Lab12: ElasticSearch Basics

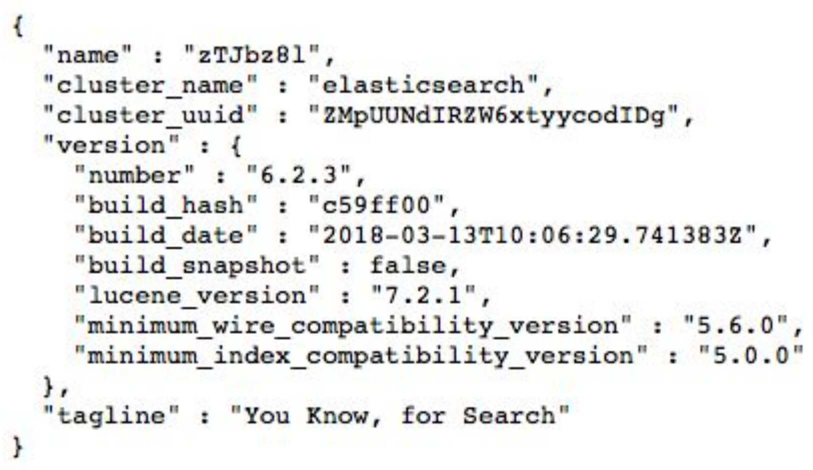
by Andriy Pyshchyk

Setup Elasticsearch + Kibana

1. Download Elasticsearch https://www.elastic.co/downloads/elasticsearch
2. Download Kibana https://www.elastic.co/downloads/kibana
3. Unpack

Run Elasticsearch

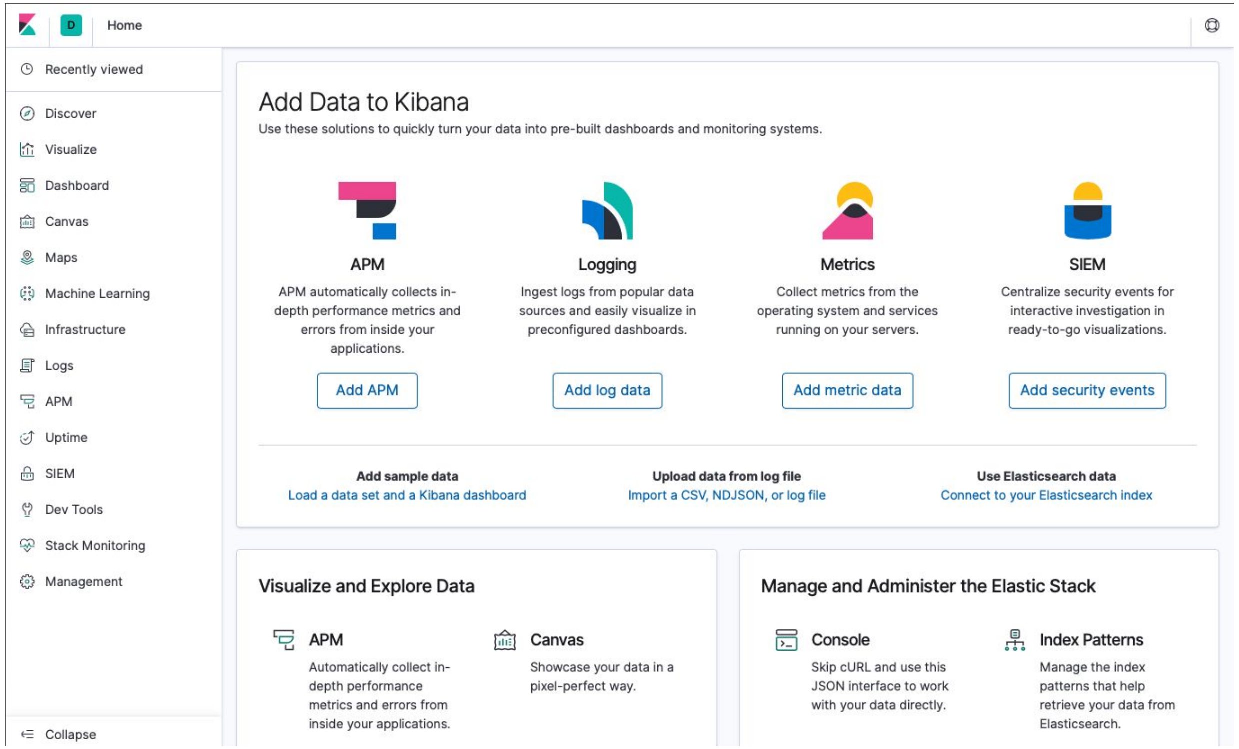
1. Run Elasticsearch bin/elasticsearch
2. Open localhost:9200



```
{
  "name" : "zTJbz8l",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "ZMpUUNdIRZW6xtyycodIDg",
  "version" : {
    "number" : "6.2.3",
    "build_hash" : "c59ff00",
    "build_date" : "2018-03-13T10:06:29.741383Z",
    "build_snapshot" : false,
    "lucene_version" : "7.2.1",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

Run Kibana

1. Run Kibana bin/kibana
2. Open localhost:5601



Add Data to Kibana

Use these solutions to quickly turn your data into pre-built dashboards and monitoring systems.

APM

APM automatically collects in-depth performance metrics and errors from inside your applications.

Add APM

Logging

Ingest logs from popular data sources and easily visualize in preconfigured dashboards.

Add log data

Metrics

Collect metrics from the operating system and services running on your servers.

Add metric data

SIEM

Centralize security events for interactive investigation in ready-to-go visualizations.

Add security events

Add sample data
Load a data set and a Kibana dashboard

Upload data from log file
Import a CSV, NDJSON, or log file

Use Elasticsearch data
Connect to your Elasticsearch index

Visualize and Explore Data

APM
Automatically collect in-depth performance metrics and errors from inside your applications.

Canvas
Showcase your data in a pixel-perfect way.

Manage and Administer the Elastic Stack

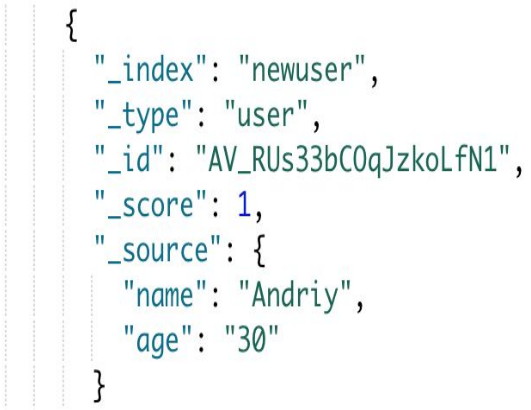
Console
Skip cURL and use this JSON interface to work with your data directly.

Index Patterns
Manage the index patterns that help retrieve your data from Elasticsearch.

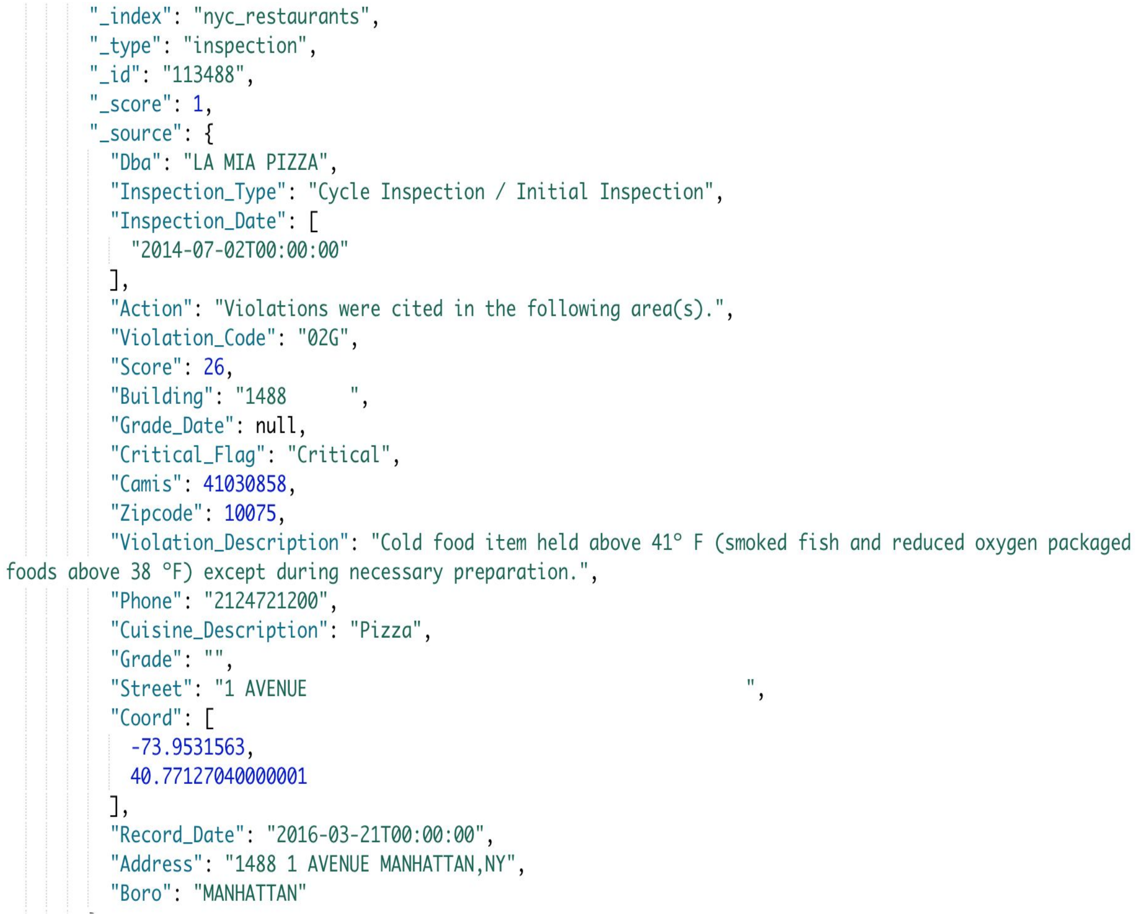
Terminology

Relation Databases		**Elasticsearch**
• Database	⟷	Index
• Table	⟷	Type
• Row	⟷	Document
• Column	⟷	Fields
• Schema	⟷	Mapping

Document sample



```
{
  "_index": "newuser",
  "_type": "user",
  "_id": "AV_RUs33bCOqJzkoLfN1",
  "_score": 1,
  "_source": {
    "name": "Andriy",
    "age": "30"
  }
```



```
"_index": "nyc_restaurants",
"_type": "inspection",
"_id": "113488",
"_score": 1,
"_source": {
  "Dba": "LA MIA PIZZA",
  "Inspection_Type": "Cycle Inspection / Initial Inspection",
  "Inspection_Date": [
    "2014-07-02T00:00:00"
  ],
  "Action": "Violations were cited in the following area(s).",
  "Violation_Code": "02G",
  "Score": 26,
  "Building": "1488      ",
  "Grade_Date": null,
  "Critical_Flag": "Critical",
  "Camis": 41030858,
  "Zipcode": 10075,
  "Violation_Description": "Cold food item held above 41° F (smoked fish and reduced oxygen packaged
foods above 38 °F) except during necessary preparation.",
  "Phone": "2124721200",
  "Cuisine_Description": "Pizza",
  "Grade": "",
  "Street": "1 AVENUE                                                        ",
  "Coord": [
    -73.9531563,
    40.77127040000001
  ],
  "Record_Date": "2016-03-21T00:00:00",
  "Address": "1488 1 AVENUE MANHATTAN,NY",
  "Boro": "MANHATTAN"
```

Mapping sample

```
PUT my_index
{
  "mappings": {
    "my_type": {
      "properties": {
        "full_text": {
          "type":  "text"
        },
        "exact_value": {
          "type":  "keyword"
        }
      }
    }
  }
}
```

ES field types

1. Basic data types (String, numeric, date, boolean, etc)
2. Geo data types (Geo-point, Geo-shape)
3. Specialized data types (IP, completion, join, alias, etc)

Analyzed vs Not Analyzed

Input: Hello Class 123

Analyzed (text)

Token1: hello **Token2:** class **Token3:** 123

Allows to search like: hello class, hello 123, etc

Pros – easy to find something in text

Cons – consumes more resources

Input: Hello Class 123

Not Analyzed (keyword)

Token1: Hello Class 123

Allows to search only: Hello Class 123

Pros – consumes less resources, very effective for search in log files

Cons – you should exactly know what you are searching for

Analyzed vs Not Analyzed use cases

Analyzed (text)

example: titles that contain the word "jobs".
query:"title:jobs".

doc1 : title:developer jobs in boston
doc2 : title:java coder jobs in vancuver
doc3 : title:unix designer jobs in austin
doc4 : title:database manager vacancies in montreal

this is going to retrieve title1, title2, title3

Not Analyzed (keyword)

example: get all the logs from machine 1**.**
query:”workstation:machine 1".

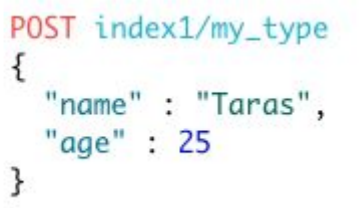
doc1: workstation:machine 1, log: somestring
doc2: workstation:machine 2, log: somestring
doc3: workstation:machine 1, log: somestring
doc4: workstation:machine 4, log: somestring

This is going to retrieve results from doc1 and doc3 only

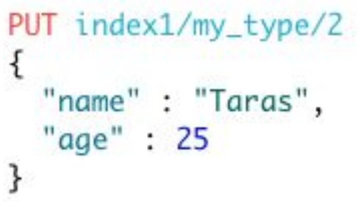
Demo

creating index, mapping, shards explanation

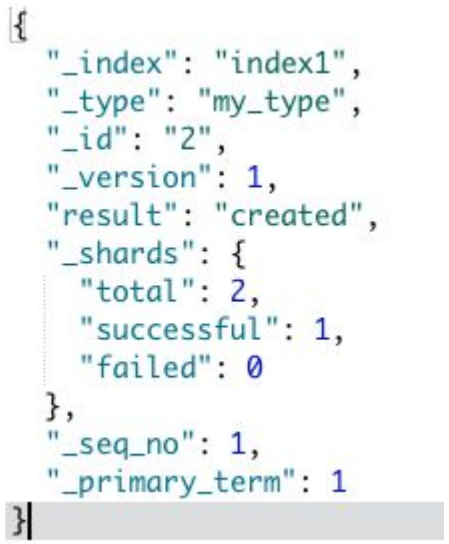
Add/update document



```
POST index1/my_type
{
  "name" : "Taras",
  "age" : 25
}
```



```
PUT index1/my_type/2
{
  "name" : "Taras",
  "age" : 25
}
```



```
{
  "_index": "index1",
  "_type": "my_type",
  "_id": "2",
  "_version": 1,
  "result": "created",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 1,
  "_primary_term": 1
}
```

Range search

Add couple documents:

```
POST index1/my_type
{
  "name" : "Bob",
  "age" : "70"
}
POST index1/my_type
{
  "name" : "Andriy",
  "age" : "30"
}
```

Query

GET /textsearch/_search

```
{  "query": {
        "range" : {
            "age" : { "gte" : 30,
                          "lte" : 69   }
        }
    }
}
```

Wildcard search

Add couple documents:

```
POST index1/my_type
{
  "name" : "Bob",
  "age" : "70"
}
POST index1/my_type
{
  "name" : "Andriy",
  "age" : "30"
}
```

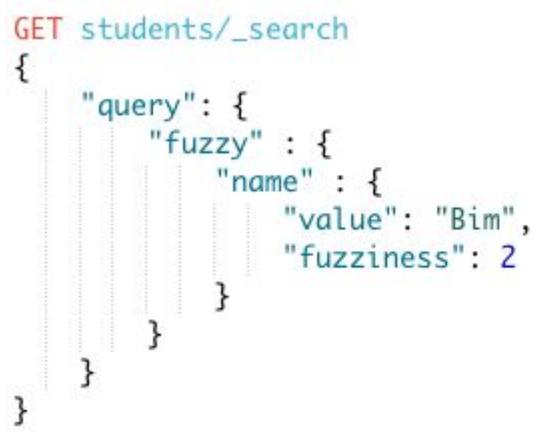
Query

```
GET /textsearch/_search
{  "query": {
        "wildcard" : {
            "name” : “b*”
        }
    }
}
```

Fuzzy search

Add couple documents:

```
POST students/student
{
  "name" : "Bob",
  "age" : "20"
}
POST students/student
{
  "name" : "Andriy",
  "age" : "30"
}
```



```
GET students/_search
{
    "query": {
        "fuzzy" : {
            "name" : {
                "value": "Bim",
                "fuzziness": 2
            }
        }
    }
}
```

Exists Query

Returns documents that have at least one non-null value in the original field:

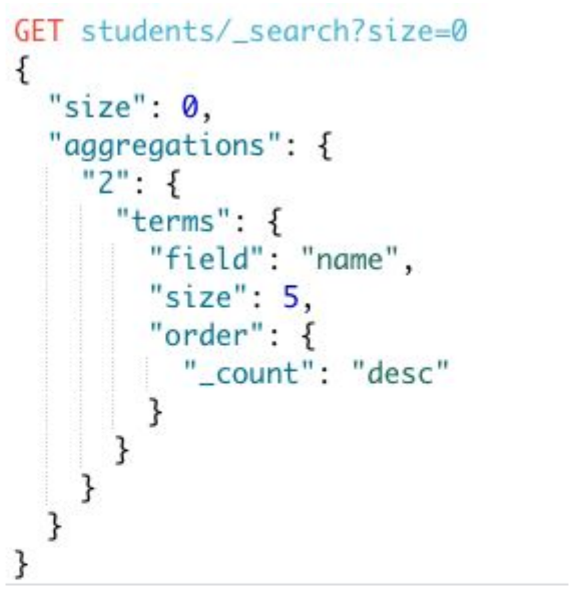
```
GET textsearch/_search
{
   "query": {
      "exists" : { "field" : "name" }
   }
}
```

Doesn’t exists Query

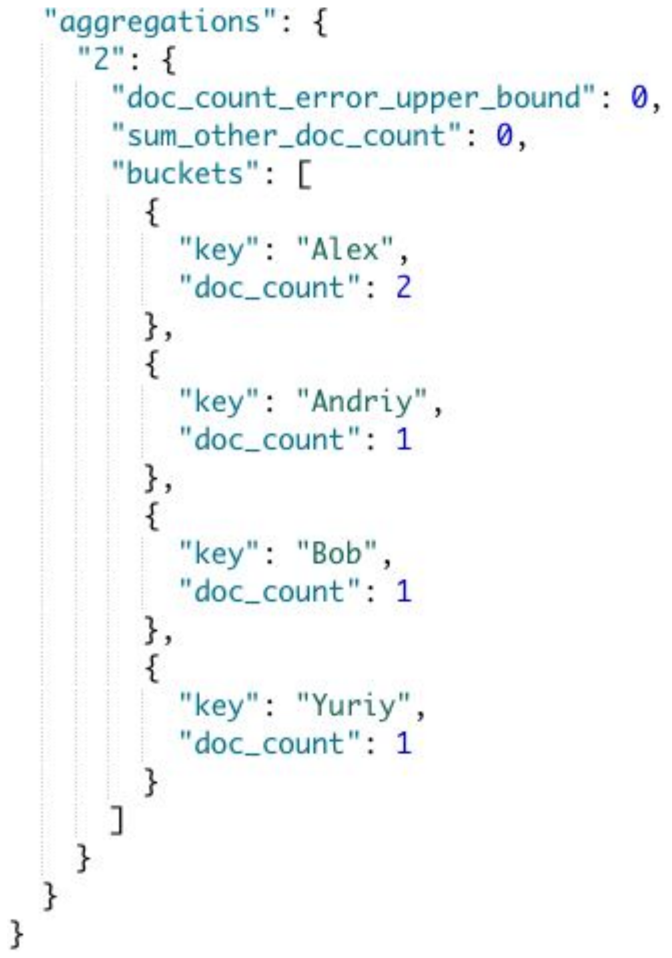
Returns documents that have at least one non-null value in the original field:

```
GET textsearch/_search
{
   "query": {
         “bool”: {
          “must_not”: {
      "exists" : { "field" : "name" }
   }
}}}
```

Aggregation (Top)



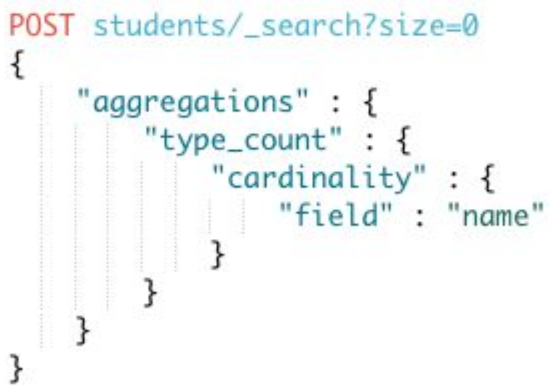
```
GET students/_search?size=0
{
  "size": 0,
  "aggregations": {
    "2": {
      "terms": {
        "field": "name",
        "size": 5,
        "order": {
          "_count": "desc"
        }
      }
    }
  }
}
```



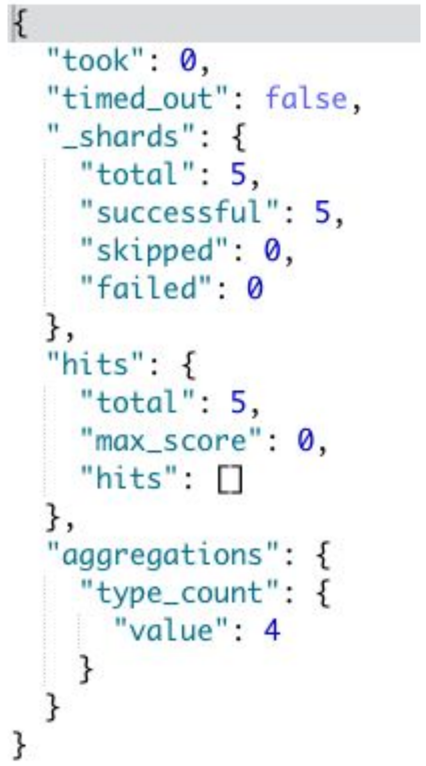
```
  "aggregations": {
    "2": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "Alex",
          "doc_count": 2
        },
        {
          "key": "Andriy",
          "doc_count": 1
        },
        {
          "key": "Bob",
          "doc_count": 1
        },
        {
          "key": "Yuriy",
          "doc_count": 1
        }
      ]
    }
  }
}
```

@Marina Popova

Aggregation (cardinality)

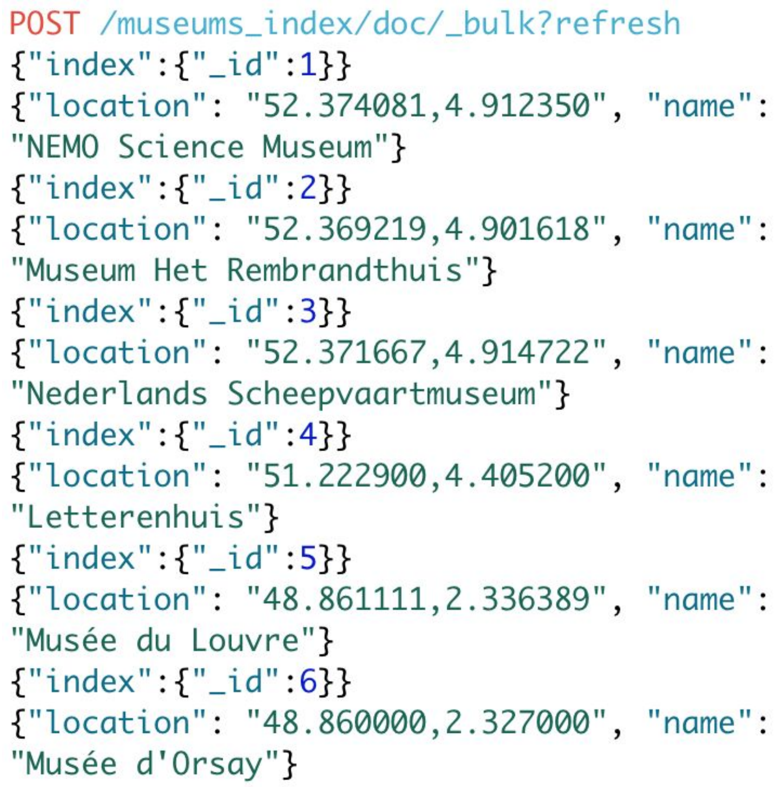


```
POST students/_search?size=0
{
    "aggregations" : {
        "type_count" : {
            "cardinality" : {
                "field" : "name"
            }
        }
    }
}
```

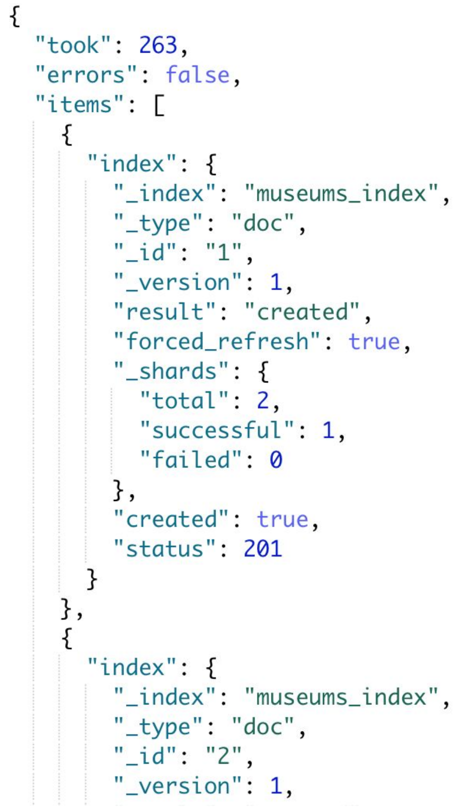


```
{
  "took": 0,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 5,
    "max_score": 0,
    "hits": []
  },
  "aggregations": {
    "type_count": {
      "value": 4
    }
  }
}
```

Bulk add

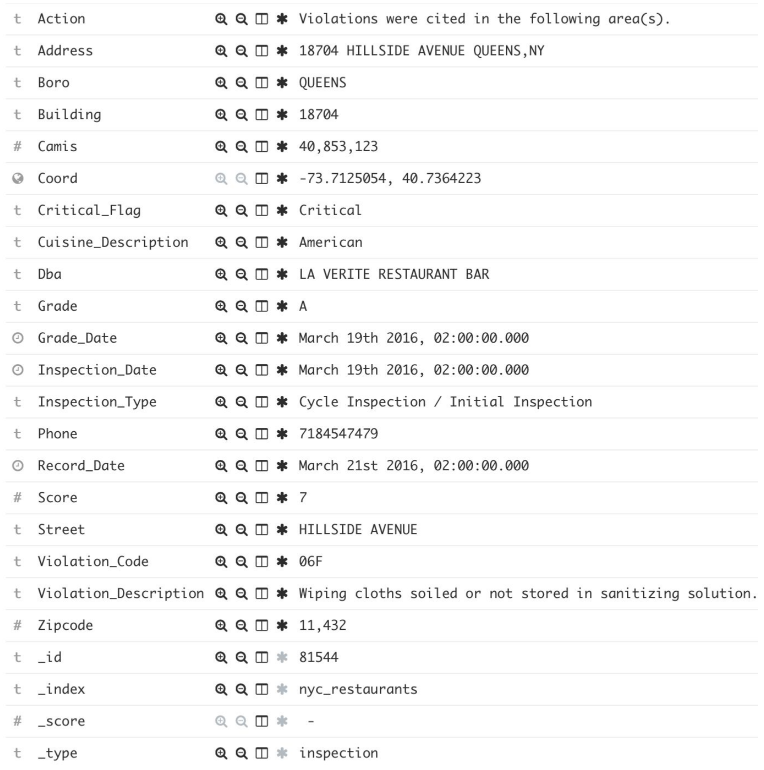


```
POST /museums_index/doc/_bulk?refresh
{"index":{"_id":1}}
{"location": "52.374081,4.912350", "name":
"NEMO Science Museum"}
{"index":{"_id":2}}
{"location": "52.369219,4.901618", "name":
"Museum Het Rembrandthuis"}
{"index":{"_id":3}}
{"location": "52.371667,4.914722", "name":
"Nederlands Scheepvaartmuseum"}
{"index":{"_id":4}}
{"location": "51.222900,4.405200", "name":
"Letterenhuis"}
{"index":{"_id":5}}
{"location": "48.861111,2.336389", "name":
"Musée du Louvre"}
{"index":{"_id":6}}
{"location": "48.860000,2.327000", "name":
"Musée d'Orsay"}
```



```
{
  "took": 263,
  "errors": false,
  "items": [
    {
      "index": {
        "_index": "museums_index",
        "_type": "doc",
        "_id": "1",
        "_version": 1,
        "result": "created",
        "forced_refresh": true,
        "_shards": {
          "total": 2,
          "successful": 1,
          "failed": 0
        },
        "created": true,
        "status": 201
      }
    },
    {
      "index": {
        "_index": "museums_index",
        "_type": "doc",
        "_id": "2",
        "_version": 1,
```

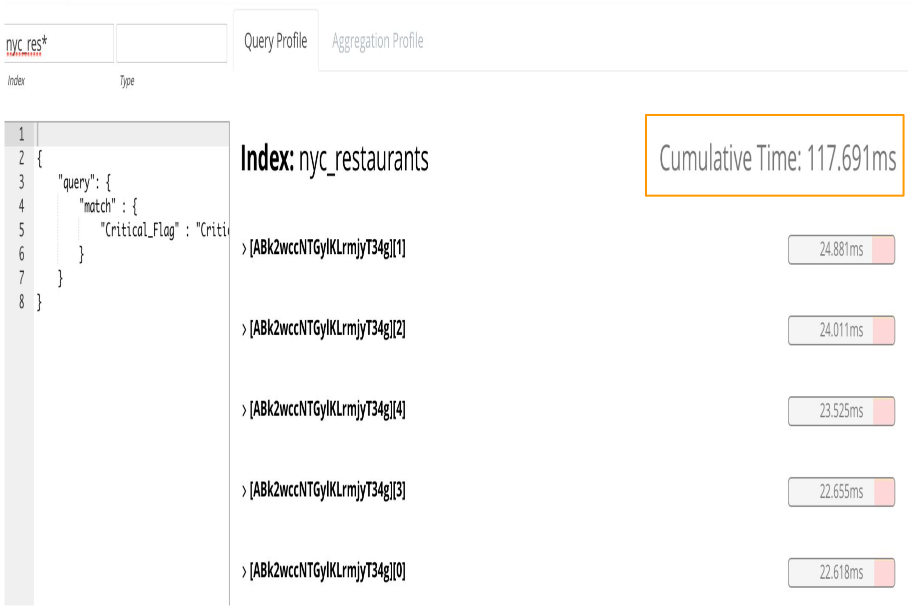
NYC Restaurant Inspection Data structure



t	Action	Violations were cited in the following area(s).
t	Address	18704 HILLSIDE AVENUE QUEENS,NY
t	Boro	QUEENS
t	Building	18704
#	Camis	40,853,123
	Coord	-73.7125054, 40.7364223
t	Critical_Flag	Critical
t	Cuisine_Description	American
t	Dba	LA VERITE RESTAURANT BAR
t	Grade	A
	Grade_Date	March 19th 2016, 02:00:00.000
	Inspection_Date	March 19th 2016, 02:00:00.000
t	Inspection_Type	Cycle Inspection / Initial Inspection
t	Phone	7184547479
	Record_Date	March 21st 2016, 02:00:00.000
#	Score	7
t	Street	HILLSIDE AVENUE
t	Violation_Code	06F
t	Violation_Description	Wiping cloths soiled or not stored in sanitizing solution.
#	Zipcode	11,432
t	_id	81544
t	_index	nyc_restaurants
#	_score	-
t	_type	inspection

@Marina Popova

Wildcard search - profiling



nyc_res*

Index Type

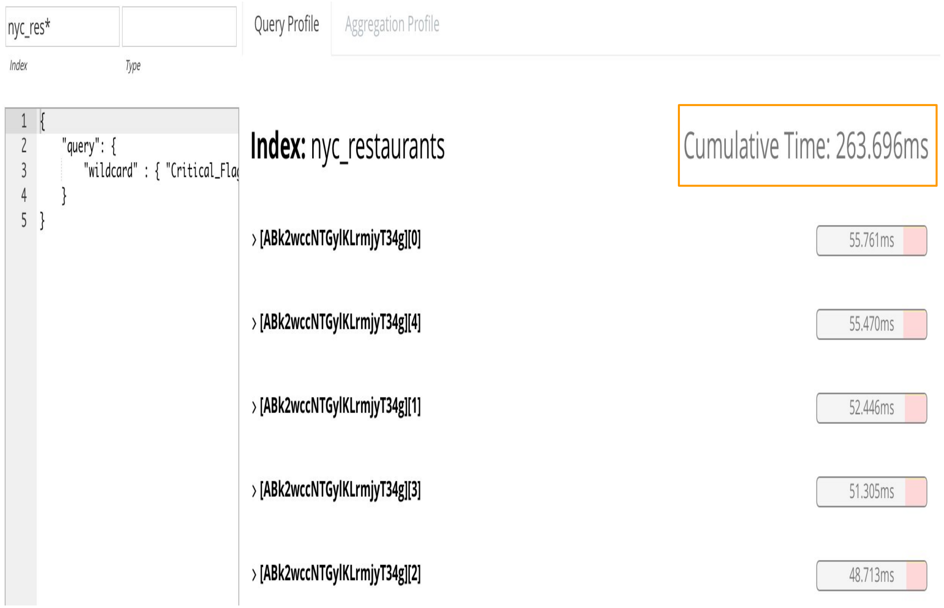
Query Profile Aggregation Profile

```
{
    "query": {
        "match" : {
            "Critical_Flag" : "Criti
        }
    }
}
```

Index: nyc_restaurants

Cumulative Time: 117.691ms

```
› [ABk2wccNTGylKLrmjyT34g][1]   24.881ms
› [ABk2wccNTGylKLrmjyT34g][2]   24.011ms
› [ABk2wccNTGylKLrmjyT34g][4]   23.525ms
› [ABk2wccNTGylKLrmjyT34g][3]   22.655ms
› [ABk2wccNTGylKLrmjyT34g][0]   22.618ms
```



nyc_res*

Index Type

Query Profile Aggregation Profile

```
{
    "query": {
        "wildcard" : { "Critical_Flag
    }
}
```

Index: nyc_restaurants

Cumulative Time: 263.696ms

```
› [ABk2wccNTGylKLrmjyT34g][0]   55.761ms
› [ABk2wccNTGylKLrmjyT34g][4]   55.470ms
› [ABk2wccNTGylKLrmjyT34g][1]   52.446ms
› [ABk2wccNTGylKLrmjyT34g][3]   51.305ms
› [ABk2wccNTGylKLrmjyT34g][2]   48.713ms
```

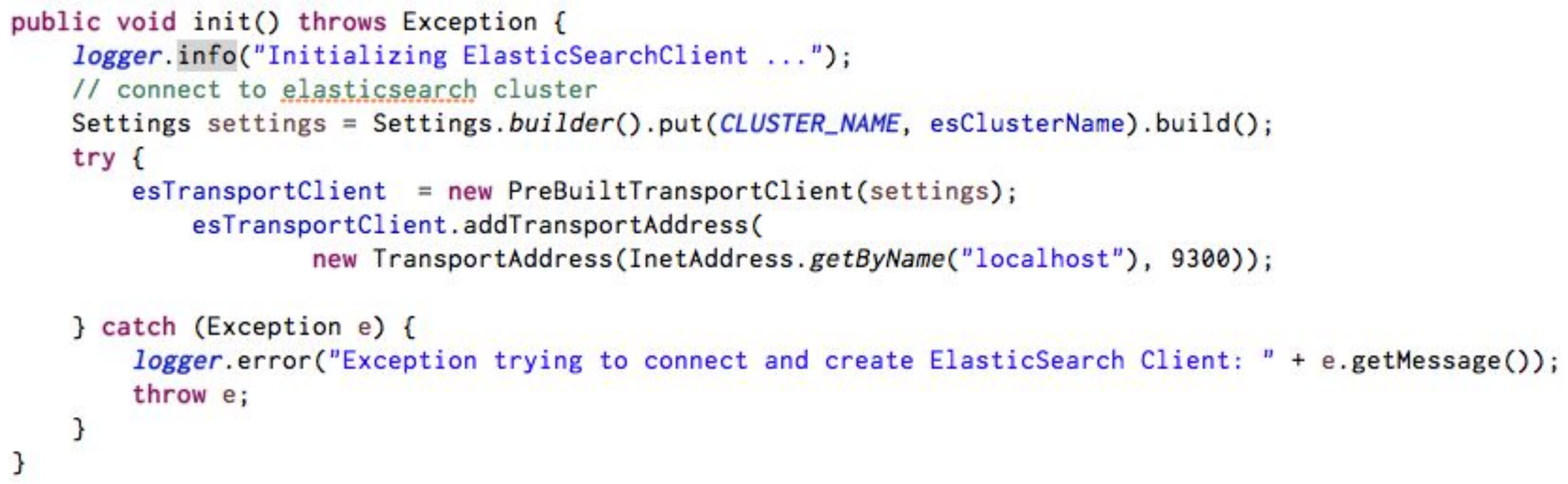
@Marina Popova

Cardinality aggregation

A single-value metrics aggregation that calculates an approximate count of distinct values.

```
GET /nyc_res*/_search?size=0
{
    "aggs" : {
        "type_count" : {
            "cardinality" : {
                "field" : "Zipcode"
            }
        }
    }
}
GET /nyc_res*/_search?size=0
{
    "aggs" : {
        "type_count" : {
            "cardinality" : {
                "field" : "Cuisine_Description"
            }
        }
    }
}
```

Java connect to Elasticsearch (versions 6+)



```
public void init() throws Exception {
    logger.info("Initializing ElasticSearchClient ...");
    // connect to elasticsearch cluster
    Settings settings = Settings.builder().put(CLUSTER_NAME, esClusterName).build();
    try {
        esTransportClient  = new PreBuiltTransportClient(settings);
            esTransportClient.addTransportAddress(
                    new TransportAddress(InetAddress.getByName("localhost"), 9300));

    } catch (Exception e) {
        logger.error("Exception trying to connect and create ElasticSearch Client: " + e.getMessage());
        throw e;
    }
}
```

Python application

```python
from datetime import datetime
from elasticsearch import Elasticsearch
es = Elasticsearch()

doc = {
    'author': 'kimchy',
    'text': 'Elasticsearch: cool. bonsai cool.,'
    'timestamp': datetime.now(),
}
res = es.index(index="test-index", doc_type='tweet', id=1, body=doc)
print(res['result'])

res = es.get(index="test-index", doc_type='tweet', id=1)
print(res['_source'])

es.indices.refresh(index="test-index")

res = es.search(index="test-index", body={"query": {"match_all": {}}})
print("Got %d Hits:" % res['hits']['total'])
for hit in res['hits']['hits']:
    print("%(timestamp)s %(author)s: %(text)s" % hit["_source"])
```

```
apysh@macbook325:~$ sudo python es.py
created
{u'text': u'Elasticsearch: cool. bonsai cool.', u'author': u'kimchy', u'timestamp':
u'2018-11-18T17:16:04.563802'}
Got 1 Hits:
2018-11-18T17:16:04.563802 kimchy: Elasticsearch: cool. bonsai cool.
```

Demo

Visualizations and Dashboards

Useful commands and links

{host}:9200/_cluster/health - shows cluster health

{host}:9200/_cluster/stats - shows cluster stats

{host}:9200/_cat - cat api, shows all possible options with cat api

{host}:9200/_cat/indices - shows all indices

Add ?v in the end of call - to have headers for table output (_cat api)

Add ?pretty - to have nice json output (cluster health)

https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html - ES documentation with good samples

https://github.com/elastic/examples/tree/master/Exploring%20Public%20Datasets/nyc_restaurants - New York City restaurant inspection data