NDBI040: PRACTICAL CLASS 9

ELASTICSEARCH

Tutor: Pavel Čontoš; December 16th 2020

(RECOMMENDED) REQUIREMENTS

- Database concepts
- JSON

- macOS / Linux command line or PuTTy / WinSCP on Windows

SERVER ACCESS

CONNECT TO NOSQL SERVER

- `ssh` on macOS / Linux
- `PuTTy` on Windows

- nosql.ms.mff.cuni.cz:42222
- Login and password send by e-mail
- Change your initial password (if not yet changed) by `passwd`

TRANSFER FILES

- `scp` on macOS / Linux
- `WinSCP` on Windows

ELASTICSEARCH

- Open Source
- Multi-model (search engine, document store)
- Based on `Apache Lucene`
- Implemented in Java
- Java API, RESTful HTTP/JSON API

- `Elastic Stash` = Elasticsearch +
  - `Logstash` data collector, parsing engine
  - `Kibana` visualization platform (next practical class)

DATA MODEL

- Cluster → Index → Document → Property

- `Cluster`
  - Collection of nodes, i.e. servers running an instance of Elasticsearch
- `Index`
  - A collection of documents and their properties
- `Document`
  - Set of properties
  - Associated with a unique identifier

HTTP API

- `cURL` tool
    - Allows to transfer data from / to a server using HTTP (or other supported protocols)

OPTIONS

- `-X command, --request command`
    - HTTP request method to be used (GET, …)
- `-d data, --data data`
    - Data to be sent to the server (implies the POST method)
- `-H header, --header header`
    - Extra headers to be included when sending the request
- `-i, --include`
    - Prints both headers and (not just) body of a response

INDEX API

- `Create` Index
  - Optionally, settings, mapping (for index or properties) and index aliases may be specified
  - Index name must be lowercase, cannot start with -, _, +, and cannot include \, /, *, ?, ", <, >, |, space, ,, #
  - `curl -X PUT "127.0.0.1:9200/$(whoami)_actors"`
  - `curl -X PUT "127.0.0.1:9200/$(whoami)_movies"`
- `Get` Index
  - Returns information about one or more indices
  - `curl -X GET "localhost:9200/_all?pretty"`
  - `curl -X GET "localhost:9200/_cat/indices?v&pretty"`
- Index `Exist`
  - `curl -I "localhost:9200/$(whoami)_actors?pretty"`

INDEX API

- `Open`/`Close` Index
  - A closed index is blocked for read/write operations
  - `curl -X POST "localhost:9200/$(whoami)_actors/_close?pretty"`
  - `curl -X POST "localhost:9200/$(whoami)_actors/_open?pretty"`
- Index `Settings`
  - `curl -X GET "localhost:9200/$(whoami)_actors/_settings?pretty"`
- Index `Stats`
  - `curl -X GET "localhost:9200/$(whoami)_actors/_stats?pretty"`
- Add Index `Alias`
  - `curl -X PUT "localhost:9200/$(whoami)_actors/_alias/$(whoami)_czechactors?pretty"`
- `Delete` Index
  - `curl -X DELETE "localhost:9200/$(whoami)_actors,$(whoami)_movies"`

INDEX API: MAPPINGS

- Definition of the way how a document and its properties are stored and indexed, i.e. its a `schema`
  - `Metadata properties` are used to customize how an associated metadata is treated, e.g. `_index`, `_id`, `_source` properties
  - Mapping contains a list of properties, each associated with its data type
  - Custom rules to control the mapping for dynamically added properties
  - See a list of data types here: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-types.html

- `Dynamic mapping`
  - Properties and its types do not need to be defined before being used
  - New properties are added automatically, just by indexing a document
- `Explicit mapping`
  - Manually defined mapping

EXAMPLE: MAPPINGS

```
curl -X PUT "localhost:9200/$(whoami)_actors?pretty" -H 'Content-Type: application/json' -d "{
  \"mappings\": {
    \"properties\": {
      \"name\": {
        \"properties\" : { \"first\" : {\"type\" : \"text\"}, \"last\" : {\"type\" : \"text\"} }
      },
      \"year\": { \"type\": \"integer\", \"index\": true }
    } },
  \"aliases\": { \"$(whoami)_actors1966\": { \"filter\": { \"term\": { \"year\": 1966 } } } } }"

curl -X PUT "localhost:9200/$(whoami)_actors/_mapping?pretty" -H 'Content-Type: application/json' -d '{
  "properties": { "movies": { "type": "keyword" } } }'

curl -X GET "localhost:9200/$(whoami)_actors/_mapping?pretty"
```

DOCUMENT API: INDEX

- Puts a JSON document to the specified index and makes it searchable
- Creates a new index with dynamic mappings if the target index does not exist
- Updates a document if the id matches to an existing document in the target index
    - Versions of documents, i.e. internal or external versioning
- Random ID is generated if not specified

```
curl -X PUT "127.0.0.1:9200/$(whoami)_actors/_doc/trojan?pretty" -H "Content-Type: application/json" -d '{
    "name": { "first": "Ivan", "last": "Trojan" },
    "year": 1964,
    "movies": [ "samotari", "medvidek", "karamazovi" ] }'
```

- Download file `data.txt` from practical class website and insert all its data to Elasticsearch

DOCUMENT API: GET

- `GET _doc, _source`
  - Retrieves a JSON document from an index
  - `GET <index>/_doc/<_id>` Retrieves single document from the particular index
  - `GET <index>/_source/<_id>` Retrieves document content only
  - `HEAD <index>/_doc/<_id>` Verifies that a document exists
  - `HEAD <index>/_source/<_id>` Verifies that a document exists

- `Multi GET _mget`
  - Extracts multiple JSON documents from an index
  - `GET /_mget`
  - `GET /<index>/_mget` Retrieves multiple documents from the particular index by ID

EXAMPLE: GET

```
curl -X GET "localhost:9200/$(whoami)_actors/_doc/sverak?_source=name.first,year&pretty"
curl -X GET "localhost:9200/$(whoami)_actors/_source/machacek/?_source_excludes=year&pretty"
curl -I "localhost:9200/$(whoami)_actors/_doc/trojan"
curl -I "localhost:9200/$(whoami)_movies/_source/zelary"

curl -X GET "localhost:9200/_mget?pretty" -H 'Content-Type: application/json' -d "{
  \"docs\": [
    { \"_index\": \"$(whoami)_actors\", \"_id\": \"trojan\" },
    { \"_index\": \"$(whoami)_actors\", \"_id\": \"machacek\" }
  ] }"

curl -X GET "localhost:9200/$(whoami)_movies/_mget?pretty" -H 'Content-Type: application/json' -d '{
  "ids" : ["medvidek", "zelary", "kolja"] }'
```

DOCUMENT API: UPDATE

- `POST _update`
- Updates a document using the script
  - Script can update, delete, or skip modifying the document
  - Access variables through the `ctx` map and `_source` property
  - `POST /<index>/_update/<_id>`

- `POST _update_by_query`
  - Update multiple documents based on `Search API` query criteria
  - `POST /<index>/_update_by_query`

EXAMPLE: UPDATE

```
curl -X POST "localhost:9200/$(whoami)_movies/_update/medvidek?pretty" -H 'Content-Type: application/json' -d '{
  "script" : "ctx._source.remove(\u0027year\u0027)" }'

curl -X POST "localhost:9200/$(whoami)_movies/_update/medvidek?pretty" -H 'Content-Type: application/json' -d '{
  "script" : {
    "source": "ctx._source.rating += params.increment",
    "lang": "painless",
    "params" : { "increment" : 10 } }, "upsert": { "counter": 1 } }'

curl -X POST "localhost:9200/$(whoami)_movies/_update_by_query?pretty" -H 'Content-Type: application/json' -d '{
  "script": {
    "source": "ctx._source.rating++",
    "lang": "painless"
  },
  "query": { "term": { "year": 2000 } } }'
```

DOCUMENT API: DELETE

- `DELETE _doc`
  - Removes a JSON document from the specified index
  - Increments version of document to ensure that document is already deleted
  - `DELETE /<index>/_doc/<_id>`

- `DELETE _delete_by_query`
  - Removes documents from index that match the query
  - Uses `Search API` to specify query criteria
  - `POST /<index>/_delete_by_query`

EXAMPLE: DELETE

```
curl -X DELETE "localhost:9200/$(whoami)_actors/_doc/geislerova?pretty"
```

```
curl -X POST "localhost:9200/$(whoami)_actors/_delete_by_query?pretty" -H 'Content-
Type: application/json' -d '{
  "query": { "match": { "name.last": "Vilhelmova" } }
}'
```

```
curl -X POST "localhost:9200/$(whoami)_actors,$(whoami)_movies/_delete_by_query?
pretty" -H 'Content-Type: application/json' -d '{
  "query": { "range" : { "year" : { "gte" : 2008 } } }
}'
```

DOCUMENT API: BULK

- Process multiple `index`, `create`, `update`, and `delete` operations in a single request
- `--data-binary` allows to submit bulk request from file
- `POST /_bulk`
- `POST /<index>/_bulk`

-
```
curl -s -H "Content-Type: application/x-ndjson" -XPOST
localhost:9200/_bulk --data-binary "@bulk.txt";
```

DOCUMENT API: REINDEX

- Makes a copy of all or subset of documents from a source index to target index
- `POST /_reindex`

```
curl -X POST "localhost:9200/_reindex?pretty" -H "Content-Type: application/json" -d "{
  \"source\": {
    \"index\": \"$(whoami)_actors\",
    \"query\": { \"range\" : { \"year\" : { \"gte\" : 1970 } } }
  },
  \"dest\": { \"index\": \"$(whoami)_youngactors\" }
}"
```

SEARCH API

- Allows to `search` and `aggregate` data stored in data streams or indices and retrieve documents that match the query
    - Search over multiple data streams and indices
    - Retrieve only selected properties
    - Sort and paginate result
    - Run an async search
- Query request body parameter accepts one or more queries written in `Query DSL`

- `GET /<index>/_search`
- `GET /_search`
- `POST /<index>/_search`
- `POST /_search`

QUERY DSL

- Domain Specific Language based on JSON to define queries

- It is an abstract syntax tree that consists of two types of clauses:
  - `Leaf query` clauses search for a value in a field (`match`, `term`, `range`)
  - `Compound query` clauses combine results (`bool`, `dis_max`) of leaf and compound clauses or alter theirs behavior (`constant_score`)

- Alternatively, the `q` parameter can be used to run a query parameter search
  - Overrides the query parameter in the request body
  - Not supports all the Query DSL queries
  - Useful for testing

MATCH ALL, MATCH NONE

- `match_all`
  - The most simple query, i.e. matches all documents
- `match_none`
  - Matches no document

```
curl -X GET "localhost:9200/$(whoami)_actors/_search?filter_path=hits.hits._source&pretty" -H
'Content-Type: application/json' -d '{
  "query" : { "match_all" : { } } ,
  "from" : 2,
  "size" : 4,
  "_source" : [ "name", "year" ],
  "sort" : { "year" : { "order" : "asc" } } }'
```

FULL TEXT QUERIES

- `match`, `match_phrase`, `match_phrase_prefix`, `match_bool_prefix`
  - Returns documents matching text, number, boolean or date value
- `multi_match`
  - Allows multi-field match queries
- `query_string`
  - Parses and queries values of properties
  - Wildcards (?, *), regular expressions, range queries, and boolean operators can be used
  - Allows to specify property name in query syntax, e.g. `name.first:(Iv?n OR Ji*i) AND (age:>70 OR year:[1950 TO *])`
- `intervals` query
  - Uses matching rules to search values from a specified property
    - `match`, `prefix`, `wildcard` (?, *), `all_of`, `any_of`, `filter` rules can be applied

EXAMPLE: FULL TEXT QUERIES

```
curl -X GET "localhost:9200/$(whoami)_actors/_search?pretty" -H 'Content-Type: application/json' -d '{
  "query": { "match": { "name.first" : "Ivan" } } }'

curl -X GET "localhost:9200/$(whoami)_*/_search?pretty" -H 'Content-Type: application/json' -d '{
  "query": { "multi_match": { "query" : "medvidek", "fields": ["title.cs", "movies"] } } }'

curl -X GET "localhost:9200/$(whoami)_actors/_search?pretty" -H 'Content-Type: application/json' -d '{
  "query": { "query_string": { "query": "Jiri OR Ivan",  "default_field": "name.first" } } }'

curl -X POST "localhost:9200/$(whoami)_movies/_search?pretty" -H 'Content-Type: application/json' -d '{
  "query": { "intervals" : { "text" : { "all_of" : {
          "intervals" : [ { "match" : { "query" : "vztah" } }, { "match" : { "query" : "poetický" } } ],
          "ordered": false  } } } } }'
```

TERM-LEVEL QUERIES

- Allows to search documents based on precise values in structured data
  - Match exact term (part of a value) stored in a field

- `exists` returns documents having defined a value for a given field
- `ids` returns documents based on theirs `_id`
- `prefix` returns documents that contain a property with a value of specified prefix
- `range` returns documents having properties value within the provided range
- `term`, `terms` returns documents having a property with an exact value (or one or more values)
- …

EXAMPLE: TERM-LEVEL QUERIES

```
curl -X GET "localhost:9200/$(whoami)_movies/_search?pretty" -H 'Content-Type: application/json' -d '{
  "query": { "exists": { "field": "actors" } } }'

curl -X GET "localhost:9200/$(whoami)_actors/_search?pretty" -H 'Content-Type: application/json' -d '{
  "query": { "ids" : { "values" : ["machacek", "trojan", "schneiderova"] } } }'

curl -X GET "localhost:9200/$(whoami)_actors/_search?pretty" -H 'Content-Type: application/json' -d '{
  "query": { "prefix": { "movies": { "value": "med" } } } }'

curl -X GET "localhost:9200/$(whoami)_*/_search?pretty" -H 'Content-Type: application/json' -d '{
  "query": { "range": { "year": { "gte": 1970, "lte": 1980 } } } }'
```

COMPOUND QUERIES

- `bool` query combines results from leaf or other compound queries, "more matches is better" approach
  - `must` query must appear in matching documents, contributes to the query (relevance) score
  - `filter` query must appear in matching documents, does not contribute to the score
  - `should` query should appear in the matching document, increases the query score
  - `must_not` query cannot appear in the matching document
- `boosting` query returns documents matching `positive` queries while `negative` queries decrease relevance
- `constant_score` packs a `filter` query and returns matching documents with the same score
- `dis_max` query returns documents that match at least one nested query, relevance by the best-matching query
- `function_score` allows to modify the score of matching documents by functions

EXAMPLE: COMPOUND QUERIES

```
curl -X POST "localhost:9200/$(whoami)_actors/_search?pretty" -H 'Content-Type: application/json' -d '{
  "query": {
    "bool" : {
      "must" : { "term" : { "movies" : "medvidek" } },
      "must_not" : { "term" : { "movies" : "karamazovi" } },
      "filter" : { "range" : { "year" : { "gte" : 1960, "lte" : 1978 } } },
      "should" : [
        { "term" : { "movies" : "samotari" } },
        { "term" : { "movies" : "kolja" } } ] } } }'

curl -X GET "localhost:9200/$(whoami)_actors/_search?pretty" -H 'Content-Type: application/json' -d '{
  "query": {
    "boosting": {
      "positive": { "match": { "name.first": "Jiri" } },
      "negative": { "range" : { "year" : { "gte" : 1960 } } },
      "negative_boost": 0.0 } } }'
```

EXAMPLE: COMPOUND QUERIES

```
curl -X GET "localhost:9200/$(whoami)_actors/_search?pretty" -H 'Content-Type: application/json' -d '{
  "query": {
    "function_score": {
      "query": { "match_all": {} },
      "boost": "5",
      "functions": [
        { "filter": { "match": { "name.first": "Jiri" } }, "weight": 30 },
        { "filter": { "match": { "movies": "medvidek" } }, "random_score": {}, "weight": 10 },
        { "filter": { "match": { "movies": "samotari" } }, "random_score": {}, "weight": 10 },
        { "filter": { "match": { "movies": "vratnelahve" } }, "random_score": {},  "weight": 10 }
      ],
      "max_boost": 500,
      "score_mode": "sum",
      "boost_mode": "avg",
      "min_score": 5 } } }'
```

AGGREGATIONS

- `Metric` aggregations calculates values like `sum` or `max` from property values

```
curl -X POST "localhost:9200/$(whoami)_actors/_search?size=0&pretty" -H 'Content-Type:
application/json' -d '{
  "aggs": {
    "minYear": { "min": { "field": "year" } },
    "maxYear": { "max": { "field": "year" } },
    "aveYear": { "avg": { "field": "year" } },
    "sumYear": { "sum": { "field": "year" } },
    "values": { "value_count": { "field": "year" } }
  } }'
```

AGGREGATIONS

- Bucket aggregations groups documents into buckets based on a criteria

```
curl -X GET "localhost:9200/$(whoami)_actors/_search?size=0&pretty" -H
'Content-Type: application/json' -d '{
  "aggs": { "actorsInMovie": { "terms": { "field": "movies" } } }
}'
```

- Pipeline takes input from other aggregations

EXERCISE 1

- Find actors with first name Jiri or Ivan who played in Medvídek and Samotáři movies
    - Use single `query_string` query
    - Return only `name` (e.g. `name.first` and `name.last`) and `movies`

EXERCISE 2

- Find all actors who played in the movie Medvídek
  - Return average `year` of birth of these actors
  - Use aggregations
  - Do not show search hits
    - E.g. use property (or parameter) size

EXERCISE 3

- Find movies filmed between `years` 2000 and 2006 such that they have drama or comedy but no romance listed in `genres` and they have a director specified
  - Construct boolean query
  - Return `title` only
  - Order the result by `ratings` in descending order and then by `years` in ascending order

EXERCISE 4

- Find movies which description contains word "`vztah`" followed by word "`milenec`" or "`vyvíjet`"
  - Construct full text query
  - Return only `title` and `text` properties
  - Sort result according to czech `title` in descending order

REFERENCES

- ElasticSearch Reference
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html
- Index API
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/indices.html
- Mapping
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-types.html
- Document API
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/docs.html
- Query DSL
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html
- Aggregations
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations.html