#

Contents

#

1. JSON Documents

2. CRUD - Create / Read / Update / Delete

a. Create

- Different ways to insert/create an index

- Bulk indexing documents

b. Read

- Basic searches

- Intermediate searches

- Sample SQL query in Elasticsearch

- Facets and aggregations

- Aggregation use cases (doc values vs inverted index?) TODO

- Sample geo search

c. Update

- Updating documents TODO

d. Delete

- Deleting documents

3. Mappings

4. Analyzers

This is the Kibana Dev tools console, we'll use this to interact with Elasticsearch

#

Elasticsearch stores documents using the JSON format

To quickly mention JSON, this is a sample JSON document

```
{
  "name" : "Elastic",
  "location" : {
    "state" : "Co",
    "zipcode" : 80006
  }
}

# A document can have many fields with values

{
  "name" : "Elastic",
  ...
  <field> : <value>
}



# And each value must be one of 6 types to be valid JSON (string, number, object, array, boolean, null)
# http://www.json.org/

# 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

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"
}
```


```
# See the structure of the JSON document, there is a geopoint, dates, and numbers

# Let's search the index using a GET command
GET /inspections/_search

# We'll dive deeper into the search API soon, for now, let's focus on indexing documents
```

A lot just happened, let's discuss

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 we'll get an error

```
PUT /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"
```

```
}

# POST creates the document's ID for us

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"
}
```

We can also specify it with PUT

PUT /inspections/_doc/12345


```
{
  "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"
}
```

Indexing the document automatically created the index for us, named "inspection"

The document is of type "report" (POST /inspection/report)

It is recommeneded to store only one type per index, as multiple types per index will not be supported in the future

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

DELETE /inspections

PUT /inspections

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

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

When you need to index a lot of docs, you should use the bulk API, you may see signficant performance benefits

POST /inspections/_bulk

{ "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":"Routin

e - 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":"2016-05-27T00:00:00.000","inspection_id":"66198_20160527","inspection_type":"Routine","inspection_score":56 }

{ "index": { "_id": 5 }}

{"business_address":"2162 24th Ave","business_city":"San Francisco","business_id":"5794","business_latitude":"37.747228","business_location":{"type":"Point","coordinates":[-122.481299,37.747228]},"business_longitude":"-122.481299","business_name":"Soup House","business_phone_number":"+14155752700","business_postal_code":"94116","business_state":"CA","inspection_date":"2016-09-07T00:00:00.000","inspection_id":"5794_20160907","inspection_score":96,"inspection_type":"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":{"type":"Point","coordinates":[-122.481299,37.747228]},"business_longitude":"-122.481299","business_name":"Soup-or-Salad","business_phone_number":"+14155752700","business_postal_code":"94116","business_state":"

CA","inspection_date":"2016-09-07T00:00:00.000","inspection_id":"5794_20160907","inspection_score":96,"inspection_type":"Routine - Unscheduled","risk_category":"Low Risk","violation_description":"Unapproved or unmaintained equipment or utensils","violation_id":"5794_20160907_103144"}

More info: https://www.elastic.co/guide/en/elasticsearch/guide/current/bulk.html

#_____

Let's go back to executing our basic search

Find *all* documents

```
GET /inspections/_search
```

#_____

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

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

Let's look for restaurants with the name San Francisco

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

```
GET /inspections/_search
```

```
{
  "query": {
    "match_phrase": {
      "business_name": "san francisco"
    }
  }
}

# Results are ranked by "relevance" (_score)
# Let's look again
GET /inspections/_search
{
  "query": {
    "match": {
      "business_name": "soup"
    }
  }
}



# More info: https://www.elastic.co/guide/en/elasticsearch/guide/current/relevance-intro.html



#_____________________________________________________
# We can also do boolean combinations of queries
# Let's find all docs with "soup" and "san francisco" in the business name

GET /inspections/_search
{
```

```
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "business_name": "soup"
          }
        },
        {
          "match_phrase": {
            "business_name": "san francisco"
          }
        }
      ]
    }
  }
}

#
# Or negate parts of a query, businesses without "soup" in the name (maybe you hate soup)

GET /inspections/_search
{
  "query": {
    "bool": {
      "must_not": [
        {
          "match": {
            "business_name": "soup"
```

```
        }
       }
     ]
   }
 }
}

#_____________________________________________________
# Combinations can be boosted for different effects
# Let's emphasize places with "soup in the name"

GET /inspections/_search
{
  "query": {
    "bool": {
      "should": [
        {
          "match_phrase": {
            "business_name": {
              "query": "soup",
              "boost" : 3
            }
          }
        },
        {
          "match_phrase": {
            "business_name": {
              "query": "san francisco"
            }
```

```
        }
       }
     ]
   }
 }
}

# Sometimes it's unclear what actually matched.
# We can highlight the matching fragments:

GET /inspections/_search
{
  "query" : {
    "match": {
      "business_name": "soup"
    }
  },
  "highlight": {
    "fields": {
      "business_name": {}
    }
  }
}

#_____________________________________________________

# 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 80
```

```
GET /inspections/_search
{
  "query": {
     "range": {
       "inspection_score": {
         "gte": 80
       }
     }
  },
  "sort": [
    { "inspection_score" : "desc" }
  ]
}

# More info: https://www.elastic.co/guide/en/elasticsearch/guide/current/structured-search.html

# We can also sort our results by "inspection_score"

# Sample SQL Query with Elasticsearch

POST /_sql?format=txt
{
    "query": "SELECT business_name, inspection_score FROM inspections ORDER BY inspection_score DESC LIMIT 5"
}

# 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

More details can be found here: https://www.elastic.co/guide/en/elasticsearch/reference/6.3/xpack-sql.html

Aggregations (one use case is faceting data) are very interesting

We won't have time to cover aggregation in depth now, but we want to get you familiar with

how they work, so you can use them on your own

Let's search for the term "soup", and bucket results by health score (similar to the facets you would see in an ebay site)

Show: 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/_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
        }
      ]
    }
  }
 }
}
```

```
# 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/_search

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

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

Error! 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 /inspections/_mapping
```

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/

```
{
    "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
    }
  }
}
```

```
      }
}
```

Now we can execute our original geo query

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

That was a very short introduction to geo queries and mappings, the goal was to get your feet wet to hopefuly go off and learn more

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, using a partial document update

```
GET /inspections/_search

POST /inspections/_doc/5/_update
{
  "doc" : {
    "flagged" : true,
    "views": 0
  }
}
```

To delete a document, we can just pass the document id to the DELETE API

```
DELETE /inspections/_doc/5
```

That completed the CRUD section

- 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


#In this index template, we've defined two fields,
#timestamp and response_code, which will be created
#when we ingest the data. We've also defined a
#dynamic runtime field mapping. Any other fields
#will be runtime fields.
PUT _index_template/my_dynamic_index
{
  "index_patterns": [
    "my_dynamic_index-*"
    ],
    "template": {
      "mappings":{
        "dynamic": "runtime",
      "properties": {
        "timestamp": {
          "type": "date",
          "format": "yyyy-MM-dd"
        },
```

```
      "response_code": {
        "type": "integer"
      }
    }
    }
  }
}
```

#The data we've ingested has three fields: timestamp, #response code, and new_tla. In the past, new_tla #wouldn't have been added because it wasn't defined in #the index template. Now it's just treated as a runtime #field.

```
POST my_dynamic_index-1/_bulk
{"index": {}}
{"timestamp": "2021-04-02", "response_code": 200, "new_tla": "data-1"}
{"index": {}}
{"timestamp": "2021-04-02", "response_code": 200, "new_tla": "data-2"}
{"index": {}}
{"timestamp": "2021-04-02", "response_code": 200, "new_tla": "data-3"}
{"index": {}}
{"timestamp": "2021-04-02", "response_code": 200, "new_tla": "data-4"}
{"index": {}}
{"timestamp": "2021-04-02", "response_code": 200, "new_tla": "data-5"}
{"index": {}}
{"timestamp": "2021-04-02", "response_code": 200, "new_tla": "data-6"}
```

#Here we're running a normal search query for new_tla. A #query can also be run with both an indexed field like #response_code and a runtime field like new_tla.

```
GET my_dynamic_index-1/_search
{
```

```
  "query": {
    "match": {
      "new_tla": "data-1"
    }
  }
}
```