
TECH
University
ebay
CHINA CENTER
OF EXCELLENCE
elastic

Elasticsearch Search Introduction

Pei Wang

Aug 2018

Agenda

- Search fundamental
- Aggregation
- Mapping
- Text analysis
- Advanced search

1

Search Fundamental

Search fundamental

Preparation

- Access go/training_cluster
- Access go/training_cluster_kibana
- Download scripts from go/training_scripts
- Click Dev Tool on Kibana page

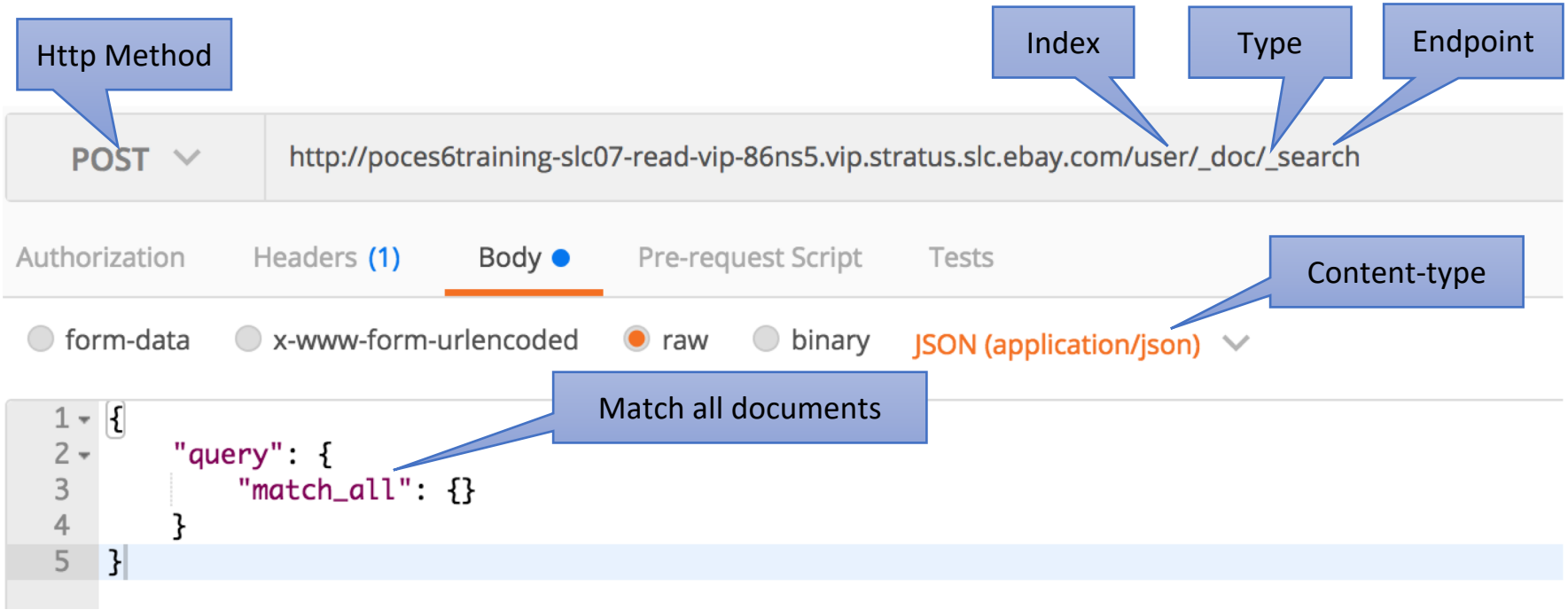
Search fundamental

Training data set

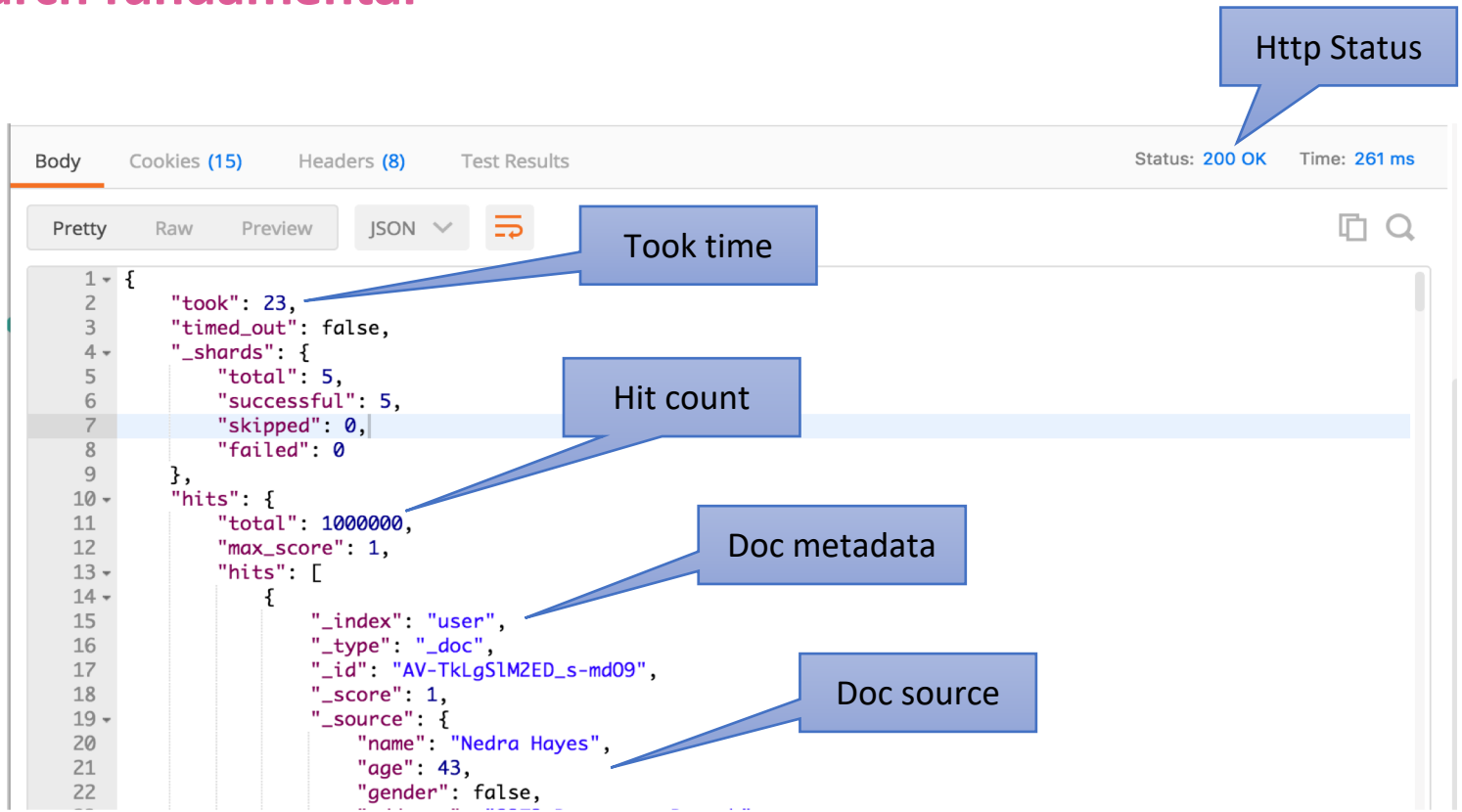
- About one million user information
- Random generated, only for test


```
            "name": "Jack Hilpert",
            "age": 65,
            "gender": true,
            "address": "7981 Kozey Stravenue",
            "phoneNumber": "456.245.6920 x747",
            "ip": "156.82.245.144",
            "ipV6": "cbe5:395b:f136:abdc:78ed:a46f:084f:03e6",
            "email": "fidel.windler@hotmail.com",
            "birthDay": "2003-09-21",
            "register_at": "2017-01-22T07:02:54.501-0700",
            "description": "Ex sunt est.Unde commodi molestias ipsum.Odio sed reprehenderit et.Quis aspernatur est et
officiis nihil ipsam.Et est reprehenderit eos.Placeat et rerum voluptas doloribus.Quibusdam sint cum mollitia.Eos cum
dolorem ea.At fuga est est vel dignissimos aut s",
            "ssnId": "579-55-5174",
            "favourite_book": {
              "author": "Verdie Kirlin",
              "publisher": "Anvil Press Poetry"
            },
            "doubleVal": -3692728743023917000,
            "longVal": 744928953693166200,
            "location": {
              "lat": -30.995067,
              "lon": 160.79841
            },
            "city": "Sar Meel",
            "company": "Beahan and Sons",
            "university": "The Georgia College",
            "customField1": "Enim molestiae temporibus est asperiores qui.Illo et culpa et expedita.Harum qui",
            "customField2": "Exercitationem voluptate voluptatum.Sunt veritatis molestias temporibus modi.Con",
            "customField3": "Consequuntur blanditiis qui deleniti eius iste vel.Aut tempora accusantium.Sit d"
        }
```

Search fundamental



Search fundamental



Specify search indices

For ES6 and later version, there is no necessary to specify type since Elasticsearch only allow on type in one index

Syntax	Indices searched
/_search	Search all indices in this cluster
/index-1/_search	Search index-1
/index-1, index-2/_search	Search index-1 and index-2
/index-*/_search	Search all indices start with “index-”

From/Size

Pagination of results can be done by using the from and size parameters.

- From, define the offset from the first result.
- Size, maximum amount of hits to be returned.

```
POST /user/_search
{
  "from":10,
  "size": 5,
  "query": {
    "match_all": {}
  }
}
```

Sort

- Allow user to add one or more sort on specific fields.
- Elasticsearch use _score as default sort field.
- Use _doc to disable sort if you really do not need it.

```
POST /user/_search
{
  "sort" : [
        "age",
        {"longVal": "desc"},
        { "birthDay" : {"order" : "desc"}},
        "_score"
    ],
  "query": {
    "match_all": {}
  }
}
```

Source filtering

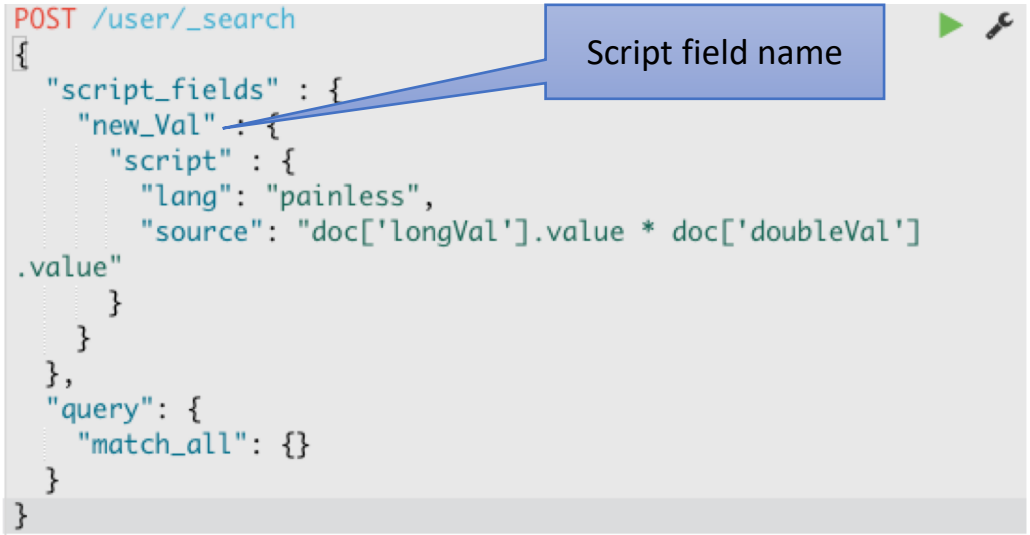
- Source field can be disabled by set _source parameter to false.
- Accept string or wildcard to control which part of _source should be returned.

```
POST /user/_search
{
  "_source":"name",
  "query": {
    "match_all": {}
  }
}
```

```
  "hits": [
    {
      "_index": "user",
      "_type": "_doc",
      "_id": "AV-TkLgSlM2ED_s-md09",
      "_score": 1,
      "_source": {
        "name": "Nedra Hayes"
      }
    },
```

Script fields

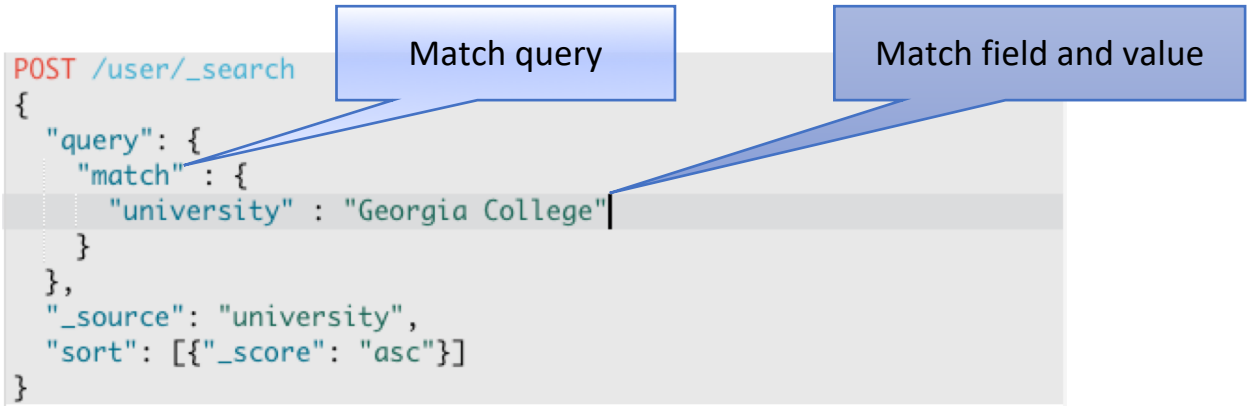
- Script fields can work on fields that are not stored and return custom values.
- Use doc[].value to access document fields



Quiz

1. **True or False:** Elasticsearch use **PUT** method for search APIs.
2. **True or False:** Elasticsearch can search multiple indices in one request.
3. **True or False:** Elasticsearch use **From/To** to control the hit result window.
4. **True or False:** Type parameter must be specified in search URL.
5. Please list the metadata for hit results in response body.
6. Run a query on user index, return all users, only “name” and “age” should be included in _source field, sort returned user by age with ascend order.

Match query



- Match query accept text/numeric/dates, analyze them and constructs a query.
- By default, match query use ”or” operator for the provided text, this query means “Georgia **OR** College”.
- If the “university” field of any user contains “Georgia” or “College”, it will be hit.

Match query

- Use operator “and” to change query to “Georgia **AND** College” .

```
POST /user/_search
{
  "query": {
    "match" : {
      "university" : {
        "query": "Georgia College",
        "operator": "and"
      }
    }
  },
  "_source": "university",
  "sort": [{"_score": "asc"}]
}
```

Match_phrase query

- Match query do not care about the term position, you can get the same result if change the term order in query.
- Match_phrase query
  - All terms must exist in field
  - The position of terms must be in same order and near each other

```
POST /user/_search
{
  "query": {
    "match_phrase" : {
      "university" : "Georgia College"
    }
  },
  "_source": "university",
  "sort": [{"_score": "asc"}]
}
```

Match_phrase query

- Slop parameter
  - How far apart terms are allowed to be

```
POST /user/_search
{
  "query": {
    "match_phrase" : {
      "university" : {
        "query": "West College",
        "slop": 1
      }
    }
  },
  "_source": "university",
  "sort": [{"_score": "asc"}]
}
```

Match_phrase_prefix query

- Allow prefix match on the last term in the text

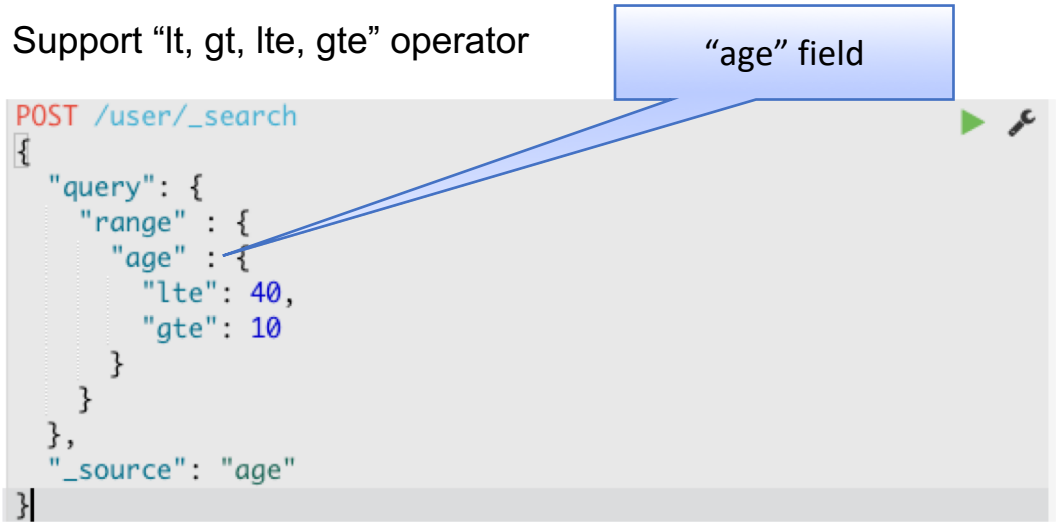
```
POST /user/_search
{
  "query": {
    "match_phrase_prefix" : {
      "university" : "Georgia Coll"
    }
  },
  "_source": "university",
  "sort": [{"_score": "asc"}]
}
```

Multiple match query

- Allow to search on multiple field
- Field name can be specified with wildcards

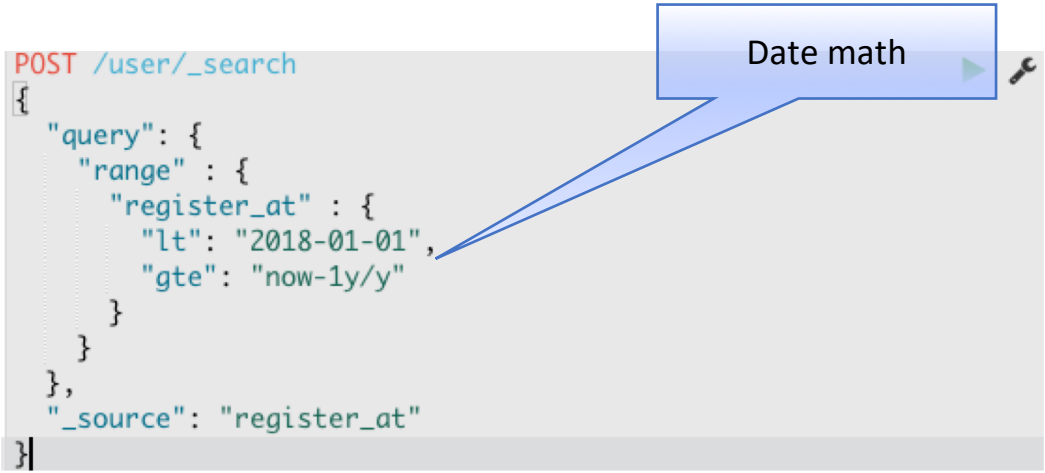
Range query

- Match documents with fields that have terms within a certain range.
- Can be applied on number/date field
- Support “lt, gt, lte, gte” operator



Range date query

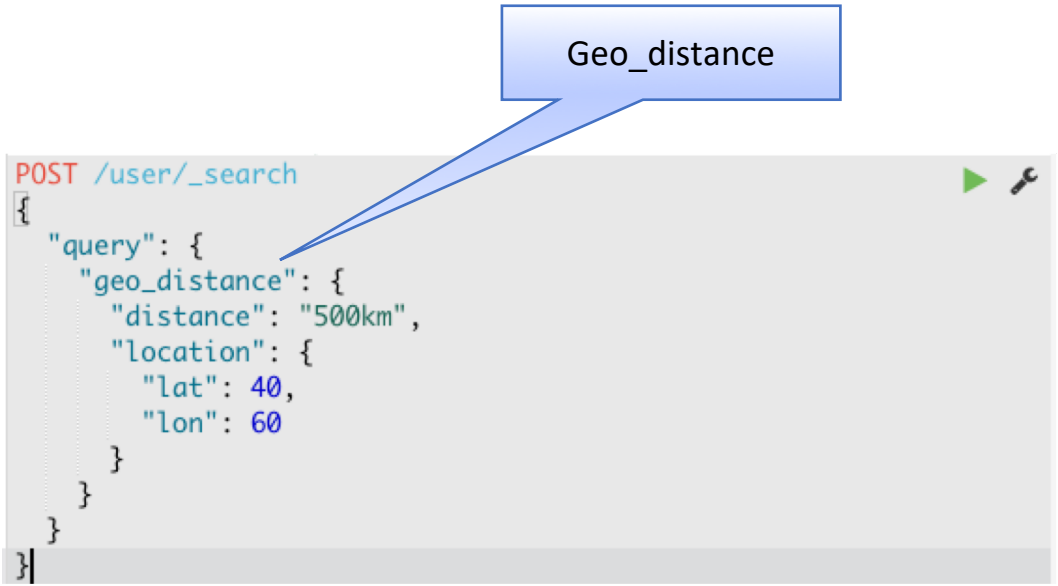
- For date type, range query can use date math
- ”now-1y/y” means current date minus one year, then round up to 2017-01-01T00:00:00



Geo query

Geo queries include

- geo_shape
- geo_bounding_box
- geo_distance
- geo_polygon



Quiz

1. **True or False:** Elasticsearch match query clause can accept text/numeric/dates.
2. **True or False:** Given a query like {“match”: {“name”: “Jack Smith”}}, user whose name is “Smith Jack” will not be returned.
3. **True or False:** match_phrase query require query text must be in the same order and next to each other.
4. What does the “slop” parameter means?
5. Please list the supported operator for range query.
6. Run a query on user index, return users whose age between 30(included) and 40(not included).
7. Run a query on user index, return users whose “university” or “name” or “address” contains “west”

Compound query

Compound query wrap other compound or leaf queries, it can combine their results and scores, or change their behavior.

Compound query includes:

- bool query
- constant_score query
- dis_max query
- function_score query
- boosting query

Bool query

- Bool query is built using one or more boolean clause queries, each clause with a typed occurrence.

Occur	Description
must	The clause query **must** appear in matching documents and will contribute to the score
must_not	The clause query **must not** appear in matching documents
should	The clause query **should optionally** appear in matching document and will contribute to the score if matched.
filter	The clause query **must** appear in matching documents and will **not** contribute to the score

Bool query

- must/must_not/should/filter clause can accept a single query or an array
- The matching document must have “Georgia” in university field, and must have “Sar Meel” in city field.
- Should clause is optional, but it will contribute to final score.
- Think about
  - What will happen if move the name query to must clause?
  - How about use “filter” instead of “should”?

```
POST /user/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
          "university": "Georgia"
          }
        },{
          "match_phrase": {
            "city": "Sar Meel"
          }
        }
      ],
      "should":{
        "match": {
          "name": "jack"
        }
      }
    }
  }
}
```

Bool query

What will happen if there are only should queries? Will it match all documents?

- If a bool query doesn't have "must" and "filter" query and only have should queries
- At least one clause must be matched (unless you specify the minimum_should_match parameter)

```
POST /user/_search
{
  "query": {
    "bool": {
      "should":[
        {"match": { "name": "Jack"}},
        {"match": { "name": "Emily"}},
        {"match": { "name": "Leo"}}
      ]
    }
  }
}
```

Bool query

- A “should_not” query
- Notice how to nest bool queries

```
POST /user/_search
{
  "query": {
    "bool": {
      "must":{"match": {"university": "Georgia"}},
      "should":{
        "bool": {
          "must_not": {
            "match":{"name":"jack"}
          }
        }
      }
    }
  },
  "sort":{"_score":"asc"}
}
```

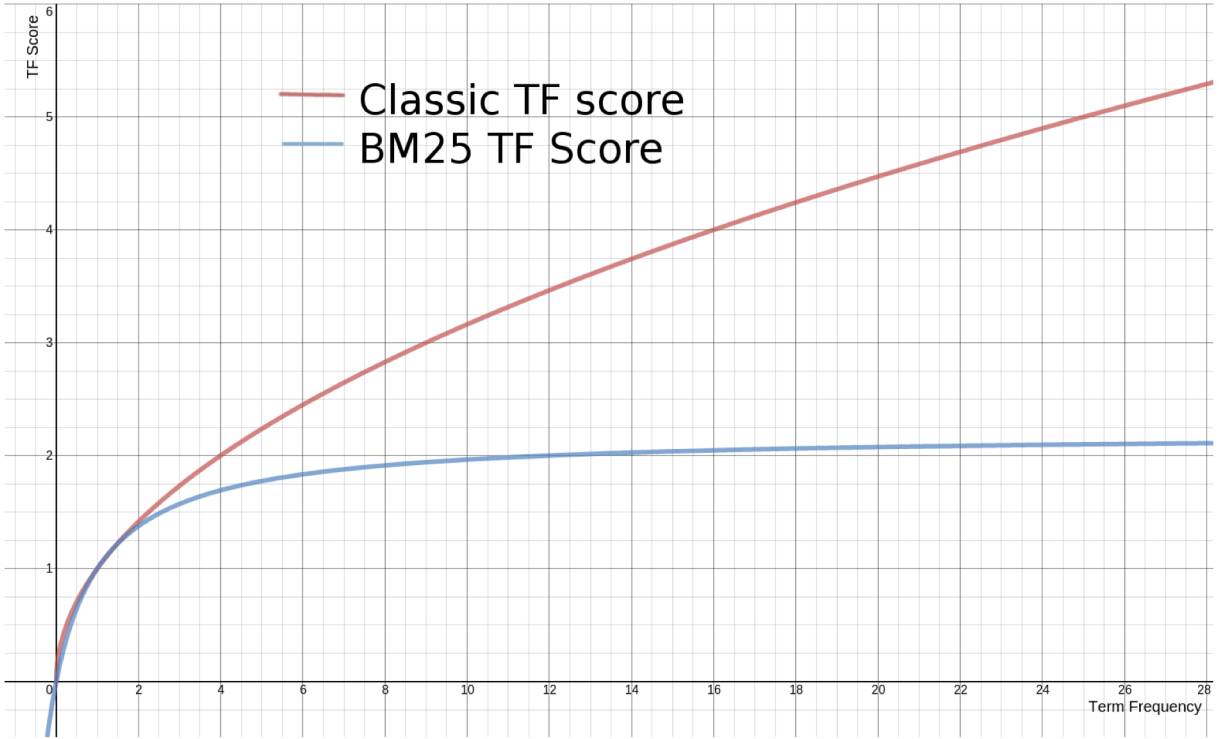
Scoring

Elasticsearch scores documents based on how closely they match the query

- The _score is computed for each document that is a hit
- The default scoring algorithm is **BM25**
- There are three factors of a document's score
  - **TF** (term frequency), The more a term appears in a field, the more important it is
  - **DF** (inverse document frequency), the more documents that contain the term, the less important the term is
  - **Field length**, shorter fields are more likely to be relevant than longer fields.

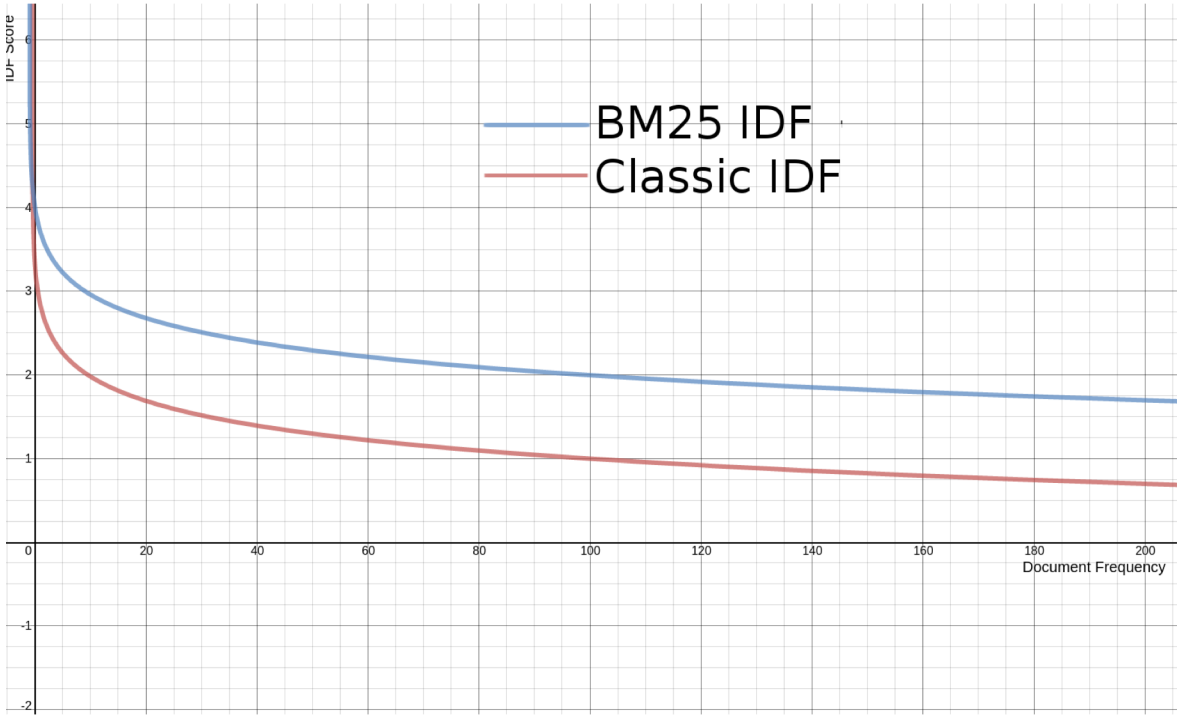
Scoring

Term Frequency



Scoring

Inverse Document Frequency



Scoring, the boost parameter

- The bool query allows a “boost” parameter to increase (or decrease) the relative weight of a clause
- If boost > 1
  - The relative weight of the clause is **increased**
- If 0< boost < 1
  - The relative weight of the clause is **decreased**
- If boost < 0
  - The clause will contribute negatively to the score, used if user do not want result match the clause

Scoring, the boost parameter

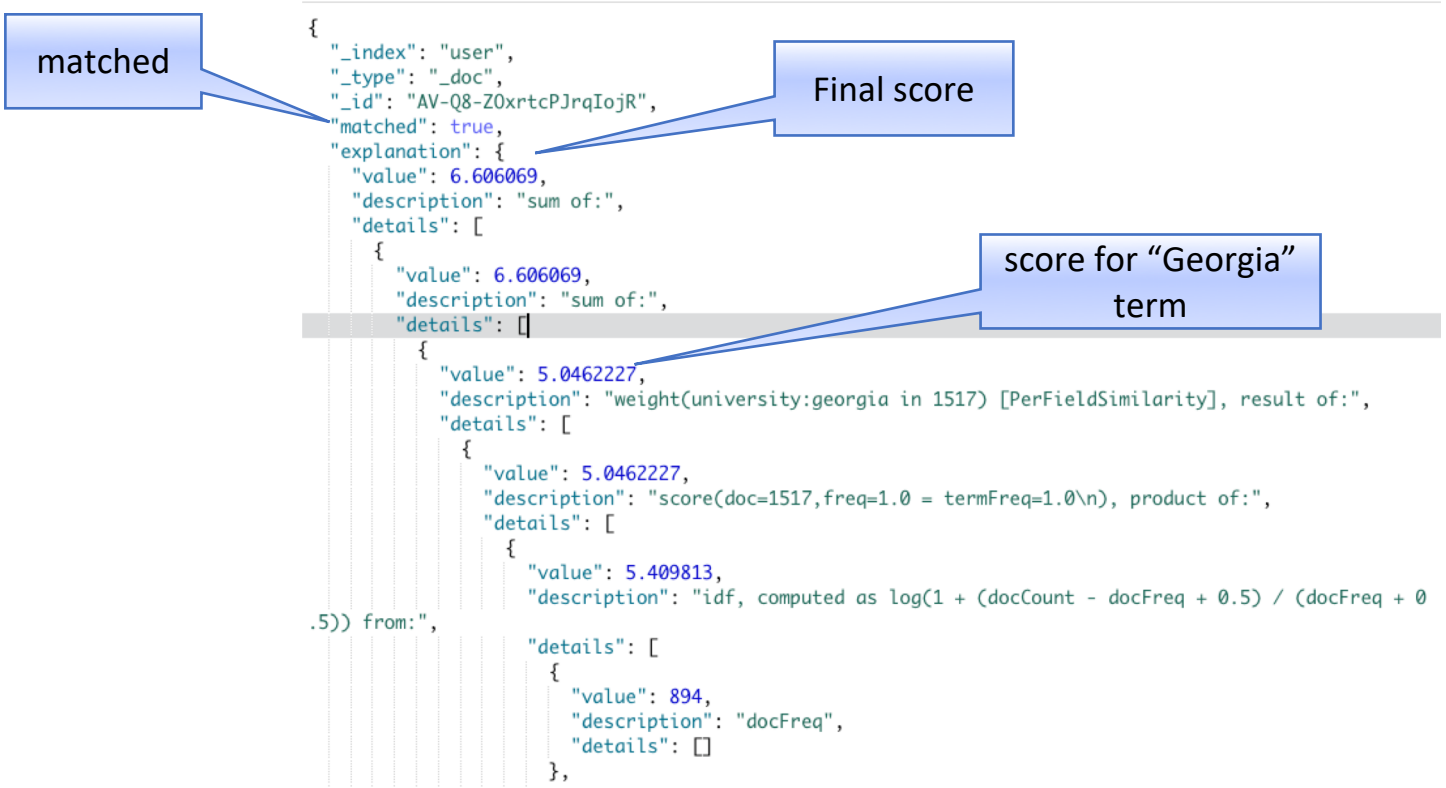
- Use negative boost parameter to simulate the “should_not” query

```
POST /user/_search
{
  "query": {
    "bool": {
      "must": [
        {"match": {"university": "Georgia"}},
        {"match_phrase": {"city": "Sar Meel"}}
      ],
      "should":{
        "match": {
          "name": {
            "query": "jack",
            "boost": -1.0
          }
        }
      }
    }
  },
  "sort":{"_score":"asc"}
}
```

Explain

- Why some documents are hit or not hit?
- Why some documents get higher score than others?
- Explain API can give you an explanation for a query and a specific document.

Explain


matched
Final score
score for “Georgia” term
{
"_index": "user",
"_type": "_doc",
"_id": "AV-Q8-ZOxrtcPJrqIojR",
"matched": true,
"explanation": {
"value": 6.606069,
"description": "sum of:",
"details": [
{
"value": 6.606069,
"description": "sum of:",
"details": [
{
"value": 5.0462227,
"description": "weight(university:georgia in 1517) [PerFieldSimilarity], result of:",
"details": [
{
"value": 5.0462227,
"description": "score(doc=1517,freq=1.0 = termFreq=1.0\n), product of:",
"details": [
{
"value": 5.409813,
"description": "idf, computed as log(1 + (docCount - docFreq + 0.5) / (docFreq + 0
.5)) from:",
"details": [
{
"value": 894,
"description": "docFreq",
"details": []
},

Quiz

1. **True or False:** Bool query can be nested in another bool query.
2. Bool query can use _____ parameter to increase or decrease document scores.
3. Elasticsearch use _____ algorithm to calculate document scores.
4. What does **TF** and **IDF** parameter means?
5. Run a query on user index, return users whose “university” field contains “Hampshire”, and city fields contains “Myr”, and “university” field should not contains “College”

Agenda

- Search fundamental
- Aggregation
- Mapping
- Text analysis
- Advanced search

2

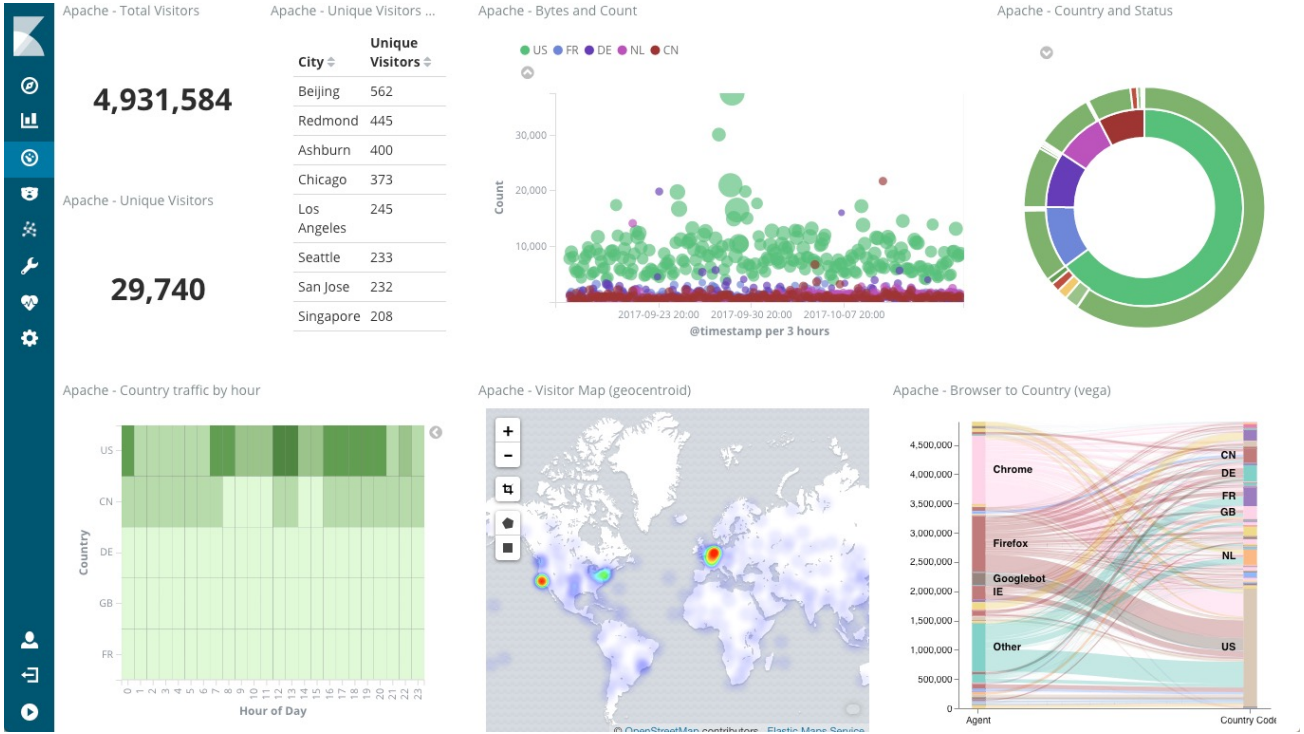
Aggregation

Aggregation

- Aggregations are a way to perform analytics on your data
- Aggregation vs Search
  - In search, we always ask for a subset of original dataset
    - Whose name is jack and live in Myr city?
  - In aggregations, we always analyze the data by asking questions about the dataset
    - How many cities our user come from and what's the top 10 cities?
    - What's the average age of our users?
- In eBay, we may ask
  - How many users access our website in last 5 minutes?
  - Which production get the most visitors in last week?

Aggregation

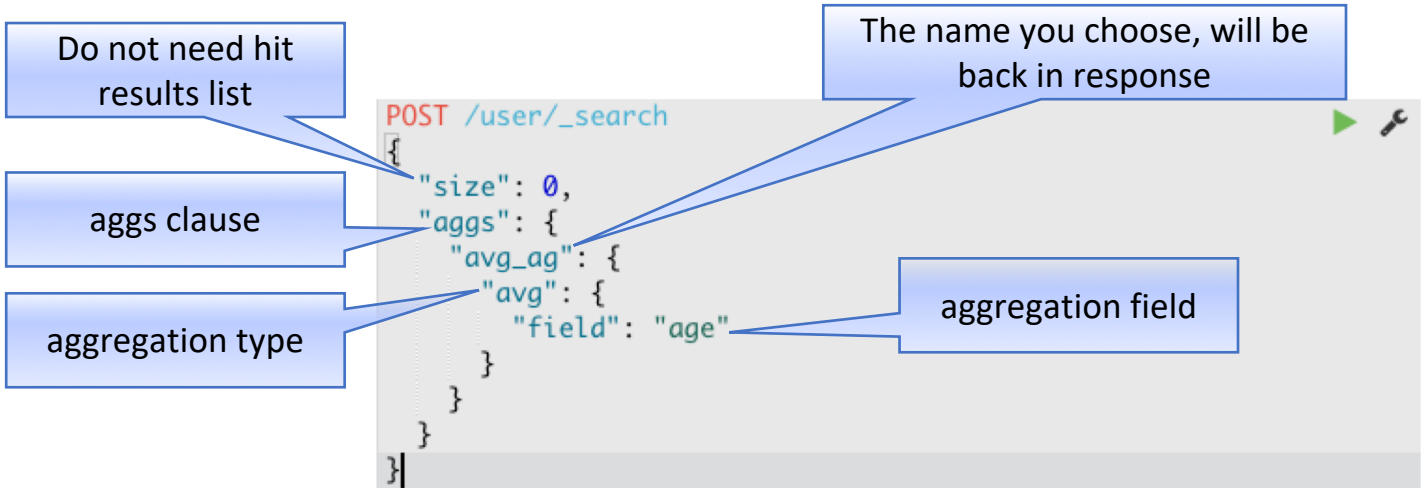
Most of Kibana visualization are aggregations



Aggregation syntax

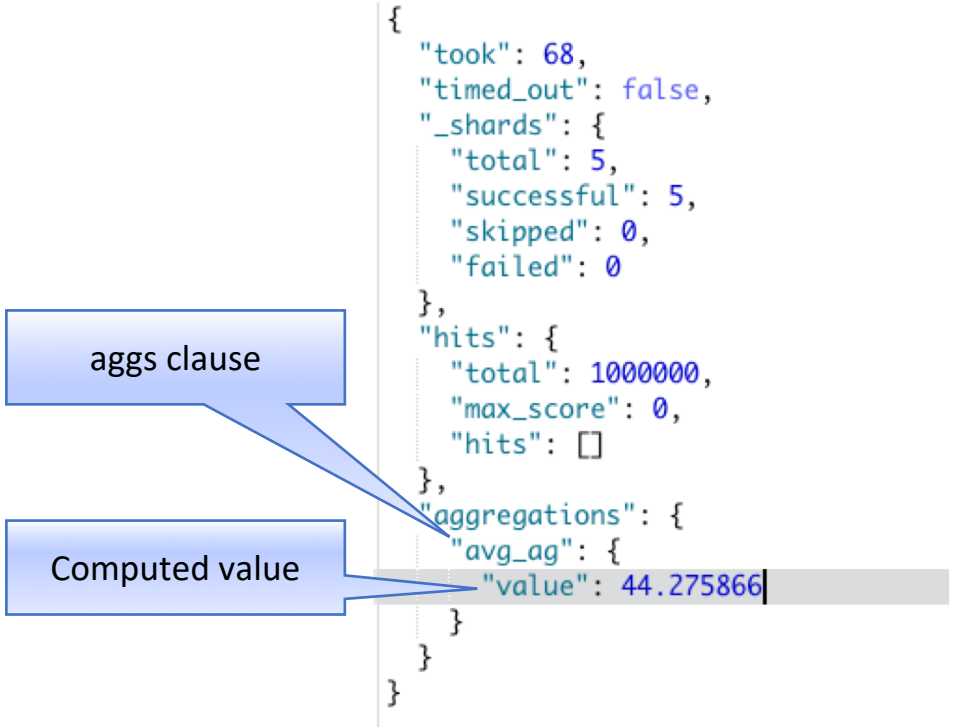
An aggregation request is a part of the search API

- The query clause defines the subset of data which we will run aggregate on



Aggregation syntax

- Aggregation result will be returned in the “aggregation” field
- The name ”avg_ag” is chosen by API caller and returned
- The average age is computed and returned



Aggregation

Four types of aggregations:

- Bucket: aggregations that combine documents that meet a given criteria into buckets
    - Bucket aggregations build buckets
- Metric: mathematical calculations performed on the fields of documents
    - Typically, metrics are calculated on buckets of data
- Matrix: aggregation that operates on multiple fields and produces a matrix result
- Pipeline: aggregate the output of other aggregations

Metrics aggregations

Avg Metric:

- The avg metric aggregation computes the average of a specified numeric field

```
POST /user/_search
{
  "size": 0,
  "query": {"match": {"city": "Myr"}},
  "aggs": {
    "avg_ag": {
      "avg": {
        "field": "age"
      }
    }
  }
}
```

```
{
  "took": 10,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 28060,
    "max_score": 0,
    "hits": []
  },
  "aggregations": {
    "avg_ag": {
      "value": 44.97013542409123
    }
  }
}
```

Metrics aggregations

Min and Max Metric:

- We can have multiple aggregations in one request

```
POST /user/_search
{
  "size": 0,
  "query": {"match": {"city": "Myr"}},
  "aggs": {
    "min_age": {"min": {"field": "age"}},
    "max_age": {"max": {"field": "age"}}
  }
}
```

```
  "aggregations": {
    "max_age": {
      "value": 89
    },
    "min_age": {
      "value": 0
    }
  }
}
```

Metrics aggregations

Stats Metric:

- Include count, min, max, avg and sum

```
POST /user/_search
{
  "size": 0,
  "query": {"match": {"city": "Myr"}},
  "aggs": {
    "stats_age": {"stats": {"field": "age"}}
  }
}
```

```
  "aggregations": {
    "stats_age": {
      "count": 28060,
      "min": 0,
      "max": 89,
      "avg": 44.97013542409123,
      "sum": 1261862
    }
  }
```

Metrics aggregations

Cardinality Metric:

- The cardinality metric aggregation is an approximation of the number of distinct values of a field

```
POST /user/_search
{
  "size": 0,
  "aggs": {
    "city_cardinality": {
      "cardinality": {
        "field": "city.keyword"
      }
    }
  }
}
```

```
  "aggregations": {
    "city_cardinality": {
      "value": 36
    }
  }
```

Metrics aggregations

Percentiles Metric:

- Percentiles show the point at which a certain percentage of observed values occur.
- For example, the 95th percentile is the value which is greater than 95% of the observed values.

```
POST /user/_search
{
  "size": 0,
  "aggs": {
    "age_percentiles": {
      "percentiles": {
        "field": "age"
      }
    }
  }
}
```

```
"aggregations": {
  "age_percentiles": {
    "values": {
      "1.0": 0,
      "5.0": 4,
      "25.0": 22,
      "50.0": 44.12485035060715,
      "75.0": 66.99999999999999,
      "95.0": 85,
      "99.0": 88
    }
  }
}
```

Bucket

- A bucket is simply a collection of documents that meet a criterion:
  - Buckets are a key element of aggregations
- For example, we want to analyze users by their age. We could put them into specific buckets like below
  - age < 25
  - 25 <= age < 60
  - age > 60
- Bucket can be nested. Suppose we want to analyze user by age and by city
  - age < 25
    - City == “Myr”

Bucket aggregations

Range aggregation

- Range bucket aggregation puts documents into buckets based on ranges of values

```
POST /user/_search
{
  "size": 0,
  "aggs": {
    "avg_range_bucket": {
      "range": {
        "field": "age",
        "ranges": [
          {"to": 25},
          {"from": 25, "to": 60},
          {"from": 60}
        ]
      }
    }
  }
}
```

```
"aggregations": {
  "avg_range_bucket": {
    "buckets": [
      {
        "key": "*-25.0",
        "to": 25,
        "doc_count": 279785
      },
      {
        "key": "25.0-60.0",
        "from": 25,
        "to": 60,
        "doc_count": 389974
      },
      {
        "key": "60.0-*",
        "from": 60,
        "doc_count": 330241
      }
    ]
  }
}
```

Bucket aggregations

Labeling Ranges

- Add a “key” field to label a range

```
POST /user/_search
{
  "size": 0,
  "aggs": {
    "avg_range_bucket": {
      "range": {
        "field": "age",
        "ranges": [
          {"key": "young","to": 25},
          {"key": "middle age","from": 25, "to": 60},
          {"key": "aged","from": 60}
        ]
      }
    }
  }
}
```

```
"aggregations": {
  "avg_range_bucket": {
    "buckets": [
      {
        "key": "young",
        "to": 25,
        "doc_count": 279785
      },
      {
        "key": "middle age",
        "from": 25,
        "to": 60,
        "doc_count": 389974
      },
      {
        "key": "aged",
        "from": 60,
        "doc_count": 330241
      }
    ]
  }
}
```

Bucket aggregations

Combine Bucket and Metrics

- Let's perform an average aggregation on each bucket

```
POST /user/_search
{
  "size": 0,
  "aggs": {
    "avg_range_bucket": {
      "range": {
        "field": "age",
        "ranges": [
          {"key": "young","to": 25},
          {"key": "middle age","from": 25, "to": 60},
          {"key": "aged","from": 60}
        ]
      },
      "aggs": {
        "avg_ag": {"avg": {"field": "age"}}
      }
    }
  }
}
```


```
"aggregations": {
  "avg_range_bucket": {
    "buckets": [
      {
        "key": "young",
        "to": 25,
        "doc_count": 279785,
        "avg_ag": {
          "value": 12.024708258126775
        }
      },
      {
        "key": "middle age",
        "from": 25,
        "to": 60,
        "doc_count": 389974,
        "avg_ag": {
          "value": 42.00876981542359
        }
      },
      {
        "key": "aged",
        "from": 60,
        "doc_count": 330241,
        "avg_ag": {
          "value": 74.27667975811605
        }
      }
    ]
  }
}
```

Date range aggregations

- Date range bucket aggregation is similar to range aggregation except it is used for **date** fields

```
POST /user/_search
{
  "size": 0,
  "aggs": {
    "avg_range_bucket": {
      "date_range": {
        "field": "register_at",
        "ranges": [
          {"to": "2017-01-01"},
          {"from": "2017-01-01", "to": "2017-06-01"},
          {"from": "2017-06-01"}
        ]
      }
    }
  }
}
```

```
"aggregations": {
  "avg_range_bucket": {
    "buckets": [
      {
        "key": "*-2017-01-01T00:00:00.000Z",
        "to": 1483228800000,
        "to_as_string": "2017-01-01T00:00:00.000Z",
        "doc_count": 151207
      },
      {
        "key": "2017-01-01T00:00:00.000Z-2017-06-01T00:00:00.000Z",
        "from": 1483228800000,
        "from_as_string": "2017-01-01T00:00:00.000Z",
        "to": 1496275200000,
        "to_as_string": "2017-06-01T00:00:00.000Z",
        "doc_count": 413885
      },
      {
        "key": "2017-06-01T00:00:00.000Z-*",
        "from": 1496275200000,
        "from_as_string": "2017-06-01T00:00:00.000Z",
        "doc_count": 434908
      }
    ]
  }
}
```

Bucket aggregations

Terms aggregation

- Terms aggregation will create bucket for every unique term it encounters of the specified field

```
POST /user/_search
{
  "size": 0,
  "aggs": {
    "city_terms_bucket": {
      "terms": {
        "field": "city.keyword",
        "size": 30
      }
    }
  }
}
```

```
"aggregations": {
  "city_terms_bucket": {
    "doc_count_error_upper_bound": 0,
    "sum_other_doc_count": 165164,
    "buckets": [
      {
        "key": "Asshai",
        "doc_count": 28080
      },
      {
        "key": "Myr",
        "doc_count": 28060
      },
      {
        "key": "Tyria",
        "doc_count": 28043
      },
```

Bucket aggregations

Nested bucket aggregation

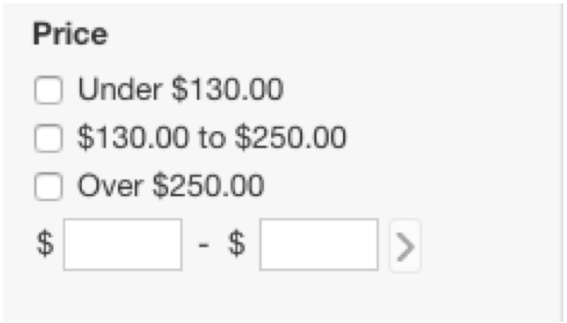
- We can define a range aggregation in the terms aggregation

```
POST /user/_search
{
  "size": 0,
  "aggs": {
    "city_terms_bucket": {
      "terms": {
        "field": "city.keyword",
        "size": 30
      },
      "aggs": {
        "age_range_bucket": {
          "range": {
            "field": "register_at",
            "ranges": [
              {"to": "2017-01-01"},
              {"from": "2017-01-01", "to": "2017-06-01"},
              {"from": "2017-06-01"}
            ]
          }
        }
      }
    }
  }
}
```

```
"aggregations": {
  "city_terms_bucket": {
    "doc_count_error_upper_bound": 0,
    "sum_other_doc_count": 165164,
    "buckets": [
      {
        "key": "Asshai",
        "doc_count": 28080,
        "age_range_bucket": {
          "buckets": [
            {
              "key": "*-2017-01-01T00:00:00.000Z",
              "to": 1483228800000,
              "to_as_string": "2017-01-01T00:00:00.000Z",
              "doc_count": 4220
            },
            {
              "key": "2017-01-01T00:00:00.000Z-2017-06-01T00:00:00.000Z",
              "from": 1483228800000,
              "from_as_string": "2017-01-01T00:00:00.000Z",
              "to": 1496275200000,
              "to_as_string": "2017-06-01T00:00:00.000Z",
              "doc_count": 11689
            },
```

Quiz

1. These pictures are got from eBay website, which aggregation should we use for them?



Quiz

1. **True or False:** Metric aggregation include avg, min, max, stats and range.
2. **Query or Aggregation:** “What are the nearby Chinese restaurants?”
3. **Query or Aggregation:** “How many error happened in last 5 minutes?”
4. If we want to put log events into buckets by its type (“error”, “warn”, “info”, “debug”), which aggregate should we use?
5. If we want to get the answer for this question “How many unique visitors came to our website today?”, which aggregation should we use? Further more, if we want to get first visit timestamp and last visit timestamp for every visitor, which aggregation should we use and where it should be
6. Run a query on user index, get average user age for every company.

Agenda

- Search fundamental
- Aggregation
- Mapping
- Text analysis
- Advanced search

3

Mapping

Mapping

- Let’s take a look on another index user2, the user2 index have the same documents with user index.
- Queries on name/city fields work well, but the IP and geo_distance queries do not work!!!

Mapping

- Example of mapping

Get /user/_mappings

```
{
  "user": {
    "mappings": {
      "_doc": {
        "properties": {
          "address": {
            "type": "text",
            "fields": {⟺}
          },
          "age": {
            "type": "long"
          },
          "birthDay": {
            "type": "date"
          },
          "city": {
            "type": "text",
            "fields": {⟺}
          },
```

Mapping

- Difference between the mappings of user and user2

user

```
"ip": {
  "type": "ip"
},
"ipV6": {
  "type": "ip"
},
"location": {
  "type": "geo_point"
},
"longVal": {
  "type": "long"
},
```

user2

```
"ip": {
  "type": "text",
  "fields": {
    "keyword": {
      "type": "keyword",
      "ignore_above": 256
    }
  }
},
"ipV6": {
  "type": "text",
  "fields": {
    "keyword": {
      "type": "keyword",
      "ignore_above": 256
    }
  }
},
"location": {
  "properties": {
    "lat": {
      "type": "float"
    },
    "lon": {
      "type": "float"
    }
  }
},
```

Mapping

- Mapping is the process of defining how a document and the fields it contains, are stored and indexed. For instance, use mappings to define:
  - Which string fields should be treated as full text fields
  - Which fields contain numbers, dates, or geolocations
  - Whether the values of all fields in the document should be indexed into the catch-all _all field
  - The format of date values
  - Custom rules to control the mapping for dynamically added fields
- Mappings belongs to index types before Elasticsearch 6, after that it belongs to index since only one type for one index

Mapping

Defining a mapping

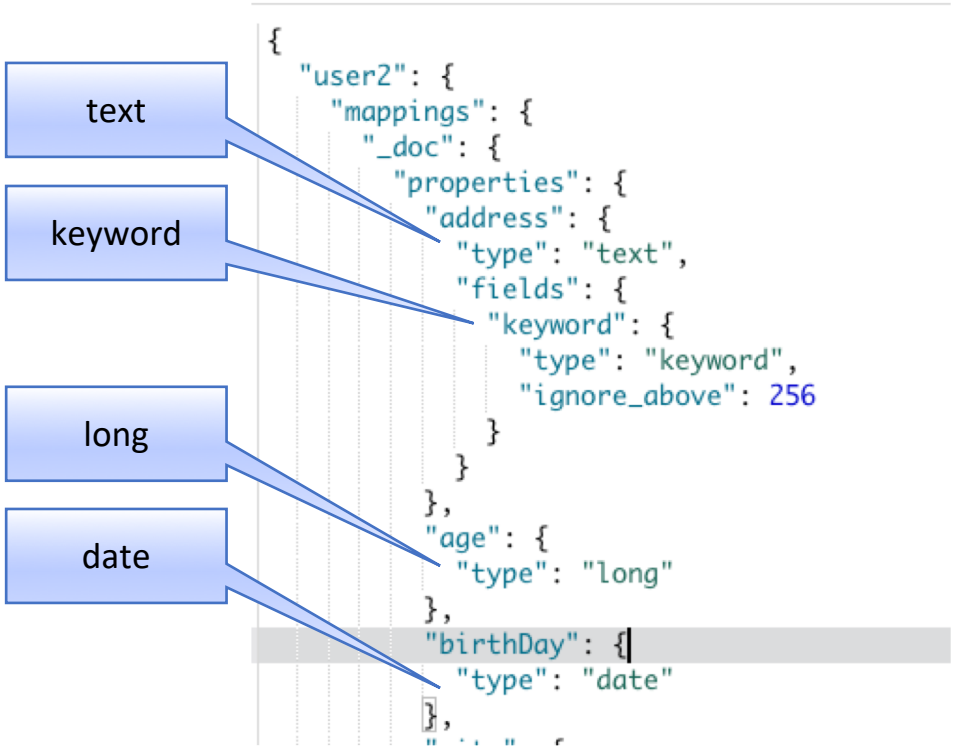
- Define mappings at index creation, or
- Update a mapping of an existing index using the Put API
- Notice, existing field mappings cannot be updated!!!

```
PUT /user3
{
  "mappings": {
    "_doc": {
      "properties": {
        "name": {"type": "text"}
      }
    }
  }
}
```

Dynamic mapping

We did not define any mapping for user2, but it really has mapping, **WHY**???

- You are not required to manually define mappings
- When a document with new field is index to an index, Elasticsearch dynamically updates the mapping based on the document.



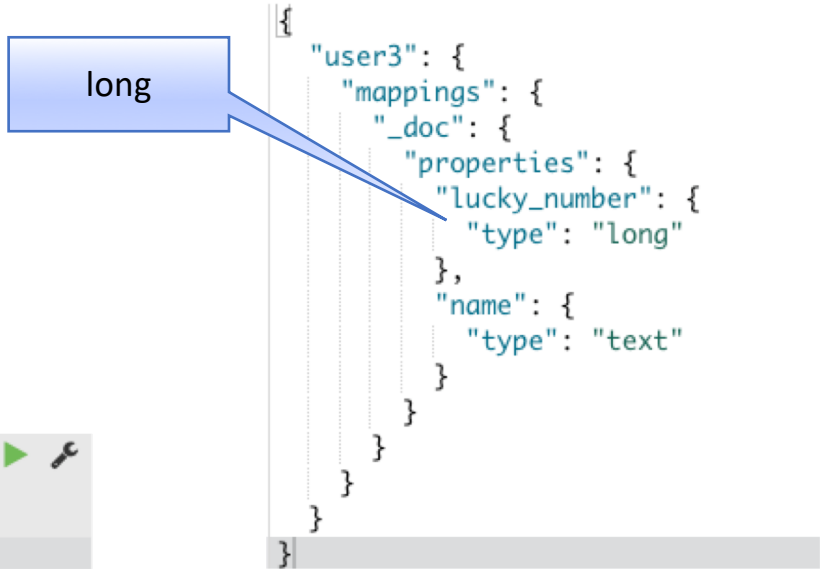
Data types

- Simple types
  - **text**: for full text (analyzed) strings
  - **keyword**: for exact value strings
  - **date**: string formatted as dates, or numeric dates
  - integer types: like **byte**, **short**, **integer**, **long**
  - floating-point numbers: **float**, **double**, **half_float**, **scaled_float**
  - **Boolean**
  - **ip**: for IPv4 or IPv6
- Hierarchical types: **object** and **nested**
- Specialized types: **geo_point**, **geo_shape** and **percolator**

Data types

Array fields

- There is no dedicated array type
  - But any field can contain multiple values



Data types

Date formats

- Use the format setting to specify the format of your date fields
  - format use either a date string like “MM/dd/YYYY” or
  - One of the built-in date formats
- Default format value is **strict_date_optional_time || epoch_millis**

```
PUT /user3/_mapping/_doc
{
  "properties": {
    "birth_day": {
      "type": "date",
      "format": "dd/MM/YYYY || epoch_millis"
    }
  }
}
```

Mapping parameters

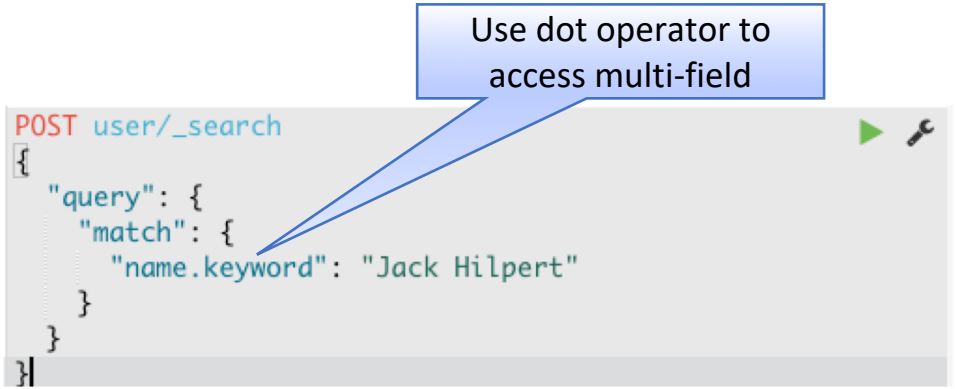
Elasticsearch allow to set mapping parameters, some frequently used parameters are lists below:

- **enabled**: Elasticsearch will try to index all of the fields except you set disabled to false explicitly. It's useful if you do not need to query or run aggregation on it.
- **format**: We've see it in Date type
- **ignore_above**: Strings longer than the ignore_above will not be indexed or stored
- **null_value**: When a field is set to null, its value will replaced by the specified value.
- **analyzer**: Very important parameter for string field, will introduce later.

Multi-Fields

- It's often useful to index the same field in different ways.
  - The default mapping type for strings is the best example.
- You can specify type and parameters for each fields, which map to the same original field
- Use dot operator to refer to a multi-field

```
"name": {
  "type": "text",
  "fields": {
    "keyword": {
      "type": "keyword",
      "ignore_above": 256
    }
  }
},
```



Index templates

- Index template allow user to define mapping and settings that will automatically be applied to **new-created** indices.
  - You can have multiple index templates
  - Templates get “merged” and applied to the created index
  - Use “order” value to control the merging process
- When a new index is created:
  - The default setting and mappings are applied
  - Apply templates, the lowest “order” template is applied first
  - Apply the setting/mapping in the payload which used in Create Index request
  - The subsequent setting/mapping will override previous templates

Add/View/Delete a template

- Use the _template endpoint to add, view and delete templates

```
PUT _template/my_template
{
  "index_patterns": "*",
  "order": 1,
  "settings": {
    "number_of_shards": 2,
    "number_of_replicas": 1
  },
  "mappings": {
    "_doc": {
      "properties": {
        "template_field": {"type": "keyword"}
      }
    }
  }
}
GET _template/my_template
DELETE _template/my_template
```

This template will be applied to all new index

This template will be applied before other templates with higher order

Quiz

1. **True or False:** Mapping only can be set when create index.
2. **True or False:** Mapping cannot be updated once it is created.
3. **True or False:** We can change field type from “short” to “long” since it’s compatible.
4. What data type would **“age”: 25** get mapped to dynamic?
5. What would happen if you index a document with “**age”: “undefined”** to user index with type “_doc”?
6. We have two index templates with “index_patterns”: “*”, the template 1 has order value **100** and it set “age” field mapping to ”**text**”, the template 2 has order value **200** and it set “age” field mapping to “**long**”, then we create a new index and set its mapping to “**short**” in create index request payload, and then we index a document with **“age”: 25**, what’s the final mapping?

Agenda

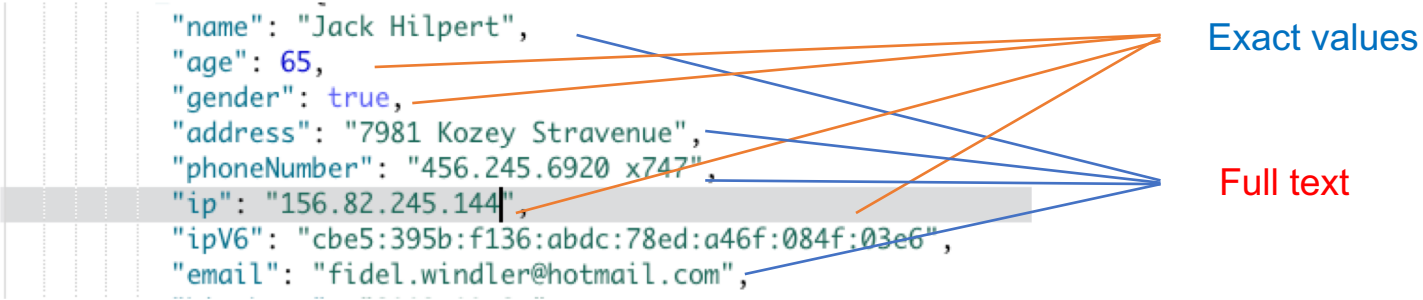
- Search fundamental
- Aggregation
- Mapping
- Text analysis
- Advanced search

4

Text Analysis

Text analysis

- In general, data in Elasticsearch can be categorized into two groups:
  - Exact values: includes numeric values, dates, and certain strings
  - Full text: unstructured text data that we want to search



- What's the difference if a value is exact or full text?

Full text is analyzed

- Analysis is the process of converting full text into terms for the inverted index
- Analysis is performed by an analyzer
  - Either a built-in analyzer
  - Or a custom analyzer you configure

Default analyzer

```
POST _analyze
{
  "analyzer": "standard",
  "text": "The 2 QUICK Brown-Foxes jumped over the lazy dog's bone."
}
```

```
{
  "tokens": [
    {
      "token": "the",
      "start_offset": 0,
      "end_offset": 3,
      "type": "<ALPHANUM>",
      "position": 0
    },
    {
      "token": "2",
      "start_offset": 4,
      "end_offset": 5,
      "type": "<NUM>",
      "position": 1
    },
    {
      "token": "quick",
      "start_offset": 6,
      "end_offset": 11,
      "type": "<ALPHANUM>",
      "position": 2
    },
```

Full text is analyzed

- Full text gets analyzed during ingestion
- Similarly, the text in your queries get analyzed using the same analyzer Either a built-in analyzer

```
POST _analyze
{
  "analyzer": "standard",
  "text": "Jack Hilpert"
}
```

```
{
  "tokens": [
    {
      "token": "jack",
      "start_offset": 0,
      "end_offset": 4,
      "type": "<ALPHANUM>",
      "position": 0
    },
    {
      "token": "hilpert",
      "start_offset": 5,
      "end_offset": 12,
      "type": "<ALPHANUM>",
      "position": 1
    }
  ]
}
```

Inverted index

- Lucene create inverted index with your documents
  - Text is broken into tokens
  - Indexed by a unique document id

1	The quick brown fox jumps over the lazy dog
2	Fast jumping spiders

token	postings
Fast	2
The	1
brown	1
fox	1
dog	1
fox	1
jumping	2
jumps	1
lazy	1
over	1
spiders	2
quick	1
the	1

Lowercase

- User might want to ignore the case, search “Fast” should return hit documents which contain “fast”
- Cast to Lowercase for all tokens

1	The quick brown fox jumps over the lazy dog
2	Fast jumping spiders

token	postings
fast	2
brown	1
fox	1
dog	1
fox	1
jumping	2
jumps	1
lazy	1
over	1
spiders	2
quick	1
the	1

Stopwords

- Search “the” rarely make sense
  - Do not build inverted index for stopwords like “a”/”the”/”is”

token	postings
fast	2
brown	1
fox	1
dog	1
fox	1
jumping	2
jumps	1
lazy	1
over	1
spiders	2
quick	1

1 The quick brown fox jumps over the lazy dog

2 Fast jumping spiders

Stemming

- “jumping” and “jumps” share the same stem: “jump”
- Only index the stem

1	The quick brown fox jumps over the lazy dog
2	Fast jumping spiders

token	postings
fast	2
brown	1
fox	1
dog	1
fox	1
jump	1,2
lazy	1
over	1
spider	2
quick	1

Synonyms

- “quick” and “fast” have similar meaning
  - Should a search for “quick” return doc 2
  - Should a search for “fast” return doc 1

1	The quick brown fox jumps over the lazy dog
2	Fast jumping spiders

token	postings
fast	1,2
brown	1
fox	1
dog	1
fox	1
jump	1,2
lazy	1
over	1
spider	2
quick	1,2

The query should analyzer too

- Search “Dog”, equals to “dog”, return document 1
- Search “jumped”, equals to “jump”, return document 1 and 2
- Search “the quick spiders” in match query equals to “quick OR spider”, return document 1 and 2

1	The quick brown fox jumps over the lazy dog
2	Fast jumping spiders

token	postings
fast	1,2
brown	1
fox	1
dog	1
fox	1
jump	1,2
lazy	1
over	1
spider	2
quick	1,2

Pre-built analyzers

- standard
- simple
- whitespace
- stop
- keyword
- pattern
- language

Anatomy of an analyzer

An analyzer consists of three parts:

- Character Filters
- Tokenizer
- Token Filters

Simple analyzer

- Simple analyzer breaks text into terms whenever it encounters a character which is not a letter. All terms are lower cased.
- Can you find the difference between “simple” and “standard” analyzer?

simple analyzer

```
POST _analyze
{
  "analyzer": "simple",
  "text": "The 2 QUICK Brown-Foxes jumped over the lazy dog's bone."
}
```


```
{
  "tokens": [
    {
      "token": "the",
      "start_offset": 0,
      "end_offset": 3,
      "type": "word",
      "position": 0
    },
    {
      "token": "quick",
      "start_offset": 6,
      "end_offset": 11,
      "type": "word",
      "position": 1
    },
    {
      "token": "brown",
      "start_offset": 12,
      "end_offset": 17,
      "type": "word",
      "position": 2
    },
```

Stop analyzer

- Similar to simple analyzer.
- remove stop words, which in English are: ***a, an, and, are, as, at, be, but, by, for, if, in, into, is, it, no, not, of, on, or, such, that, the, their, then, there, these, they, this, to, was, will, with***

stop analyzer

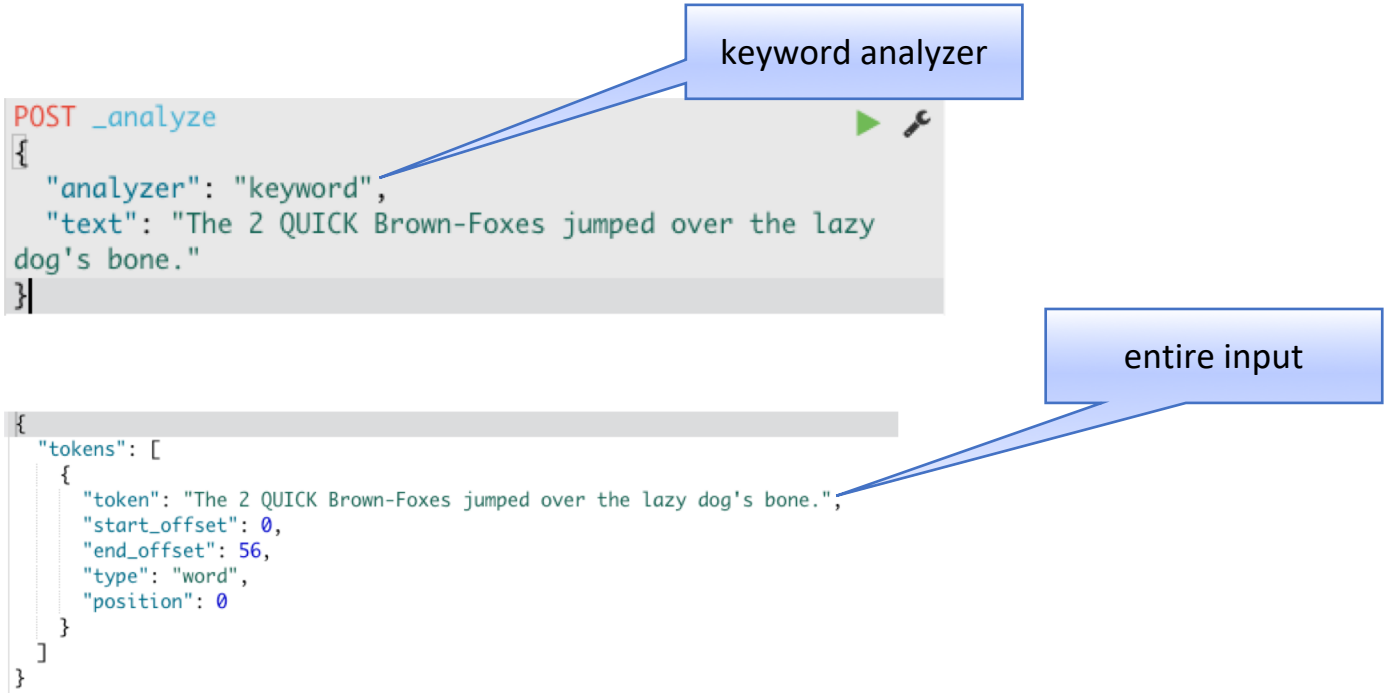
```
POST _analyze
{
  "analyzer": "stop",
  "text": "The 2 QUICK Brown-Foxes jumped over the lazy dog's bone."
}
```

“the” disappeared


```
{
  "tokens": [
    {
      "token": "quick",
      "start_offset": 6,
      "end_offset": 11,
      "type": "word",
      "position": 1
    },
    {
      "token": "brown",
      "start_offset": 12,
      "end_offset": 17,
      "type": "word",
      "position": 2
    },
    {
      "token": "foxes",
      "start_offset": 18,
      "end_offset": 23,
      "type": "word",
      "position": 3
    },
```

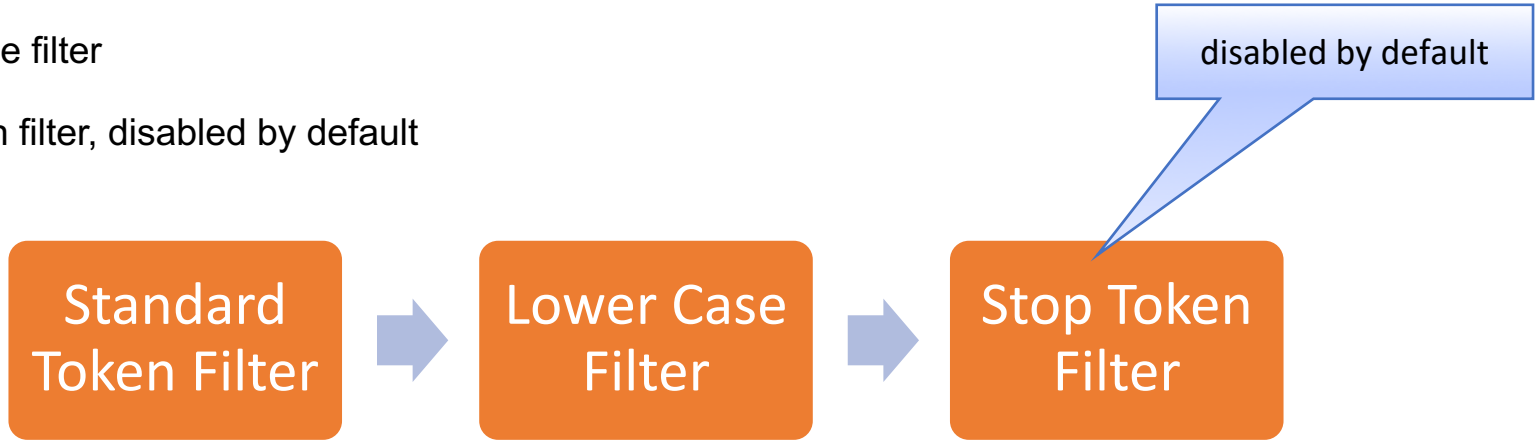
Keyword analyzer

- Keyword analyze just return the entire input string as a single token.



Standard analyzer

- The default analyzer.
- Standard token filter, provides grammar based tokenization (based on the Unicode Text Segmentation algorithm) and works well for most languages
- Lower case filter
- Stop token filter, disabled by default



Language analyzer

- A set of analyzer aimed at analyzing specific language text.
  - 30 + languages supported

French analyzer

```
POST _analyze
{
  "analyzer": "french",
  "text": "Le renard brun rapide saute par-dessus le chien paresseux"
}
```


```
{
  "tokens": [
    {
      "token": "renard",
      "start_offset": 3,
      "end_offset": 9,
      "type": "<ALPHANUM>",
      "position": 1
    },
    {
      "token": "brun",
      "start_offset": 10,
      "end_offset": 14,
      "type": "<ALPHANUM>",
      "position": 2
    },
    {
      "token": "rapid",
      "start_offset": 15,
      "end_offset": 21,
      "type": "<ALPHANUM>",
      "position": 3
    },
```

Customize analyzer

- A set of analyzer aimed at analyzing specific language text.
  - Have a try to see the difference between customized analyzer and standard analyzer

```
PUT test_index
{
  "settings": {
    "analysis": {
      "analyzer": {
        "my_analyzer": {
          "type": "custom",
          "char_filter":["html_strip"],
          "tokenizer": "standard",
          "filter" : ["lowercase", "snowball"]
        }
      }
    }
  }
}
```

analyzer name

analyzer type

strip html tags

customized analyzer

```
POST test_index/_analyze
{
  "analyzer": "my_analyzer",
  "text": "The 2 QUICK <em>Brown-Foxes</em> jumped over the lazy dog's bone."
}
```

Quiz

1. _____ is the process of converting full text into terms for the inverted index.
2. Why does it matter if a field is an exact value vs. full text?
3. What are the three components that make up an analyzer?
4. How many unique tokens we will get if use standard analyzer for text “Peter's question is 'how a fly flies?’” What about use stop analyzer? keyword analyzer? standard analyzer + stop analyzer? standard analyzer + snowball analyzer?

Quiz

1. _____ is the process of converting full text into terms for the inverted index.
2. Why does it matter if a field is an exact value vs. full text?
3. What are the three components that make up an analyzer?
4. How many unique tokens we will get if use standard analyzer for text “Peter's question is 'how a fly flies?’” What about use stop analyzer? keyword analyzer? standard analyzer + stop analyzer? standard analyzer + snowball analyzer?

Agenda

- Search fundamental
- Aggregation
- Mapping
- Text analysis
- Advanced search

5

Advanced Search

Scroll

- What if you want more than a ”page” of results?
    - Like retrieving all user information document from our user index

- Scroll API allows you to take a snapshot of large number of results from a single search request
    - Notice the word “snapshot”

- Scroll searches are a 4-step process:

Scroll

1. Initiate the scroll

- To initiate a scroll search, add the scroll parameter to yo[illegible]y
  - specify how long Elasticsearch should keep th[illegible]

initiate scroll, the context will be invalid after 5 minutes

```
POST /user/_search?scroll=5m
{
  "query": {"match_all": {}}
}
```

scroll id


```
{
  "_scroll_id": "DnF1ZXJ5VGhlbkZldGNoBQAAAAAAAAKsFmRnZzUtb
nBtVGFpU0VqV1pnM21BelEAAAAAAAAABGBZodmtQSnVwbFJYeXpjTWZPTnF
PdTVnAAAAAAAAARcWaHZrUEp1cGxSWHl6Y01mT05xT3U1ZwAAAAAAAACLF
nJma1dZVlFrUlU2c1NtaE5BZEhPTEEAAAAAAAACrRZkZ2c1LW5wbVRhaVN
FaldaZzNtQXpR",
  "took": 11,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 1000000,
    "max_score": 1,
    "hits": [
```

Scroll

1. Initiate the scroll

- To initiate a scroll search, add the scroll parameter to your search query
- Specify how long Elasticsearch should keep the search context

initiate scroll, the context will be invalid after 5 minutes

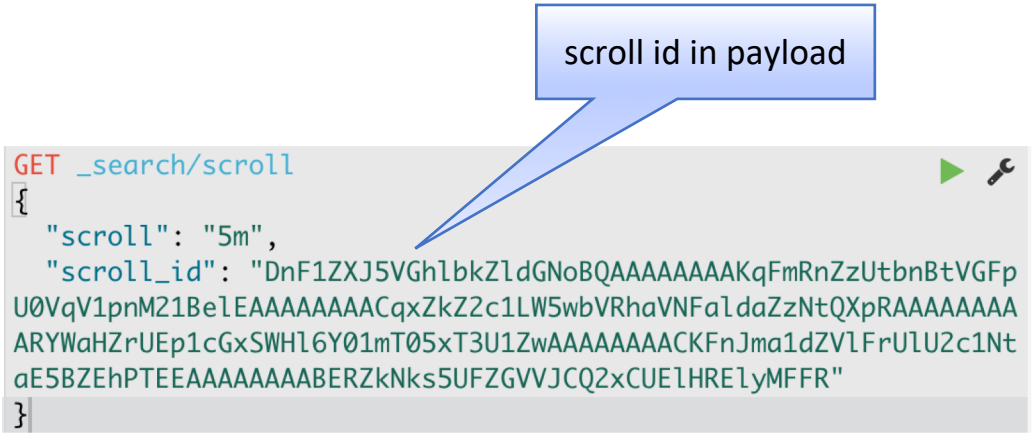
```
POST /user/_search?scroll=5m
{
  "query": {"match_all": {}}
}
```

scroll id


```
{
  "_scroll_id": "DnF1ZXJ5VGhlbkZldGNoBQAAAAAAAAKsFmRnZzUtb
nBtVGFpU0VqV1pnM21BelEAAAAAAAABGBZodmtQSnVwbFJYeXpjTWZPTnF
PdTVnAAAAAAAAARcWaHZrUEp1cGxSWHl6Y01mT05xT3U1ZwAAAAAAAACLF
nJma1dZVlFrUlU2c1NtaE5BZEhPTEEAAAAAAAAACrRZkZ2c1LW5wbVRhaVN
FaldaZzNtQXpR",
  "took": 11,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 1000000,
    "max_score": 1,
    "hits": [
```

Scroll

2. Get scroll id from response
3. Retrieve more documents by scroll id. Repeat step 2 and 3 until get enough documents



Scroll

4. Clear scroll context

   - The scroll context will be clear automatically or by manual

Suggest

Suggester

- Do you mean …?
- Auto-complete

Elasticsearch has three types of suggesters

- Term suggester: “do you mean” feature on terms
- Phrase sugester: “do you mean” feature on phrases
- Completion suggester: for implementing an auto-complete feature

Term suggester

- Term suggester suggests terms based on edit distance
  - The suggested terms are provided per analyzed token (each term in the submitted text)

```
POST /user/_search
{
  "query": {"match": {"name": "Jaci"}},
  "suggest": {
    "my_suggestion": {
      "text": "Jaci",
      "term" : {
        "field":"name"
      }
    }
  }
}
```

query is optional

text

```
"suggest": {
  "my_suggestion": [
    {
      "text": "jaci",
      "offset": 0,
      "length": 4,
      "options": [
        {
          "text": "jack",
          "score": 0.75,
          "freq": 358
        },
        {
          "text": "jace",
          "score": 0.75,
          "freq": 357
        },
        {
          "text": "jacobi",
          "score": 0.5,
          "freq": 2151
        },
        {
          "text": "jast",
          "score": 0.5,
          "freq": 2130
```

Term suggester

- max_edits determines the maximum edit distance for suggestions
  - The default value is 2

```
POST /user/_search
{
  "query": {"match": {"name": "Jaci"}},
  "suggest": {
    "my_suggestion": {
      "text": "Jaci",
      "term" : {
        "field":"name",
        "max_edits": 1
      }
    }
  }
}
```

only edit 1 char

```
"suggest": {
  "my_suggestion": [
    {
      "text": "jaci",
      "offset": 0,
      "length": 4,
      "options": [
        {
          "text": "jack",
          "score": 0.75,
          "freq": 358
        },
        {
          "text": "jace",
          "score": 0.75,
          "freq": 357
        }
      ]
    }
  ]
}
```

Term suggester

- ”Jack” scored higher because it only need edit 1 character, but “jacobi” is more frequent
- Use “sort”: “frequency”

```
POST /user/_search
{
  "query": {"match": {"name": "Jaci"}},
  "suggest": {
    "my_suggestion": {
      "text": "Jaci",
      "term" : {
        "field":"name",
        "sort": "frequency"
      }
    }
  }
}
```

sort by frequency

```
"suggest": {
  "my_suggestion": [
    {
      "text": "jaci",
      "offset": 0,
      "length": 4,
      "options": [
        {
          "text": "jacobi",
          "score": 0.5,
          "freq": 2151
        },
        {
          "text": "jast",
          "score": 0.5,
          "freq": 2130
        },
        {
          "text": "jack",
          "score": 0.75,
          "freq": 156
        },
```

Highlighting

Highlighter enable you to get highlighted snippets from one or more fields in your search results so you can show user where the query matches are.

```
POST /user/_search
{
  "query": {"match": {"university": "Georgia"}},
  "_source": "university",
  "highlight": {
    "fields": {"university": {}}
  }
}
```

highlight

```
"hits": [
  {
    "_index": "user",
    "_type": "_doc",
    "_id": "AV-Q88VFlM2ED_s-G78c",
    "_score": 5.0462227,
    "_source": {
      "university": "West Georgia Academy"
    },
    "highlight": {
      "university": [
        "West <em>Georgia</em> Academy"
      ]
    }
  },
```

highlight in result

Quiz

1. **True or False:** Scroll context will expire in a short time, the time length can be specified by scroll parameter.
2. **True or False:** For scroll api, we only need to get scroll id from the scroll initiate response, then we can use it in all subsequence requests.
3. What is the three types of suggesters?
4. Someone is seeking for suggestion for text “Shanghe” in city field, will the “Shanghai” term in the suggest list? What if we set the max_edits parameter to 1?
5. Run a suggest query on user index, ask for suggestion for text “Hally” in name field.

Thanks

Wang, Pei

Dai, Catherine

Hu, Hongsi