Design Pathshala

Exercise

In this Exercise of designpathsahala we will learn the basics of Elasticsearch.

Please insure that Elasticsearch is up and running by typing below command on console of the node where Elasticsearch is installed

```
curl localhost:9200
```

Open Kibana UI, by typing the URL: http://{ip_of_kibana_node}:5601/

You have to fill in the brackets like this (…) in different exercise to complete the exercises.

1. Insert our first document

Let's index our first JSON document! When we say index, we mean store in Elasticsearch. We'll use restaurant food safety violations from the City of San Francisco, let's index one document.

```
POST /inspections/_doc
{
 "business_address": "660 Sacramento St",
 "business_city": "San Francisco",
 "business_id": "2228",
 "business_latitude": "37.793698",
 "business_location": {
  "type": "Point",
  "coordinates": [
   -122.403984,
   37.793698
  ]
 },
 "business_longitude": "-122.403984",
 "business_name": "Tokyo Express",
 "business_postal_code": "94111",
 "business_state": "CA",
 "inspection_date": "2016-02-04T00:00:00.000",
 "inspection_id": "2228_20160204",
 "inspection_type": "Routine",
 "inspection_score":96,
 "risk_category": "Low Risk",
 "violation_description": "Unclean nonfood contact surfaces",
```

```
  "violation_id": "2228_20160204_103142"
}
```


Please, see the structure of the JSON document, there is a geopoint, dates, and numbers

2. Search the document

Let's search the index using a GET command and _search endpoint. Please fill in the braces to search.

GET (…)

3. PUT vs POST

Elasticsearch uses a REST API, and it matters whether we use POST vs PUT
PUT requires an id for the document, as part of the URL. If we run the following, what will happen? Lets check!
Fill in the index endpoint before hitting the request for index name: inspections and type: _doc .


```
PUT ( .. )
{
  "business_address": "660 Sacramento St",
  "business_city": "San Francisco",
  "business_id": "2228",
  "business_latitude": "37.793698",
  "business_location": {
    "type": "Point",
    "coordinates": [
      -122.403984,
      37.793698
    ]
  },
  "business_longitude": "-122.403984",
  "business_name": "Tokyo Express",
  "business_postal_code": "94111",
  "business_state": "CA",
  "inspection_date": "2016-02-04T00:00:00.000",
  "inspection_id": "2228_20160204",
  "inspection_type": "Routine",
  "inspection_score":96,
  "risk_category": "Low Risk",
  "violation_description": "Unclean nonfood contact surfaces",
  "violation_id": "2228_20160204_103142"
}
```

Error!!
Why?

4. Automatic document ID creation

Lets try out POST to create the document's ID for us. Fill in the index endpoint before hitting the request for index name: inspections and type: _doc .

POST (…)


```
{
 "business_address": "660 Sacramento St",
 "business_city": "San Francisco",
 "business_id": "2228",
 "business_latitude": "37.793698",
 "business_location": {
  "type": "Point",
  "coordinates": [
   -122.403984,
   37.793698
  ]
 },
 "business_longitude": "-122.403984",
 "business_name": "Tokyo Express",
 "business_postal_code": "94111",
 "business_state": "CA",
 "inspection_date": "2016-02-04T00:00:00.000",
 "inspection_id": "2228_20160204",
 "inspection_type": "Routine",
 "inspection_score":96,
 "risk_category": "Low Risk",
 "violation_description": "Unclean nonfood contact surfaces",
 "violation_id": "2228_20160204_103142"
}
```


5. PUT with document id

Try to hit PUT with explicit document id: 12345 and also fill in the index endpoint before hitting the request for index name: inspections and type: _doc .

PUT (…)


```
{
```

```
 "business_address": "660 Sacramento St",
 "business_city": "San Francisco",
 "business_id": "2228",
 "business_latitude": "37.793698",
 "business_location": {
  "type": "Point",
  "coordinates": [
   -122.403984,
   37.793698
  ]
 },
 "business_longitude": "-122.403984",
 "business_name": "Tokyo Express",
 "business_postal_code": "94111",
 "business_state": "CA",
 "inspection_date": "2016-02-04T00:00:00.000",
 "inspection_id": "2228_20160204",
 "inspection_type": "Routine",
 "inspection_score":96,
 "risk_category": "Low Risk",
 "violation_description": "Unclean nonfood contact surfaces",
 "violation_id": "2228_20160204_103142"
}
```


6. Index creation

Indexing the document automatically created the index for us, named "inspection". The document is of type "_doc" (POST /inspection/_doc)
It is recommeneded to store only one type per index, as multiple types per index are not not more supported by Elasticsearch.

Instead of dynamically creating the index based on the first document we add, we can create the index beforehand, to set certain settings

Lets first delete the index: inspections

DELETE (…)

Now create a new index by name: inspections

(…)


```
{
 "settings": {
  "index.number_of_shards": 1,
  "index.number_of_replicas": 0
```

```
 }
}
```

We'll are useing 1 shard for this example, and no replicas, we probably wouldn't want to do this in production

7. Bulk API

When you need to index a lot of docs, you should use the bulk API, you may see signficant performance benefits. Fill in the index endpoint before hitting the request for index name: inspections and type: _doc .

(…)


```
{ "index": { "_id": 1 }}
{"business_address":"315 California St","business_city":"San Francisco","business_id":"24936","business_latitude":"37.793199","business_location":{"type":"Point","coordinates":[-122.400152,37.793199]},"business_longitude":"-122.400152","business_name":"San Francisco Soup Company","business_postal_code":"94104","business_state":"CA","inspection_date":"2016-06-09T00:00:00.000","inspection_id":"24936_20160609","inspection_score":77,"inspection_type":"Routine - Unscheduled","risk_category":"Low Risk","violation_description":"Improper food labeling or menu misrepresentation","violation_id":"24936_20160609_103141"}
{ "index": { "_id": 2 }}
{"business_address":"10 Mason St","business_city":"San Francisco","business_id":"60354","business_latitude":"37.783527","business_location":{"type":"Point","coordinates":[-122.409061,37.783527]},"business_longitude":"-122.409061","business_name":"Soup Unlimited","business_postal_code":"94102","business_state":"CA","inspection_date":"2016-11-23T00:00:00.000","inspection_id":"60354_20161123","inspection_type":"Routine", "inspection_score": 95}
{ "index": { "_id": 3 }}
{"business_address":"2872 24th St","business_city":"San Francisco","business_id":"1797","business_latitude":"37.752807","business_location":{"type":"Point","coordinates":[-122.409752,37.752807]},"business_longitude":"-122.409752","business_name":"TIO CHILOS GRILL","business_postal_code":"94110","business_state":"CA","inspection_date":"2016-07-05T00:00:00.000","inspection_id":"1797_20160705","inspection_score":90,"inspection_type":"Routine - Unscheduled","risk_category":"Low Risk","violation_description":"Unclean nonfood contact surfaces","violation_id":"1797_20160705_103142"}
{ "index": { "_id": 4 }}
{"business_address":"1661 Tennessee St Suite 3B","business_city":"San Francisco Whard Restaurant","business_id":"66198","business_latitude":"37.75072","business_location":{"type":"Point","coordinates":[-122.388478,37.75072]},"business_longitude":"-
```

122.388478","business_name":"San Francisco
Restaurant","business_postal_code":"94107","business_state":"CA","inspection_date":"201
6-05-
27T00:00:00.000","inspection_id":"66198_20160527","inspection_type":"Routine","inspecti
on_score":56 }
{ "index": { "_id": 5 }}
{"business_address":"2162 24th Ave","business_city":"San
Francisco","business_id":"5794","business_latitude":"37.747228","business_location":{"typ
e":"Point","coordinates":[-122.481299,37.747228]},"business_longitude":"-
122.481299","business_name":"Soup
House","business_phone_number":"+14155752700","business_postal_code":"94116","busi
ness_state":"CA","inspection_date":"2016-09-
07T00:00:00.000","inspection_id":"5794_20160907","inspection_score":96,"inspection_typ
e":"Routine - Unscheduled","risk_category":"Low
Risk","violation_description":"Unapproved or unmaintained equipment or
utensils","violation_id":"5794_20160907_103144"}
{ "index": { "_id": 6 }}
{"business_address":"2162 24th Ave","business_city":"San
Francisco","business_id":"5794","business_latitude":"37.747228","business_location":{"typ
e":"Point","coordinates":[-122.481299,37.747228]},"business_longitude":"-
122.481299","business_name":"Soup-or-
Salad","business_phone_number":"+14155752700","business_postal_code":"94116","busin
ess_state":"CA","inspection_date":"2016-09-
07T00:00:00.000","inspection_id":"5794_20160907","inspection_score":96,"inspection_typ
e":"Routine - Unscheduled","risk_category":"Low
Risk","violation_description":"Unapproved or unmaintained equipment or
utensils","violation_id":"5794_20160907_103144"}

Let's go back to executing our basic search and find *all* documents inserted.

(…) /inspections/_doc/_search

8. Let's find all inspection reports for places that sell soup

```
GET /inspections/_doc/_search
{
  "query": {
    ( … ): {
      "business_name": ( … )
    }
  }
}
```

9. Look for restaurants with the name San Francisco

Since San Francisco is two words, we'll use match_phrase

(…)

Can you see results are ranked by "relevance" (_score)
Let's look again the previous search and compare the relevance score of both the search.

```
GET /inspections/_doc/_search
{
  "query": {
    "match": {
      "business_name": "soup"
    }
  }
}
```

10. Find all docs with "soup" and "san francisco" in the business name

We can do boolean combinations of queries like this

```
GET /inspections/_doc/_search
{
  "query": {
    "bool": {
      ( … ): [
        {
          "match": {
            ( … )
          }
        },
        {
          "match_phrase": {
            "business_name": "san francisco"
          }
        }
      ]
    }
  }
}
```

11. Maybe you hate soup

SO negate parts of a query, search for businesses without "soup" in the name ()

```
GET /inspections/_doc/_search
( … )
```

12. Emphasize places with "soup in the name"

Combinations can be boosted for different effects. Lets try to boost the results 3 times which contain soup, as compared to “san francisco”.

```
GET /inspections/_doc/_search
{
  "query": {
    "bool": {
      "should": [
        {
          "match_phrase": {
            "business_name": {
              "query": "soup",
              ( … )
            }
          }
        },
        {
          "match_phrase": {
            "business_name": {
              "query": "san francisco"
            }
          }
        }
      ]
    }
  }
}
```

13. Highlighting

Sometimes it's unclear what actually matched. We can highlight the matching fragment. Lets highlight the business_name in the query output.

```
GET /inspections/_doc/_search
```

```
{
  "query" : {
    "match": {
      "business_name": "soup"
    }
  },
( … )
  }
}
```

14. Find soup companies with a health score greater than 80

Finally, we can perform filtering, when we don't need text analysis (or need to do exact matches, range queries, etc.)
Let's find soup companies with a health score greater than (equal to) 80. Also, sort the results with inspection_score decending.

```
GET /inspections/_doc/_search
{
  "query": {
     "range": {
       ( … )
     }
  },
  "sort": [
    ( … )
  ]
}
```

15. SQL queries

Try to run simple SQL query with Elasticsearch. Query business_name, inspection_score from inspections index

```
POST /_xpack/sql?format=txt
{
   "query": "( … )"
}
```

There are Multiple methods to query Elasticsearch with SQL

- Through the rest endpoints (as seen above)
- Through the included CLI tool in the /bin directory of Elasticsearch
- JDBC Elasticsearch client

16. Aggregations

Let's search for the term "soup", and bucket results by health score (similar to the facets you would see in an ebay site)s
https://www.ebay.com/sch/i.html?_from=R40&_trksid=p2380057.m570.l1313.TR12.TRC2.A0.H0.Xwatch.TRS0&_nkw=watch&_sacat=0

```
GET /inspections/_doc/_search
{
  "query": {
    "match": {
      "business_name": "soup"
    }
  }
 ,"aggregations" : {
    "inspection_score" : {
      "range" : {
        "field" : "inspection_score",
        "ranges" : [
          {
            "key" : "0-80",
            "from" : 0,
            "to" : 80
          },
          {
            "key" : "81-90",
            "from" : 81,
            "to" : 90
          },
          {
            "key" : "91-100",
            "from" : 91,
            "to" : 100
          }
        ]
      }
    }
  }
}
```

17. Let's find soup restaurants closest to us!

Geo search is another powerful tool for search. Let's find soup restaurants closest to us! We have the geo point within the document, let's use it

GET /inspections/_doc/_search

Let's execute the follow geo query, to sorted restaurants by distance by us

```
GET /inspections/_doc/_search
{
  "query": {
    "match": { "business_name": "soup"}
  },
  "sort": [
    {
      "_geo_distance": {
        "coordinates": {
          "lat":  37.783527,
          "lon": -122.409061
        },
        "order":       "asc",
        "unit":        "km"
      }
    }
    ]
}
```

Error! Why?

Elasticsearch doesn't know the field is a geopoint. We must define this field as a geo point using mappings
Mapping are helpful for defining the structure of our document, and more efficiently storing/searching the data within our index
We have numbers/dates/strings, and geopoints, let's see what elasticsearch thinks our mapping is

GET (…)

Let's change the mapping, delete our index, and perform our bulk import again
In production scenarios, you may prefer to use the reindex API, you can add new mapping fields without needing to migrate the data

DELETE inspections

PUT /inspections

```
PUT inspections/_mapping/_doc
{
```

```
"properties": {
  "business_address": {
   "type": "text",
   "fields": {
    "keyword": {
     "type": "keyword",
     "ignore_above": 256
    }
   }
  },
  "business_city": {
   "type": "text",
   "fields": {
    "keyword": {
     "type": "keyword",
     "ignore_above": 256
    }
   }
  },
  "business_id": {
   "type": "text",
   "fields": {
    "keyword": {
     "type": "keyword",
     "ignore_above": 256
    }
   }
  },
  "business_latitude": {
   "type": "text",
   "fields": {
    "keyword": {
     "type": "keyword",
     "ignore_above": 256
    }
   }
  },
  "coordinates": {
    "type": "geo_point"
  },
  "business_longitude": {
   "type": "text",
   "fields": {
    "keyword": {
     "type": "keyword",
     "ignore_above": 256
    }
```

```
  }
 },
 "business_name": {
  "type": "text",
  "fields": {
   "keyword": {
    "type": "keyword",
    "ignore_above": 256
   }
  }
 },
 "business_phone_number": {
  "type": "text",
  "fields": {
   "keyword": {
    "type": "keyword",
    "ignore_above": 256
   }
  }
 },
 "business_postal_code": {
  "type": "text",
  "fields": {
   "keyword": {
    "type": "keyword",
    "ignore_above": 256
   }
  }
 },
 "business_state": {
  "type": "text",
  "fields": {
   "keyword": {
    "type": "keyword",
    "ignore_above": 256
   }
  }
 },
 "inspection_date": {
  "type": "date"
 },
 "inspection_id": {
  "type": "text",
  "fields": {
   "keyword": {
    "type": "keyword",
    "ignore_above": 256
```

```
          }
         }
        },
        "inspection_score": {
         "type": "long"
        },
        "inspection_type": {
         "type": "text",
         "fields": {
          "keyword": {
           "type": "keyword",
           "ignore_above": 256
          }
         }
        },
        "risk_category": {
         "type": "text",
         "fields": {
          "keyword": {
           "type": "keyword",
           "ignore_above": 256
          }
         }
        },
        "violation_description": {
         "type": "text",
         "fields": {
          "keyword": {
           "type": "keyword",
           "ignore_above": 256
          }
         }
        },
        "violation_id": {
         "type": "text",
         "fields": {
          "keyword": {
           "type": "keyword",
           "ignore_above": 256
          }
         }
        }
       }
}
```

Lets, re-run our original geo query

```
GET /inspections/_doc/_search
{
 "query": {
  "match": { "business_name": "soup"}
 },
 "sort": [
  {
   "_geo_distance": {
    "business_location": {
     "lat":  37.800175,
     "lon": -122.409081
    },
    "order":       "asc",
    "unit":        "km",
    "distance_type": "plane"
   }
  }
  ]
}
```

18. Updates

Let's finish the CRUD components, we covered C, and R, let's show show to update and delete documents

Let's add a flagged field to one of our documents with value true, using a partial document update

```
GET /inspections/_doc/_search

POST /inspections/_doc/5/( … )
{
  "doc" : {
    ( … )
    "views": 0
  }
}
```

19. Delete document

To delete a document, we can just pass the document id to the DELETE API. Lets delete the document id 5 for for index name: inspections and type: _doc .

(…)

That completed the CRUD section

20. Analyzers

Text analysis is core to Elasticsearch, and very important to understand
As you saw a mapping configuration for data types in the previous example, you can also configure an analyzer per field or an entire index!

Analysis = tokenization + token filters

Tokenization breaks sentences into discrete tokens

```
GET /inspections/_analyze
{
  "tokenizer": "standard",
  "text": "my email address test123@company.com"
}

GET /inspections/_analyze
{
  "tokenizer": "whitespace",
  "text": "my email address test123@company.com"
}

GET /inspections/_analyze
{
  "tokenizer": "standard",
  "text": "Brown fox brown dog"
}
```

And filters manipulate those tokens

```
GET /inspections/_analyze
{
  "tokenizer": "standard",
  "filter": ["lowercase"],
  "text": "Brown fox brown dog"
}
```

There is a wide variety of filters.

```
GET /inspections/_analyze
{
```

```
  "tokenizer": "standard",
  "filter": ["lowercase", "unique"],
  "text": "Brown brown brown fox brown dog"
}

# More info:
https://www.elastic.co/guide/en/elasticsearch/guide/current/_controlling_analysis.html
```