Elastic Stack을 활용한 Data Dashboard 만들기

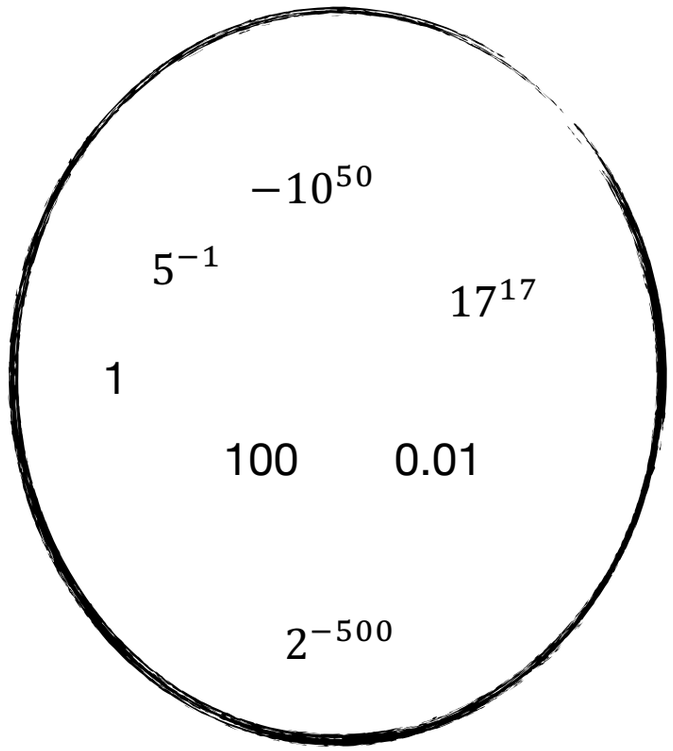
Week 3 - Elasticsearch API를 활용해보자

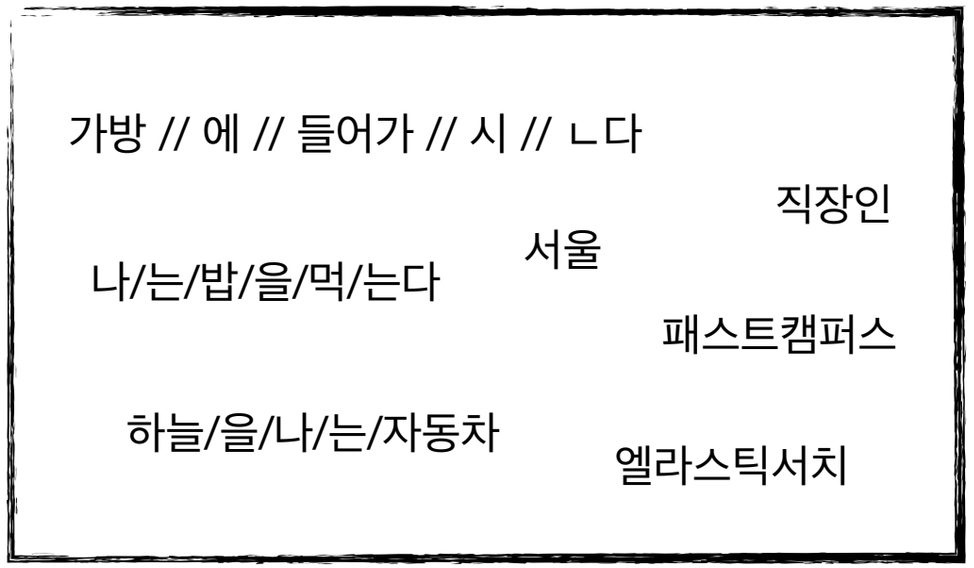
Fast Campus

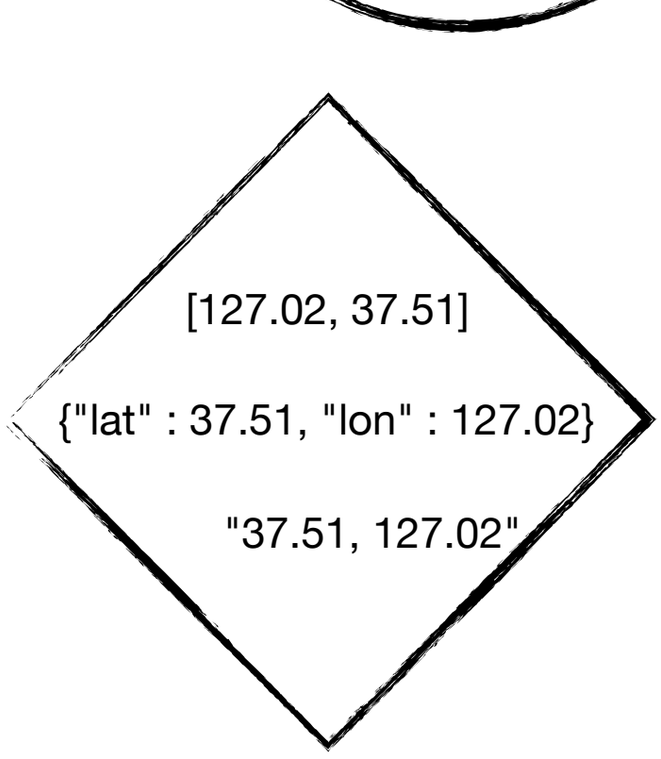
목차

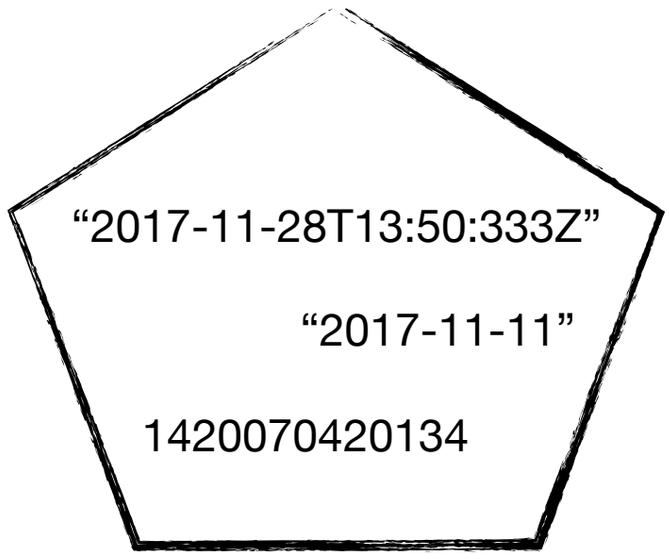
Dashboard에 데이터를 넣을 때 알면 좋은 (최소한의) Datatype을 살펴보자

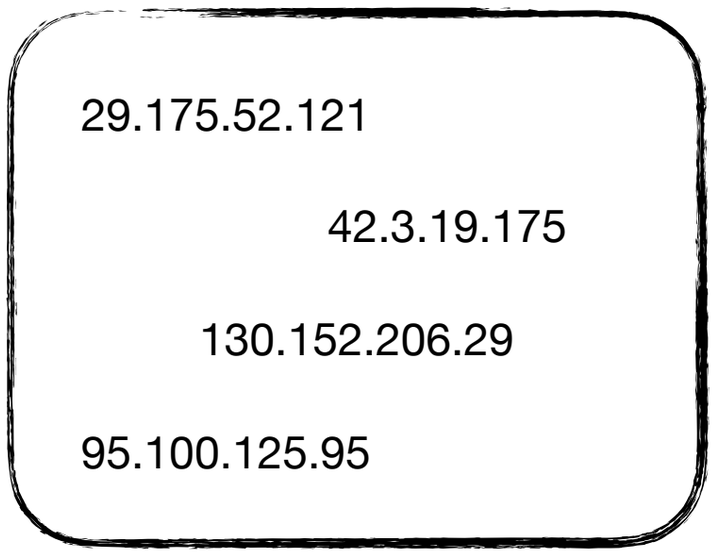
Data Type


-10^{50}
5^{-1}
17^{17}
1
100
0.01
2^{-500}


가방 // 에 // 들어가 // 시 // ㄴ다
직장인
서울
나/는/밥/을/먹/는다
패스트캠퍼스
하늘/을/나/는/자동차
엘라스틱서치

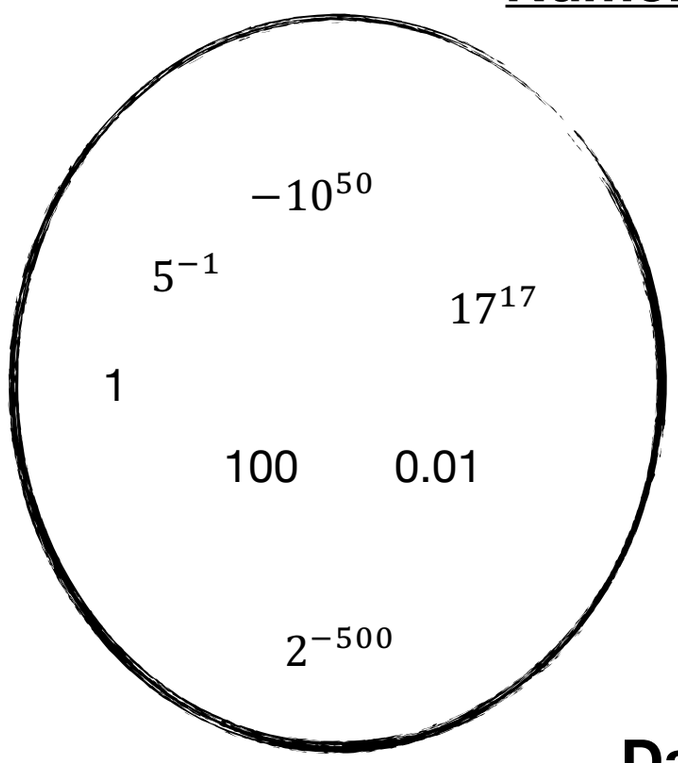

[127.02, 37.51]
{"lat" : 37.51, "lon" : 127.02}
"37.51, 127.02"


“2017-11-28T13:50:333Z”
“2017-11-11”
1420070420134

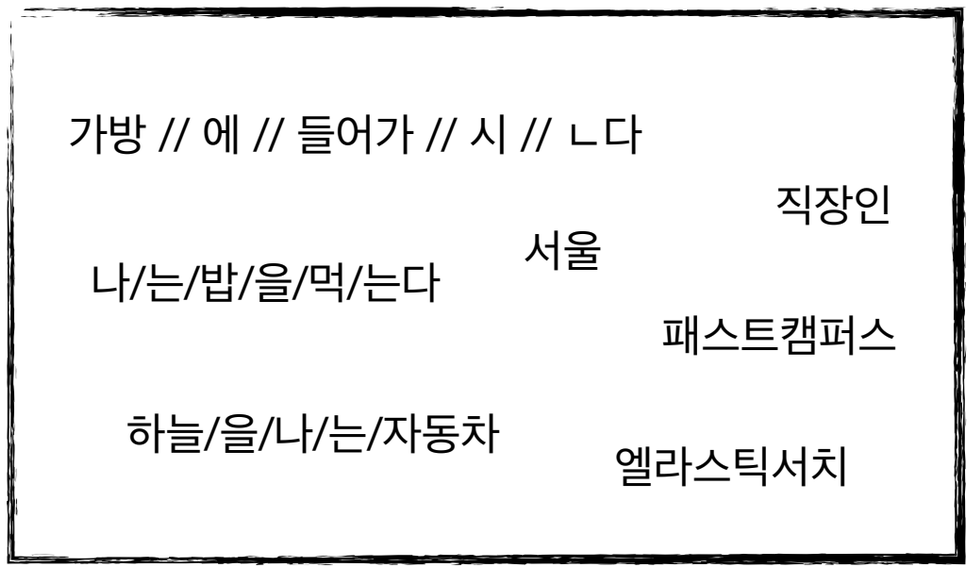

29.175.52.121
42.3.19.175
130.152.206.29
95.100.125.95

Data Type

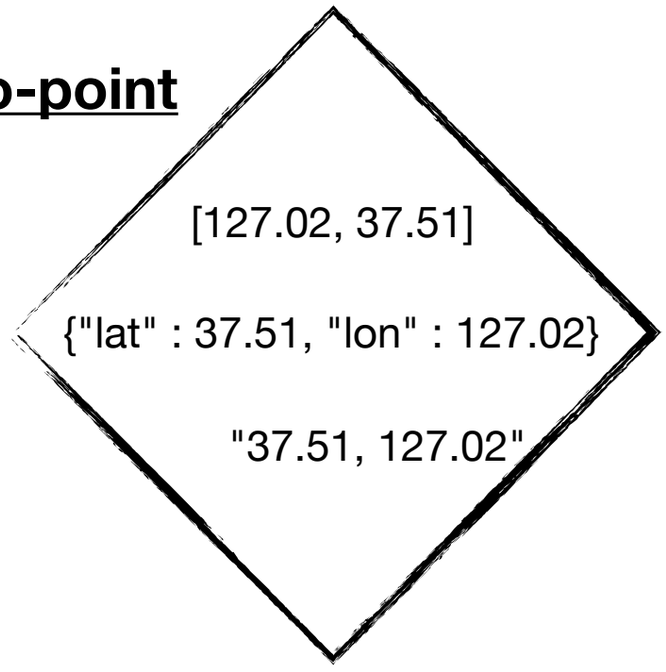
Numeric



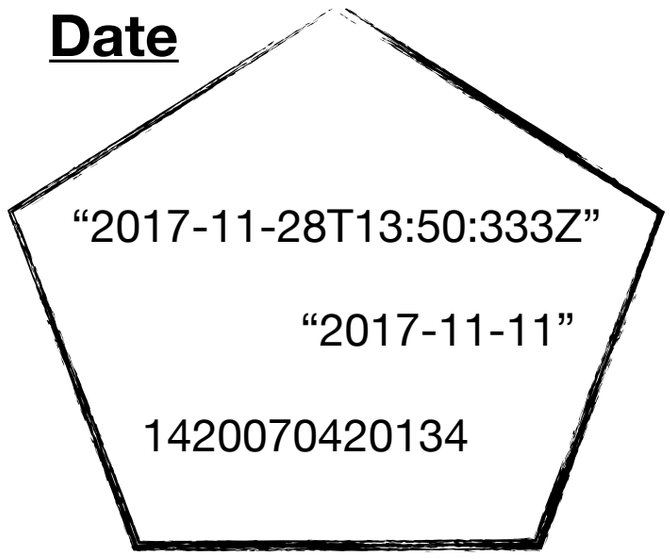
String



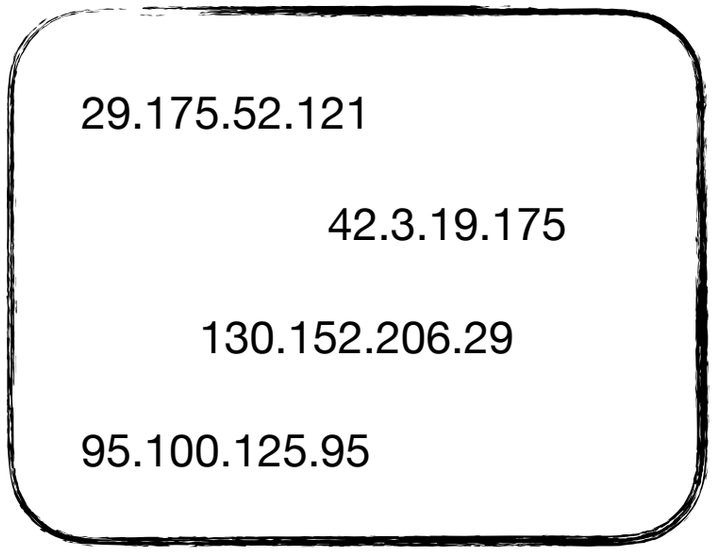
Geo-point



Date



IP



이 때 주의할 Type은 Numeric과 String이다.

Geo-point, Date, IP는 Format이 다양할 뿐 Type 자체는 1개다

Data Type - Numeric

Numeric datatypes

edit

The following numeric types are supported:

`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.
`float`	A single-precision 32-bit IEEE 754 floating point.
`half_float`	A half-precision 16-bit IEEE 754 floating point.
`scaled_float`	A floating point that is backed by a `long` and a fixed scaling factor.

Data Type - Numeric

Numeric datatypes

edit

The following numeric types are supported:

`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.
`float`	A single-precision 32-bit IEEE 754 floating point.
`half_float`	A half-precision 16-bit IEEE 754 floating point.
`scaled_float`	A floating point that is backed by a `long` and a fixed scaling factor.

어떤 Type을 사용해야 될까?

Data Type - Numeric

- 정수형 : 값의 크기에 따라 분류
- 부동소수점 : Precision 정도에 따라 분류

Numeric Field 값의 성격을 안다면 그 값에 맞는 Field를 선택하고,
그렇지 않을 경우 아래처럼 사용하자

- 정수형 : **Long**
- 부동소수점 : **Double**

Elasticsearch가 검색엔진인 만큼,
String Field 선택은 매우 중요하다!

Data Type - String

Keyword : 입력 String Field의 값을 한 단위로 보고 싶은 경우
Text : 입력 String Field를 더 작은 단위로 분석하고 싶은 경우

	Keyword로 설정	Text로 설정
가방에 들어가신다	가방에 들어가신다	가방 // 에 // 들어가 // 시 // ㄴ다
나는 밥을 먹는다	나는 밥을 먹는다	나 // 는 // 밥 // 을 // 먹 // 는다
패스트캠퍼스	패스트캠퍼스	패스트 // 캠퍼스
엘라스틱서치	엘라스틱서치	엘라스틱 // 서치

Kibana Dev Tools를 살펴보자

Dev Tools

Dev Tools History Settings Help

Console

```
1  GET shopping/_search
2  {"query":{"match_all":{}}}
3
4  GET shopping/_search
5  {
6    "query": {
7      "match_all": {}
8    }
9  }
```

Copy as cURL
Auto indent

```
1  {
2    "took": 0,
3    "timed_out": false,
4    "_shards": {
         "total": 5,
         "successful": 5,
         "skipped": 0,
         "failed": 0
9    },
10   "hits": {
11     "total": 20222,
12     "max_score": 1,
13     "hits": [
14       {
15         "_index": "shopping",
16         "_type": "shopping",
17         "_id": "AV-iDKZcRJy4v-Hns1Sk",
18         "_score": 1,
19         "_source": {
20           "접수번호": "277",
21           "주문시간": "2016-04-11T04:28:14",
22           "수령시간": "2016-04-11T07:18:14",
23           "예약여부": "예약",
24           "배송메모": "상품 이상",
25           "고객ip": "130.152.206.29",
26           "고객성별": "남성",
27           "고객나이": 40,
28           "물건좌표": "36.56404885993116, 129.87454556889554",
29           "고객주소_시도": "서울특별시",
30           "구매사이트": "g마켓",
31           "판매자평점": 3,
32           "상품분류": "스웨터",
33           "상품가격": 10000,
34           "상품개수": 1,
35           "결제카드": "시티"
36         }
37       },
38       {
39         "_index": "shopping",
40         "_type": "shopping",
41         "_id": "AV-iDKahRJy4v-Hns1Sp",
42         "_score": 1,
```

Dev Tools

과거에 작성한 API 이력 조회

cURL 명령어로 복사

Console : Elasticsearch API 작성

Output Pane : 작성한 Elasticsearch API 결과 조회

다양한 Elasticsearch API 중에서 Indices, Document, Search API를 알아보자

Indices API로 무얼 할 수 있을까?

API - Indices

Indices API로 무얼 할 수 있을까?

- **Index 생성/삭제**
- **Mapping 설정**

API - Indices

Index 생성

`PUT {Index 이름}` → `PUT week3_higee`

Index 삭제

`DELETE {Index 이름}` → `DELETE week3_higee`

API - Indices

Index 생성 후 Mapping 추가

```
PUT {Index 이름}/_mapping/{Type 이름}
{
  "properties": {
    "{Field 이름}" : {
      "type" : "{Field Type}"
    }
  }
}
```

```
PUT week3_higee/_mapping/week3_higee
{
  "properties": {
    "가격" : {
      "type" : "integer"
    }
  }
}
```

API - Indices

Index 생성하면서 Mapping 추가

```
PUT {Index 이름}
{
  "mappings": {
    "{Type 이름}": {
      "properties": {
        "{Field1 이름}" : {
          "type": "{Field1 Type}"
        },
        "{Field2 이름}": {
          "type" : "{Field2 Type}"
        }
      }
    }
  }
}
```



```
PUT week3_higee_mapping_test
{
  "mappings": {
    "week3_higee": {
      "properties": {
        "가격" : {
          "type": "integer"
        },
        "날짜": {
          "type" : "date"
        }
      }
    }
  }
}
```

Template 활용해서 자동으로 Mapping 추가

```
PUT _template/{Template 이름}
{
  "template": "{Index Pattern}",
  "mappings": {
    "{Type 이름}": {
      "properties": {
        "Field1 이름": {
          "type": "Field1 Type"
        },
        "Field2 이름": {
          "type": "Field2 Type"
        }
      }
    }
  }
}
```



```
PUT _template/template_higee
{
  "template": "higee-log*",
  "mappings": {
    "my_type": {
      "properties": {
        "가격": {
          "type": "integer"
        },
        "날짜": {
          "type": "date"
        }
      }
    }
  }
}
```

API - Indices

기존 Mapping에 새로운 Field Mapping 추가하기

```
PUT {Index 이름}/_mapping/{Type 이름}
{
  "properties": {
    "{Field 이름}" : {
      "type" : "{Field Type}"
    }
  }
}
```

```
PUT week3_higee/_mapping/week3_higee
{
  "properties": {
    "나이" : {
      "type" : "integer"
    }
  }
}
```

API - Indices

Mapping 확인

`GET /{Index 이름}/_mapping/{Type 이름}` → `GET week3_higee/_mapping/week3_higee`

Document API로 무얼 할 수 있을까?

Document API로 무얼 할 수 있을까?

- Document 생성 Ⓒ
- Document 조회 Ⓡ
- Document 수정 Ⓤ
- Document 삭제 Ⓓ
- Document 복사

Document 추가

```
PUT {Index 이름}/{Type 이름}/{ID}
{
  "{Field 이름}" : {Value}
}
```

```
PUT week3_higee/week3_higee/1
{
  "가격" : 10000,
  "나이" : 17
}
```

```
PUT week3_higee/week3_higee/2
{
  "가격" : 2000,
  "나이" : 20
}
```

```
PUT week3_higee/week3_higee/3
{
  "가격" : 1000,
  "나이" : 25
}
```

ID로 Document 조회

※ Query로 Document 조회하는 건 Search API

`GET {Index 이름}/{Type 이름}/{ID}` → `GET week3_higee/week3_higee/1`

ID로 Document 삭제

`DELETE {Index 이름}/{Type 이름}/{ID}` → `DELETE week3_higee/week3_higee/1`

Query로 Document 삭제

```
POST {Index 이름}/_delete_by_query
{
  "query": {
    "match": {
      "{Field 이름}": "{Value}"
    }
  }
}
```

```
POST week3_higee/_delete_by_query
{
  "query": {
    "match": {
      "나이": 17
    }
  }
}
```

ID로 Document 수정

```
POST {Index 이름}/{Type 이름}/{ID}/_update
{
  "doc": {
    "{Field}" : {Value}
  }
}
```

```
POST week3_higee/week3_higee/2/_update
{
  "doc": {
    "가격" : 20000
  }
}
```

ID로 Document 수정 (Upsert)

```
POST {Index 이름}/{Type 이름}/{ID}/_update
{
  "doc" : {
    "{Field 이름}" : "{Value}"
  },
  "doc_as_upsert" : true
}
```

```
POST week3_higee/week3_higee/2/_update
{
  "doc" : {
    "가격" : "50000"
  },
  "doc_as_upsert" : true
}
```

```
POST week3_higee/week3_higee/777/_update
{
  "doc" : {
    "가격" : "50000"
  },
  "doc_as_upsert" : true
}
```

Query로 Document 수정

```
POST {Index 이름}/{Type 이름}/_update_by_query
{
  "script": {
    "source": "ctx._source[{Field 이름}] = Value
  },
  "query": {
    "term": {
      "{Field 이름}": "Value"
    }
  }
}
```



```
POST week3_higee/week3_higee/_update_by_query
{
  "script": {
    "source": "ctx._source['가격'] = 7777"
  },
  "query": {
    "term": {
      "나이": 17
    }
  }
}
```

API - Document

Index 내 모든 Document 복사

```
POST _reindex
{
  "source": {
    "index": "{복사하려는 원본 Index 이름}"
  },
  "dest": {
    "index": "{복사본을 저장할 Index 이름}"
  }
}
```



```
POST _reindex
{
  "source": {
    "index": "week3_higee"
  },
  "dest": {
    "index": "week3_higee_reindex"
  }
}
```

Index 내 일부 Document 복사

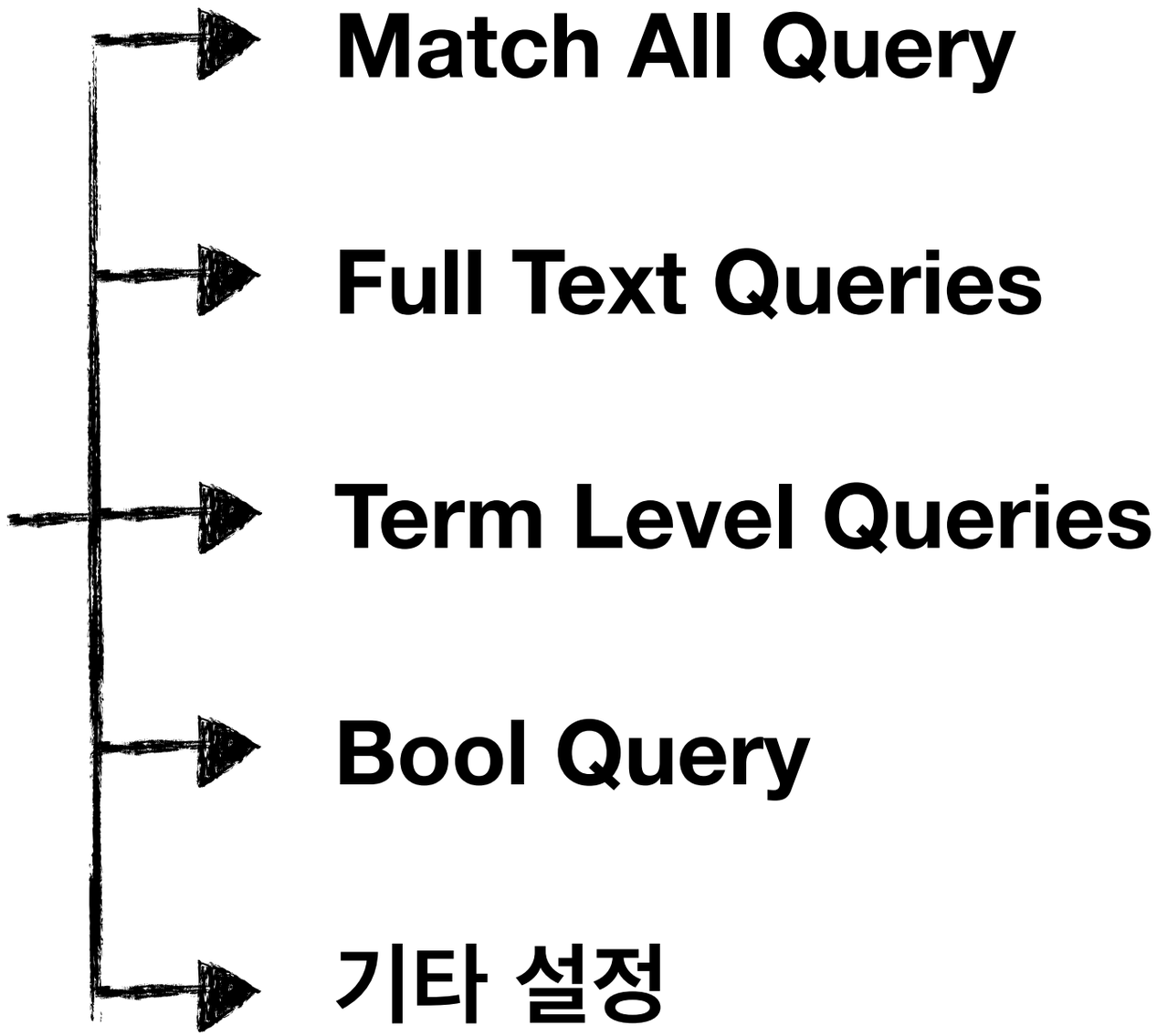
```
POST _reindex
{
  "source": {
    "index": "{복사하려는 Index 이름}",
    "type" : "{복사하려는 Type 이름}",
    "query": {
      "term": {
        "{Field 이름}": "{Value}"
      }
    }
  },
  "dest": {
    "index": "{복사본을 저장할 Index 이름}"
  }
}
```

```
POST _reindex
{
  "source": {
    "index": "week3_higee",
    "type" : "week3_higee",
    "query": {
      "term": {
        "나이": 17
      }
    }
  },
  "dest": {
    "index": "week3_higee_reindex2"
  }
}
```

Search API로 무얼 할 수 있을까?

Search API로 무얼 할 수 있을까?

(특히 Request Body Search)



API - Search

모든 Documents 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query" : {
    "match_all" : {}
  }
}
```

```
GET shopping/shopping/_search
{
  "query" : {
    "match_all" : {}
  }
}
```

API - Search


```
{
  "took": 0,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 20222,
    "max_score": 1,
    "hits": [
      {
        "_index": "shopping",
        "_type": "shopping",
        "_id": "AV-iDKZcRJy4v-Hns1Sk",
        "_score": 1,
        "_source": {
          "접수번호": "277",
          "주문시간": "2016-04-11T04:28:14",
          "고객ip": "130.152.206.29",
          "물건좌표": "36.56, 129.87",
          "판매자평점": 3,
          "상품분류": "스웨터",
          "상품가격": 10000,
        }
      }
    ]
  }
}
```


① Elasticsearch 검색 소요시간 (millisecond)

② 검색결과가 time out에 걸렸는지 표시

③ 몇 개의 shards가 검색되었는지 표시

④ 검색된 Documents의 개수

⑤ 실제 Documents 내용

검색어와 정확히 일치하는 Document 조회

```
PUT {Index 이름}/{Type 이름}/{ID}
{
  "{Field 이름}" : {Value}
}
```

```
PUT week3_higee/week3_higee/1
{
  "가격" : 10000,
  "나이" : 17
}
```

API - Search

검색어 중 적어도 1개와 정확히 일치하는 Document 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query" : {
    "terms" : {
      "{Field 이름}" : ["{Value}", "{Value}"]
    }
  }
}
```



```
GET shopping/shopping/_search
{
  "query" : {
    "terms" : {
      "상품분류" : ["셔츠", "스웨터"]
    }
  }
}
```

특정 접두어로 시작하는 Document 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query": {
    "prefix" : {
      "{Field 이름}" : "{Value}"
    }
  }
}
```

```
GET shopping/shopping/_search
{
  "query": {
    "prefix" : {
      "고객주소_시도" : "경상"
    }
  }
}
```

Wildcard Expression 만족하는 Documents 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query": {
    "wildcard" : {
      "{Field 이름}" : "{Value}"
    }
  }
}
```

```
GET shopping/shopping/_search
{
  "query": {
    "wildcard" : {
      "고객주소_시도" : "경**도"
    }
  }
}
```

API - Search

검색어와 유사한 Documents 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query": {
    "fuzzy" : {
      "{Field 이름}" : "{Value}"
    }
  }
}
```

```
GET {Index 이름}/{Type 이름}/_search
{
  "query": {
    "fuzzy" : {
      "{Field 이름}" : "{Value}"
    }
  }
}
```

특정 Numeric Field가 임의의 범위 내에 있는 Documents 조회

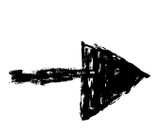
```
GET {Index 이름}/{Type 이름}/_search
{
  "query": {
    "range": {
      "{Field 이름}": {
        "gte": "{Value},
        "lte": "{Value},
      }
    }
  }
}
```

```
GET shopping/shopping/_search
{
  "query": {
    "range": {
      "주문시간": {
        "gte": "2017-02-15"
      }
    }
  }
}
```

API - Search

Lucene Query Syntax를 만족하는 Documents 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query" : {
    "query_string" : {
      "query" : "{LUCENE QUERY}"
    }
  }
}
```



```
GET shopping/shopping/_search
{
  "query" : {
    "query_string" : {
      "query": "고객나이 : [10 TO 25]"
    }
  }
}
```

API - Search

non-null value가 있는 Documents 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query": {
    "exists" : {
      "field" : "{Field 이름}"
    }
  }
}
```

```
GET shopping/shopping/_search
{
  "query": {
    "exists" : {
      "field" : "상품분류"
    }
  }
}
```

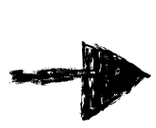
Boolean으로 묶인 조건(들)을 만족하는 Documents 조회

```
GET shopping/shopping/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "term": { "결제카드":  "시티" }
        }
      ],
      "must_not": [
        { "term": { "구매사이트": "11번가" }},
        { "range": {
            "상품가격" : {"gte" :  20000 }}}
      ]
    }
  }
}
```

API - Search

조회된 Documents의 몇 번째~몇 번째 Documents 조회할지 설정

```
GET {Index 이름}/{Type 이름}/_search
{
  "from" : {몇 번째부터},
  "size" : {몇 개},
  "query" : {
    "match_all" : {}
  }
}
```



```
GET shopping/shopping/_search
{
  "from" : 0,
  "size" : 1,
  "query" : {
    "match_all" : {}
  }
}
```

Documents를 특정 기준으로 정렬

```
GET {Index 이름}/{Type 이름}/_search
{
  "query": {
    "match_all": {}
  },
  "sort": [
    {
      "{FIELD 이름}": {
        "order": "{정렬기준}"
      }
    }
  ]
}
```



```
GET shopping/shopping/_search
{
  "query": {
    "match_all": {}
  },
  "sort": [
    {
      "판매자평점": {
        "order": "desc"
      }
    }
  ]
}
```

Documents에서 조회하고 싶은 Field 설정

```
GET {Index 이름}/{Type 이름}/_search
{
  "_source": "{Field 이름}",
  "query" : {
    "match_all" : {}
  }
}
```

```
GET shopping/shopping/_search
{
  "_source": "구매사이트",
  "query" : {
    "match_all" : {}
  }
}
```

```
GET shopping/shopping/_search
{
  "_source": {
    "includes" : ["고객*", "구매사이트"],
    "excludes" : "상품*"
  },
  "query" : {
    "match_all" : {}
  }
}
```

Query 결과를 Page 별로 조회

```
POST shopping/shopping/_search?scroll=10m
{
  "size": 500,
  "query": {
    "match" : {
      "상품분류" : "셔츠"
    }
  }
}
```

```
POST  /_search/scroll
{
  "scroll" : "10m",
  "scroll_id" : "{_scroll_id}"
}
```

결과로 나온 **scroll_id**를 입력

API

모든 Query를 외워야 되나?

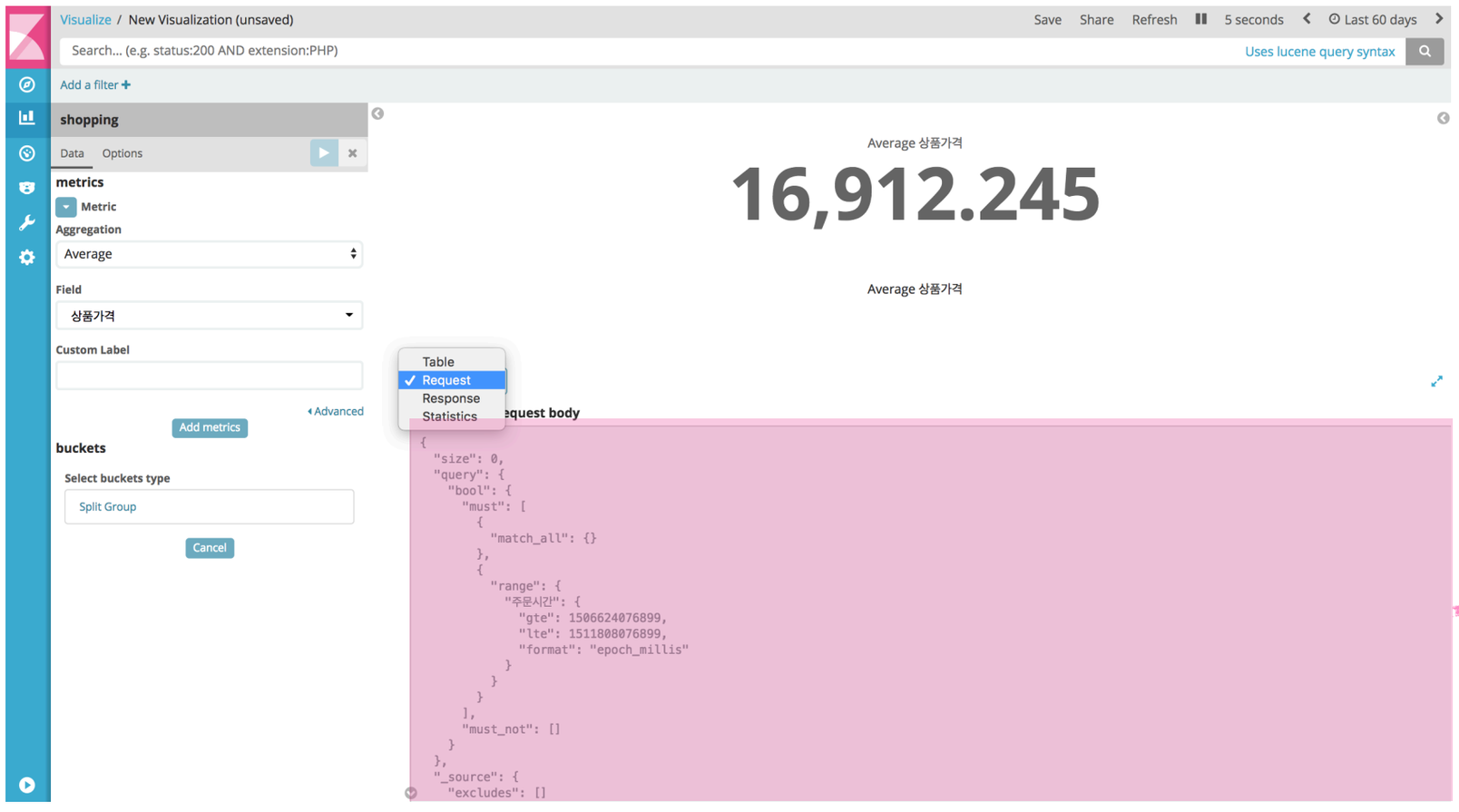
(Elasticsearch Query 결과를 시각화하는)
Kibana를 적극 활용하자

API

상품가격 Field의 평균을 구한다고 하자.

문제는 Aggregation Query는 아직 접하지도 않았다.

이 때 **Visualization Spy**를 보면 복사/붙여넣기가 가능한 코드를 볼 수 있다



Elasticsearch Request Body!!!

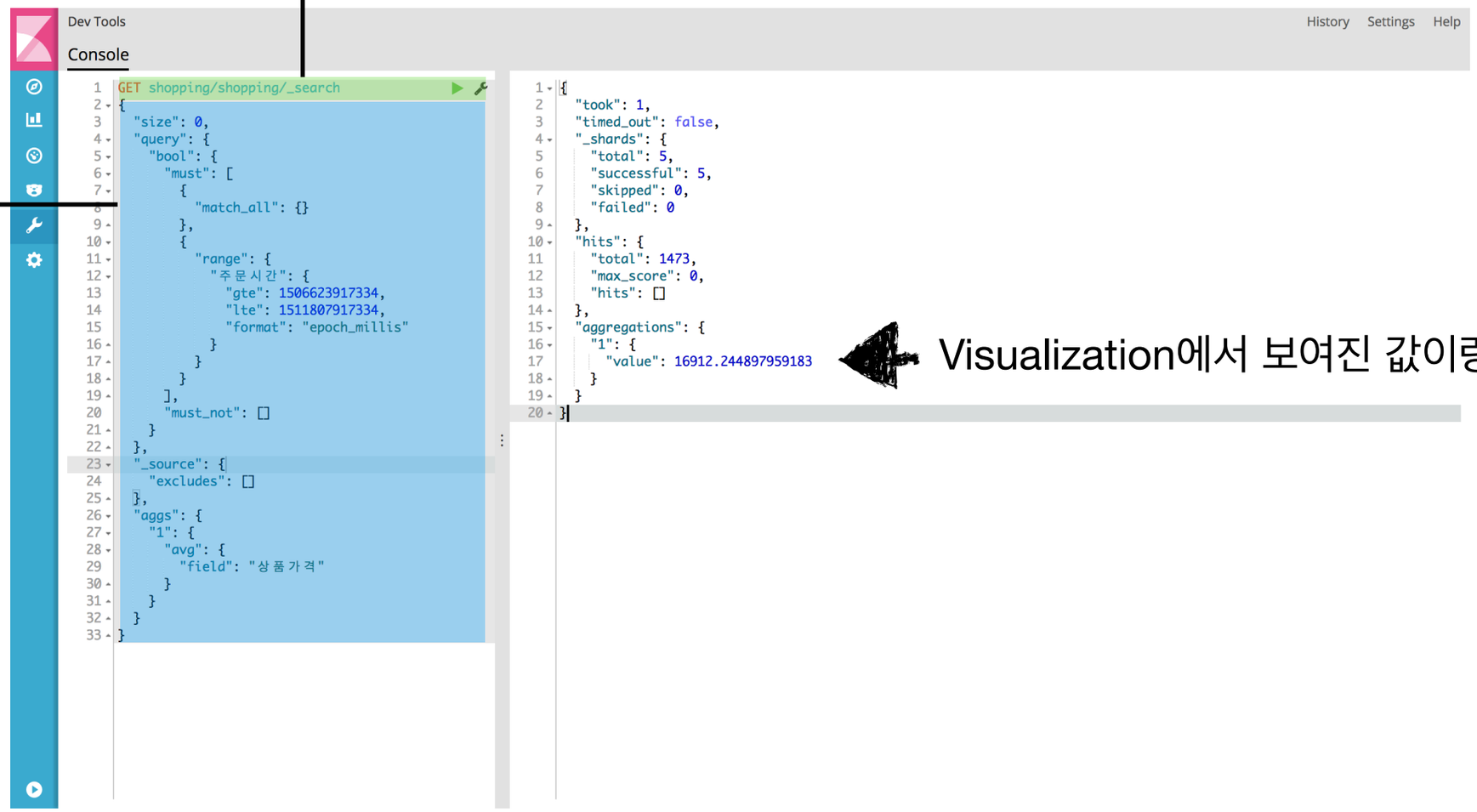
API

앞장에서 봤던 Elasticsearch Request Body 전체를 복사하자 (Ctrl + C)

그리고 Dev Tools로 이동하자

그 다음 복사한 Elasticsearch Request Body를 붙여넣자

마지막으로 가장 윗줄에 GET method를 붙여주자



Visualization에서 보여진 값이랑 일치한다!

API

그렇다면 Sum Aggregation은 어떻게 할까?

API

```
GET shopping/shopping/_search
{
  "size": 0,
  "query": {
    "bool": {
      "must": [
        {
          "match_all": {}
        },
        {
          "range": {
            "주문시간": {
              "gte": 1506623917334,
              "lte": 1511807917334,
              "format": "epoch_millis"
            }
          }
        }
      ],
      "must_not": []
    }
  },
  "_source": {
    "excludes": []
  },
  "aggs": {
    "1": {
      "avg": {
        "field": "상품가격"
      }
    }
  }
}
```

이처럼
- Visualize
- Visualization Spy
- Request
- Dev Tools

방식으로 접근하면
빠르게 원하는 query를 작성할 수 있다.

→ **avg**라고 되어 있는 부분을 **sum**으로 바꾸면 된다!

설치 - AWS EC2 Instance 생성

우선 AWS EC2 Instance를 생성해보자 (클릭)

설치 - Elastic Stack 설치

Elastic Stack을 설치해보자 (클릭)

~~혹시~~ 설치를 다 해버렸다면 다음 문제를 풀어보자!

보너스 - 문제

1. week3_exercise_{id}라는 이름의 Index를 생성하고 (id : email 앞자리) 다음과 같이 mapping을 추가하자
   - Field : 데이터이용량, Datatype : long
   - Field : 날짜, Datatype : date
   - Field : 어플리케이션, Datatype : keyword
2. Documents API를 이용해서 다음의 Documents를 넣어보자
   - {“데이터이용량” : 10000, “날짜” : “2017-11-27T13:00”, “어플리케이션” : “페이스북”}
   - {“데이터이용량” : 15000, “날짜” : “2017-11-27T15:00”, “어플리케이션” : “지메일”}
3. 데이터 사용량을 Bytes 형식으로 표시되게 해보자 (예: 14.648KB, 9.766KB)
4. Reindex API를 이용해서 위의 Index에서 “어플리케이션” : “페이스북” 인 document를 wee3_exercise_{id}_reindex로 옮겨보자
5. Search API로 shopping Index에서 상품가격이 5000~10000 사이인 Documents를 조회해보자
6. Search API로 shopping Index에서 다음과 같은 조건을 만족하는 Documents를 조회해보자
   - 상품분류가 “셔츠”, “니트”, “스웨터” 중 하나이고, 주문날짜가 2017-05-01 ~ 2017-07-01 사이이지만,
   - 고객주소_시도가 “서울특별시”가 아니다
7. Search API로 shopping Index에서 주문날짜를 최신순으로 정렬하고 가장 앞 5개를 출력해보자
8. Search API로 shopping Index에서 상품가격이 작은순으로 정렬하고 최신순으로 정렬하고 가장 앞 5개의 상품분류를 출력해보자
9. Search API로 shopping Index에서 2017년 11월 동안 상품가격의 최대값을 구해보자
10. Search API로 shopping Index에서 2017년 11월 동안 일별로 평균 상품가격이 높은 구매사이트 5개 별로 평균 상품가격을 구해보자