20        },
21        "user": {
22          "type": "text",
23          "fields": {
24            "keyword": {
25              "type": "keyword",
26              "ignore_above": 256
27            }
28          }
29        }
30      }
31    },
32    "settings": {
33      "index": {
34        "creation_date": "1560784413305",
35        "number_of_shards": "1",
36        "number_of_replicas": "1",
37        "uuid": "3_dLqxdORo6qdICvdfu9Qg",
38        "version": {
39          "created": "7010199"
40        }
41      }
42    },
43    "provided_name": "twitter"
44  }
45 }
```



The screenshot shows the Postman interface. The top bar includes a 'GET' request to 'localhost:9200/twitter'. The response is a JSON object with the following structure:

```
1 {
2   "twitter": {
3     "aliases": {},
4     "mappings": {
5       "properties": {
6         "message": {
7           "type": "text",
8           "fields": {
9             "keyword": {
10              "type": "keyword",
11              "ignore_above": 256
12            }
13          }
14        },
15        "post_date": {
16          "type": "date"
17        },
18        "user": {
19          "type": "text",
20          "fields": {
21            "keyword": {
22              "type": "keyword",
23              "ignore_above": 256
24            }
25          }
26        }
27      }
28    },
29    "settings": {
30      "index": {
31        "creation_date": "1560784413305",
32        "number_of_shards": "1",
33        "number_of_replicas": "1",
34        "uuid": "3_dLqxdORo6qdICvdfu9Qg",
35        "version": {
36          "created": "7010199"
37        }
38      }
39    },
40    "provided_name": "twitter"
41  }
42 }
```

# | 데이터 모델링



# 데이터 모델링

- ElasticSearc에서의 모델링이란?
  - 색인할 때 문서의 데이터 유형에 따라 필드에 적절한 데이터 타입 지정하는 과정을 매핑이라 함
  - 매핑은 색인될 문서의 데이터 모델링이라고도 함
    - 매핑 방법
      - 매핑 필드 설정 후 적용
      - Elasticsearch가 자동 설정
        - 권장하지 않음
        - 문자와 숫자가 무작위 따라서 색인 불가
    - 인덱스 생성시 항상 명시적인 매핑 설정

## || 맵핑 API

# 데이터 모델링

## 매핑 정보 확인

GET /encore/\_mapping

```
1 {
2   "encore" : {
3     "mappings" : {
4       "properties" : {
5         "id" : {
6           "type" : "text",
7           "fields" : {
8             "keyword" : {
9               "type" : "keyword",
10              "ignore_above" : 256
11            }
12          }
13        }
14      }
15    }
16  }
17 }
```

## 문제점 찾아보기?

```
1 # id는 text 타입
2 PUT /encore/_doc/1
3 {
4   "id" : "25"
5 }
6
7 PUT /encore/_doc/2
8 {
9   "id" : "k23"
10 }
11
12 # id는 숫자 타입
13 PUT /encore/_doc/3
14 {
15   "id" : 3
16 }
17
18 GET /encore/_search
```

```
16 "hits" : [
17   {
18     "_index" : "encore",
19     "_type" : "_doc",
20     "_id" : "1",
21     "_score" : 1.0,
22     "_source" : {
23       "id" : "25"
24     }
25   },
26   {
27     "_index" : "encore",
28     "_type" : "_doc",
29     "_id" : "2",
30     "_score" : 1.0,
31     "_source" : {
32       "id" : "k23"
33     }
34   },
35   {
36     "_index" : "encore",
37     "_type" : "_doc",
38     "_id" : "3",
39     "_score" : 1.0,
40     "_source" : {
41       "id" : 3
42     }
43   }
44 ]
```

## 매핑 정보 설정시 고려사항

---

- 문자열을 분석할 것인가?
- \_source에 어떤 필드를 정의할 것인가?
- 날짜 필드를 가지는 필드는 무엇인가?
- 매핑에 정의되지 않고 유입되는 필드는 어떻게 처리할 것인가?

# Mapping

- 매핑 예시

```
PUT my_index ❶
{
  "mappings": {
    "properties": { ❷
      "title": { "type": "text" }, ❸
      "name": { "type": "text" }, ❹
      "age": { "type": "integer" }, ❺
      "created": {
        "type": "date", ❻
        "format": "strict_date_optional_time||epoch_millis"
      }
    }
  }
}
```

- ❶ Create an index called `my_index`.
- ❷ Specify the fields or *properties* in the mapping.
- ❸ Specify that the `title` field contains `text` values.
- ❹ Specify that the `name` field contains `text` values.
- ❺ Specify that the `age` field contains `integer` values.
- ❻ Specify that the `created` field contains `date` values in two possible formats.

<https://www.elastic.co/guide/en/elasticsearch/reference/7.1/mapping.html>

# Mapping

- kibana를 활용한 index 생성 및 매핑 정보 조회

```
1 PUT my_index
2 {
3   "mappings": {
4     "properties": {
5       "title": {
6         "type": "text"
7       },
8       "name": {
9         "type": "text"
10      },
11      "age": {
12        "type": "integer"
13      },
14      "created": {
15        "type": "date",
16        "format":
17          "strict_date_optional_time||epoch_millis"
18      }
19    }
20 }
```

GET my\_index/\_mapping

```
1 {
2   "my_index" : {
3     "mappings" : {
4       "properties" : {
5         "age" : {
6           "type" : "integer"
7         },
8         "created" : {
9           "type" : "date"
10        },
11        "name" : {
12          "type" : "text"
13        },
14        "title" : {
15          "type" : "text"
16        }
17      }
18    }
19  }
20 }
```

# Mapping Field datatypes

- 공식 문서를 활용한 학습
- [https://www.elastic.co/guide/en/elasticsearch/reference/7.1/mapping.html#\\_field\\_datatypes](https://www.elastic.co/guide/en/elasticsearch/reference/7.1/mapping.html#_field_datatypes)

## Field datatypes

Each field has a data type which can be:

- a simple type like `text`, `keyword`, `date`, `long`, `double`, `boolean` or `ip`.
- a type which supports the hierarchical nature of JSON such as `object` or `nested`.
- or a specialised type like `geo_point`, `geo_shape`, or `completion`.


## Text datatype

A field to index full-text values, such as the body of an email or the description of a product. These fields are *analyzed*, that is they are passed through an *analyzer* to convert the string into a list of individual terms before being indexed. The analysis process allows Elasticsearch to search for individual words *within* each full text field. Text fields are not used for sorting and seldom used for aggregations (although the *significant text aggregation* is a notable exception).

If you need to index structured content such as email addresses, hostnames, status codes, or tags, it is likely that you should rather use a `keyword` field.

Below is an example of a mapping for a text field:

```
PUT my_index
{
  "mappings": {
    "properties": {
      "full_name": {
        "type": "text"
      }
    }
  }
}
```

[Copy as cURL](#) [View in Console](#) 

## 매핑 parameter

no	Parameter	설명
1	analyzer	해당 필드의 데이터를 형태소 분석하겠다는 의미 색인과 검색 시 지정한 분석기로 형태소 분석 수행 text 데이터 타입의 field는 analyzer 매핑 parameter를 기본적으로 수행해야 함
2	normalizer	term query에 분석기를 사용하기 위해 사용
3	coerce	색인 시 자동 변환 허용 여부를 결정 "10" 과 같은 숫자 형태의 문자열이 integer 타입의 field에 적용시 Elasticsearch는 자동으로 형변환
4	doc_values	ElasticSearch에서 사용하는 기본 캐시 text 타입을 제외한 모든 타입에서 기본적으로 doc_values 사용
5	format	날짜/시간을 문자열로 표시
...		



# 매핑 parameter

- "coreco" : false 설정인 경우

1

```
1 DELETE my_index
2
3 PUT my_index
4 {
5   "mappings": {
6     "properties": {
7       "number_one": {
8         "type": "integer"
9       },
10      "number_two": {
11        "type": "integer",
12        "coerce": false
13      }
14    }
15  }
16 }
17
18 PUT my_index/_doc/1
19 {
20   "number_one": "10"
21 }
22
23 PUT my_index/_doc/2
24 {
25   "number_two": "10"
26 }
```

정상 실행

2

```
1 {
2   "error": {
3     "root_cause": [
4       {
5         "type": "mapper_parsing_exception",
6         "reason": "failed to parse field [number_two] of ty
7       }
8     ],
9     "type": "mapper_parsing_exception",
10    "reason": "failed to parse field [number_two] of type [
11    "caused_by": {
12      "type": "illegal_argument_exception",
13      "reason": "Integer value passed as String"
14    }
15  },
16  "status": 400
```

error 발생

3

```
1 GET my_index/_search
2 {
3   "took" : 1,
4   "timed_out" : false,
5   "_shards" : {
6     "total" : 1,
7     "successful" : 1,
8     "skipped" : 0,
9     "failed" : 0
10  },
11  "hits" : {
12    "total" : {
13      "value" : 1,
14      "relation" : "eq"
15    },
16    "max_score" : 1.0,
17    "hits" : [
18      {
19        "_index" : "my_index",
20        "_type" : "_doc",
21        "_id" : "1",
22        "_score" : 1.0,
23        "_source" : {
24          "number_one" : "10"
25        }
26      }
27    ]
28  }
```

모두 조회

## || 메타 필드

# 메타 필드

- Elasticsearch에서 생성한 문서에서 제공하는 특별한 field
- 메타 데이터를 저장하는 특수 목적의 field
- 검색시 다양한 형태로 제어할 수 있도록 이용 가능

```
GET bank/_doc/1
```

```
1 {  
2   "_index" : "bank",  
3   "_type" : "_doc",  
4   "_id" : "1",  
5   "_version" : 2,  
6   "_seq_no" : 1000,  
7   "_primary_term" : 4,  
8   "found" : true,  
9   "_source" : {  
10    "account_number" : 1,  
11    "balance" : 39225,  
12    "firstname" : "Amber",  
13    "lastname" : "Duke",  
14    "age" : 32,  
15    "gender" : "M",  
16    "address" : "880 Holmes Lane",  
17    "employer" : "Pyrami",  
18    "email" : "amberduke@pyrami.com",  
19    "city" : "Brogan",  
20    "state" : "IL"  
21  }  
22 }  
23
```

1. \_index 메타 field : 해당 문서가 속한 index의 이름 보유
2. \_type 메타 field : 해당 문서가 속한 타입 정보 보유
3. \_id 메타 field : 문서를 식별하는 유일한 key
4. \_source 메타 field : 문서의 원본 데이터 제공

## || field 데이터 타입

# field 데이터 타입

---

- 데이터 타입 종류
  - 문자열 : keyword, text
  - 일반 타입 : data, long, double, integer, Boolean, ip
  - JSON 특성의 데이터 타입 : 객체 또는 중첩문
  - 특수 데이터 타입 : geo\_point, geo\_shape

# keyword 데이터 타입

- 키워드 형태로 사용할 데이터에 적합한 데이터 타입
- 별도의 분석기를 거치지 않고 원문 그대로 색인하기 때문에 특정 코드나 키워드 등 정형화된 콘텐츠에 주로 사용

- 다음 항목에 많이 사용

- 검색시 필터링 되는 항목
- 정렬이 필요한 항목
- 집계(aggregation) 해야 하는 항목

- 예시

- 'elastic search'라는 문자열이 keyword 타입으로 되어 있을 경우
- 검색시 : elastic search로만 가능, elastic 또는 search로는 검색 불가

```
1 # keyword 데이터 타입
```

```
2 DELETE my_index
```

```
3 PUT my_index
```

```
4 {
5   "mappings": {
6     "properties": {
7       "tags": {
8         "type": "keyword"
9       }
10    }
11  }
12 }
```

```
13 GET my_index/_mapping
```

```
1 {
2   "my_index" : {
3     "mappings" : {
4       "properties" : {
5         "tags" : {
6           "type" : "keyword"
7         }
8       }
9     }
10  }
11 }
```

## Text 데이터 타입

---

- 색인 시 지정된 분석기가 컬럼의 데이터를 문자열 데이터로 인식 및 분석
- 별도의 분석기를 정의하지 않을 경우 Standard Analyzer 사용
- 영화의 제목이나 영화의 설명글과 같이 문장 형태의 데이터에 사용하기 적합한 데이터 타입
- 특징
  - 전문 검색이 가능
    - 전체 text가 토큰화되어 생성되며 특정 단어를 검색하는 것이 가능

# Array 데이터 타입

- 데이터 매핑 구조
  - 1차원 – 하나의 field에 하나의 값이 매핑
  - 2차원 – 하나의 field에 여러 개의 값이 매핑
- 적용 방식
  - 문자열, 숫자 처럼 일반적인 값을 지정할 수도 있지만 객체 형태로도 정의 가능
  - 주의 사항
    - Array 타입에 저장되는 값 모두 같은 타입으로만 구성해야 함

- an array of strings: [ "one", "two" ]
- an array of integers: [ 1, 2 ]
- an array of arrays: [ 1, [ 2, 3 ] ] which is the equivalent of [ 1, 2, 3 ]
- an array of objects: [ { "name": "Mary", "age": 12 }, { "name": "John", "age": 10 } ]

<https://www.elastic.co/guide/en/elasticsearch/reference/7.1/array.html>



# Numeric 데이터 타입

- 숫자 데이터 타입
- 데이터 크기에 알맞은 타입을 제공함으로써 색인과 검색을 효율적으로 처리하기 위함

타입	설명
long	A signed 64-bit integer with a minimum value of $-2^{63}$ and a maximum value of $2^{63}-1$ .
integer	A signed 32-bit integer with a minimum value of $-2^{31}$ and a maximum value of $2^{31}-1$ .
short	A signed 16-bit integer with a minimum value of -32,768 and a maximum value of 32,767.
byte	A signed 8-bit integer with a minimum value of -128 and a maximum value of 127.
double	A double-precision 64-bit IEEE 754 floating point number, restricted to finite values.
float	A single-precision 32-bit IEEE 754 floating point number, restricted to finite values.
half_float	A half-precision 16-bit IEEE 754 floating point number, restricted to finite values.
scaled_float	A floating point number that is backed by a long, scaled by a fixed double scaling factor.

<https://www.elastic.co/guide/en/elasticsearch/reference/7.1/number.html>

# Date 데이터 타입

- JSON 포맷에서 문자열로 처리
- 날짜는 다양하게 표현 가능
  - 구문 분석될 수 있도록 날짜 문자열 형식을 명시적으로 설정 해야 함
  - 별도의 형식으로 지정하지 않을 경우 "yyyy-MM-ddTHH:mm:ssZ"로 지정됨
- 참고
  - "yyyy-MM-ddTHH:mm:ssZ : ISO\_INSTANT 포맷의 날짜 형식

```
5 PUT my_index
6 {
7   "mappings": {
8     "properties": {
9       "date": {
10        "type": "date"
11      }
12    }
13  }
14 }
```

정상 실행

```
16 PUT my_index/_doc/1
17 {
18   "date": "2019-01-01"
19 }
20 }
```

에러 발생

```
45 PUT my_index
46 {
47   "mappings": {
48     "properties": {
49       "date": {
50        "type": "date",
51        "format": "yyyy-MM-dd HH:mm:ss"
52      }
53    }
54  }
55 }
```

# Range 데이터 타입

- 범위가 있는 데이터를 저장할 경우 사용

```
6 # integer_range : 최솟값과 최댓값을 갖는 부호 있는 32bit 정수범위
7 # date_range : 64bit 정수 형태의 밀리초로 표시되는 날짜값의 범위
8 PUT range_index
9 {
10  "settings": {
11    "number_of_shards": 2
12  },
13  "mappings": {
14    "properties": {
15      "expected_attendees": {
16        "type": "integer_range"
17      },
18      "time_frame": {
19        "type": "date_range",
20        "format": "yyyy-MM-dd HH:mm:ss||yyyy-MM-dd||epoch_millis"
21      }
22    }
23  }
24 }
```

integer\_range와 date\_range로 정의

```
1 {
2   "acknowledged" : true,
3   "shards_acknowledged" : true,
4   "index" : "range_index"
5 }
6
```

```
25
26 # 유효 범위 : 10~20/2019년 10월 31일 12시 2019년 11월 1일 00:00:00까지
27 PUT range_index/_doc/1?refresh
28 {
29   "expected_attendees" : {
30     "gte" : 10,
31     "lte" : 20
32   },
33   "time_frame" : {
34     "gte" : "2019-10-31 12:00:00",
35     "lte" : "2019-11-01"
36   }
37 }
38
39
```

integer\_range와 date\_range 데이터 입력

```
1 {
2   "_index" : "range_index",
3   "_type" : "_doc",
4   "_id" : "1",
5   "_version" : 1,
6   "result" : "created",
7   "forced_refresh" : true,
8   "_shards" : {
9     "total" : 2,
10    "successful" : 1,
11    "failed" : 0
12  },
13   "_seq_no" : 0,
14   "_primary_term" : 1
15 }
```

# Range 데이터 타입

```
40 # 정상 조회
41 GET range_index/_search
42 {
43   "query" : {
44     "term" : {
45       "expected_attendees" : {
46         "value": 12
47       }
48     }
49   }
50 }
```

12 로 데이터 검색시 성공

```
1 {
2   "took" : 0,
3   "timed_out" : false,
4   "_shards" : {
5     "total" : 2,
6     "successful" : 2,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : {
12      "value" : 1,
13      "relation" : "eq"
14    },
15    "max_score" : 1.0,
16    "hits" : [
17      {
18        "_index" : "range_index",
19        "_type" : "_doc",
20        "_id" : "1",
21        "_score" : 1.0,
22        "_source" : {
23          "expected_attendees" : {
24            "gte" : 10,
25            "lte" : 20
26          },
27          "time_frame" : {
28            "gte" : "2019-10-31 12:00:00",
29            "lte" : "2019-11-01"
30          }
31        }
32      }
33    ]
34  }
35 }
```

```
52
53 # 조회시 해당 데이터가 미 존재
```

```
54 GET range_index/_search
55 {
56   "query" : {
57     "term" : {
58       "expected_attendees" : {
59         "value": 22
60       }
61     }
62   }
63 }
```

22 로 데이터 검색시 데이터 미존재

```
1 {
2   "took" : 0,
3   "timed_out" : false,
4   "_shards" : {
5     "total" : 2,
6     "successful" : 2,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : {
12      "value" : 0,
13      "relation" : "eq"
14    },
15    "max_score" : null,
16    "hits" : [ ]
17  }
18 }
```

# Boolean 데이터 타입

- 참과 거짓이라는 두 논리값을 보유한 데이터 타입
- Boolean 데이터 타입
  - 참 : true or "true"
  - 거짓 : false or "false"

```
38
39 # boolean data가 아닌 다른 데이터 저장시 에러 발생
40 POST my_index/_doc/5
41 {
42   "is_published": "non-boolean data"
43 }
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
```

```
1 {
2   "error": {
3     "root_cause": [
4       {
5         "type": "mapper_parsing_exception",
6         "reason": "failed to parse field [is_published] of type [boolean] in document
          with id '5'"
7       }
8     ],
9     "type": "mapper_parsing_exception",
10    "reason": "failed to parse field [is_published] of type [boolean] in document with
      id '5'",
11    "caused_by": {
12      "type": "illegal_argument_exception",
13      "reason": "Failed to parse value [non-boolean data] as only [true] or [false]
        are allowed."
14    }
15  },
16  "status": 400
17 }
```

# Object 데이터 타입

- JSON 포맷의 문서는 내부 객체를 계층적으로 포함
- 문서의 filed는 단순히 값을 가질 수도 있지만  
복잡한 형태의 또 다른 문서 포함도 가능
- 방식
  - 다른 타입과 달리 특정 키워드 사용하지 않음

```
1 # Object 데이터 타입
2
3 DELETE my_index
4
5 # manager field 하위에 age, name field 구조로 생성
6 PUT my_index/_doc/1
7 {
8   "region": "US",
9   "manager": {
10     "age": 30,
11     "name": {
12       "first": "John",
13       "last": "Smith"
14     }
15   }
16 }
```

```
1 {
2   "_index" : "my_index",
3   "_type" : "_doc",
4   "_id" : "1",
5   "_version" : 1,
6   "result" : "created",
7   "_shards" : {
8     "total" : 2,
9     "successful" : 1,
10    "failed" : 0
11  },
12   "_seq_no" : 0,
13   "_primary_term" : 1
14 }
15 }
```

```
GET my_index/_mapping
{
  "my_index" : {
    "mappings" : {
      "properties" : {
        "manager" : {
          "properties" : {
            "age" : {
              "type" : "long"
            },
            "name" : {
              "properties" : {
                "first" : {
                  "type" : "text",
                  "fields" : {
                    "keyword" : {
                      "type" : "keyword",
                      "ignore_above" : 256
                    }
                  }
                },
                "last" : {
                  "type" : "text",
                  "fields" : {
                    "keyword" : {
                      "type" : "keyword",
                      "ignore_above" : 256
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

```
GET my_index/_doc/1
1 {
2   "_index" : "my_index",
3   "_type" : "_doc",
4   "_id" : "1",
5   "_version" : 1,
6   "_seq_no" : 0,
7   "_primary_term" : 1,
8   "found" : true,
9   "_source" : {
10    "region" : "US",
11    "manager" : {
12      "age" : 30,
13      "name" : {
14        "first" : "John",
15        "last" : "Smith"
16      }
17    }
18  }
19 }
```

# Object 데이터 타입

- 데이터 저장

```
22 # key-value 구조로 데이터 저장
23 PUT my_index/_doc/2
24 {
25   "region": "US",
26   "manager.age": 30,
27   "manager.name.first": "John",
28   "manager.name.last": "Smith"
29 }
30 GET my_index/_doc/2
```

```
1 {
2   "_index" : "my_index",
3   "_type" : "_doc",
4   "_id" : "2",
5   "_version" : 1,
6   "result" : "created",
7   "_shards" : {
8     "total" : 2,
9     "successful" : 1,
10    "failed" : 0
11  },
12   "_seq_no" : 1,
13   "_primary_term" : 1
14 }
```

```
1 {
2   "_index" : "my_index",
3   "_type" : "_doc",
4   "_id" : "2",
5   "_version" : 1,
6   "_seq_no" : 1,
7   "_primary_term" : 1,
8   "found" : true,
9   "_source" : {
10    "region" : "US",
11    "manager.age" : 30,
12    "manager.name.first" : "John",
13    "manager.name.last" : "Smith"
14  }
15 }
```

# Object 데이터 타입

- top-level mappings & inner object field

```
33 DELETE my_index
34
35 PUT my_index
36 {
37   "mappings": {
38     "properties": {
39       "region": {
40         "type": "keyword"
41       },
42       "manager": {
43         "properties": {
44           "age": { "type": "integer" },
45           "name": {
46             "properties": {
47               "first": { "type": "text" },
48               "last": { "type": "text" }
49             }
50           }
51         }
52       }
53     }
54   }
55 }
56
57 GET my_index/_mapping
```

```
1 {
2   "my_index" : {
3     "mappings" : {
4       "properties" : {
5         "manager" : {
6           "properties" : {
7             "age" : {
8               "type" : "integer"
9             },
10            "name" : {
11              "properties" : {
12                "first" : {
13                  "type" : "text"
14                },
15                "last" : {
16                  "type" : "text"
17                }
18              }
19            }
20          }
21        },
22        "region" : {
23          "type" : "keyword"
24        }
25      }
26    }
27  }
28 }
```



## 참고 문헌 및 사이트

---

- <https://www.elastic.co/guide/index.html>
- 엘라스틱서치 실무가이드, 위키북스 [ 권택환 외 5인]

Kim Hye Kyung