Modern Database Systems

Practicals. Elasticsearch

Doc. RNDr. Irena Holubova, Ph.D.
holubova@ksi.mff.cuni.cz

- Can be used for all kinds of documents
- Near real-time search
  - Slight latency (approx. 1 second) from the time you index (or update or delete) a document until the time it becomes searchable
- Index = collection of documents with similar characteristics
  - e.g., customer data, product catalogue, …
  - Has a name
  - In a cluster there can be any number of indices
- Indices can be divided into shards
  - Each shard can have replicas
  - Rebalancing and routing are done automatically
- Each node can act as a coordinator to delegate operations to the respective shards

Basic Operations

```
GET /_cat/indices?v
```

- Get all indices

```
PUT /customer?pretty
```

- Create index “customer” (and pretty print the result, if any)

```
PUT /customer/_doc/1?pretty
{ "name": "John Doe" }
```

- Index the given document with ID = 1

```
GET /customer/_doc/1?pretty
```

- Get document with ID = 1

```
DELETE /customer/_doc/1?pretty
```

- Delete document with ID = 1

```
DELETE /customer
```

- Delete index “customer”

Data Modification

- ID of a document
  - If an existing is used: the document is replaced (and re-indexed)
  - If a different is used: a new document is stored
    - The same one twice
  - If none is specified: a random ID is generated
- Document updates
  - No in-place updates
  - A document is deleted and a new one is created and indexed

Data Modification

```
POST /customer/_doc/1/_update?pretty
{ "doc": { "name": "Jane Doe" } }
```

- Change value of field “name” of document with ID = 1

```
POST /customer/_doc/1/_update?pretty
{ "doc": { "name": "Jane Doe", "age": 20 } }
```

- … and add a new field

```
POST /customer/_doc/1/_update?pretty
{ "script" : "ctx._source.age += 5" }
```

- … or use a script to specify the change

ctx._source = document content
ctx._index = document metadata
…

Batch Processing

```
POST /customer/_doc/_bulk?pretty
 {"index":{"_id":"1"}} {"name": "John Doe" }
 {"index":{"_id":"2"}} {"name": "Jane Doe" }
```

- Index two documents

```
POST /customer/_doc/_bulk?pretty
{"update":{"_id":"1"}}
  {"doc": { "name": "John Doe becomes Jane Doe" } }
{"delete":{"_id":"2"}}
```

- Update the first document, delete the second document

Search API


```
{ "account_number": 0,
  "balance": 16623,
  "firstname": "Bradshaw",
  "lastname": "Mckenzie",
  "age": 29,
  "gender": "F",
  "address": "244 Columbus Place",
  "employer": "Euron",
  "email": "bradshawmckenzie@euron.com",
  "city": "Hobucken",
  "state": "CO" }
```


- Sample data set

Search API

- Search parameters can be sent by:
  - REST request URI
  - REST request body
    - More expressive
    - More readable (JSON)?

```
GET /bank/_search?q=*&sort=account_number:asc&pretty
```

- Search (`_search`) in the `bank` index,
- match all the documents (`q=*`),
- sort the results using the `account_number` field of each document in an ascending order (`sort=account_number:asc`)

Search API


```
{
  "took" : 9,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1000,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
```

```
{
 "_index" : "holubova_bank",
 "_type" : "_doc",
 "_id" : "51",
 "_score" : 1.0,
 "_source" : {
   "account_number" : 51,
   "balance" : 14097,
   "firstname" : "Burton",
   "lastname" : "Meyers",
   "age" : 31,
   "gender" : "F",
   "address" : "334 River Street",
   "employer" : "Bezal",
   "email" : "burtonmeyers@bezal.com",
   "city" : "Jacksonburg",
   "state" : "MO"
 }
}, …
```

Search API

- In the result we will see:
  - `took` – time in milliseconds to execute the search
  - `timed_out` – if the search timed out or not
  - `_shards` – how many shards were searched
    - Total, successful, failed, skipped
  - `hits` – search results
    - `hits.total` – total number of documents matching our search criteria
    - `hits.hits` – actual array of search results
      - Default: first 10 documents
    - `hits.sort` – sort key for results
  - …

Search API

```
GET /bank/_search
{ "query": { "match_all": {} },
  "sort": [ { "account_number": "asc" } ] }
```

- The same exact search using the request body method

- When all search results are returned, Elasticsearch does not maintain any kind of server-side resources or open cursors etc.
  - Contrary to, e.g., traditional relational databases

Query DSL

- Domain specific language
- JSON-style

```
GET /bank/_search
{ "query": { "match_all": {} },
  "from": 10,  // starting index
  "size": 10,  // number of results
  "_source": ["account_number", "balance"]
               // include to the result
  "sort": { "balance": { "order": "desc" } }
 }
```

https://www.elastic.co/guide/en/elasticsearch/reference/6.5/query-dsl.html

Query DSL

```
"query": { "match": { "account_number": 20 } }
```

- Return the account numbered 20

```
"query": { "match": { "address": "mill" } }
```

- Return all accounts containing the term "`mill`" in the address

```
"query": { "match": { "address": "mill lane" } }
```

- Return all accounts containing the term "`mill`" or "`lane`" in the address

```
"query": { "match_phrase": { "address": "mill lane"
  } }
```

- Return all accounts containing the phrase "`mill lane`" in the address

Query DSL – Bool Query

- Bool query allows us to compose smaller queries into bigger queries using Boolean logic

```
"query": { "bool":
  { "must": [
    { "match": { "address": "mill" } },
    { "match": { "address": "lane" } } ] } }
```

- Return all accounts containing "`mill`" **and** "`lane`" in the address

```
"query": { "bool":
  { "should": [
    { "match": { "address": "mill" } },
    { "match": { "address": "lane" } } ] } }
```

- Return all accounts containing "`mill`" **or** "`lane`" in the address

Query DSL – Bool Query

```
"query": { "bool": {
  "must_not": [
    { "match": { "address": "mill" } },
    { "match": { "address": "lane" } } ] } }
```

- Return all accounts that contain **neither** "mill" **nor** "lane" in the address

```
"query": { "bool": {
  "must": [ { "match": { "age": "40" } } ],
  "must_not": [ { "match": { "state": "ID" } } ] } }
```

- Return all accounts of anybody who is 40 years old but doesn't live in ID(aho):

Query DSL – Filters

- `_score` field in the search results
  - Relative measure of how well the document matches the search query
    - The bigger, the more relevant
    - Practical scoring function evaluates it from 0 to `max_score` for the set
      - Idea: more relevant documents =
        - a) with a higher term frequency, and
        - b) contain more unique uses of the term compared to other documents in the index
  - When queries filter the set, it is not evaluated
    - Y/N depending on the filter

```
"query": {
  "bool": { "must": { "match_all": {} },
  "filter": {
    "range": { "balance": {
      "gte": 20000,
      "lte": 30000 } } } } }
```

- Return all accounts with balances between 20000 and 30000

Query DSL – Aggregations

- Ability to group and extract statistics
  - Like SQL GROUP BY
- We can execute searches returning both hits and aggregated results
  - No round tripping

```
GET /bank/_search {
  "size": 0,   // not show search hits
  "aggs": {
    "group_by_state": {
      "terms": { "field": "state.keyword" } } } }
```

- Group all the accounts by state, and returns the top 10 (default) states sorted by count descending (default)

Query DSL – Aggregations

```
GET /bank/_search {
  "size": 0,
  "aggs": {
    "group_by_state": {
      "terms": { "field": "state.keyword" },
      "aggs": {
        "average_balance": {
          "avg": { "field": "balance" } } } } } }
```

- Calculate the average account balance by state
  - Uses nested aggregations (`average_balance` in `group_by_state`)

Assignment

- Chose your unique problem domain
  - E.g., the results of football matches of various teams
- For your selected problem domain, think about an application that uses Elasticsearch for storing and querying your data.
- Submit a script with respective commands + explanatory comments

References

- Document APIs:
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/docs.html
- Search APIs:
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/search.html