COMP810 Data Warehousing and Big Data

Search and Analyse Big Data with Elasticsearch (II)

Dr Weihua Li

Outline

- Review
- Analyzer and Text Analysis
- Inverted Index
- Stop Words and Stemming
- Match Query and Term Query
- Aggregations
- Elasticsearch Clients

Review

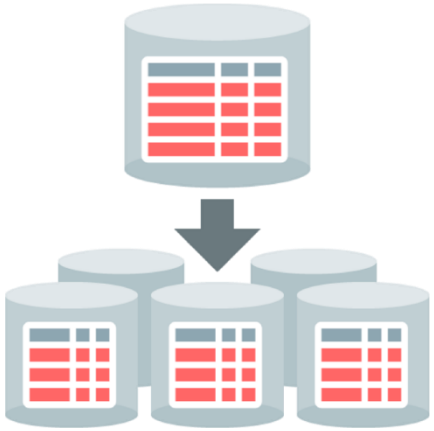
- Elasticsearch
  - Elasticsearch is a highly scalable open-source full-text search and analytics engine based on the Apache Lucene library.
- Kibana
  - Kibana acts as the user interface (UI) for monitoring, managing, and securing an Elastic Stack cluster.

- Backend Components

Node **Cluster** **Document** **Index** **Shard and Replicas**

Review (Cont.)

- Sharding
  - Sharding is a way to divide indices into smaller pieces



- Replication
  - Replication works by creating copies of shards, referred to as replica shards

- Elasticsearch API

GET / _cat / indices ? v

Method API Command Parameter

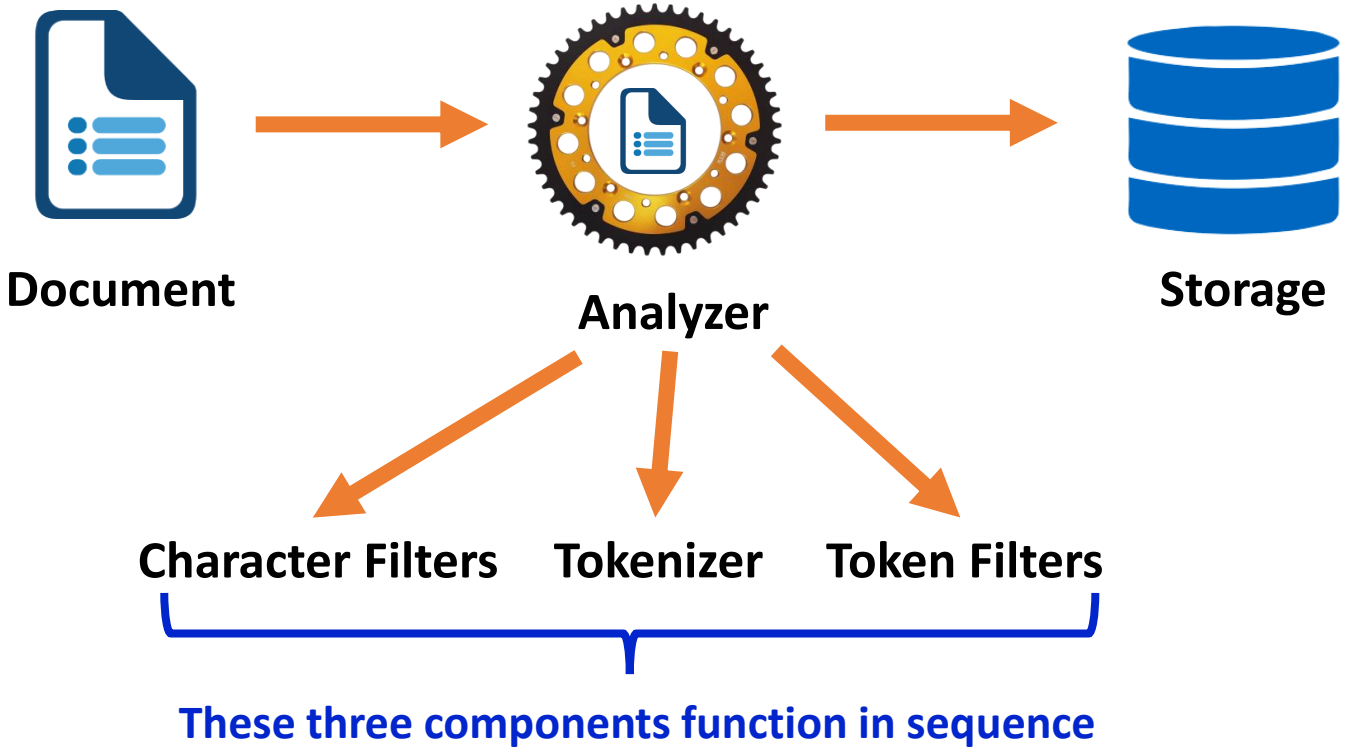
Use Elasticsearch API - Basics

- See the cluster status
  - GET /_cluster/health
  - GET /_cat/nodes?v
  - GET /_cat/indices?v
- Create an index named sales
  - PUT /sales
- Create a document
  - PUT /sales/_doc/123
    ```
    {
      "orderid": "123",
      "orderAmount" : "500"
    }
    ```
- Update a document
  ```
  POST sales/_doc/123/
  {
    "name": "Leo",
    "skills": "Data, AI and Programming"
  }
  ```
- Search a document based on id
  - GET /sales/_doc/123
- Delete a document
  - DELETE /sales/_doc/123
- Delete index
  - DELETE /sales/

Analyzer and Text Analysis

Analyzer

- Applicable to Text fields/values
- Text values are analysed when indexing documents
- Bult-in analyzers, e.g., standard, simple, whitespace, keyword, pattern



Analyzer (Cont.)

- Character Filters
  - Add, remove or change characters, e.g., html_strip filter, remove all the html tags
- Tokenizer
  - Separate a piece of text into smaller units called tokens, e.g., tokenize a string, i.e., split it into tokens.
- Token Filters
  - receive the output from tokenizer as input, add, remove or modify tokens. E.g., *lowercase* filter, *stemmer* filter.

Token Filters ("lowercase")

Input: "I", "REALLY", "like", "COMP810"
Output: ["i", "really", "like", "comp810"]

Tokenizer (standard)

Input: "I REALLY like COMP810!"
Output: ["I", "REALLY", "like", "COMP810"]

Character Filters (none)

Input: "I REALLY like COMP810!"
Output: "I REALLY like COMP810!"

By applying standard analyzer

Analyzer (Cont.)

- Analyze API in Elasticsearch: POST /_analyze
- Blow example shows how the standard analyzer works

```
POST _analyze
{
  "text": "I REALLY like COMP810! This course is very
    INTERESTING :-)",
  "analyzer": "standard"
}
```

```
POST _analyze
{
  "text": "I REALLY like COMP810! This course is very
    INTERESTING :-)",
  "char_filter": [],
  "tokenizer": "standard",
  "filter": ["lowercase"]
}
```

Inverted Index

- Indicate mapping between terms and the documents that contain them
- Inverted Index is the primary data structure
  - A document represents a JSON structure
  - Each text field has a separate inverted index data structure

Sentence

“I REALLY like COMP810!”

Tokens

[“i”, “really”, “like”, “comp810”]

TERM	DOC #1
i	X
really	X
like	X
comp810	X

DOC #1 "I REALLY like COMP810!" → ["i", "really", "like", "comp810"]

DOC #2 "I found COMP810 interesting :-)" → ["i", "found", "comp810", "interesting"]

DOC #3 "My favourite course is COMP810 !" → ["my", "favourite", "course", "is", "comp810"]

TERM	DOC #1	DOC #2	DOC #3
i	X	X	
really	X		
like	X		
comp810	X	X	X
found		X	
interesting		X	
my			X
favourite			X
course			X
is			X

- Inverted index is primarily associated with text field.
- There will be two inverted indices associated with text fields in the below example.

```
{
  "name": "Coffee Maker",
  "description": "Makes coffee super fast!",
  "price": 64,
  "in_stock": 10,
  "created_at": "2009-11-08T14:21:51Z"
}
```

```
{
  "name": "Toaster",
  "description": "Makes delicious toasts...",
  "price": 49,
  "in_stock": 4,
  "created_at": "2007-01-29T09:44:15Z"
}
```

name field

TERM	DOCUMENT #1	DOCUMENT #2
coffee	X	
maker	X	
toaster		X

description field

TERM	DOCUMENT #1	DOCUMENT #2
coffee		X
delicious	X	
fast		X
makes	X	X
super		X
toasts	X	

Stop Words

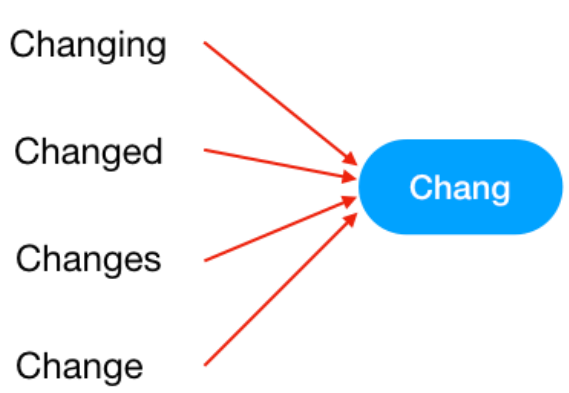
- Words that provide little or no value for the relevance scoring and are filtered out during the text analysis
- Common words such as “a”, “the”, “at”, “of”, “on”, etc.
- Nowadays, it appears less common for removing them due to the improvement of the relevance algorithms
- ES treats removing stop words as a token filter

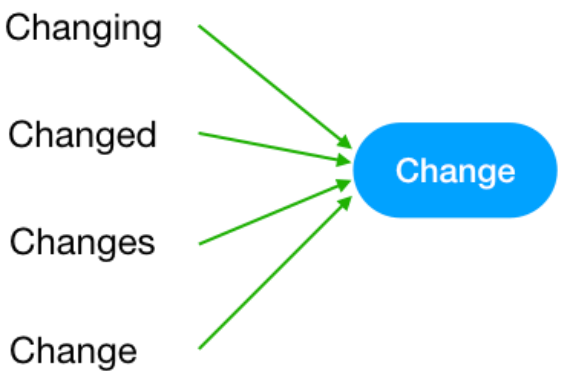
```
POST /_analyze
{
  "text": "My favourite course is COMP810 interesting!",
  "char_filter": [],
  "tokenizer": "standard",
  "filter": ["stop"]
}
```

```
{
  "tokens" : [
    {
      "token" : "My",
      "start_offset" : 0,
      "end_offset" : 2,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    {
      "token" : "favourite",
      "start_offset" : 3,
      "end_offset" : 12,
      "type" : "<ALPHANUM>",
      "position" : 1
    },
    {
      "token" : "course",
      "start_offset" : 13,
      "end_offset" : 19,
      "type" : "<ALPHANUM>",
      "position" : 2
    },
    {
      "token" : "COMP810",
      "start_offset" : 23,
      "end_offset" : 30,
      "type" : "<ALPHANUM>",
      "position" : 4
    },
    {
      "token" : "interesting",
      "start_offset" : 31,
      "end_offset" : 42,
      "type" : "<ALPHANUM>",
      "position" : 5
    }
  ]
}
```

Stemming

- Stemming is the process of reducing a word to its root (base) form. This ensures variants of a word match during a search.
- Stemming is language-dependent but often involves removing prefixes and suffixes from words.
- Stemming is treated as one of the token filters in Elasticsearch.





Stemming Vs Lemmatization

- **Apply Standard Analyzer without Stemming**

"I loved drinking bottles of wine on last year's vacation."

Past tense **Gerund** **Plural** **Possession**

```
POST /stemming_test/_doc
{
  "description": "I loved drinking
    bottles of wine on last year's
    vacation."
}
```

Standard Analyzer

TERM	DOC #1
i	X
loved	X
drinking	X
bottles	X
of	X
wine	X
on	X
last	X
year's	X
vacation	X

```
GET /stemming_test/_search
{
  "query":{
    "match": {
      "description": "loves"
    }
  }
}
```

No matching results!

- **Apply Custom Analyzer with Stemming**

"I loved drinking bottles of wine on last year's vacation."

"I love drink bottl of wine on last year vacat."

```
POST /stemming_test/_doc
{
  "description": "I loved drinking
    bottles of wine on last year's
    vacation."
}
```

```
GET /stemming_test/_search
{
  "query":{
    "match": {
      "description": "drinking"
    }
  }
}
```

Custom Analyzer with Stemming

TERM	DOC #1
i	X
love	X
drink	X
bottl	X
of	X
wine	X
on	X
last	X
year	X
vacat	X

Stemming (Cont.)

- Test stemmer token in Elasticsearch

```
POST /_analyze
{
  "text": "My favourite course is COMP810 interesting!",
  "char_filter": [],
  "tokenizer": "standard",
  "filter": [ "lowercase", "stemmer" ]
}
```


```
{
  "tokens" : [
    {
      "token" : "my",
      "start_offset" : 0,
      "end_offset" : 2,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    {
      "token" : "favourit",
      "start_offset" : 3,
      "end_offset" : 12,
      "type" : "<ALPHANUM>",
      "position" : 1
    },
    {
      "token" : "cours",
      "start_offset" : 13,
      "end_offset" : 19,
      "type" : "<ALPHANUM>",
      "position" : 2
    },
    {
      "token" : "is",
      "start_offset" : 20,
      "end_offset" : 22,
      "type" : "<ALPHANUM>",
      "position" : 3
    },
    {
      "token" : "comp810",
      "start_offset" : 23,
      "end_offset" : 30,
      "type" : "<ALPHANUM>",
      "position" : 4
    },
    {
      "token" : "interest",
      "start_offset" : 31,
      "end_offset" : 42,
      "type" : "<ALPHANUM>",
      "position" : 5
    }
  ]
}
```

Apply Custom Analyzer

- Define a custom analyzer by using a combination of character filters, tokenizer and token filters.
- Create an index by adding a custom analyzer

```
PUT /analyzer_test/
{
  "settings": {
    "analysis": {
      "analyzer": {
        "my_custom_analyzer": {
          "type": "custom",
          "char_filter": ["html_strip"],
          "tokenizer": "standard",
          "filter": [
            "lowercase",
            "stop",
            "stemmer",
            "asciifolding"
          ]
        }
      }
    }
  }
}
```

```
POST /analyzer_test/_analyze
{
  "analyzer": "my_custom_analyzer",
  "text": "<body>I found that COMP810 @AuT is interesting!</body>"
}
```

```
{
  "tokens" : [
    {
      "token" : "i",
      "start_offset" : 6,
      "end_offset" : 7,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    {
      "token" : "found",
      "start_offset" : 8,
      "end_offset" : 13,
      "type" : "<ALPHANUM>",
      "position" : 1
    },
    {
      "token" : "comp810",
      "start_offset" : 19,
      "end_offset" : 26,
      "type" : "<ALPHANUM>",
      "position" : 3
    },
    {
      "token" : "aut",
      "start_offset" : 28,
      "end_offset" : 31,
      "type" : "<ALPHANUM>",
      "position" : 4
    },
    {
      "token" : "interest",
      "start_offset" : 35,
      "end_offset" : 46,
      "type" : "<ALPHANUM>",
      "position" : 6
    }
  ]
}
```

Apply Custom Analyzer (Cont.)

```
PUT /analyzer_test/_mapping
{
  "properties":{
    "description": {
      "type": "text",
      "analyzer": "my_custom_analyzer"
    }
  }
}
```

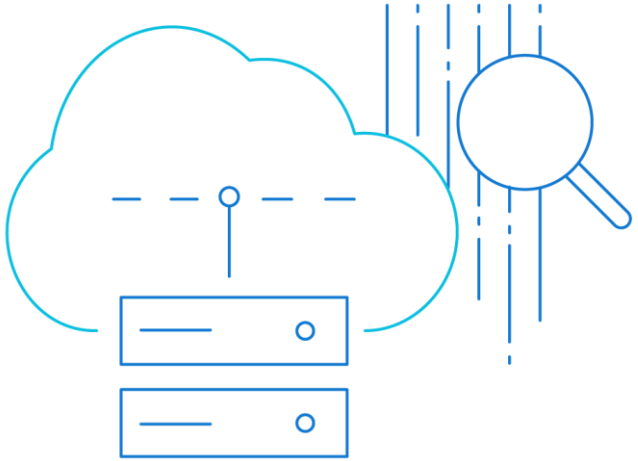
```
POST /analyzer_test/_doc
{
  "description": "<body>I found that
    COMP810 @AuT is interesting!</body>"
}
```

```
GET /analyzer_test/_search
{
  "query":{
    "match": {
      "description": "<div>interested</div>"
    }
  }
}
```

Can find the document

```
GET /analyzer_test/_search
{
  "query":{
    "match": {
      "description": "<div>that</div>"
    }
  }
}
```

Can NOT find the document



Query and Aggregation

Match Query

- Match Query returns documents that match a provided text, number, date or Boolean value.
- The match query is the standard query for performing a full-text search, including options for fuzzy matching.

```
GET kibana_sample_data_ecommerce/_search
{
  "size": 10,
  "query": {
    "match": {
      "products.product_name": "Vest"
    }
  }
}
```


```
"products" : [
  {
    "base_price" : 16.99,
    "discount_percentage" : 0,
    "quantity" : 1,
    "manufacturer" : "Pyramidustries",
    "tax_amount" : 0,
    "product_id" : 23760,
    "category" : "Women's Clothing",
    "sku" : "ZO0161001610",
    "taxless_price" : 16.99,
    "unit_discount_amount" : 0,
    "min_price" : 8.49,
    "_id" : "sold_product_579778_23760",
    "discount_amount" : 0,
    "created_on" : "2016-12-22T20:11:02+00:00",
    "product_name" : "Vest - black",
    "price" : 16.99,
    "taxful_price" : 16.99,
    "base_unit_price" : 16.99
  },
```

Term Query

- Term query returns documents that contain an exact term in a provided field.
- Use term query to find documents based on a precise value such as a price, a product ID, or a username.
- Avoid using the term query for text fields.

```
GET kibana_sample_data_ecommerce/_search
{
  "size": 1,
  "query": {
    "term": {
      "_id": "oybIJ3QBnbCmhWzQfmKX"
    }
  }
}
```


```
"hits" : [
  {
    "_index" : "kibana_sample_data_ecommerce",
    "_type" : "_doc",
    "_id" : "oybIJ3QBnbCmhWzQfmKX",
    "_score" : 1.0,
    "_source" : {
      "category" : [
        "Men's Clothing"
      ],
      "currency" : "EUR",
      "customer_first_name" : "Eddie",
      "customer_full_name" : "Eddie Underwood",
      "customer_gender" : "MALE",
      "customer_id" : 38,
      "customer_last_name" : "Underwood",
      "customer_phone" : "",
      "day_of_week" : "Monday",
```

Aggregations

- Statistics derived from your data are often needed when your aggregated document is large. The statistics aggregation allows you to get a min, max, sum, avg, and count of data in a single go. The statistics aggregation structure is similar to that of the other aggregations.

```
"aggs": {
  "name_of_aggregation": {
   "type_of_aggregation": {
    "field": "document_field_name"
}
```

- **aggs:** This keyword shows that you are using an aggregation.
- **name_of_aggregation:** the name of aggregation which the user defines.
- **type_of_aggregation:** the type of aggregation being used.
- **field:** the field keyword.
- **document_field_name:** the column name of the document being targeted.

Aggregations (cont.)

- Example: show the stats for the quantity field—min, max, avg, sum, and count values.

```
GET /kibana_sample_data_ecommerce/_search
{
  "size": 0,
  "aggs": {
    "quantity_stats": {
      "stats": {
        "field": "total_quantity"
      }
    }
  }
}
```


```
"hits" : {
  "total" : {
    "value" : 4675,
    "relation" : "eq"
  },
  "max_score" : null,
  "hits" : [ ]
},
"aggregations" : {
  "quantity_stats" : {
    "count" : 4675,
    "min" : 1.0,
    "max" : 8.0,
    "avg" : 2.1585026737967916,
    "sum" : 10091.0
  }
}
```

Filter Aggregation

- Filter documents into a single bucket.
- Often this will be used to narrow down the current aggregation context to a specific set of documents.

```
GET /kibana_sample_data_ecommerce/_search
{
  "size": 0,
  "aggs": {
    "User_based_filter": {
      "filter": {
        "term": {
          "user": "eddie"
        }
      },
      "aggs": {
        "avg_price": {
          "avg": {
            "field": "products.price"
          }
        }
      }
    }
  }
}
```

```
"aggregations" : {
  "User_based_filter" : {
    "doc_count" : 100,
    "avg_price" : {
      "value" : 34.85423743206522
    }
  }
}
```

Term Aggregation

- The terms aggregation generates buckets by field values.
- Once a field is selected, it will generate buckets for each of the values and place all of the records separately.

```
GET /kibana_sample_data_ecommerce/_search
{
  "size": 0,
  "aggs": {
    "Terms_Aggregation": {
      "terms": {
        "field": "user"
      }
    }
  }
}
```


```
"aggregations" : {
  "Terms_Aggregation" : {
    "doc_count_error_upper_bound" : 0,
    "sum_other_doc_count" : 2970,
    "buckets" : [
      {
        "key" : "elyssa",
        "doc_count" : 348
      },
      {
        "key" : "abd",
        "doc_count" : 188
      },
      {
        "key" : "wilhemina",
        "doc_count" : 170
      },
      {
        "key" : "rabbia",
        "doc_count" : 158
      },
```

Elasticsearch Clients

- Communicate with an Elasticsearch cluster in a language of your choice. We currently support the official Elasticsearch clients, including:
- Java REST Client, Java API, JavaScript, Ruby, Go, .NET, PHP, Perl, Python, eland Client, Rust API …

```
<dependency>
    <groupId>org.elasticsearch</groupId>
    <artifactId>elasticsearch</artifactId>
    <version>${elasticsearch.version}</version>
</dependency>

<dependency>
    <groupId>org.elasticsearch.client</groupId>
    <artifactId>elasticsearch-rest-high-level-client</artifactId>
    <version>${elasticsearch.version}</version>
</dependency>
```

```
private static final String HOST = "localhost";
private static final int PORT_ONE = 9200;
private static final int PORT_TWO = 9201;
private static final String SCHEME = "http";

private static RestHighLevelClient restHighLevelClient;

public static synchronized RestHighLevelClient getESClient() {
    if (restHighLevelClient == null) {
        restHighLevelClient = new RestHighLevelClient(
                RestClient.builder(
                        new HttpHost(HOST, PORT_ONE, SCHEME),
                        new HttpHost(HOST, PORT_TWO, SCHEME)));
    }
    return restHighLevelClient;
 }
```

References

- Bo Andersen, *Complete Guide to Elasticsearch*
- Giovanni Pagano Dritto, *an Overview on Elasticsearch and it usage*, 28 Mar 2019
- Elasticsearch Reference, https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html
- Getting Started with Apache Tika, https://tika.apache.org/1.19.1/gettingstarted.html