

Lecture 3: Overview of Elasticsearch

Friday, June 7, 2019 11:43 AM

4. Lecture 3: Overview of Elasticsearch

Elasticsearch is Document Oriented

- **Insert** Documents
- **Delete** Documents
- **Retrieve** Documents
- **Analyze** Documents
- **Search** Documents



Inverted Index

Maps words to the actual document locations of where they occur

Docs

1. The quick brown fox jumped over the lazy dog
2. Quick brown foxes leap over lazy dogs in summer

To create an inverted index, we first split the `content` field of each document into separate words (which we call *terms*, or *tokens*), create a sorted list of all the unique terms, and then list in which document each term appears. The result looks something like this:

Term	Doc_1	Doc_2
Quick	X	X
The	X	
brown	X	X
dog	X	
dogs		X
fox	X	
foxes		X
in		X
jumped	X	
lazy	X	X
leap		X
over	X	X
quick	X	
summer		X
the	X	

Getting Started

- + You Know, for Search...
- + Life Inside a Cluster
- + Data In, Data Out
- + Distributed Document Store
- + Searching—The Basic Tools
- Mapping and Analysis

Exact Values Versus Full Text

Inverted Index

Analysis and Analyzers

Mapping

Complex Core Field Types

+ Full-Body Search

+ Sorting and Relevance

+ Distributed Search Execution

+ Index Management

+ Inside a Shard

+ Search in Depth

Docs

dogs		X
fox	X	
foxes		X
in		X
jumped	X	
lazy	X	X
leap		X
over	X	X
quick	X	
summer		X
the	X	

Complex Core Field Types

- + Full-Body Search
- + Sorting and Relevance
- + Distributed Search Execution
- + Index Management
- + Inside a Shard

+ Search in Depth

+ Dealing with Human Language

+ Aggregations

+ Geolocation

+ Modeling Your Data

+ Administration, Monitoring, and Deployment

Now, if we want to search for `quick brown`, we just need to find the documents in which each term appears:

Term	Doc_1	Doc_2
brown	X	X
quick	X	
Total	2	1

Both documents match, but the first document has more matches than the second. If we apply a naive *similarity algorithm* that just counts the number of matching terms, then we can say that the first document is a better match—is *more relevant* to our query—than the second document.

Elasticsearch is Document Oriented

```
{  
    "total": 2,  
    "employees": [  
        {  
            "name": "David",  
            "id": 1101,  
            "salary": 41000,  
            "hiredate": "October 3, 1998",  
            "department": "admin"  
        },  
        {  
            "name": "Michelle",  
            "id": 1101,  
            "salary": 45000,  
            "hiredate": "April 7, 2008",  
            "department": "Research"  
        }  
    ]  
}
```

```
{  
    "total": 6,  
    "Type": "employees",  
    "documents": [  
        {  
            "name": "David",  
            "id": 1101,  
            "salary": 41000,  
            "hiredate": "October 3, 1998",  
            "department": "admin"  
        },  
        {  
            "name": "Michelle",  
            "id": 1103,  
            "salary": 45000,  
            "hiredate": "April 7, 2008",  
            "department": "Research"  
        },  
        {  
            "name": "Cassandra",  
            "id": 1102,  
            "salary": 68000,  
            "hiredate": "January 12, 2001",  
            "department": "Sales"  
        },  
        {  
            "name": "Brian",  
            "id": 1104,  
            "salary": 37000,  
            "hiredate": "August 19, 2012",  
            "department": "Admin"  
        },  
        {  
            "name": "Jason",  
            "id": 1105,  
            "salary": 92000,  
            "hiredate": "March 15, 2013",  
            "department": "Sales"  
        },  
        {  
            "name": "Robert",  
            "id": 1106,  
            "salary": 43000,  
            "hiredate": "January 11, 2014",  
            "department": "Sales"  
        }  
    ]  
}
```

Elasticsearch is
Document Oriented

Employees Table in a Relational Database

EMPLOYEE_ID	NAME	SALARY	HIREDATE	DEPARTMENT
1101	David	41000	October 3, 1998	Admin
1102	Cassandra	68000	January 12, 2001	Sales
1103	Michelle	45000	April 7, 2008	Research
1104	Brian	37000	August 19, 2012	Admin
1105	Jason	92000	March 15, 2013	Sales
1106	Robert	43000	January 11, 2014	Sales

- <https://www.elastic.co/guide/en/elasticsearch/guide/current/document.html>

Lecture 4: Indexing, Retrieving, and Deleting Documents

Friday, June 7, 2019 2:04 PM

How Elasticsearch relates to a database

Relational DB	Elasticsearch 6
Table	Index
Row	Document
Column	Field

Index > Type > Document > Field

- **Vehicles (index)**

- Car (document 1)
- Car (document 2)
- Car (document n)

```
{  
  "type": "car",  
  "documents": [  
    {  
      "id": 912843,  
      "make": "Honda",  
      "Color": "red",  
      "purchase Date": "Oct 3, 1998",  
      "Milage": 120000  
    },  
    {  
      "id": 925063,  
      "make": "Toyota",  
      "Color": "Blue",  
      "purchase Date": "Sept 22, 2008",  
      "Milage": 18000  
    }  
  ]  
}
```



- Inserting = Indexing

```
PUT /{index}/{type}/{id}
{
    "field1": "value1",
    "field2": "value2",
    ...
}
```

- `_version` is incremented via overwriting the existing document
- anything with an `_` is considered a meta element
- frequent updates, additions, and deprecations
- documents are immutable
- PUT, POST, GET, HEAD, DELETE

The screenshot shows a terminal window with two panes. The left pane contains the following JSON document:

```
1 PUT /vehicles/car/123
2 {
3     "make": "Honda",
4     "Color": "blue",
5     "HP": 250,
6     "milage": 19000,
7     "price": 19300.97
8 }
```

The right pane shows the response from the server, which includes the updated document and its metadata:

```
1 {
2     "_index": "vehicles",
3     "_type": "car",
4     "_id": "123",
5     "_version": 6,
6     "result": "updated",
7     "_shards": {
8         "total": 2,
9         "successful": 1,
10        "failed": 0
11    },
12    "created": false
13 }
```

```
1 POST /vehicles/car/123/_update
2 {
3     "doc": {
4         "driver": "Tom"
5     }
6 }
```

```
1 { "_index": "vehicles", "_type": "car", "_id": "123", "_version": 8, "result": "updated", "_shards": { "total": 2, "successful": 1, "failed": 0 } }
2
3
4
5
6
7
8
9
10
11
12 }
```

```
1 DELETE /vehicles/car/123
```

```
1 { "found": true, "_index": "vehicles", "_type": "car", "_id": "123", "_version": 9, "result": "deleted", "_shards": { "total": 2, "successful": 1, "failed": 0 } }
2
3
4
5
6
7
8
9
10
11
12
13 }
```

- when you **DELETE** a document, you can no longer access it via **GET**, but it still exists
 - Elasticsearch marks those documents as **DELETED**, and every so often behind the scenes, it collects all documents marked as **DELETED** and wipes them off the disk

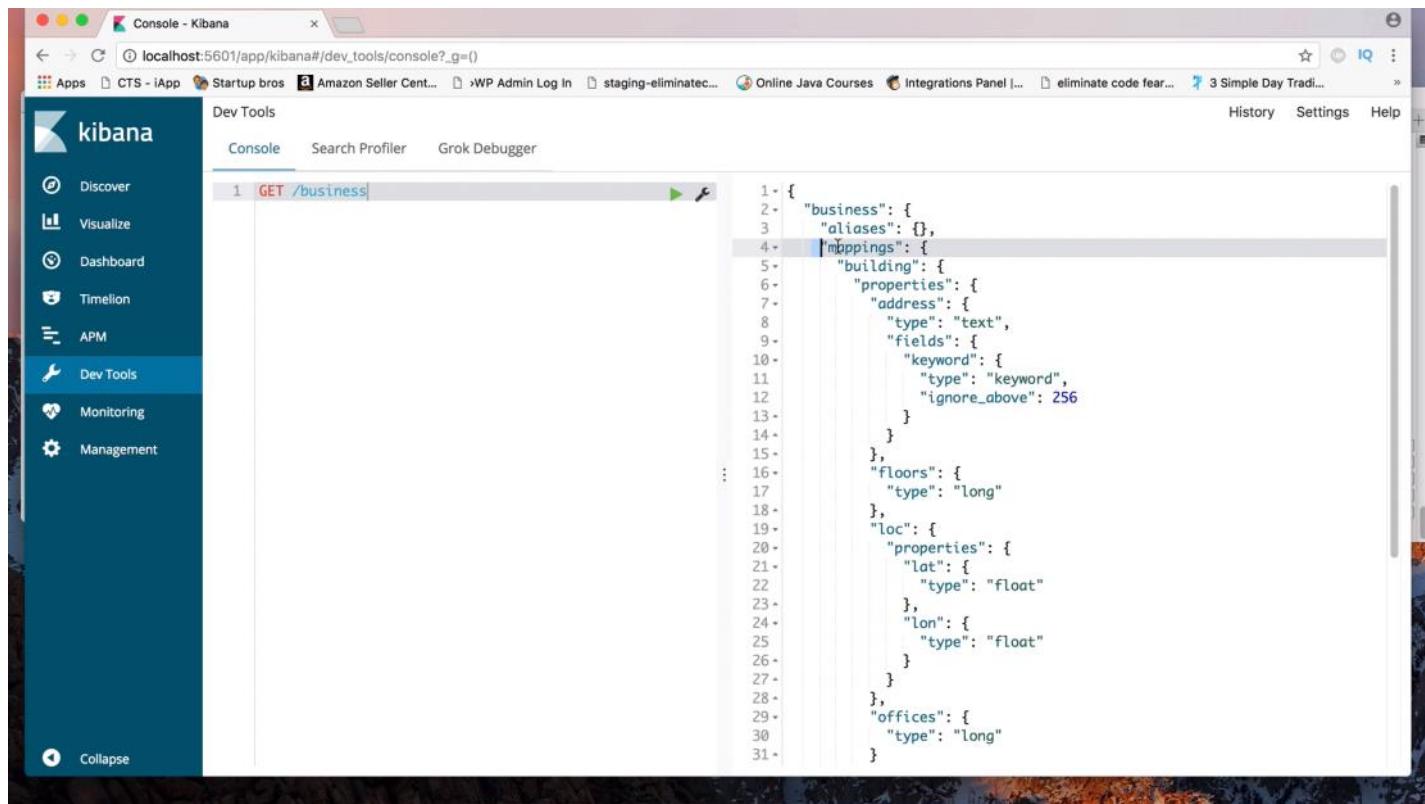
```
1 GET /vehicles
33     "ignore_above": 256
34     }
35   }
36 }
37   "milage": {
38     "type": "long"
39   },
40   "price": {
41     "type": "float"
42   }
43 }
44 }
45 },
46   "settings": {
47     "index": {
48       "creation_date": "1495762580661",
49       "number_of_shards": "5",
50       "number_of_replicas": "1",
51       "uuid": "qjhySZ4iR4CVRDuF0-5DIw",
52       "version": {
53         "created": "5040099"
54       },
55       "provided_name": "vehicles"
56     }
57   }
58 }
59 }
```

- doing a GET command on an index will provide the structure of that index, auto-generated via Elasticsearch
- *make sure to look out for ES/Kibana updates in interfacing!*

Lecture 5: Components of an Index

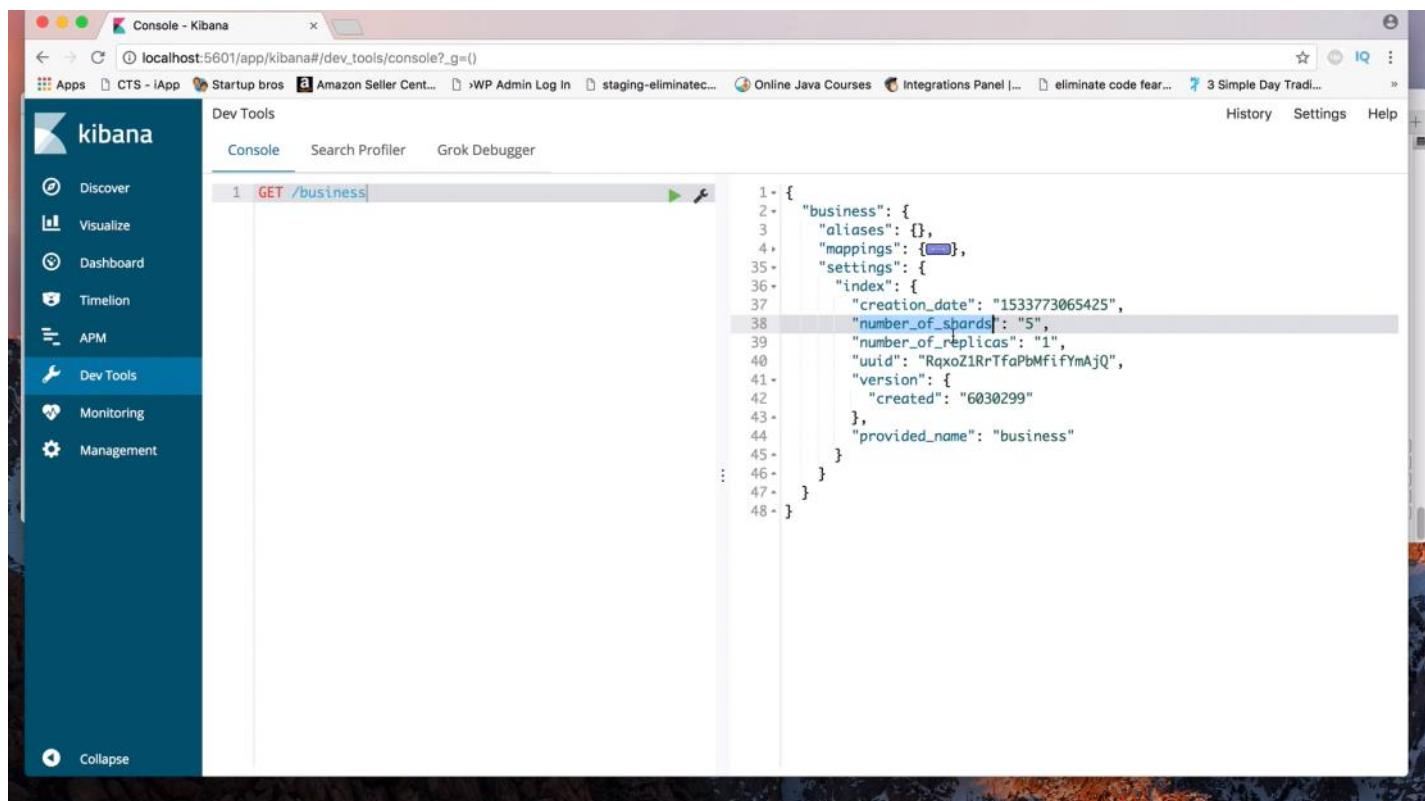
Friday, June 7, 2019 3:52 PM

- **Index name, aliases, mappings (structure of fields w/properties [auto-generated]), settings (shards, replicas, IDs, etc.)**



The screenshot shows the Kibana Dev Tools Console interface. The left sidebar has 'Dev Tools' selected. The main area shows a 'Console' tab with the URL 'localhost:5601/app/kibana#/dev_tools/console?_g=()' and a query 'GET /business'. The response is a JSON object with the following content:

```
1. {  
2.   "business": {  
3.     "aliases": {},  
4.     "mappings": {  
5.       "building": {  
6.         "properties": {  
7.           "address": {  
8.             "type": "text",  
9.             "fields": {  
10.               "keyword": {  
11.                 "type": "keyword",  
12.                 "ignore_above": 256  
13.               }  
14.             }  
15.           },  
16.           "floors": {  
17.             "type": "long"  
18.           },  
19.           "loc": {  
20.             "properties": {  
21.               "lat": {  
22.                 "type": "float"  
23.               },  
24.               "lon": {  
25.                 "type": "float"  
26.               }  
27.             }  
28.           },  
29.           "offices": {  
30.             "type": "long"  
31.           }  
32.     }  
33.   }  
34. }
```



The screenshot shows the Kibana Dev Tools Console interface. The left sidebar has 'Dev Tools' selected. The main area shows a 'Console' tab with the URL 'localhost:5601/app/kibana#/dev_tools/console?_g=()' and a query 'GET /business'. The response is a JSON object with the following content, highlighting the 'settings' field:

```
1. {  
2.   "business": {  
3.     "aliases": {},  
4.     "mappings": {},  
5.     "settings": {  
6.       "index": {  
7.         "creation_date": "1533773065425",  
8.         "number_of_shards": "5",  
9.         "number_of_replicas": "1",  
10.        "uuid": "RqxoZ1RrTfaPbMflfYmAjQ",  
11.        "version": {  
12.          "created": "6030299"  
13.        },  
14.        "provided_name": "business"  
15.      }  
16.    }  
17.  }  
18. }
```

- ES able to dynamically change index structure to accommodate new fields

The screenshot shows the Kibana Dev Tools Console interface. On the left, there's a sidebar with various navigation options like Discover, Visualize, Dashboard, Timeline, APM, Dev Tools (which is selected), Monitoring, and Management. The main area is titled "Dev Tools" and has tabs for Console, Search Profiler, and Grok Debugger. In the "Console" tab, a code editor displays a PUT request:

```
1 PUT /business/employee/330
2 {
3   "name": "Richard Bell",
4   "title": "Senior Accountant",
5   "salary_usd": 115000.00,
6   "hiredate": "Jan 19, 2013"
7
8 }
```

After the request, an error response is shown:

```
1 {
2   "error": {
3     "root_cause": [
4       {
5         "type": "illegal_argument_exception",
6         "reason": "Rejecting mapping update to [business] as the final
mapping would have more than 1 type: [employee, building]"
7       }
8     ],
9     "type": "illegal_argument_exception",
10    "reason": "Rejecting mapping update to [business] as the final
mapping would have more than 1 type: [employee, building]"
11  },
12  "status": 400
13 }
```

- For ES6, multi-mapping to a specific index is NOT allowed
- Must make a new index to map a new type of object

The screenshot shows the Kibana Dev Tools Console interface. The sidebar and tabs are identical to the previous screenshot. The "Console" tab contains a GET request:

```
1 GET business/building/_search
2
```

After the request, a successful search response is displayed:

```
1 {
2   "took": 0,
3   "timed_out": false,
4   "_shards": {
5     "total": 5,
6     "successful": 5,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": 2,
12    "max_score": 1,
13    "hits": [
14      {
15        "_index": "business",
16        "_type": "building",
17        "_id": "110",
18        "_score": 1,
19        "_source": {
20          "address": "57 New Dover Ln",
21          "floors": 10,
22          "offices": 21,
23          "loc": {
24            "lat": 40.707519,
25            "lon": -74.00856
26          }
27        }
28      },
29      {
30        "_index": "business",
31        "_type": "building",
32        ...
33      }
34    ]
35  }
```

- HITS array contains all documents that are indexed

The screenshot shows the Kibana Dev Tools Console interface. On the left, there's a sidebar with various navigation options like Discover, Visualize, Dashboard, Timelion, APM, Dev Tools (which is selected), Monitoring, and Management. The main area is titled 'Dev Tools' and has tabs for Console, Search Profiler, and Grok Debugger. In the 'Console' tab, a code editor displays a GET request to the 'business/_search' index. The query part of the request is a 'term' query: "query": {"term": {"address": "pen"}}, which is highlighted in blue. The response from the server is shown below, containing the search results for documents where the address field contains 'pen'. One result is highlighted in blue, showing details like '_index': 'business', '_type': 'building', '_id': '217', '_score': 0.2876821, and '_source': { 'address': '11 Pen Ave', 'floors': 5, 'offices': 7, 'price': 450000, 'loc': { 'lat': 40.693519, 'lon': -73.98856 } }.

```

1 GET business/_search
2 {
3   "query": {
4     "term": {
5       "address": "pen"
6     }
7   }
8 }

```

```

1 {
2   "took": 0,
3   "timed_out": false,
4   "_shards": {
5     "total": 5,
6     "successful": 5,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": 1,
12    "max_score": 0.2876821,
13    "hits": [
14      {
15        "_index": "business",
16        "_type": "building",
17        "_id": "217",
18        "_score": 0.2876821,
19        "_source": {
20          "address": "11 Pen Ave",
21          "floors": 5,
22          "offices": 7,
23          "price": 450000,
24          "loc": {
25            "lat": 40.693519,
26            "lon": -73.98856
27          }
28        }
29      }
30    ]
31  }

```

- The ABOVE code is known as a term query
- Here, the query specifies a certain term/property to search for, and matches the string provided with the ones in the indices

This screenshot is similar to the previous one but shows a different query type. The code editor now contains a 'match_all' query: "query": {"match_all": {}}, which is highlighted in blue. The response shows results for all documents in the 'business' index, with two hits found. The first hit is highlighted in blue, showing details like '_index': 'business', '_type': 'building', '_id': '110', '_score': 1, and '_source': { 'address': '57 New Dover Ln', 'floors': 10, 'offices': 21, 'loc': { 'lat': 40.707519, 'lon': -74.00856 } }.

```

1 GET business/_search
2 {
3   "query": {
4     "match_all": {}
5   }
6 }

```

```

1 {
2   "took": 0,
3   "timed_out": false,
4   "_shards": {
5     "total": 5,
6     "successful": 5,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": 2,
12    "max_score": 1,
13    "hits": [
14      {
15        "_index": "business",
16        "_type": "building",
17        "_id": "110",
18        "_score": 1,
19        "_source": {
20          "address": "57 New Dover Ln",
21          "floors": 10,
22          "offices": 21,
23          "loc": {
24            "lat": 40.707519,
25            "lon": -74.00856
26          }
27        }
28      },
29      {
30        "_index": "business",
31        "_type": "building",
32        ...
33      }
34    ]
35  }

```

- The ABOVE is known as a match_all query, which provides all the documents in the index (similar to just searching)

```
on red to green - Ready
log [20:35:51.731] [info][status][plugin:searchprofiler@6.3.2] Status change
d from red to green - Ready
log [20:35:51.732] [info][status][plugin:ml@6.3.2] Status changed from red t
o green - Ready
log [20:35:51.732] [info][status][plugin:tile...]
red to green - Ready
log [20:35:51.733] [info][status][plugin:watc...
red to green - Ready
log [20:35:51.734] [info][status][plugin:inde...
ged from red to green - Ready
log [20:35:51.734] [info][status][plugin:grap...
d to green - Ready
log [20:35:51.735] [info][status][plugin:secu...
red to green - Ready
log [20:35:51.735] [info][status][plugin:grok...
from red to green - Ready
log [20:35:51.736] [info][status][plugin:logs...
red to green - Ready
log [20:35:51.736] [info][status][plugin:repo...
n red to green - Ready
log [20:35:54.795] [info][license][xpack] Imp...
Elasticsearch for the [monitoring] cluster: mode:
```

- Go to wrench icon, click "Copy as cURL"
- Paste in terminal to retrieve same index and documentation
- Add "?pretty" to the end of the request to format the response

Lecture 6: Distributed Execution of Requests

Tuesday, June 11, 2019 11:21 AM

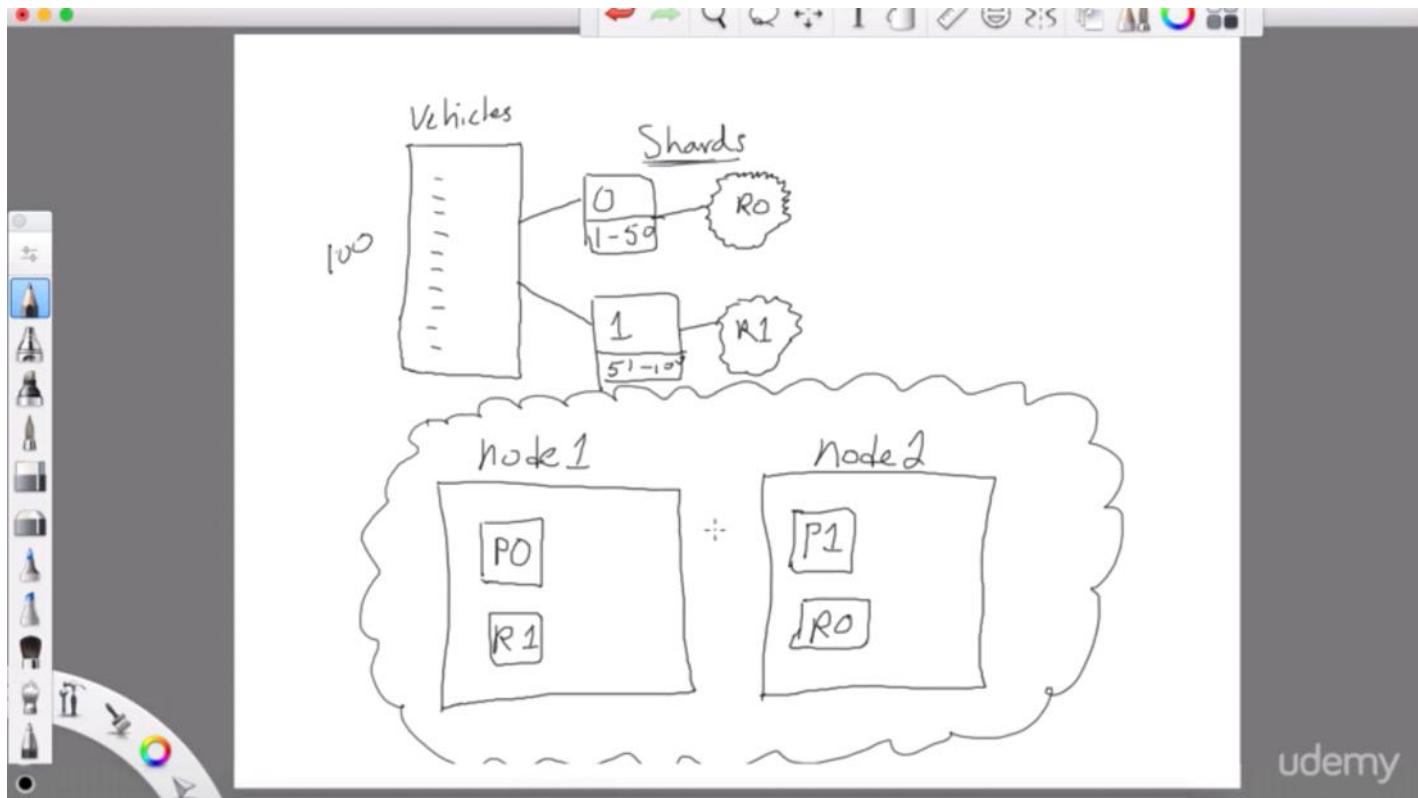
- ES is a distributed technology, meaning that devices on the same network running ES can fluidly communicate with each other, resulting in a cluster
- A cluster can efficiently distribute large search problems to various nodes, reducing runtime and increasing throughput



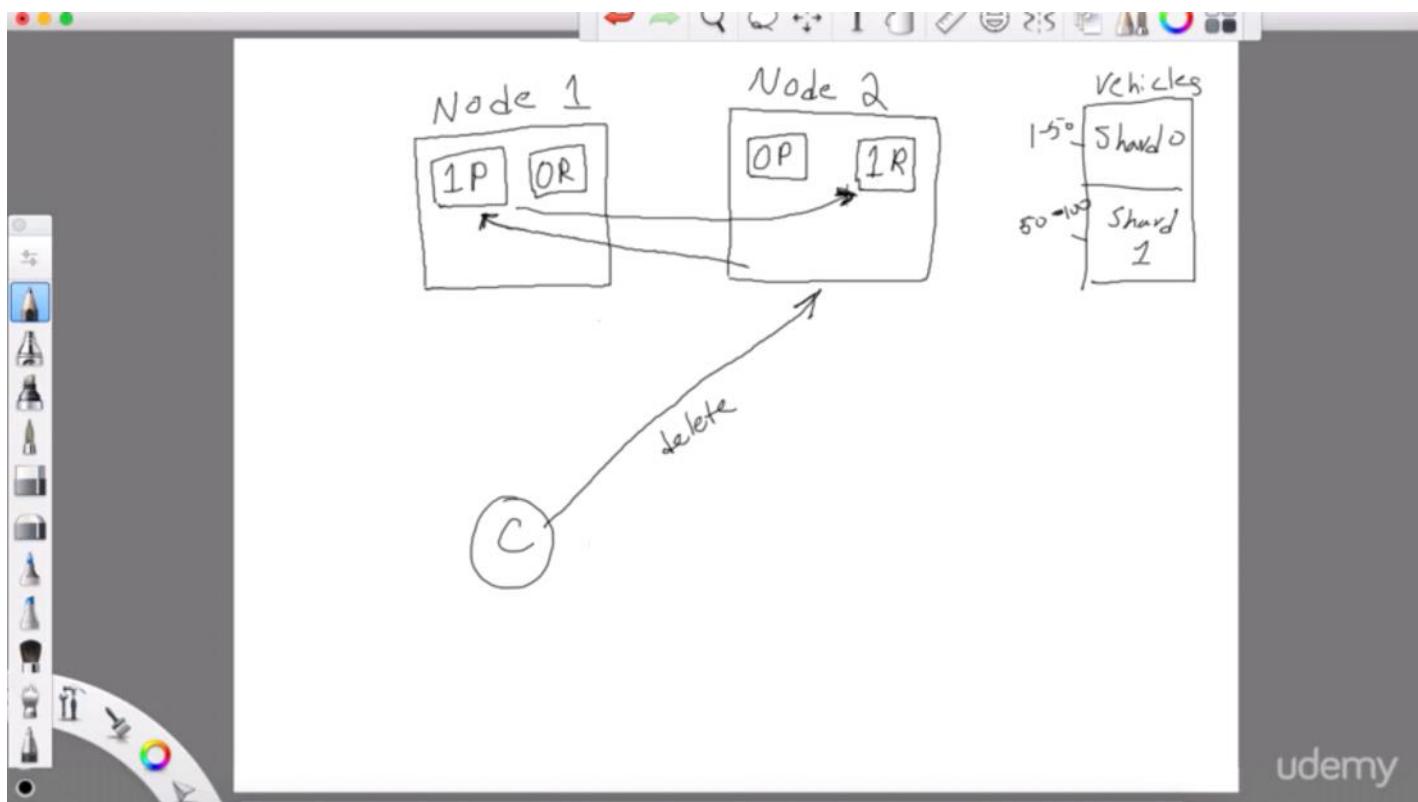
```
Untitled @ 113.8 % - Not logged in

        }
    },
    "settings": {
        "index": {
            "creation_date": "1495992850651",
            "number_of_shards": "5", ←
            "number_of_replicas": "1",
            "uuid": "_v6dYh0eSkGL66uWLizwSw",
            "version": {
                "created": "5040099"
            },
            "provided_name": "business"
        }
    }
}
```

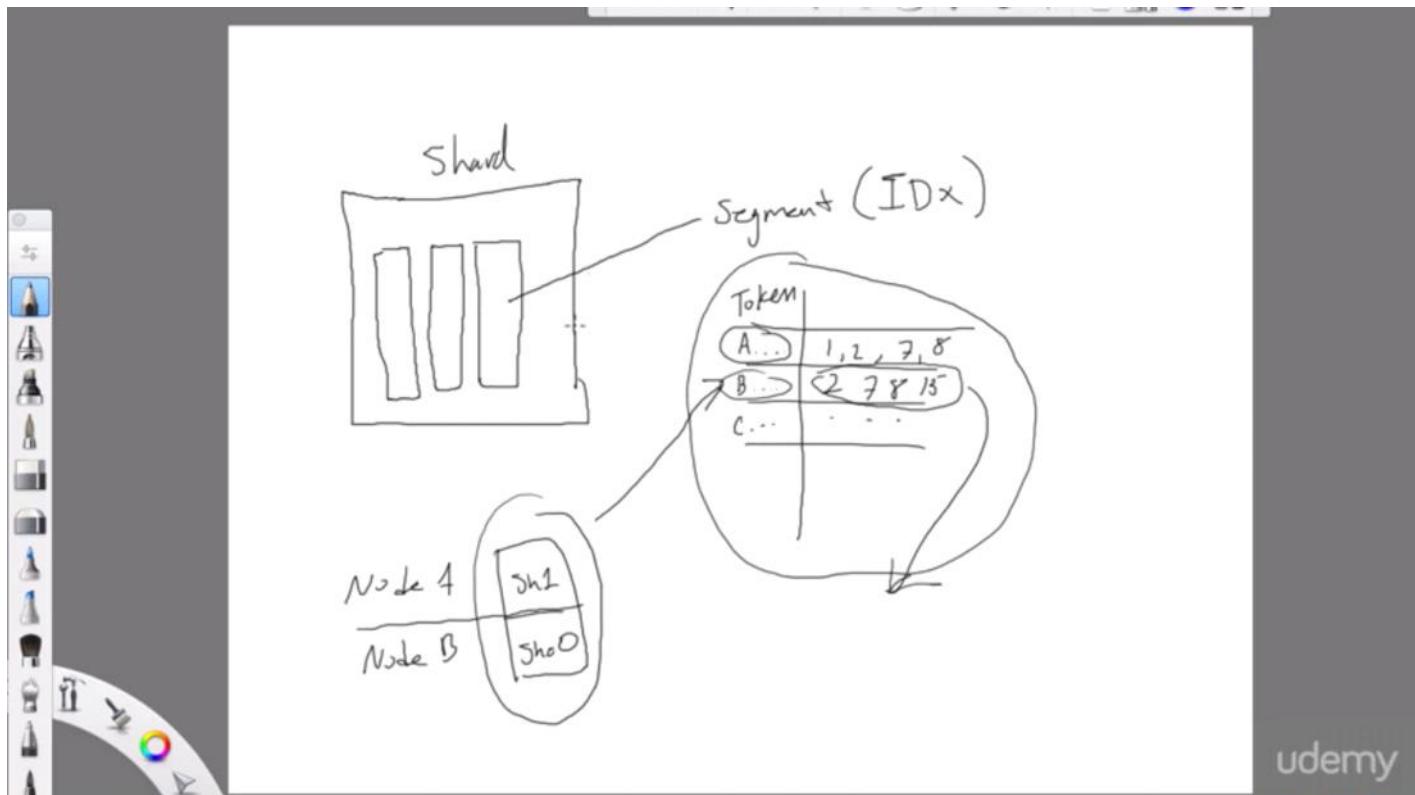
- Physical representation of an index is a shard
- Shards contain data from multiple documents (via docIDs)
- "P" represents a primary shard (P0, P1)
- Also have replica shards, which contains same exact documents as the ones in the shards (R0, R1)
- Replicas used for data distribution, w/o replica shards, would create isolated bubbles



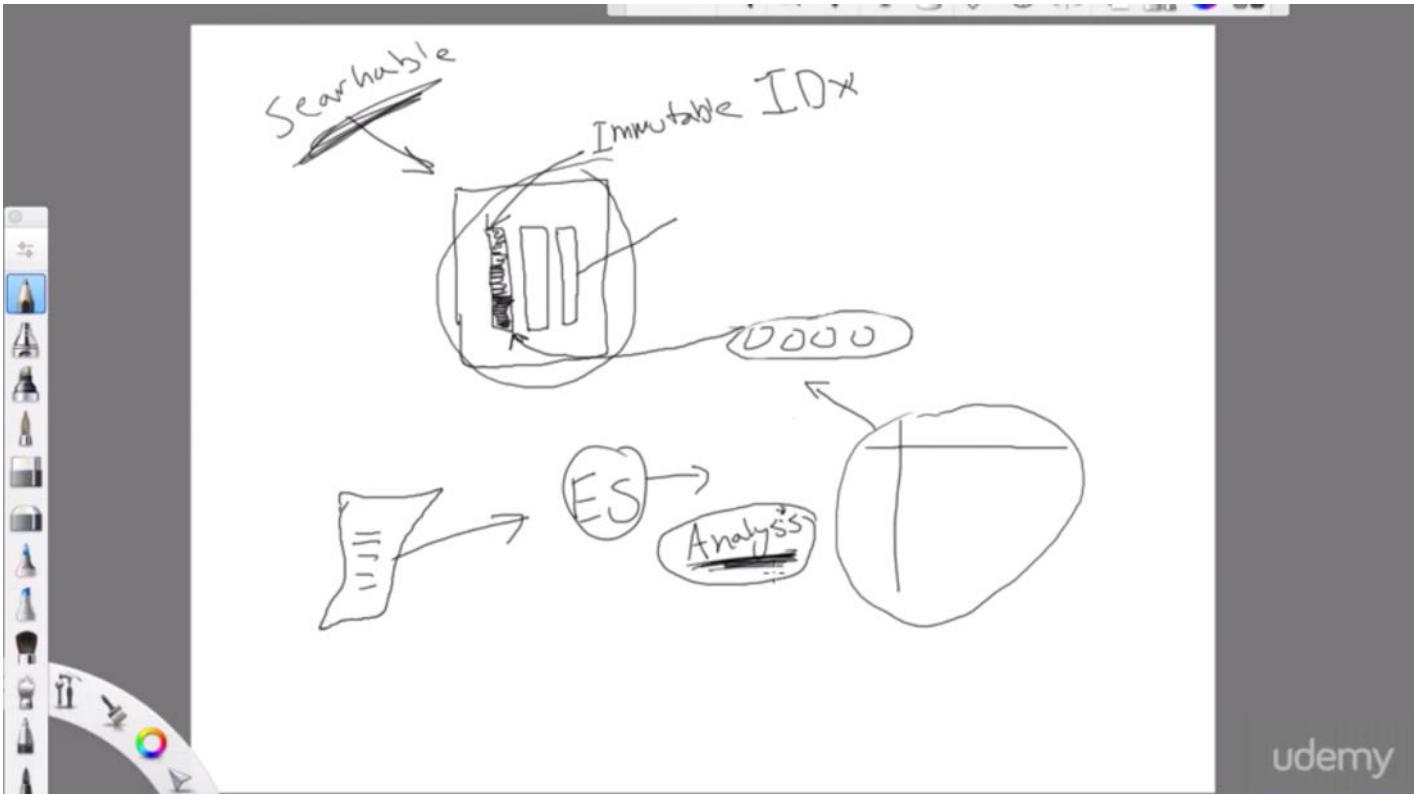
- Given a request, the nodes use the ID of the document requested to first find the proper primary shard, and then updates each of its replica shards in turn (indexing, deleting, getting, etc.)
- This is a form of load-balancing using an algorithm call Round-robin scheduling (RR), where the process bounces amongst nodes instead of simply hammering on one node.



- A shard is a lucene index
- A shard is a physical container; it can have multiple segments (inverted indices)



- ES formulates an inverted index when it's given documents
- ES is made of shards, which are made of segments
- Given a JSON document (data) sent to ES, ES will deconstruct the data via a step known as text analysis to convert text to tokens to feed the inverted index
- Once it forms an inverted index, it gets sent to a memory buffer, which gets populated with each document that passes through
- The buffer gets committed to a specific segment, which is how those segments get confirmed
- For each committed buffer, a new one rises up to continue processing incoming documents
- Once a segment gets completed, the segment becomes an *immutable inverted index* (can create more segments, but can't overwrite existing segments)

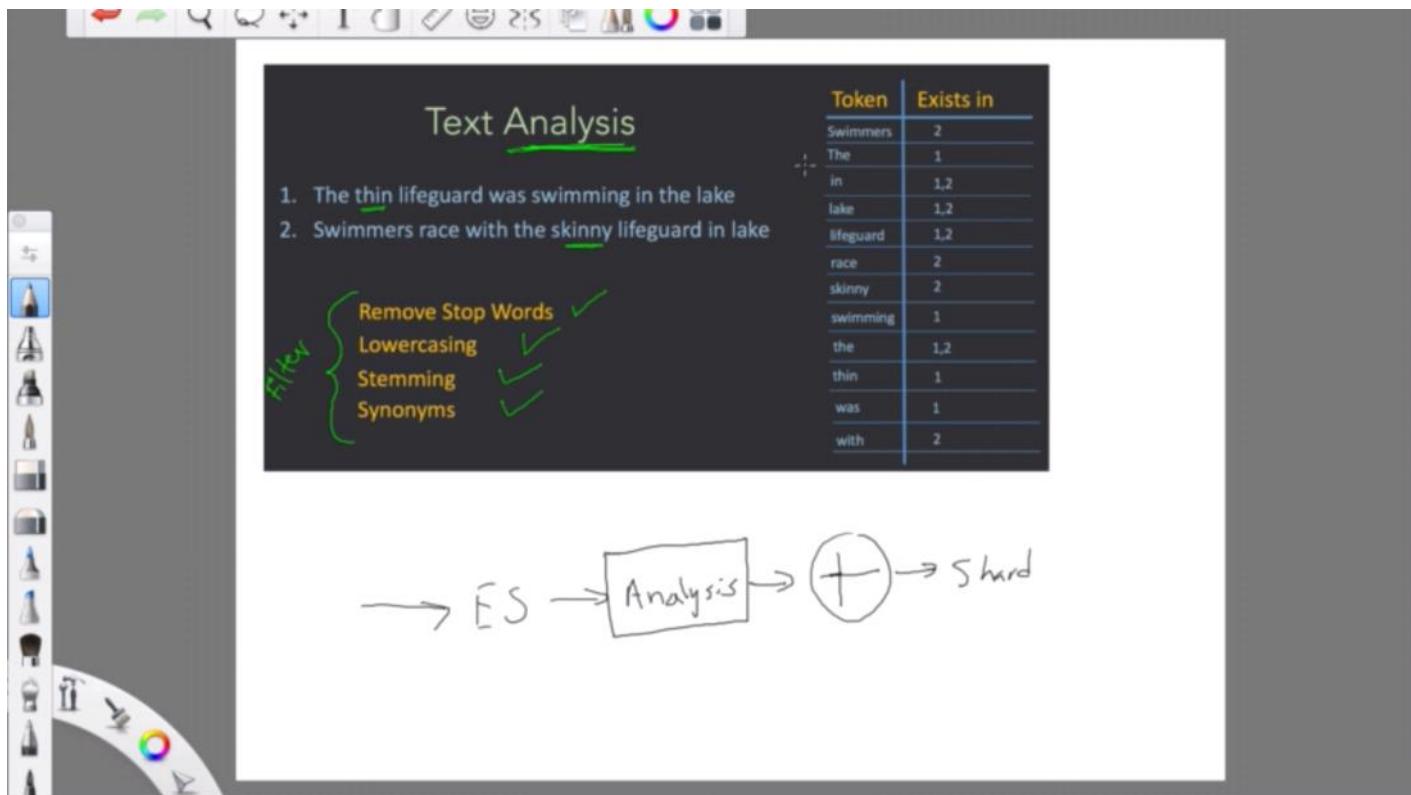


udemy

Lecture 7: Text Analysis for Indexing and Searching

Tuesday, June 11, 2019 2:59 PM

- ES --> Analysis --> Inverted Index (IDx) --> Shard (Segment)
- This process is key for indexing, retrieving, etc.
- Text Analysis (Filters):
 - Remove Stop Words ("useless" words)
 - Lowercasing
 - Stemming (root words prioritized)
 - Synonyms



- Analyzer (for both Indexing and Querying)
 - Tokenizer
 - Filter

Text Analysis

1. The thin lifeguard was swimming in the lake
2. Swimmers race with the skinny lifeguard in lake

Filter { Remove Stop Words ✓
Lowercasing ✓
Stemming ✓
Synonyms ✓

Token	Exists in
Swimmers	2
The	1
in	1,2
lake	1,2
lifeguard	1,2
race	2
skinny	2
swimming	1
the	1,2
thin	1
was	1
with	2

```

graph LR
    Text --> Analyzer[Analyzer  
Tokenizer ①  
Filter ②]
    Analyzer --> Index((index))
    
```

Text Analysis

1. The thin lifeguard was swimming in the lake
2. Swimmers race with the skinny lifeguard in lake

Filter { Remove Stop Words ✓
Lowercasing ✓
Stemming ✓
Synonyms ✓

Token	Exists in
Swimmers	2
The	1
in	1,2
lake	1,2
lifeguard	1,2
race	2
skinny	2
swimming	1
the	1,2
thin	1
was	1
with	2

```

graph LR
    Text --> Analyzer[Analyzer  
Tokenizer ①  
Filter ②]
    Analyzer --> Index((index))
    
```

Search
Text → Analyzer
match: "the thin"
Result = "thin"

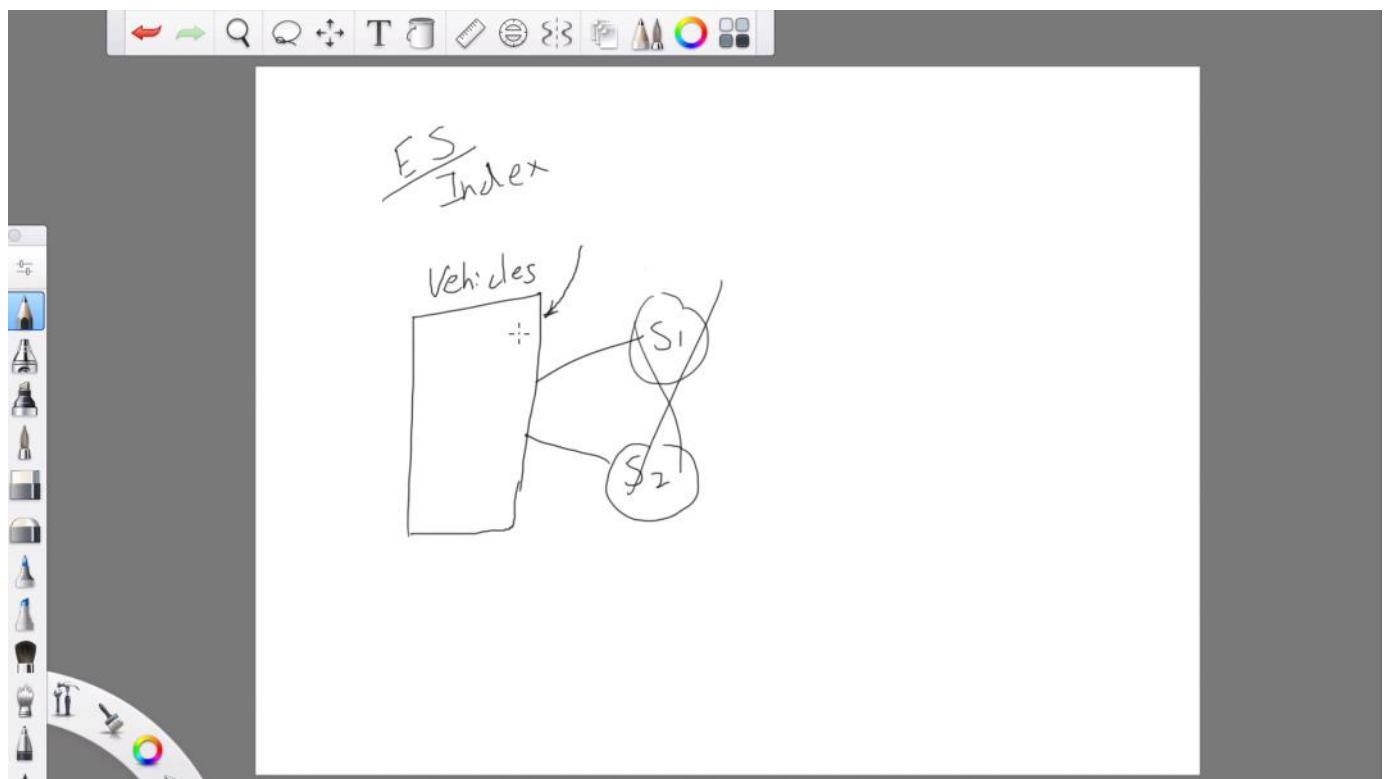
Configure the Index Structure

```
{  
  "settings": {  
    ...  
  },  
  
  "mappings": {  
    ...  
  }  
}
```

Lecture 8: Index Settings and Mappings (Part 1)

Tuesday, July 9, 2019 3:22 PM

- Not as much of a focus on shards, more concerned with the logical representation of said data and how to load data into each index --> higher level usability standpoint

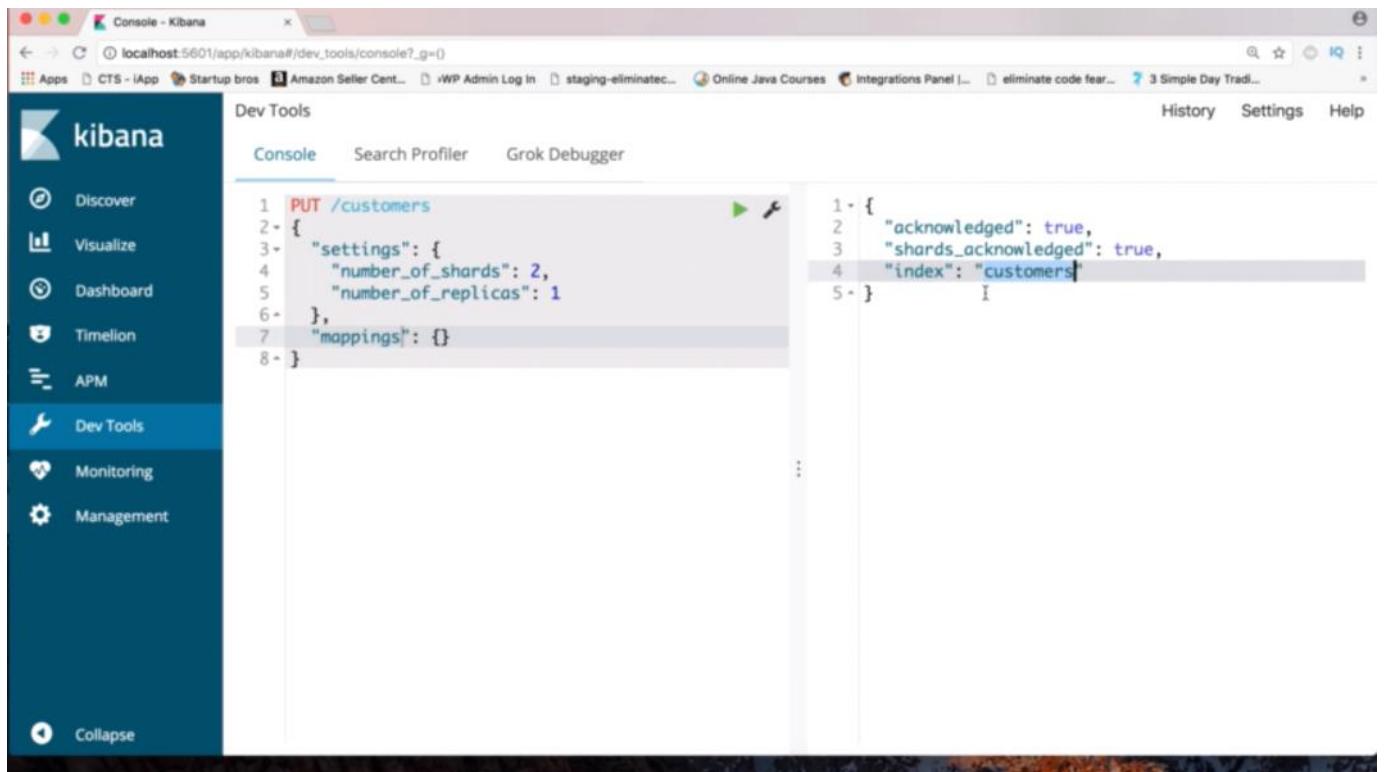


Console - Kibana

localhost:5601/app/kibana#/dev_tools/console?_g=()

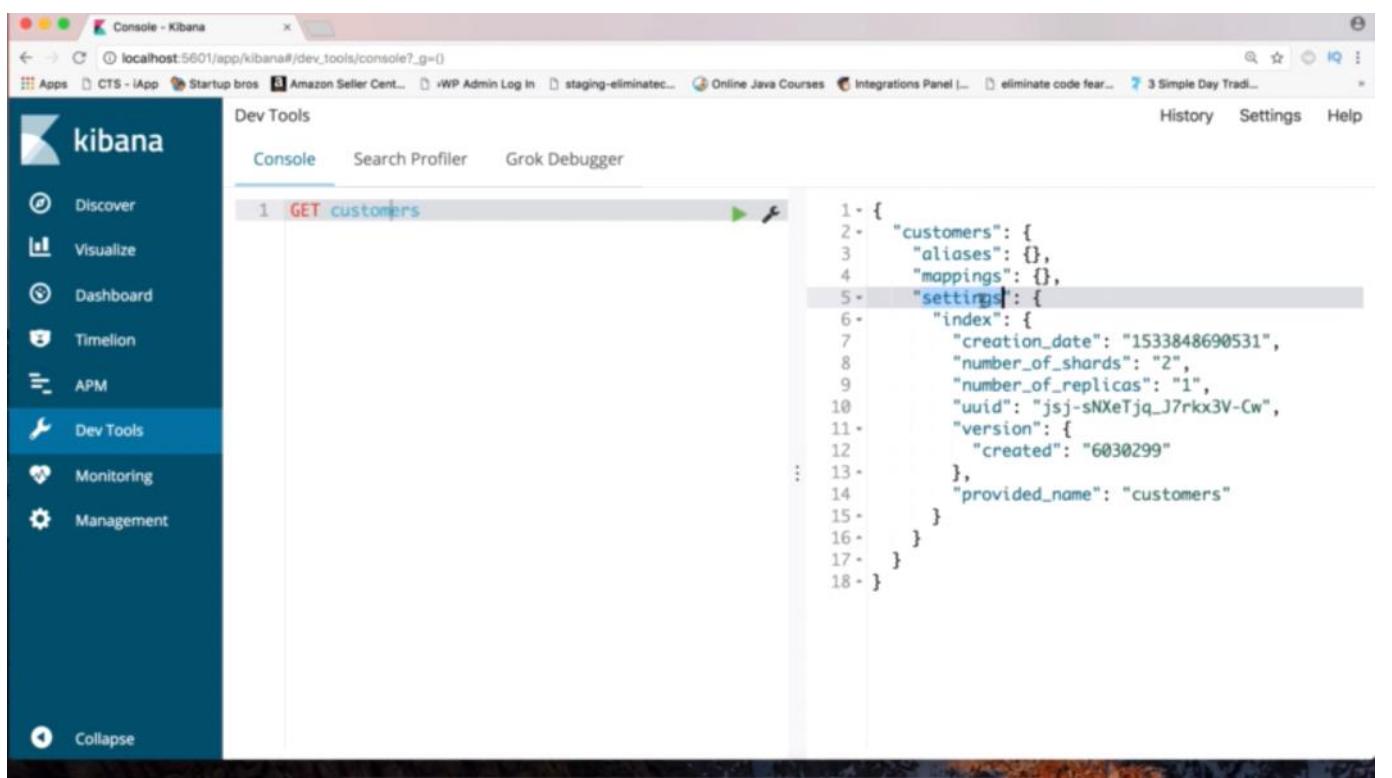
get customers

```
1 - {  
2 -   "error": {  
3 -     "root_cause": [  
4 -       {  
5 -         "type": "index_not_found_exception",  
6 -         "reason": "no such index",  
7 -         "resource.type": "index_or_alias",  
8 -         "resource.id": "customers",  
9 -         "index_uuid": "_na_",  
10 -        "index": "customers"  
11 -      }  
12 -    ],  
13 -    "type": "index_not_found_exception",  
14 -    "reason": "no such index",  
15 -    "resource.type": "index_or_alias",  
16 -    "resource.id": "customers",  
17 -    "index_uuid": "_na_",  
18 -    "index": "customers"  
19 -  },  
20 -  "status": 404  
21 - }
```



Kibana Dev Tools interface showing a PUT request to the /customers index.

```
PUT /customers
{
  "settings": {
    "number_of_shards": 2,
    "number_of_replicas": 1
  },
  "mappings": {}
}
```



Kibana Dev Tools interface showing a GET request to the /customers index.

```
GET customers
```

```
1 - {
  2 -   "customers": {
  3 -     "aliases": {},
  4 -     "mappings": {},
  5 -     "settings": {
  6 -       "index": {
  7 -         "creation_date": "1533848690531",
  8 -         "number_of_shards": "2",
  9 -         "number_of_replicas": "1",
  10 -        "uuid": "jsj-sNXeTjq_J7rlx3V-Cw",
  11 -        "version": {
  12 -          "created": "6030299"
  13 -        },
  14 -        "provided_name": "customers"
  15 -      }
  16 -    }
  17 -  }
  18 - }
```

- Analyzers are used for splitting text to be stored in the inverted index

The screenshot shows the Kibana Dev Tools interface. On the left, the sidebar has 'Dev Tools' selected. The main area has three tabs: 'Console', 'Search Profiler', and 'Grok Debugger'. The 'Console' tab is active, displaying a code editor with a red 'PUT' icon. The code is:

```
1 PUT /customers
2 {
3   "mappings": {
4     "online": {
5       "properties": {
6         "gender": {
7           "type": "text",
8           "analyzer": "standard"
9         },
10        "age": {
11          "type": "integer"
12        },
13        "total_spent": {
14          "type": "float"
15        },
16        "is_new": {
17          "type": "boolean"
18        },
19        "name": {
20          "type": "text",
21          "analyzer": "standard"
22      }
23    }
24  },
25}
```

To the right, the response is shown in a monospaced text area:

```
1 {
2   "customers": {
3     "aliases": {},
4     "mappings": {},
5     "settings": {
6       "index": {
7         "creation_date": "1533848690531",
8         "number_of_shards": "2",
9         "number_of_replicas": "1",
10        "uuid": "jsj-sNXeTjq_J7rkx3V-Cw",
11        "version": {
12          "created": "6030299"
13        },
14        "provided_name": "customers"
15      }
16    }
17  }
18 }
```

The screenshot shows the Kibana Dev Tools interface. On the left, the sidebar has 'Dev Tools' selected. The main area has three tabs: 'Console', 'Search Profiler', and 'Grok Debugger'. The 'Console' tab is active, displaying a code editor with a red 'PUT' icon. The code is identical to the one in the previous screenshot:

```
1 PUT /customers
2 {
3   "mappings": {
4     "online": {
5       "properties": {
6         "gender": {
7           "type": "text",
8           "analyzer": "standard"
9         },
10        "age": {
11          "type": "integer"
12        },
13        "total_spent": {
14          "type": "float"
15        },
16        "is_new": {
17          "type": "boolean"
18        },
19        "name": {
20          "type": "text",
21          "analyzer": "standard"
22      }
23    }
24  },
25  "settings": {
26    "number_of_shards": 2,
27    "number_of_replicas": 1
28  }
29}
```

To the right, the response is shown in a monospaced text area:

```
1 {
2   "customers": {
3     "aliases": {},
4     "mappings": {},
5     "settings": {
6       "index": {
7         "creation_date": "1533848690531",
8         "number_of_shards": "2",
9         "number_of_replicas": "1",
10        "uuid": "jsj-sNXeTjq_J7rkx3V-Cw",
11        "version": {
12          "created": "6030299"
13        },
14        "provided_name": "customers"
15      }
16    }
17  }
18 }
```

- **PUT /customers**
- {
- "mappings": {
- "online": {
- "properties": {
- "gender": {
- "type": "text",
- "analyzer": "standard"

```
},
"age": {
  "type": "integer"
},
"total_spent": {
  "type": "float"
},
"is_new": {
  "type": "boolean"
},
"name": {
  "type": "text",
  "analyzer": "standard"
}
},
"settings": {
  "number_of_shards": 2,
  "number_of_replicas": 1
}
}
• {
  "acknowledged": true,
  "shards_acknowledged": true,
  "index": "customers"
}
```

Lecture 9: Index Settings and Mappings (Part 2)

Tuesday, June 11, 2019 3:22 PM

```
1 PUT customers/online/124
2 {
3     "name": "Mary Cranford",
4     "address": "310 Clark Ave",
5     "gender": "female",
6     "age": 34,
7     "total_spent": 550.75,
8     "is_new": false
9 }
```

```
1 { "_index": "customers", "_type": "online", "_id": "124", "_version": 1, "result": "created", "_shards": { "total": 2, "successful": 1, "failed": 0 }, "created": true }
11 }
12
13 }
```

The screenshot shows the Elasticsearch Dev Tools interface. On the left, a code editor window displays a PUT request to the 'customers/online/124' endpoint with a JSON payload containing customer information. On the right, the results of the request are shown, indicating a successful creation of the document with ID '124'. The response includes details about the index, type, ID, version, result, shards, and creation status.

- Elastic will take additional fields and auto-edit the schema/mapping in a dynamic manner
- Can configure Elasticsearch to multiple levels of dynamic mapping/strictness

```
1 GET customers
```

dynamic :
If set to **false** - indexing field will be Ignored
If set to **strict** - indexing field will throw error

```
2 { "customers": { "aliases": {}, "mappings": { "online": { "properties": { "address": { "type": "text", "fields": { "keyword": { "type": "keyword", "ignore_above": 256 } } } }, "age": { "type": "integer" }, "gender": { "type": "string", "analyzer": "standard" }, "is_new": { "type": "boolean" }, "name": { "type": "text", "analyzer": "standard" }, "total_spent": { "type": "float" } } } } }
```

The screenshot shows the Elasticsearch Dev Tools interface again. A GET request is made to the 'customers' endpoint. A callout box highlights the 'dynamic' setting in the mapping configuration. It defines three levels of dynamic mapping: 'false' (indexing field will be ignored), 'strict' (indexing field will throw an error), and 'dynamic' (the current setting). The mapping configuration for the 'online' type is detailed, showing properties for address, age, gender, is_new, name, and total_spent, each with specific types and analyzers.

```
1 GET customers/_mapping/online
2 {
3     "dynamic": false
4 }
```

```
1 {
2     "acknowledged": true
3 }
```

```
1 PUT /customers/online/124
2 {
3     "name": "Mary Cranford",
4     "address": "310 Clark Ave",
5     "gender": "female",
6     "age": 34,
7     "total_spent": 550.75,
8     "is_new": false,
9     "retired":true
10 }
```

```
1 {
2     "error": {
3         "root_cause": [
4             {
5                 "type": "strict_dynamic_mapping_exception",
6                 "reason": "mapping set to strict, dynamic
introduction of [retired] within [online] is not
allowed"
7             }
8         ],
9         "type": "strict_dynamic_mapping_exception",
10        "reason": "mapping set to strict, dynamic
introduction of [retired] within [online] is not
allowed"
11     },
12     "status": 400
13 }
```

- <https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html>
- Can test how analyzers work

```

1 POST _analyze
2 {
3   "analyzer": "whitespace",
4   "text": "The quick brown fox."
5 }

1 POST _analyze
2 {
3   "analyzer": "standard",
4   "text": "The quick brown fox."
5 }

```

The screenshot shows two code snippets in a terminal or IDE interface. The left snippet uses the 'whitespace' analyzer, which splits the text into tokens based on whitespace. The right snippet uses the 'standard' analyzer, which splits the text into tokens based on non-alphanumeric characters (like spaces). Both snippets are identical except for the analyzer name.

```

1 POST _analyze
2 {
3   "analyzer": "whitespace",
4   "text": "The quick brown fox."
5 }

1 POST _analyze
2 {
3   "analyzer": "standard",
4   "text": "The quick brown fox."
5 }

```

The screenshot shows two code snippets in a terminal or IDE interface. The left snippet uses the 'whitespace' analyzer, which splits the text into tokens based on whitespace. The right snippet uses the 'standard' analyzer, which splits the text into tokens based on non-alphanumeric characters (like spaces). Both snippets are identical except for the analyzer name.

- Where standard analyzer considers consecutive alphanumeric strings as one object, simple analyzer splits on any non-alphabetic character (i.e. numbers and punctuation)

```

1 POST _analyze
2 {
3   "analyzer": "simple",
4   "text": "The quick brown fox."
5 }

1+ [
2+   "tokens": [
3+     {
4+       "token": "the",
5+       "start_offset": 0,
6+       "end_offset": 3,
7+       "type": "word",
8+       "position": 0
9+     },
10+    {
11+      "token": "quick",
12+      "start_offset": 4,
13+      "end_offset": 9,
14+      "type": "word",
15+      "position": 1
16+    },
17+    {
18+      "token": "brown",
19+      "start_offset": 13,
20+      "end_offset": 18,
21+      "type": "word",
22+      "position": 2
23+    },
24+    {
25+      "token": "fox",
26+      "start_offset": 22,
27+      "end_offset": 25,
28+      "type": "word",
29+      "position": 3
30+    }
]

```

- The english analyzer splits on punctuation and can split root words from gerunds and other forms

```

1 POST _analyze
2 {
3   "analyzer": "english",
4   "text": "The quick running fox."
5 }

1+ [
2+   "tokens": [
3+     {
4+       "token": "quick",
5+       "start_offset": 4,
6+       "end_offset": 9,
7+       "type": "<ALPHANUM>",
8+       "position": 1
9+     },
10+    {
11+      "token": "run",
12+      "start_offset": 10,
13+      "end_offset": 17,
14+      "type": "<ALPHANUM>",
15+      "position": 2
16+    },
17+    {
18+      "token": "fox",
19+      "start_offset": 18,
20+      "end_offset": 21,
21+      "type": "<ALPHANUM>",
22+      "position": 3
23+    }
24+  ]
25+

```

- Can define custom analyzer, which depends on tokenizer, filter, mappings, properties, etc.

The screenshot shows a web browser window with the URL https://www.elastic.co/guide/en/elasticsearch/reference/5.4/_analyze.html. The page title is "Testing analyzers | Elasticsearch Reference [5.4] | Elastic". The main content area is titled "PUT my_index" and contains the following JSON code:

```
PUT my_index
{
  "settings": {
    "analysis": {
      "analyzer": {
        "std_folded": {
          "type": "custom",
          "tokenizer": "standard",
          "filter": [
            "lowercase",
            "asciifolding"
          ]
        }
      }
    },
    "mappings": {
      "my_type": {
        "properties": {
          "my_text": {
            "type": "text",
            "analyzer": "std_folded"
          }
        }
      }
    }
}
```

To the right of the code, there is a sidebar with a tree-like navigation menu:

- Testing analyzers
 - Analyzers
 - Normalizers
 - Tokenizers
 - Token Filters
 - Character Filters
- Modules
- Index Modules
- Ingest Node
- How To
- Testing
- Glossary of terms
- Release Notes
- A. Painless API Reference

Lecture 10: Search DSL Query Context (Part 1)

Tuesday, June 11, 2019 3:23 PM

- Elasticsearch is HTTP and JSON
- We provide the HTTP URL endpoint for indexing and for querying
- The body of the request contains that JSON to send as part of the request

```
1 GET courses
```

```
1 · {  
2 ·   "courses": {  
3 ·     "aliases": {},  
4 ·     "mappings": {  
5 ·       "classroom": {  
6 ·         "properties": {  
7 ·           "course_description": {  
8 ·             "type": "text",  
9 ·             "fields": {  
10 ·               "keyword": {  
11 ·                 "type": "keyword",  
12 ·                 "ignore_above": 256  
13 ·               }  
14 ·             }  
15 ·           },  
16 ·           "course_publish_date": {  
17 ·             "type": "date"  
18 ·           },  
19 ·           "title": {  
20 ·             "type": "text",  
21 ·             "fields": {  
22 ·               "keyword": {  
23 ·                 "type": "keyword",  
24 ·                 "ignore_above": 256  
25 ·               }  
26 ·             }  
27 ·           },  
28 ·           "professor": {  
29 ·             "properties": {  
30 ·               "department": {  
31 ·                 "type": "text"  
32 ·               }  
33 ·             }  
34 ·           }  
35 ·         }  
36 ·       }  
37 ·     }  
38 ·   }  
39 · }
```

Those are that's a specific structure and this is known as the DSL for lastic search.

- One objective: match your search criteria and provide some kind of relevancy score
 - Ex. Searching for the term "technology" in documents where documents that contain more occurrences of the word are considered more relevant
- Query context is used for full text searches
 - Match documents that contain the search criteria as well as specify how well that document matched the particular search criteria by providing that relevancy score

```
1 GET courses
```



```
1 {
```

```
2   "courses": {  
3     "aliases": {},  
4  
5     "properties": {  
6       "course_description": {  
7         "type": "text",  
8         "fields": {  
9           "keyword": {  
10             "type": "keyword",  
11             "ignore_above": 256  
12           }  
13         },  
14       },  
15     },  
16     "course_publish_date": {  
17       "type": "date"  
18     },  
19     "name": {  
20       "type": "text",  
21       "fields": {  
22         "keyword": {  
23           "type": "keyword",  
24           "ignore_above": 256  
25         }  
26       },  
27     },  
28     "professor": {  
29       "properties": {  
30         "department": {}  
31       }  
32     }  
33   }  
34 }
```

Query Context

Filter Context

- Both of these can also be combined to form more complex queries

- Filter context

- Can be combined together with query context to form one large query

```
1 GET /courses/_search  
2 {  
3   "query": {  
4     "match_all": {}  
5   }  
6 }
```

```
1 {  
2   "took": 0,  
3   "timed_out": false,  
4   "_shards": {  
5     "total": 5,  
6     "successful": 5,  
7     "failed": 0  
8   },  
9   "hits": {  
10    "total": 10,  
11    "max_score": 1,  
12    "hits": [  
13      {  
14        "_index": "courses",  
15        "_type": "classroom",  
16        "_id": "5",  
17        "_score": 1,  
18        "_source": {  
19          "name": "Theatre 410",  
20          "room": "T18",  
21          "professor": {  
22            "name": "Sebastian Hern",  
23            "department": "art",  
24            "faculty_type": "part-time",  
25            "email": ""  
26          },  
27        }  
28      },  
29      {  
30        "score": 47,  
31        "course_publish_date": "2013-01-27",  
32        "course_description": "The 410 is an advanced elective course dissecting the various  
33      }  
34    ]  
35  }  
36 }
```

These are the different documents that we get are right here instead of this array structures.

- All the documents inside the array structure are called hits
- _score field holds the relevancy score

```

1 GET /courses/_search
2 {
3   "query": {
4     "match": {"name" : "computer" }
5   }
6 }

```

But anyway let's hit run and there we go.

```

1 [ 2   "took": 0, 3   "timed_out": false, 4   "_shards": { 5     "total": 5, 6     "successful": 5, 7     "failed": 0 8   }, 9   "hits": { 10    "total": 2, 11    "max_score": 0.8142733, 12    "hits": [ 13      { 14        "_index": "courses", 15        "_type": "classroom", 16        "_id": "4", 17        "_score": 0.8142733, 18        "_source": { 19          "name": "Computer Science 101", 20          "room": "C12", 21          "professor": { 22            "name": "Gregg Payne", 23            "department": "engineering", 24            "faculty_type": "full-time", 25            "email": "payneg@onuni.com" 26          }, 27          "students_enrolled": 33, 28          "course_publish_date": "2013-08-27", 29          "course_description": "CS 101 is a first year computer science introduction teaching"

```

- Score based off of "significance" within the field specified
 - Ex. A term's score within a field could be based off of # chars

```

1 GET /courses/_search
2 {
3   "query": {
4     "match": {"name" : "computer" }
5   }
6 }

```

a classic search you know you can memorize those formulas but, you don't really need to know those formulas

```

12   "hits": [ 13     { 14       "_index": "courses", 15       "_type": "classroom", 16       "_id": "7", 17       "_score": 0.81546724, 18       "_source": { 19         "name": "Computer computer 250", 20         "room": "C8", 21         "professor": { 22           "name": "Gregg Payne", 23           "department": "engineering", 24           "faculty_type": "part-time", 25           "email": "payneg@onuni.com" 26         }, 27         "students_enrolled": 33, 28         "course_publish_date": "2012-08-20", 29         "course_description": "cpt Int 250 gives students an integrated and rigorous picture of applied computer science, as it comes to play in the construction of a simple yet powerful computer system. " 30       } 31     }, 32     { 33       "_index": "courses", 34       "_id": "4", 35       "_score": 0.8142733, 36       "_source": { 37         "name": "Computer Science 101", 38         "room": "C12", 39         "professor": { 40           "name": "Gregg Payne", 41           "department": "engineering", 42           "faculty_type": "full-time", 43           "email": "payneg@onuni.com" 44         }, 45         "students_enrolled": 33, 46         "course_publish_date": "2013-08-27", 47         "course_description": "CS 101 is a first year computer science introduction teaching"

```

- For multiple criteria searches, you must have a "must" structure -- this allows for multiple variable confirmations simultaneously
- You also require an enclosing "bool" statement, without which the query will not run properly
- GET /courses/_search**

```
"query": {  
  "bool": {  
    "must": [  
      {"match": {"name": "computer"}},  
      {"match": {"room": "c8"}},  
      ]  
    }  
  }  
}
```

Lecture 11: Search DSL Query Context (Part 2)

Tuesday, June 11, 2019 3:23 PM

- With must comes must not

```
1 GET /courses/_search
2 {
3   "query": {
4     "bool": {
5       "must": [
6         {"match": {"name": "accounting"}},
7         {"match": {"room": "e3"}}
8       ],
9
10      "must_not": [
11        {"match": {"professor.name": "bill"}}
12      ],
13
14      "should": [
15        {"match": {"name": "computer"}}
16      ]
17    }
18  }
19}
20
```

8 * },
9 * "hits": {
10 * "total": 1,
11 * "max_score": 1.379555,
12 * "hits": [
13 * {
14 * "_index": "courses",
15 * "_type": "classroom",
16 * "_id": "1",
17 * "_score": 1.379555,
18 * "_source": {
19 * "name": "Accounting 101",
20 * "room": "E3",
21 * "professor": {
22 * "name": "Thomas Baszo",
23 * "department": "finance",
24 * "faculty_type": "part-time",
25 * "email": "baszot@onuni.com"
26 * },
27 * "students_enrolled": 27,
28 * "course_publish_date": "2015-01-19",
29 * "course_description": "Act 101 is a course
from the business school on the introduction to
accounting that teaches students how to read and
compose basic financial statements"
30 * }
31 *]
32 *]
33 * }
34 * }

It's pretty much almost always ignored by classic search unless we specify a special field.

- With must not comes should which is ^^

- "minimum_should_match" refers to a query where the should statement must be true in at least one hit

```
1 GET /courses/_search
2 {
3   "query": {
4     "bool": {
5       "must": [
6         {"match": {"name": "accounting"}}
7       ],
8
9       "must_not": [
10        {"match": {"professor.name": "bill"}}
11      ],
12
13       "should": [
14        {"match": {"room": "e7"}}
15      ],
16      "minimum_should_match": 1
17    }
18  }
19}
20}
```

29 "course_description": "Act 101 is a course
from the business school on the introduction to
accounting that teaches students how to read and
compose basic financial statements"
30 }
31],
32 {
33 "_index": "courses",
34 "_type": "classroom",
35 "_id": "9",
36 "_score": 0.6099695,
37 "_source": {
38 "name": "Tax Accounting 200",
39 "room": "E7",
40 "professor": {
41 "name": "Thomas Baszo",
42 "department": "finance",
43 "faculty_type": "part-time",
44 "email": "baszot@onuni.com"
45 },
46 "students_enrolled": 17,
47 "course_publish_date": "2016-06-15",
48 "course_description": "Tax Act 200 is an
intermediate course covering various aspects of tax
law"
49 }
50 }
51],
52 }

And if I set a minimum should match one that's going to only return this document.

- Multi-match query consists of searching over multiple fields

```

1 GET /courses/_search
2 {
3   "query": {
4     "multi_match": {
5       "fields": ["name", "professor.department"],
6       "query": "accounting"
7     }
8   }
9 }
```

this particular score so it's sorted based on score the highest one is on top or highest document.

```

5   "total": 5,
6   "successful": 5,
7   "failed": 0
8 },
9 "hits": {
10   "total": 4,
11   "max_score": 1.2039728,
12   "hits": [
13     {
14       "_index": "courses",
15       "_type": "classroom",
16       "_id": "8",
17       "_score": 1.2039728,
18       "_source": {
19         "name": "Accounting Info Systems 350",
20         "room": "E3",
21         "professor": {
22           "name": "Bill Cage",
23           "department": "accounting",
24           "faculty_type": "full-time",
25           "email": "cageb@onuni.com"
26         },
27         "students_enrolled": 19,
28         "course_publish_date": "2014-05-15",
29         "course_description": "Act Sys 350 is an advanced course from the business school taken by final year accounting majors that covers the subject of business incurred costs and how to record them in financial statements"
30       }
31     }
32   ]
33 }
```

- "match_phrase" queries for a specific phrase contained in a specific field (requires complete tokens & complete words)
- "match_phrase_prefix" can be used to figure out partial tokens

```

1 GET /courses/_search
2 {
3   "query": {
4     "match_phrase_prefix": {
5       "course_description": "from the business school
taken by fin"
6     }
7   }
8 }
```

```

5   "total": 1,
6   "max_score": 4.6659784,
7   "hits": [
8     {
9       "_index": "courses",
10      "_type": "classroom",
11      "_id": "6",
12      "_score": 4.6659784,
13      "_source": {
14        "name": "Cost Accounting 400",
15        "room": "E7",
16        "professor": {
17          "name": "Bill Cage",
18          "department": "accounting",
19          "faculty_type": "full-time",
20          "email": "cageb@onuni.com"
21        },
22        "students_enrolled": 31,
23        "course_publish_date": "2014-12-31",
24        "course_description": "Cst Act 400 is an advanced course from the business school taken by final year accounting majors that covers the subject of business incurred costs and how to record them in financial statements"
25      }
26    }
27  ]
28 }
```

There's also something called a range query.

- A range query is used to search for documents that exist within a given range

```

1 GET /courses/_search
2 {
3   "query": {
4     "range": {
5       "students_enrolled": {
6         "gt": 20
7       }
8     }
9   }
10 }
```

```

5   "total": 1,
6   "successful": 1,
7   "failed": 0
8 },
9 "hits": {
10   "total": 1,
11   "max_score": 1.2039728,
12   "hits": [
13     {
14       "_index": "courses",
15       "_type": "classroom",
16       "_id": "8",
17       "_score": 1.2039728,
18       "_source": {
19         "name": "Accounting Info Systems 350",
20         "room": "E3",
21         "professor": {
22           "name": "Bill Cage",
23           "department": "accounting",
24           "faculty_type": "full-time",
25           "email": "cageb@onuni.com"
26         },
27         "students_enrolled": 19,
28         "course_publish_date": "2014-05-15",
29         "course_description": "Act Sys 350 is an advanced course from the business school taken by final year accounting majors that covers the subject of business incurred costs and how to record them in financial statements"
30       }
31     }
32   ]
33 }
```

```

1 GET /courses/_search
2 {
3   "query": {
4     "range":{
5       "students_enrolled": {
6         "gte": 10,
7         "lte": 30
8       }
9     }
10   }
11 }
```

course from the business school on the introduction to accounting that teaches students how to read and compose basic financial statements"

```

106   }
107   },
108   {
109     "_index": "courses",
110     "_type": "classroom",
111     "_id": "3",
112     "_score": 1,
113     "_source": {
114       "name": "Anthropology 230",
115       "room": "G11",
116       "professor": {
117         "name": "Devin Cranford",
118         "department": "history",
119         "facultry_type": "full-time",
120         "email": "devinc@onuni.com"
121       },
122       "students_enrolled": 22,
123       "course_publish_date": "2013-08-27",
124       "course_description": "Ant 230 is an intermediate course on human societies and culture and their development. A focus on the Mayans civilization is rooted in this course"
125     }
126   }
127 }
```

This is greater than or equal to.

- "gte" - greater than or equal to, "gt" - greater than, "lte" - less than or equal to, "lt" - less than

```

1 GET /courses/_search
2 {
3   "query": {
4     "range":{
5       "course_publish_date": {
6         "gte": "2013-08-27"
7       }
8     }
9   }
10 }
```

final year accounting majors that covers the subject of business incurred costs and how to record them in financial statements"

```

106   }
107   },
108   {
109     "_index": "courses",
110     "_type": "classroom",
111     "_id": "1",
112     "_score": 1,
113     "_source": {
114       "name": "Accounting 101",
115       "room": "E3",
116       "professor": {
117         "name": "Thomas Baszo",
118         "department": "finance",
119         "facultry_type": "part-time",
120         "email": "baszot@onuni.com"
121       },
122       "students_enrolled": 27,
123       "course_publish_date": "2015-01-19",
124       "course_description": "Act 101 is a course from the business school on the introduction to accounting that teaches students how to read and compose basic financial statements"
125     }
126   }
127 }
```

But if I do greater than or equal to that's going to include this date as well.

```

1 GET /courses/_search
2 {
3     "query": {
4         "bool": {
5             "must": [
6                 {"match": {"name": "accounting"}}
7             ],
8             "must_not": [
9                 {"match": {"room": "e7"}}
10            ],
11            "should": [
12                {"range": {
13                    "students_enrolled": {
14                        "gte": 10,
15                        "lte": 20
16                    }
17                }}
18            ],
19        },
20        "minimum_should_match": 1
21    }
22 }
23 }

5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

```

nits": 1,
 "total": 1,
 "max_score": 1.6099695,
 "hits": [
 {
 "_index": "courses",
 "_type": "classroom",
 "_id": "8",
 "_score": 1.6099695,
 "_source": {
 "name": "Accounting Info Systems 350",
 "room": "E3",
 "professor": {
 "name": "Bill Cage",
 "department": "accounting",
 "facutly_type": "full-time",
 "email": "cageb@onuni.com"
 },
 "students_enrolled": 19,
 "course_publish_date": "2014-05-15",
 "course_description": "Act Sys 350 is an advanced course providing students a practical understanding of an accounting system in database technology. Students will use MS Access to build a transaction ledger system"
 }
 }
]
 }

All right so we work through a lot of examples delving into the query syntax in the Querrey DSL.

Lecture 12: Search DSL Filter Context

Tuesday, June 11, 2019 3:23 PM

- **Filter query** used for filtering documents that we need
- Filter queries are cached, so Elasticsearch will cache previously run queries

```
1 GET /courses/_search
2 {
3   "query": {
4     "bool": {
5       "filter": {
6         "bool": {
7           "must": [
8             { "match": {"professor.name": "bill"} },
9             { "match": {"name": "accounting"} }
10            ]
11          }
12        }
13      }
14    }
15  }
16}
17
```

8 },
9 "hits": {
10 "total": 2,
11 "max_score": 0,
12 "hits": [
13 {
14 "_index": "courses",
15 "_type": "classroom",
16 "_id": "8",
17 "_score": 0,
18 "_source": {
19 "name": "Accounting Info Systems 350",
20 "room": "E3",
21 "professor": {
22 "name": "Bill Cage",
23 "department": "accounting",
24 "facultly_type": "full-time",
25 "email": "cageb@onuni.com"
26 },
27 "students_enrolled": 19,
28 "course_publish_date": "2014-05-15",
29 "course_description": "Act Sys 350 is an advanced course providing students a practical understanding of an accounting system in database technology. Students will use MS Access to build a transaction ledger system"
30 }

And again look at the max score here.

33 "_index": "courses",

- Aspects within a filter query will NOT impact the relevancy score
- Within a multi-match, you can use field boosting to alter the guidelines of the relevancy score

```
11           "gte": 12
12         }
13       }
14     ]
15   }
16 }
17 }
18 },
19
20 "should": [
21   {"match": {"room": "e3"}},
22   {
23     "range": {
24       "students_enrolled": {
25         "gte": 13,
26         "lte": 14
27       }
28     }
29   },
30   {
31     "multi_match": {
32       "query": "market",
33       "fields": [ "name", "course_description^2" ]
34     }
35   }
36 ]
37 }
38 }
39
40
```

5 "total": 5,
6 "successful": 5,
7 "failed": 0
8 },
9 "hits": {
10 "total": 10,
11 "max_score": 1.8784901,
12 "hits": [
13 {
14 "_index": "courses",
15 "_type": "classroom",
16 "_id": "2",
17 "_score": 1.8784901,
18 "_source": {
19 "name": "Marketing 101",
20 "room": "E4",
21 "professor": {
22 "name": "William Smith",
23 "department": "finance",
24 "facultly_type": "part-time",
25 "email": "wills@onuni.com"
26 },
27 "students_enrolled": 18,
28 "course_publish_date": "2015-06-21",
29 "course_description": "Mkt 101 is a course from the business school on the introduction to marketing that teaches students the fundamentals of market analysis, customer retention and online advertisements"
30 }

with a factor of two.

- You can alter relevancy via the "^" symbol
- Queries are gonna be slower than filters
- Syntax wise the filter is very similar to the query where the same components can apply to both
- For EXACT matches with no care for relevancy, use FILTER
- For RELEVANCY SCORING and for FULL TEXT SEARCH, use a QUERY

Lecture 13: Aggregations DSL (Part 1)

Tuesday, June 11, 2019 3:24 PM

- <https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-bulk.html>
 - **BULK API** can be used to send multiple documents as part of the same request
1. POST /vehicles/cars/_bulk
 2. { "index": {} }
 3. { "price": 10000, "color": "white", "make": "honda", "sold": "2016-10-28", "condition": "okay" }
 4. { "index": {} }
 5. { "price": 20000, "color": "white", "make": "honda", "sold": "2016-11-05", "condition": "new" }
 6. { "index": {} }
 7. { "price": 30000, "color": "green", "make": "ford", "sold": "2016-05-18", "condition": "new" }
 8. { "index": {} }
 9. { "price": 15000, "color": "blue", "make": "toyota", "sold": "2016-07-02", "condition": "good" }
 10. { "index": {} }
 11. { "price": 12000, "color": "green", "make": "toyota", "sold": "2016-08-19", "condition": "good" }
 12. { "index": {} }
 13. { "price": 18000, "color": "red", "make": "dodge", "sold": "2016-11-05", "condition": "good" }
 14. { "index": {} }
 15. { "price": 80000, "color": "red", "make": "bmw", "sold": "2016-01-01", "condition": "new" }
 16. { "index": {} }
 17. { "price": 25000, "color": "blue", "make": "ford", "sold": "2016-08-22", "condition": "new" }
 18. { "index": {} }
 19. { "price": 10000, "color": "gray", "make": "dodge", "sold": "2016-02-12", "condition": "okay" }
 20. { "index": {} }
 21. { "price": 19000, "color": "red", "make": "dodge", "sold": "2016-02-12", "condition": "good" }
 22. { "index": {} }
 23. { "price": 20000, "color": "red", "make": "chevrolet", "sold": "2016-08-15", "condition": "good" }
 24. { "index": {} }
 25. { "price": 13000, "color": "gray", "make": "chevrolet", "sold": "2016-11-20", "condition": "okay" }
 26. { "index": {} }
 27. { "price": 12500, "color": "gray", "make": "dodge", "sold": "2016-03-09", "condition": "okay" }
 28. { "index": {} }
 29. { "price": 35000, "color": "red", "make": "dodge", "sold": "2016-04-10", "condition": "new" }
 30. { "index": {} }
 31. { "price": 28000, "color": "blue", "make": "chevrolet", "sold": "2016-08-15", "condition": "new" }
 32. { "index": {} }
 33. { "price": 30000, "color": "gray", "make": "bmw", "sold": "2016-11-20", "condition": "good" }

```

1 GET /vehicles/cars/_search
2 {
3   "from": 0,
4   "size": 5,
5
6   "query": {
7     "match_all": {}
8   },
9
10  "sort": [
11    {"price": {"order": "desc"}}
12  ]
13 }

```

Well there's a mechanism for queries It's called the query and fetch.

```

12  "hits": [
13    {
14      "_index": "vehicles",
15      "_type": "cars",
16      "_id": "AVyLIItUrq0M1E3BufUu6",
17      "_score": null,
18      "_source": {
19        "price": 80000,
20        "color": "red",
21        "make": "bmw",
22        "sold": "2016-01-01",
23        "condition": "new"
24      },
25      "sort": [
26        80000
27      ],
28    },
29    {
30      "_index": "vehicles",
31      "_type": "cars",
32      "_id": "AVyLIItUrq0M1E3BufUvB",
33      "_score": null,
34      "_source": {
35        "price": 35000,
36        "color": "red",
37        "make": "dodge",
38        "condition": "new"
39      },
40    },
41    "sort": [

```

- How does the paginate and sorting process work?
 - Mechanism called the “query and fetch”
- In this order:
 - “query” is the query portion
 - “sort” sorts the displayed entries
 - “from” and “size” configure the amount and form of displayed hits
- Aggregations are about gaining insights and analytics from your data
 - This is achieved through summarization

```

1 GET /vehicles/cars/_count
2 {
3   "query": {
4     "match": {"make": "dodge"}
5   }
6
7 }

```

```

1 {
2   "count": 5,
3   "_shards": {
4     "total": 5,
5     "successful": 5,
6     "failed": 0
7   }
8 }

```

OK so five cars are manufactured by Dodge.

- More powerful syntax for aggregations:
 - "aggs" (short for aggregation query)

```

1 GET /vehicles/cars/_search
2 {
3
4   "aggs": {
5     "popular_cars": {
6       "terms": {
7         "field": "make.keyword" [
8           ]
9         }
10      }
11    }
12  }
13 }
```

the inverted index so we can have sentences you know entire
 emails or blog posts into the text field.

```

7   "failed": 0
8   },
9   "hits": { [REDACTED],
10  "aggregations": {
11    "popular_cars": {
12      "doc_count_error_upper_bound": 0,
13      "sum_other_doc_count": 0,
14      "buckets": [
15        {
16          "key": "dodge",
17          "doc_count": 5
18        },
19        {
20          "key": "chevrolet",
21          "doc_count": 3
22        },
23        {
24          "key": "bmw",
25          "doc_count": 2
26        },
27        {
28          "key": "ford",
29          "doc_count": 2
30        },
31        {
32          "key": "honda",
33          "doc_count": 2
34        },
35        {
36          "key": "toyota",
37          "doc_count": 2
38        }
39      ],
40      "popular_cars": {
41        "doc_count_error_upper_bound": 0,
42        "sum_other_doc_count": 0,
43        "buckets": [
44          {
45            "key": "dodge",
46            "doc_count": 5,
47            "max_price": {
48              "value": 35000
49            },
50            "avg_price": {
51              "value": 18900
52            }
53          },
54          {
55            "key": "chevrolet",
56            "doc_count": 3,
57            "max_price": {
58              "value": 28000
59            },
60            "avg_price": {
61              "value": 20333.333333333332
62            }
63          },
64          {
65            "key": "bmw",
66            "doc_count": 2,
67            "max_price": {
68              "value": 80000
69            },
70            "avg_price": {
71              "value": 40000
72            }
73          }
74        ],
75        "popular_cars": {
76          "doc_count_error_upper_bound": 0,
77          "sum_other_doc_count": 0,
78          "buckets": [
79            {
80              "key": "dodge",
81              "doc_count": 5
82            },
83            {
84              "key": "chevrolet",
85              "doc_count": 3
86            },
87            {
88              "key": "bmw",
89              "doc_count": 2
90            },
91            {
92              "key": "ford",
93              "doc_count": 2
94            },
95            {
96              "key": "honda",
97              "doc_count": 2
98            },
99            {
100             "key": "toyota",
101             "doc_count": 2
102           }
103         ],
104         "popular_cars": {
105           "doc_count_error_upper_bound": 0,
106           "sum_other_doc_count": 0,
107           "buckets": [
108             {
109               "key": "dodge",
110               "doc_count": 5
111             },
112             {
113               "key": "chevrolet",
114               "doc_count": 3
115             },
116             {
117               "key": "bmw",
118               "doc_count": 2
119             },
120             {
121               "key": "ford",
122               "doc_count": 2
123             },
124             {
125               "key": "honda",
126               "doc_count": 2
127             },
128             {
129               "key": "toyota",
130               "doc_count": 2
131             }
132           ],
133           "popular_cars": {
134             "doc_count_error_upper_bound": 0,
135             "sum_other_doc_count": 0,
136             "buckets": [
137               {
138                 "key": "dodge",
139                 "doc_count": 5
140               },
141               {
142                 "key": "chevrolet",
143                 "doc_count": 3
144               },
145               {
146                 "key": "bmw",
147                 "doc_count": 2
148               },
149               {
150                 "key": "ford",
151                 "doc_count": 2
152               },
153               {
154                 "key": "honda",
155                 "doc_count": 2
156               },
157               {
158                 "key": "toyota",
159                 "doc_count": 2
160               }
161             ],
162             "popular_cars": {
163               "doc_count_error_upper_bound": 0,
164               "sum_other_doc_count": 0,
165               "buckets": [
166                 {
167                   "key": "dodge",
168                   "doc_count": 5
169                 },
170                 {
171                   "key": "chevrolet",
172                   "doc_count": 3
173                 },
174                 {
175                   "key": "bmw",
176                   "doc_count": 2
177                 },
178                 {
179                   "key": "ford",
180                   "doc_count": 2
181                 },
182                 {
183                   "key": "honda",
184                   "doc_count": 2
185                 },
186                 {
187                   "key": "toyota",
188                   "doc_count": 2
189                 }
190               ],
191               "popular_cars": {
192                 "doc_count_error_upper_bound": 0,
193                 "sum_other_doc_count": 0,
194                 "buckets": [
195                   {
196                     "key": "dodge",
197                     "doc_count": 5
198                   },
199                   {
200                     "key": "chevrolet",
201                     "doc_count": 3
202                   },
203                   {
204                     "key": "bmw",
205                     "doc_count": 2
206                   },
207                   {
208                     "key": "ford",
209                     "doc_count": 2
210                   },
211                   {
212                     "key": "honda",
213                     "doc_count": 2
214                   },
215                   {
216                     "key": "toyota",
217                     "doc_count": 2
218                   }
219                 ],
220                 "popular_cars": {
221                   "doc_count_error_upper_bound": 0,
222                   "sum_other_doc_count": 0,
223                   "buckets": [
224                     {
225                       "key": "dodge",
226                       "doc_count": 5
227                     },
228                     {
229                       "key": "chevrolet",
230                       "doc_count": 3
231                     },
232                     {
233                       "key": "bmw",
234                       "doc_count": 2
235                     },
236                     {
237                       "key": "ford",
238                       "doc_count": 2
239                     },
240                     {
241                       "key": "honda",
242                       "doc_count": 2
243                     },
244                     {
245                       "key": "toyota",
246                       "doc_count": 2
247                     }
248                   ],
249                   "popular_cars": {
250                     "doc_count_error_upper_bound": 0,
251                     "sum_other_doc_count": 0,
252                     "buckets": [
253                       {
254                         "key": "dodge",
255                         "doc_count": 5
256                       },
257                         ...
258                     ],
259                     "popular_cars": {
260                       "doc_count_error_upper_bound": 0,
261                       "sum_other_doc_count": 0,
262                       "buckets": [
263                         {
264                           "key": "dodge",
265                           "doc_count": 5
266                         },
267                         ...
268                       ],
269                     }
270                   ]
271                 ]
272               ]
273             ]
274           ]
275         ]
276       ]
277     ]
278   ]
279 }
```

- We use "make.keyword" to ensure that Elasticsearch treats whatever text is within the text field as one word instead of multiple terms (which would be the case w/ just "make")

```

1 GET /vehicles/cars/_search
2 {
3
4   "aggs": {
5     "popular_cars": {
6       "terms": {
7         "field": "make.keyword"
8       },
9       "aggs": {
10         "avg_price": {
11           "avg": {
12             "field": "price"
13           }
14         },
15         "max_price": {
16           "max": {
17             "field": "price"
18           }
19         }
20       }
21     }
22   }
23 }
24 }
```

That's the most expensive dodge that we have the most expensive
 Chevrolet we have is 28000 the most

```

140  "popular_cars": [
141    "doc_count_error_upper_bound": 0,
142    "sum_other_doc_count": 0,
143    "buckets": [
144      {
145        "key": "dodge",
146        "doc_count": 5,
147        "max_price": {
148          "value": 35000
149        },
150        "avg_price": {
151          "value": 18900
152        }
153      },
154      {
155        "key": "chevrolet",
156        "doc_count": 3,
157        "max_price": {
158          "value": 28000
159        },
160        "avg_price": {
161          "value": 20333.333333333332
162        }
163      },
164      {
165        "key": "bmw",
166        "doc_count": 2,
167        "max_price": {
168          "value": 80000
169        },
170        "avg_price": {
171          "value": 40000
172        }
173      }
174    ],
175    "popular_cars": [
176      "doc_count_error_upper_bound": 0,
177      "sum_other_doc_count": 0,
178      "buckets": [
179        {
180          "key": "dodge",
181          "doc_count": 5
182        },
183        {
184          "key": "chevrolet",
185          "doc_count": 3
186        },
187        {
188          "key": "bmw",
189          "doc_count": 2
190        },
191        {
192          "key": "ford",
193          "doc_count": 2
194        },
195        {
196          "key": "honda",
197          "doc_count": 2
198        },
199        {
200          "key": "toyota",
201          "doc_count": 2
202        }
203      ],
204      "popular_cars": [
205        "doc_count_error_upper_bound": 0,
206        "sum_other_doc_count": 0,
207        "buckets": [
208          {
209            "key": "dodge",
210            "doc_count": 5
211          },
212          {
213            "key": "chevrolet",
214            "doc_count": 3
215          },
216          {
217            "key": "bmw",
218            "doc_count": 2
219          },
220          {
221            "key": "ford",
222            "doc_count": 2
223          },
224          {
225            "key": "honda",
226            "doc_count": 2
227          },
228          {
229            "key": "toyota",
229            "doc_count": 2
230          }
231        ],
232        "popular_cars": [
233          "doc_count_error_upper_bound": 0,
234          "sum_other_doc_count": 0,
235          "buckets": [
236            {
237              "key": "dodge",
238              "doc_count": 5
239            },
240            {
241              "key": "chevrolet",
242              "doc_count": 3
243            },
244            {
245              "key": "bmw",
246              "doc_count": 2
247            },
248            {
249              "key": "ford",
249              "doc_count": 2
250            },
251            {
252              "key": "honda",
253              "doc_count": 2
254            },
255            {
256              "key": "toyota",
257              "doc_count": 2
258            }
259          ],
260          "popular_cars": [
261            "doc_count_error_upper_bound": 0,
262            "sum_other_doc_count": 0,
263            "buckets": [
264              {
265                "key": "dodge",
266                "doc_count": 5
267              },
268              ...
269            ],
270            "popular_cars": [
271              "doc_count_error_upper_bound": 0,
272              "sum_other_doc_count": 0,
273              "buckets": [
274                {
275                  "key": "dodge",
276                  "doc_count": 5
277                },
278                ...
279              ],
280            }
281          ]
282        ]
283      ]
284    ]
285  ]
286 }
```

- You can use queries to limit the range of the aggregation and simplify data analysis

```

1 GET /vehicles/cars/_search
2 {
3
4 }
```

```

1 GET /vehicles/cars/_search
2 {
3
4   "query": {
5     "match": {"color": "red"}
6   },
7
8   "aggs": {
9     "popular_cars": {
10       "terms": {
11         "field": "make.keyword"
12       },
13       "aggs": {
14         "avg_price": {
15           "avg": {
16             "field": "price"
17           }
18         },
19         "max_price": {
20           "max": {
21             "field": "price"
22           }
23         },
24         "min_price": {
25           "min": {
26             "field": "price"
27           }
28       }
29     }
30   }
}

```

It's the color red and the Shiver a lot.

```

96   }
97   },
98   {
99     "key": "bmw",
100    "doc_count": 1,
101    "max_price": {
102      "value": 80000
103    },
104    "min_price": {
105      "value": 80000
106    },
107    "avg_price": {
108      "value": 80000
109    }
110  },
111  {
112    "key": "chevrolet",
113    "doc_count": 1,
114    "max_price": {
115      "value": 20000
116    },
117    "min_price": {
118      "value": 20000
119    },
120    "avg_price": {
121      "value": 20000
122    }
123  }
124
125
}

```

- Running the aggregation will also show the documents in case you want to access the raw data via a query, filter, or for some other purpose
- To NOT display the individual documents/hits, either minimize them in the console OR set `["size": 0]` to show zero docs

Lecture 14: Aggregations DSL (Part 2)

Tuesday, June 11, 2019 3:24 PM

- For getting all of the statistics such as avg, min, max, sum, etc., we can use stats instead of running commands individually

```
1 GET /vehicles/cars/_search
2 {
3
4   "size":0,
5
6   "query": {
7     "match": {"color": "red"}
8   },
9
10  "aggs": {
11    "popular_cars": {
12      "terms": {
13        "field": "make.keyword"
14      },
15      "aggs": {
16        "stats_on_price": {
17          "stats": {
18            "field": "price"
19          }
20        }
21      }
22    }
23  }
24}
25
26 }
```

Notice that for Dodge it's giving the minimum maximum average
for BMD is giving the minimum maximum

```
4 _snaras": t
5   "total": 5,
6   "successful": 5,
7   "failed": 0
8 },
9 "hits": {
10   "total": 5,
11   "max_score": 0,
12   "hits": []
13 },
14 "aggregations": {
15   "popular_cars": {
16     "doc_count_error_upper_bound": 0,
17     "sum_other_doc_count": 0,
18     "buckets": [
19       {
20         "key": "dodge",
21         "doc_count": 3,
22         "stats_on_price": {
23           "count": 3,
24           "min": 18000,
25           "max": 35000,
26           "avg": 24000,
27           "sum": 72000
28         }
29       },
30       {
31         "key": "bmw",
32         "doc_count": 1,
33         "stats_on_price": {
34           "count": 1
35         }
36       }
37     ]
38   }
39 }
```

- Structure known as buckets and metrics
- Creating a bucket called "popular_cars"
- Defining the metric to be "stats_on_price"

```
1 GET /vehicles/cars/_search
2 {
3
4   "size":0,
5
6   "aggs": {
7     "popular_cars": {
8       "terms": {
9         "field": "make.keyword"
10      },
11      "aggs": {
12        "sold_date_ranges": {
13          "range": {
14            "field": "sold",
15            "ranges": [
16              { "from": "2016-01-01", "to": "2016-05-18" },
17              { "from": "2016-05-18", "to": "2017-01-01" }
18            ]
19          }
20        }
21      }
22    }
23  }
24}
25
26 }
```

That's how you learn.

```
72 BUCKETS : L
73   [
74     {
75       "key": "2016-01-01T00:00:00.000Z",
76       "from": 1451606400000,
77       "from_as_string": "2016-01-01T00
78       :00:00.000Z",
79       "to": 1463529600000,
80       "to_as_string": "2016-05-18T00:00
81       .000Z",
82       "doc_count": 1
83     },
84     {
85       "key": "2016-05-18T00:00:00.000Z",
86       "from": 1463529600000,
87       "from_as_string": "2016-05-18T00
88       :00:00.000Z",
89       "to": 1483228800000,
90       "to_as_string": "2017-01-01T00:00
91       .000Z",
92       "doc_count": 1
93     }
94   ],
95   [
96     {
97       "key": "ford",
98       "doc_count": 2,
99     }
100   ]
101 }
```

```

26
27.     }
28.   }
29. }
30. }
```

That's how you learn.

```

      "key": "ford",
      "doc_count": 2,
      "sold_date_ranges": {
        "buckets": [

```

```

5
6.   "aggs": {
7.     "popular_cars": {
8.       "terms": {
9.         "field": "make.keyword"
10.        },
11.       "aggs": {
12.         "sold_date_ranges": {
13.           "range": {
14.             "field": "sold",
15.             "ranges": [
16.               {
17.                 "from": "2016-01-01",
18.                 "to": "2016-05-18"
19.               },
20.               {
21.                 "from": "2016-05-18",
22.                 "to": "2017-01-01"
23.               }
24.             ]
25.           }
26.         }
27.       }
28.     }
29.   }
30. }
```

```

      "key": "dodge",
      "doc_count": 2,
      "sold_date_ranges": {
        "buckets": [
          {
            "key": "2016-01-01T00:00:00.000Z-2016-05-18T00:00:00.000Z",
            "from": 1451606400000,
            "from_as_string": "2016-01-01T00:00:00.000Z",
            "to": 1463529600000,
            "to_as_string": "2016-05-18T00:00:00.000Z",
            "doc_count": 4,
            "avg_price": {
              "value": 19125
            }
          },
          {
            "key": "2016-05-18T00:00:00.000Z-2017-01-01T00:00:00.000Z",
            "from": 1463529600000,
            "from_as_string": "2016-05-18T00:00:00.000Z",
            "to": 1483228800000,
            "to_as_string": "2017-01-01T00:00:00.000Z",
            "doc_count": 1,
            "avg_price": {
              "value": 18000
            }
          }
        ]
      }
    }
  }
}
```

and that vehicles price was 18000 cable.

```

1 GET /vehicles/cars/_search
2 {
3
4   "size": 0,
5
6.   "aggs": {
7.     "car_conditions": {
8.       "terms": {
9.         "field": "condition.keyword"
10.        },
11.       "aggs": {
12.         "avg_price": {
13.           "avg": {
14.             "field": "price"
15.           }
16.         }
17.       }
18.     }
19.   }
20. }
```

```

10.   "total": 16,
11.   "max_score": 0,
12.   "hits": []
13. },
14.   "aggregations": {
15.     "car_conditions": {
16.       "doc_count_error_upper_bound": 0,
17.       "sum_other_doc_count": 0,
18.       "buckets": [
19.         {
20.           "key": "good",
21.           "doc_count": 6,
22.           "avg_price": {
23.             "value": 19000
24.           }
25.         },
26.         {
27.           "key": "new",
28.           "doc_count": 5,
29.           "avg_price": {
30.             "value": 36333.333333333336
31.           }
32.         },
33.         {
34.           "key": "okay",
35.           "doc_count": 4,
36.           "value": 11375
37.         }
38.       ]
39.     }
40.   }
41. }
```

We've got six new vehicles the average price is thirty six thousand three hundred thirty three.

```

1 GET /vehicles/cars/_search
2 {
3
4 "size":0,
5
6 "aggs": {
7   "car_conditions": {
8     "terms": {
9       "field": "condition.keyword"
10    },
11    "aggs": {
12      "avg_price": {
13        "avg": {
14          "field": "price"
15        }
16      },
17      "make": {
18        "terms": {
19          "field": "make.keyword"
20        }
21      }
22    }
23  }
24 }
25 }
26 }
27 }
```

We got one shover lot that is in good condition.

```

1 GET /vehicles/cars/_search
2 {
3
4 "size":0,
5
6 "aggs": {
7   "car_conditions": {
8     "terms": {
9       "field": "condition.keyword"
10    },
11    "aggs": {
12      "avg_price": {
13        "avg": {
14          "field": "price"
15        }
16      },
17      "make": {
18        "terms": {
19          "field": "make.keyword"
20        },
21        "aggs": {
22          "min_price": {"min": {"field": "price"}},
23          "max_price": {"max": {"field": "price"}}
24        }
25      }
26    }
27  }
28 }
29 }
```

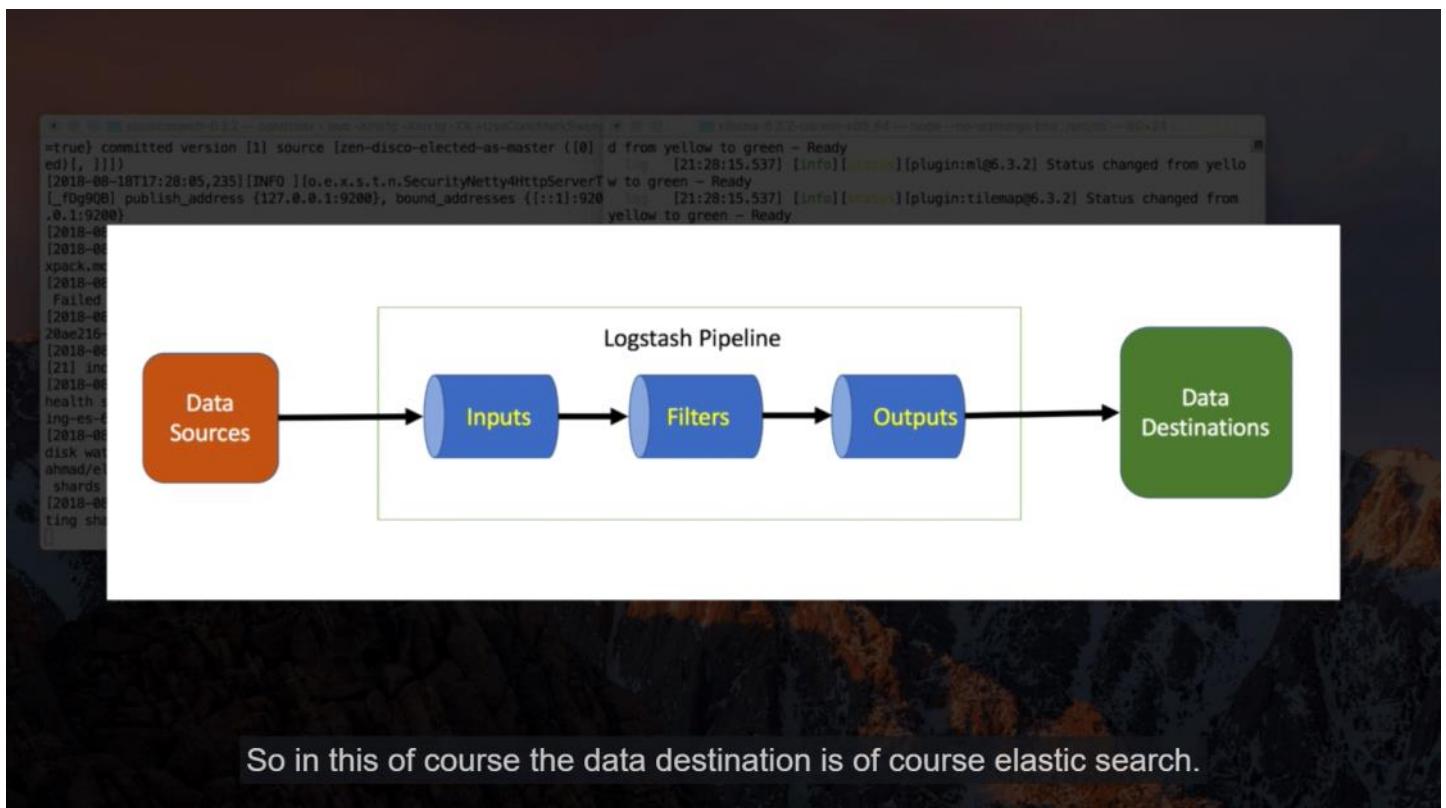
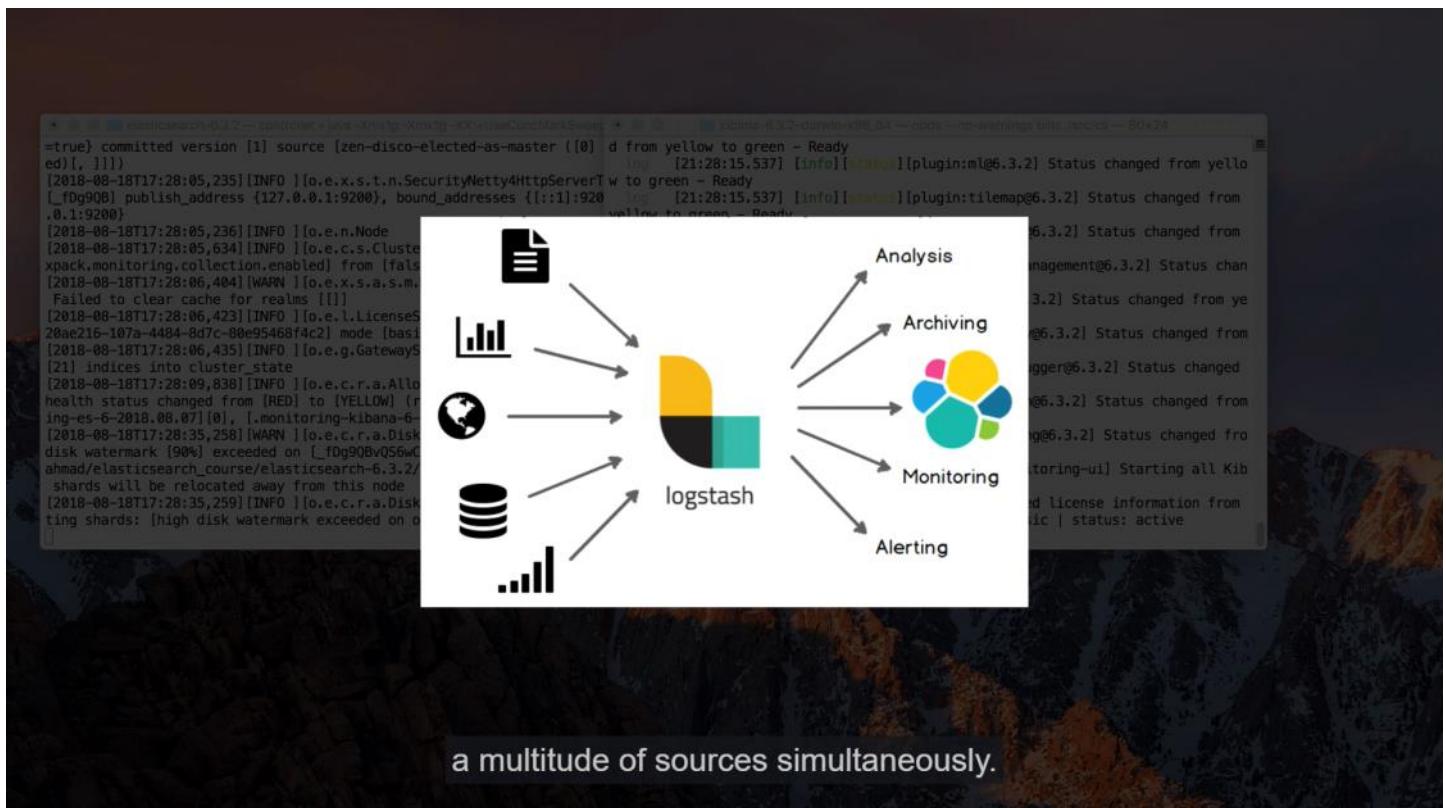
So notice how powerful this query language is.

- Concurrently running a query and an aggregation based off that query is known as aggregation scoping and can be done in real time (much faster than current relational databases)

Lecture 15: Download and Configure Logstash

Wednesday, July 10, 2019 8:15 AM

- **Logstash** is a very powerful open source data processing pipeline engine and it's used to ingest data from a multitude of sources simultaneously



- Configured in the JRuby language

```

input
{
  ...
}

filter
{
  ...
}

output
{
  ...
}

```

This syntax we're going to be going over how to write it and configure log stash to work with our cluster.

- To test logstash:

```

Last login: Sat Aug 18 16:59:40 on ttys000
~: $ cd elasticsearch_course/
~/elasticsearch_course: $ ls
elasticsearch-6.3.2/      logstash-6.3.2/
kibana-6.3.2-darwin-x86_64/ logstash-6.3.2.tar.gz
~/elasticsearch_course: $ cd logstash-6.3.2
~/elasticsearch_course/logstash-6.3.2: $ ls
CONTRIBUTORS      bin/
Gemfile           config/
Gemfile.lock      data/
LICENSE.txt       lib/
NOTICE.TXT        logstash-core/
~/elasticsearch_course/logstash-6.3.2: $ bin/logstash -e 'input { stdin {} } output { stdout {} }'
Unrecognized VM option 'UseParNewGC'
Error: Could not create the Java Virtual Machine.
Error: A fatal exception has occurred. Program will exit.
~/elasticsearch_course/logstash-6.3.2: $

```

Includes the [full set of tree features](#).

View the detailed release notes [here](#).
Not the version you're looking for? View [past releases](#).
The pure Apache 2.0 licensed distribution is available [here](#).
Java 8 is required for Logstash 6.x and 5.x.
program Glugs.

Lecture 16: Overview of Logstash & Indexing Apache Logs

Wednesday, July 10, 2019 8:59 AM

```
untitled
10
11 filter
12 {
13     grok{
14         match => {
15             "message" => "%{COMBINEDAPACHELOG}"
16         }
17     }
18     mutate{
19         convert => { "bytes" => "integer" }
20     }
21     date {
22         match => [ "timestamp", "dd/MMM/YYYY:HH:mm:ss Z" ]
23         locale => en
24         remove_field => "timestamp"
25     }
26     geoip {
27         source => "clientip"
28     }
29     useragent {
30         source => "agent"
31         target => "useragent"
32     }
33 }
34
35
36 output
37 {
38     stdout {
39         codec
40     }
41 }
```

But real quickly in the filter section we can put various plug ins and codec in this particular case there

- In configuration file, filter section, we can instantiate multiple plugins
 - grok, mutate, date, geoip, useragent
 - Grok - regex matcher (built-in vars that are predefined (APACHELOG))
 - <https://github.com/elastic/logstash/blob/v1.4.2/patterns/grok-patterns>
 - Mutate - convert data types
 - Date - formats dates from the log into a certain format specified
 - Geoip - finds IP address from the log
 - Useragent - stores whether the user is using a phone or tablet or PC

The screenshot shows a web browser displaying the official Elasticsearch documentation for the Grok filter plugin. The URL is <https://www.elastic.co/guide/en/logstash/6.2/plugins-filters-grok.html>. The page title is "Grok filter plugin". Key sections include "Description", "Getting Help", and "On this page" which lists various Logstash-related topics.

Description

Parse arbitrary text and structure it.

Grok is a great way to parse unstructured log data into something structured and queryable. This tool is perfect for syslog logs, apache and other webserver logs, mysql logs, and in general, any log format that is generally written for humans and not computer consumption.

So definitely read through this documentation as well in combination with that file and it will make

If you need help building patterns to match your logs, you will find the

On this page

- Getting Help
- Description
- Grok Basics
- Regular Expressions
- Custom Patterns
- Grok Filter Configuration Options
- Common Options

Logstash Reference: 6.2

- Logstash Introduction
- Getting Started with Logstash
- Setting Up and Running Logstash
- Setting Up X-Pack

- Change path to "\Users\A647655\Downloads\data\logs\logs"

Lecture 17: Kibana Visualizations and Dashboards Overview

Wednesday, July 10, 2019 9:52 AM

- After running the conf file, each dot that appears after basically represents a particular event or ingestion that logstash went through to index the log entry

A screenshot of a web browser displaying a Kibana dashboard. The top tab shows 'elk-index-size-tests/logs.gz' and the bottom tab shows 'localhost:5601/app/kibana#/management/kibana/index?_g={}'. The main content area displays a log entry from Elasticsearch:

```
[2018-08-18T19:29:53,756] [INFO] logstash-2014.06.24] creating [logstash-2014.06.24], shards [5]/[1], mappings [{}]
```

Below the log entry, there is a list of indices:

- logstash-2014.06.09
- logstash-2014.06.10
- logstash-2014.06.11

A tooltip is overlaid on the screen, reading: "a particular event or ingestion that logstash went through to index the log entry. Rows per page: 10".

- localhost:9200/logstash-*/_count to find total number of docs and shards that logstash parsed

A screenshot of a web browser displaying a Kibana management interface. The left sidebar shows navigation options: Discover, Visualize, Dashboard, Timelion, APM, Dev Tools, Monitoring, and Management. The Management option is selected. The main content area is a table showing Elasticsearch indices and their metrics:

Name	Health	Status	Primaries	Replicas	Docs count	Storage size	Primary storage size
logstash-2014.06.08	● yellow	open	5	1	13055	9.5mb	9.5mb
logstash-2014.06.13	● yellow	open	5	1	10866	7.9mb	7.9mb
logstash-2014.06.05	● yellow	open	5	1	8512	6.8mb	6.8mb
logstash-2014.05.29	● yellow	open	5	1	7894	6.5mb	6.5mb
logstash-2014.06.03	● yellow	open	5	1	7556	6mb	6mb
business	● yellow	open	5	1	2	9.8kb	9.8kb
logstash-2014.06.10	● yellow	open	5	1	10612	8.3mb	8.3mb
logstash-2014.06.06	● yellow	open	5	1	11214	8.2mb	8.2mb
logstash-2014.05.28	● yellow	open	5	1	1006	1.6mb	1.6mb
logstash-2014.06.14	● yellow	open	5	1	10370	8mb	8mb

A tooltip is overlaid on the screen, reading: "the number of replicas is 1 so we didn't mess around with any of those and you could just keep clicking".

the number of replicas is 1 so we didn't mess around with any of those and you could just keep clicking

Rows per page: 10

1 2 3 4

Show All

- By default the number of primaries is 5 and the number of replicas is 1

Index management

Update your Elasticsearch indices individually or in bulk.

Search:

Name	Health	S
customers	yellow	o
employees	yellow	o
logstash-2014.06.02	yellow	o
logstash-2014.06.17	yellow	o
logstash-2014.06.18	yellow	o
logstash-2014.05.30	yellow	o

logstash-2014.06.02

Summary Settings Mapping Stats Edit settings

```
89     type: "date"
90   },
91   "@version": {
92     "type": "keyword"
93   },
94   "agent": {
95     "type": "text",
96     "norms": false,
97     "fields": {
98       "keyword": {
99         "type": "keyword",
100        "ignore_above": 256
101      }
102    }
103  },
104  "auth": {
105    "type": "text",
106    "norms": false,
107    "fields": {
108      "keyword": {
109        "type": "keyword",
110        "ignore_above": 256
111      }
112    }
113 }
```

OK we've got off GOP client IP all of this so it's generating a bunch of these you know country code

X Close

Manage

Show All

- You must define an index pattern and choose an option for time filter in order to create your index pattern for visualizations and dashboard generation

Kibana Management / Kibana

Step 1 of 2: Define index pattern

Index pattern

logstash-*

You can use a * as a wildcard in your index pattern.
You can't use spaces or the characters \, /, ?, <, >, |.

✓ Success! Your index pattern matches 31 indices.

logstash-2014.05.28
logstash-2014.05.29
logstash-2014.05.30
logstash-2014.05.31
logstash-2014.06.01
logstash-2014.06.02
logstash-2014.06.03

31 indices that we have here.

Kibana Management / Kibana

Index Patterns Saved Objects Reporting Advanced Settings

No default index pattern. You must select or create one to continue.

Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

X Include system indices

Step 2 of 2: Configure settings

You've defined logstash-* as your index pattern. Now you can specify some settings before we create it.

Time Filter field name Refresh

The Time Filter will use this field to filter your data by time.
You can choose not to have a time field, but you will not be able to narrow down your data by a time range.

> Show advanced options

Back Create index pattern

So we'll just pick time stamp and you can click on create index pattern.

The screenshot shows the Kibana Management interface. On the left, a sidebar lists various features: Discover, Visualize, Dashboard, Timelion, APM, Dev Tools, Monitoring, and Management. The Management option is currently selected. The main area is titled "logstash-*". It displays a table of fields with the following columns: Name, Type, Format, Searchable, and AggregatableExcluded. The fields listed are:

Name	Type	Format	Searchable	AggregatableExcluded
request	string		●	
request.keyword	string		●	●
response	string		●	
response.keyword	string		●	●

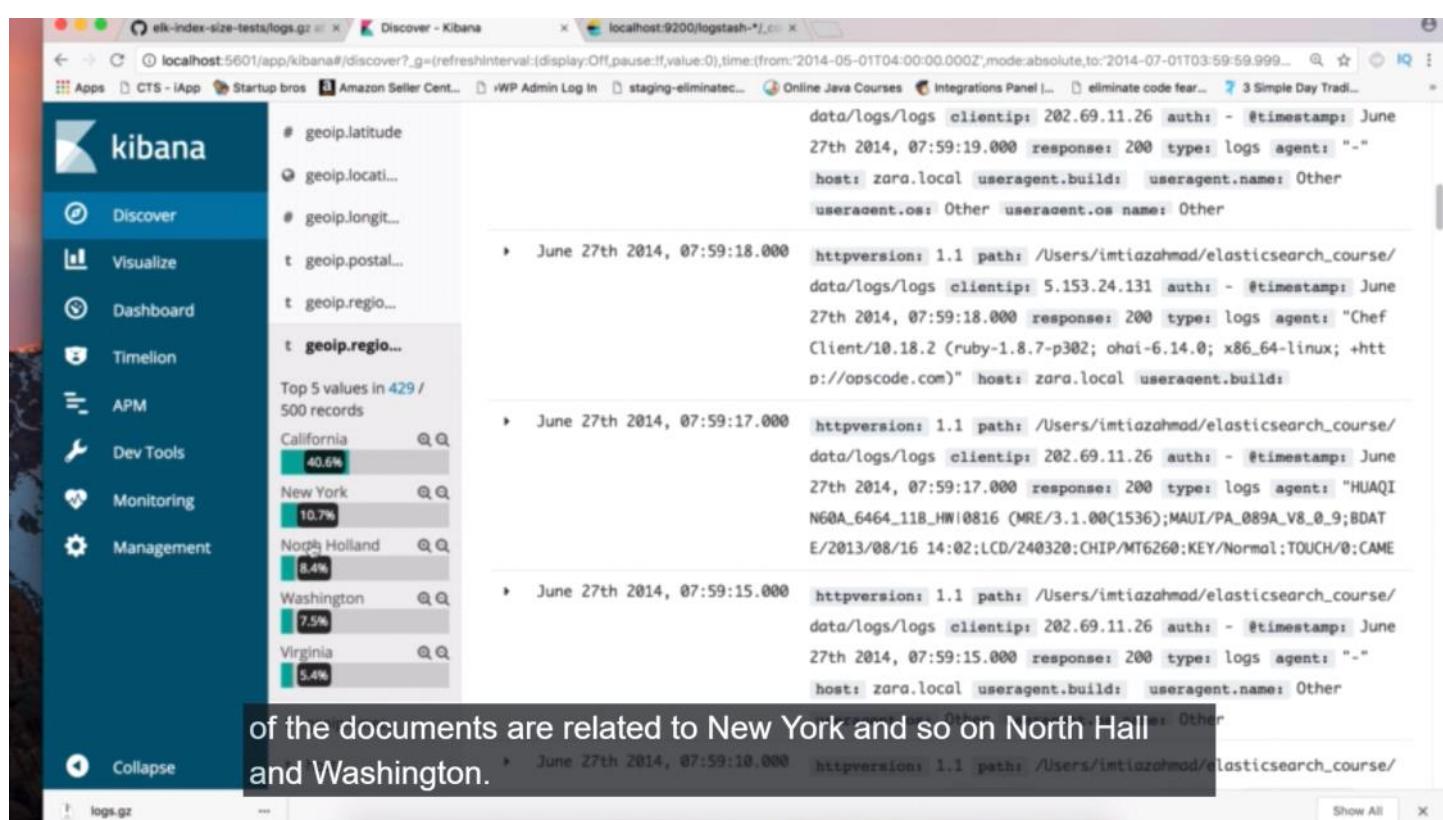
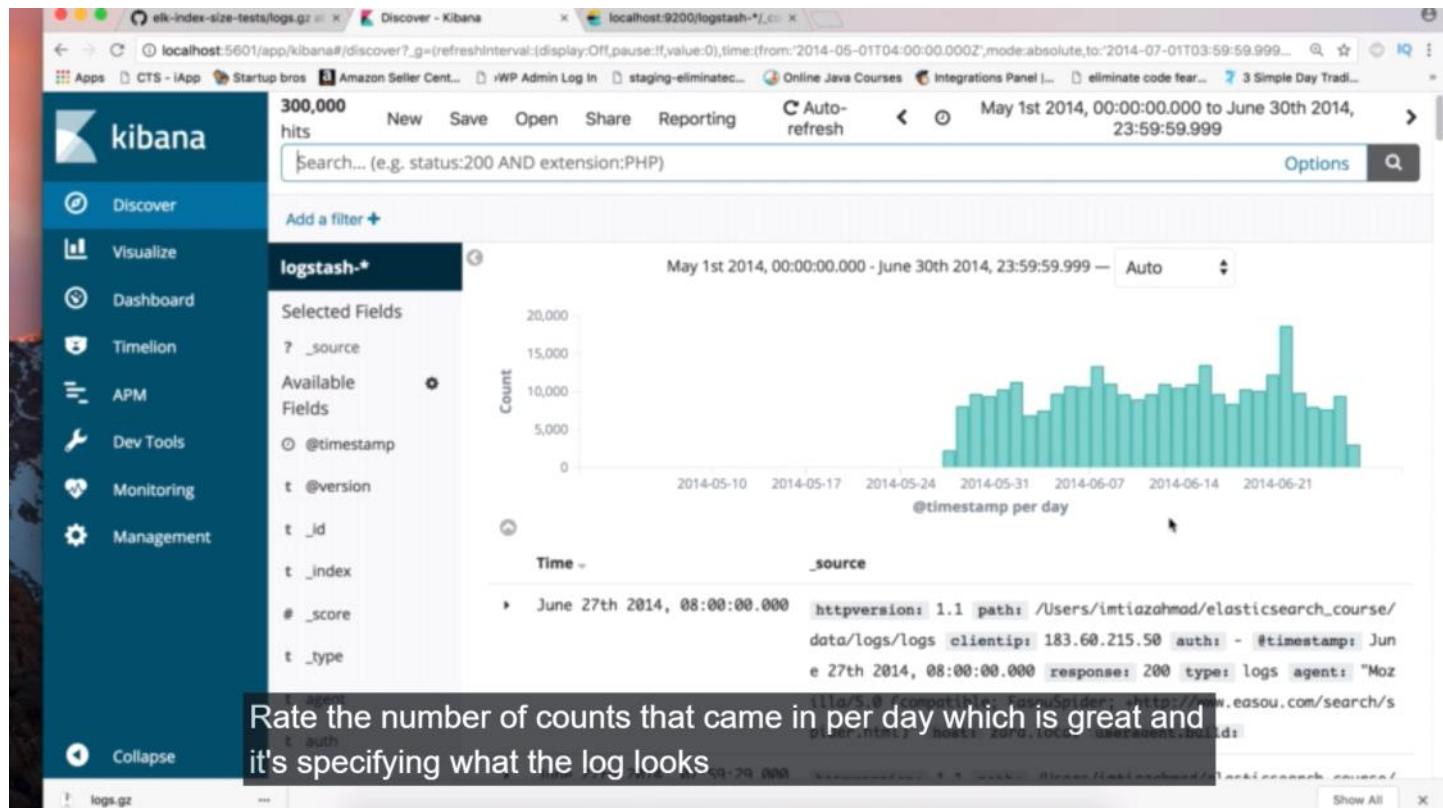
A note at the top states: "This page lists every field in the **logstash-*** index and the field's associated core type as recorded by Elasticsearch. To change a field type, use the Elasticsearch Mapping API".

These are all fields that belong on a given log stash index document.

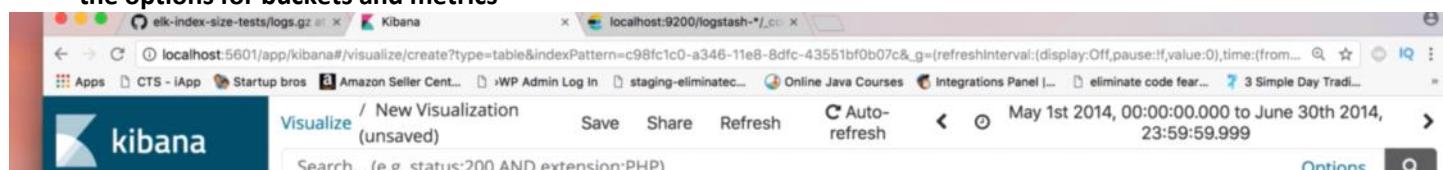
The screenshot shows the Kibana Discover interface. The sidebar is identical to the previous screen. The main area has a search bar with the placeholder "Search... (e.g. status:200 AND extension:PHP)". Below the search bar, there is a section titled "logstash-*" with two tabs: "Selected Fields" and "Available Fields". The "Available Fields" tab is selected. It displays a message: "No results match your search criteria". Below this, there are sections for "Expand your time range" and "Refine your query". The "Refine your query" section contains examples of Elasticsearch queries:

- Find requests that contain the number 200, in any field
- Find 200 in the status field
- Find all status codes between 400-499

So we need to pick a time range for the data that we are interested in.



- Creating a data table is as easy as clicking "Visualize", choosing our index pattern, and filling in the options for buckets and metrics



The screenshot shows the Kibana Visualize interface with a table visualization titled "logstash-*". The table lists cities based on their count, with Amsterdam at the top. A modal window displays the same data in a table format.

	Count
Amsterdam	11,283
Leander	9,108
San Jose	8,229
Campinas	7,868
Beijing	6,799

A prominent message box states: "11000 documents came from Amsterdam so that's 11000 log entries that have to do with Amsterdam."

The screenshot shows the Kibana Dashboard interface with a donut chart visualization titled "test_dashboard". The chart shows the distribution of cities, with Amsterdam being the largest segment. A legend on the right identifies the cities by color.

City	Count
Amsterdam	11,283
Leander	9,108
San Jose	8,229
Campinas	7,868
Beijing	6,799

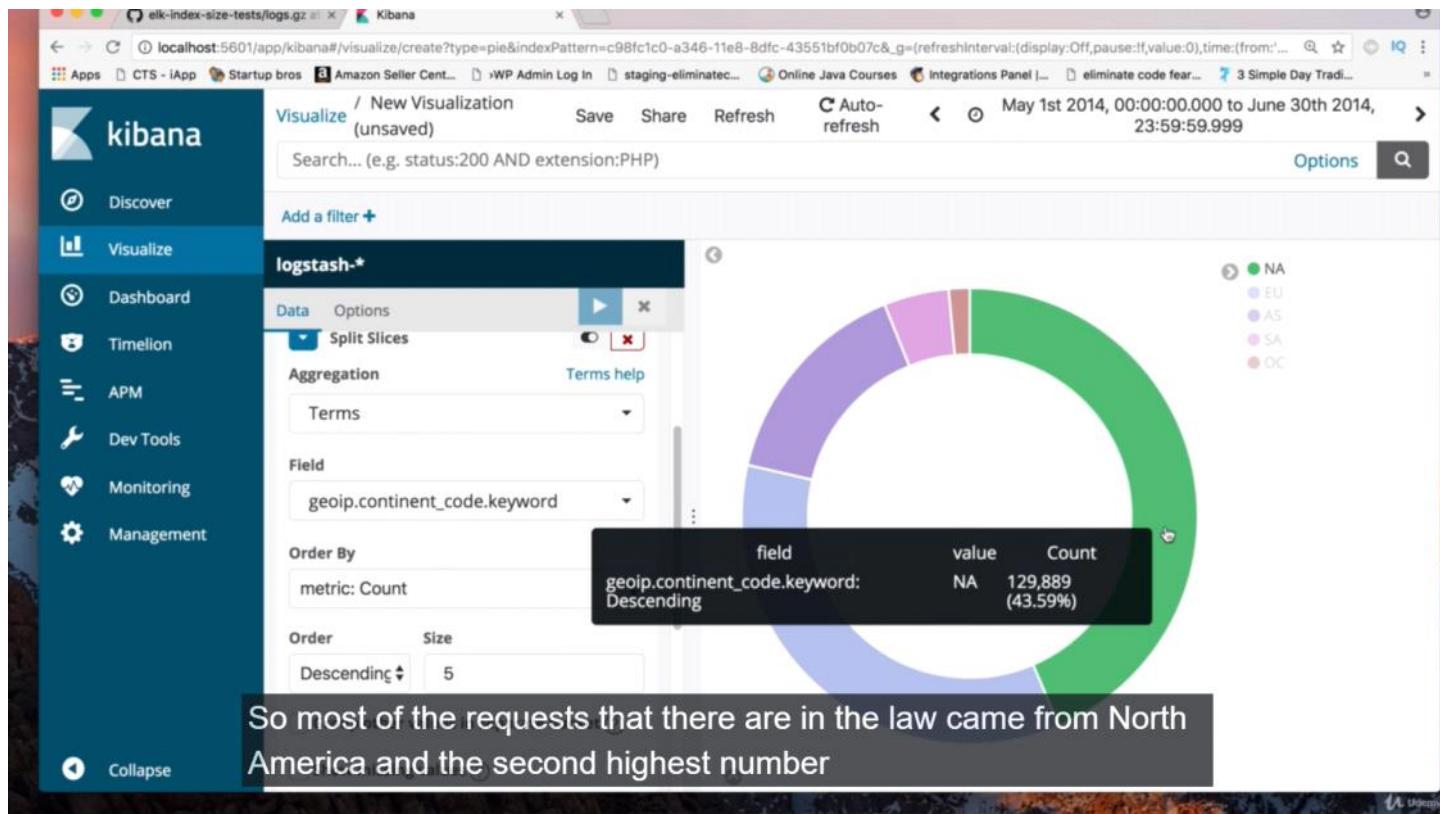
A message box states: "We're going to get into the details of how to create visualizations in the next lecture."

- Visualizations, dashboards, and more!

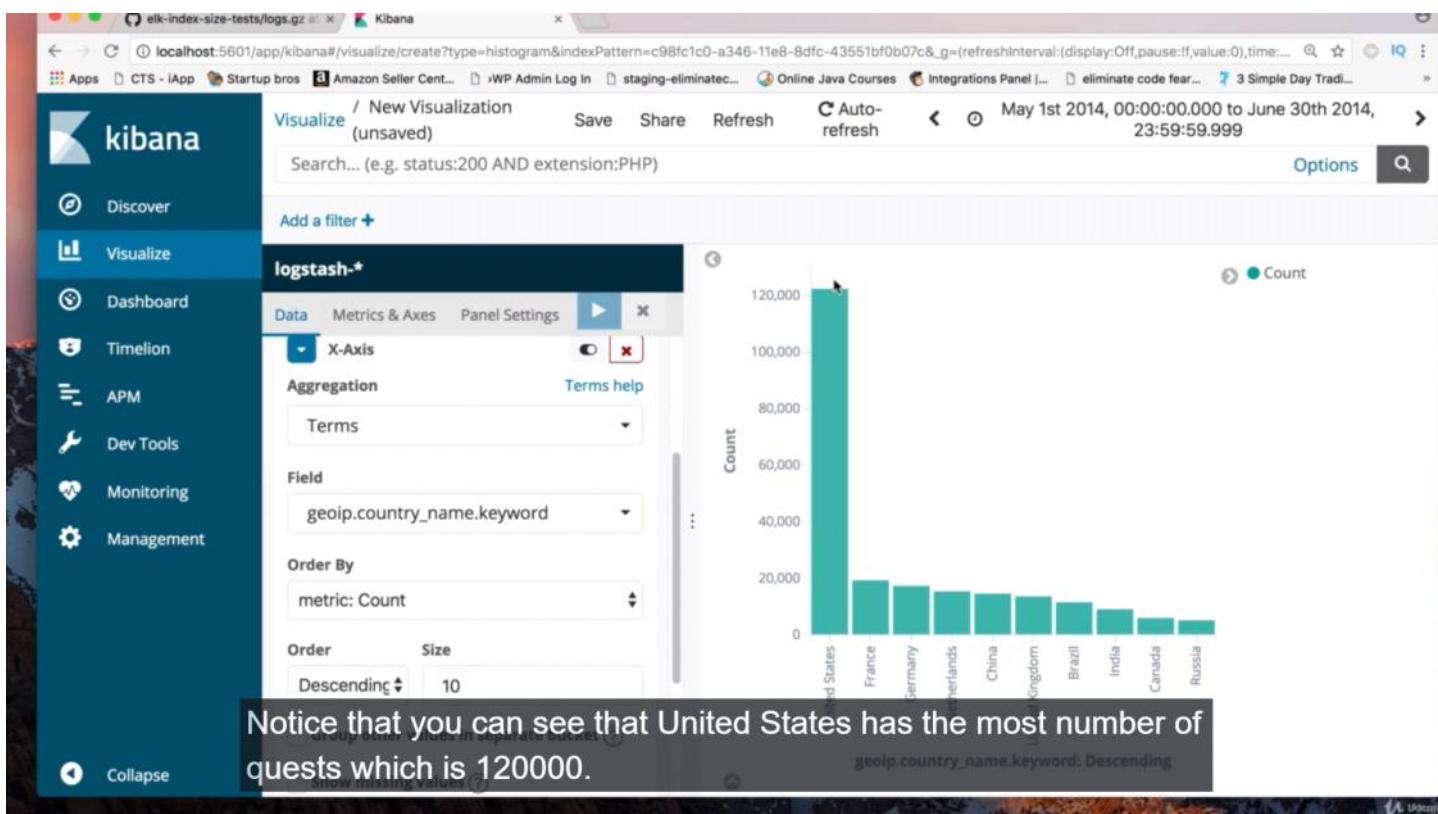
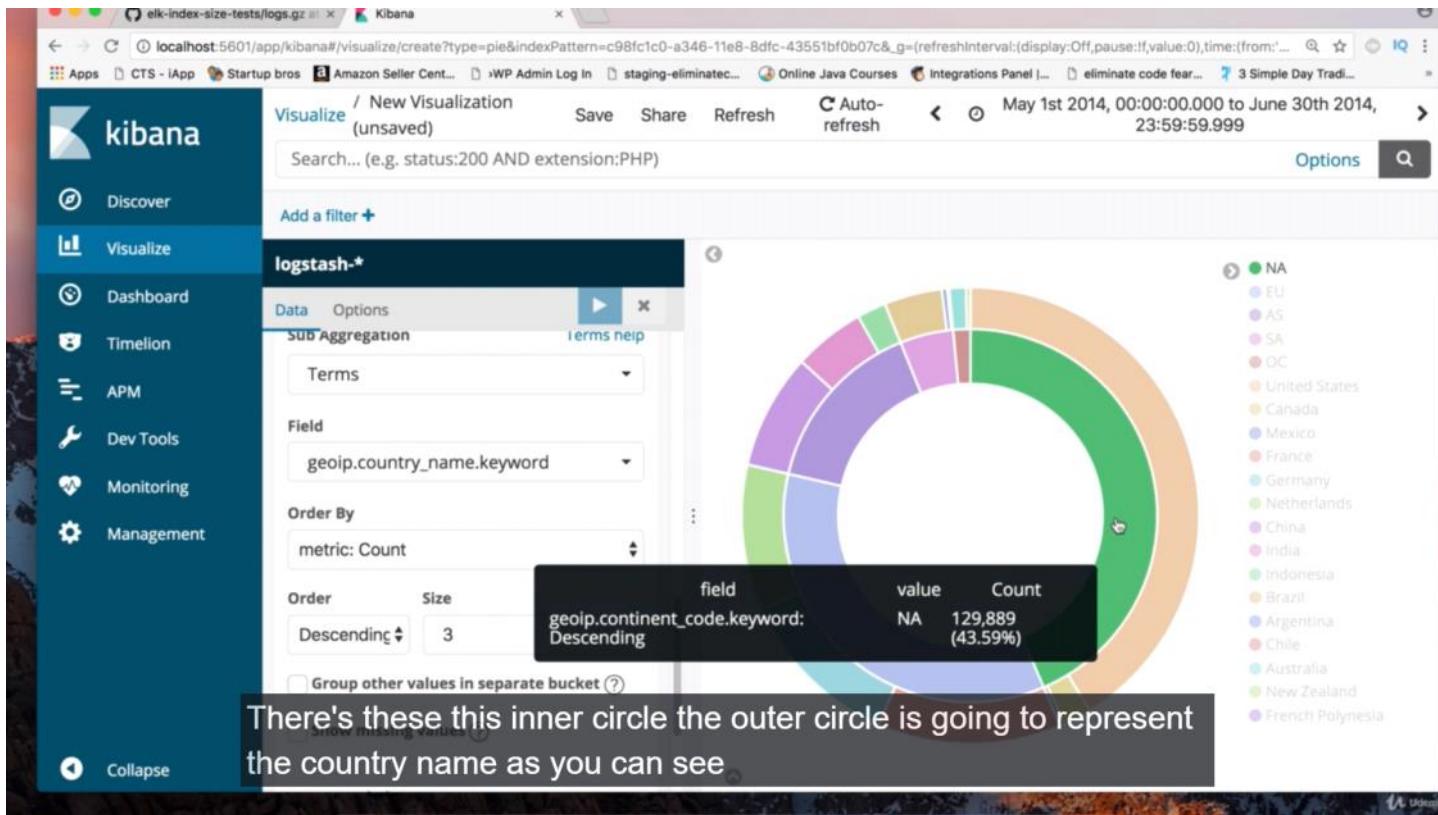
Lecture 18: More on Visualizations and Dashboards

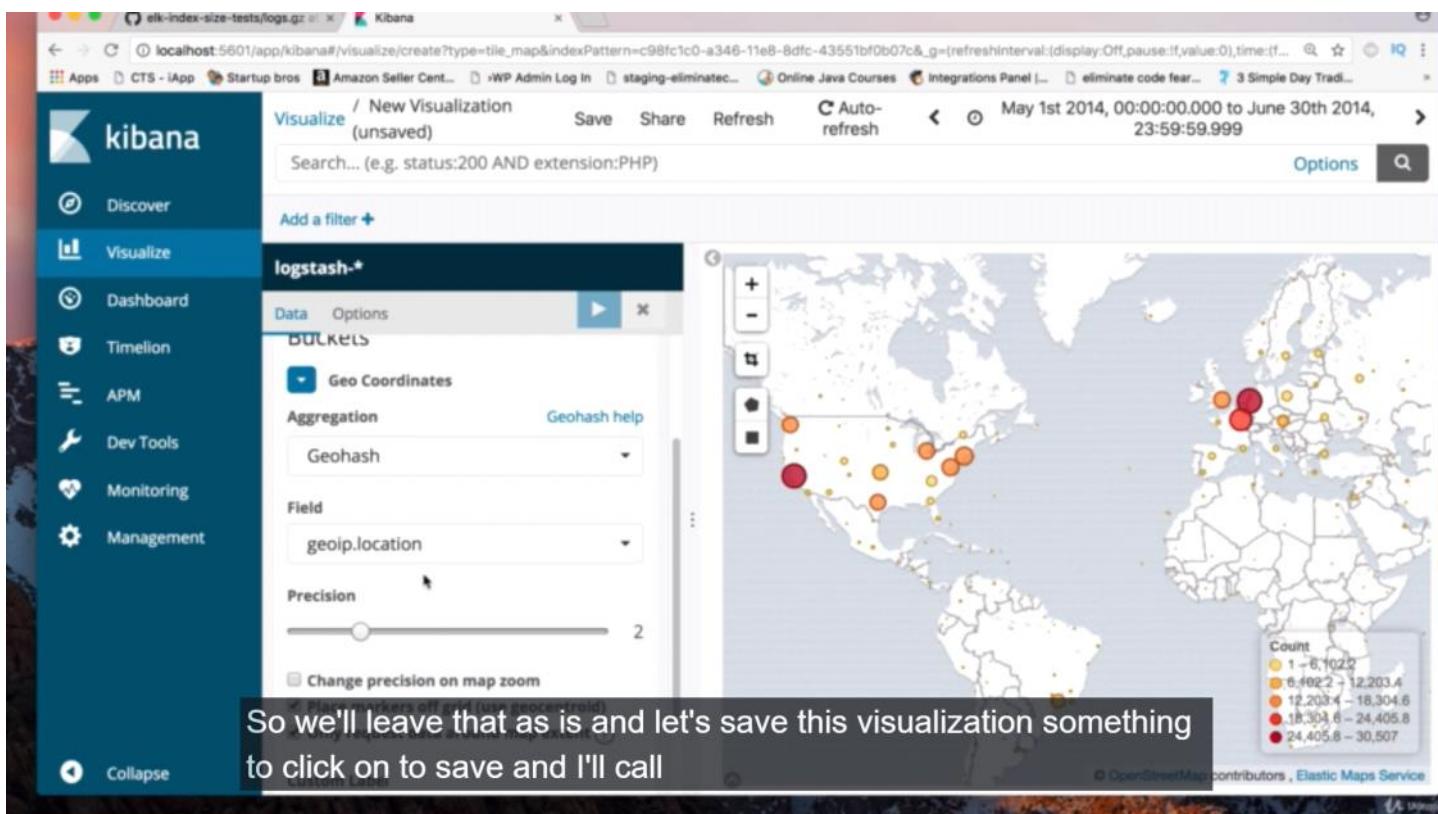
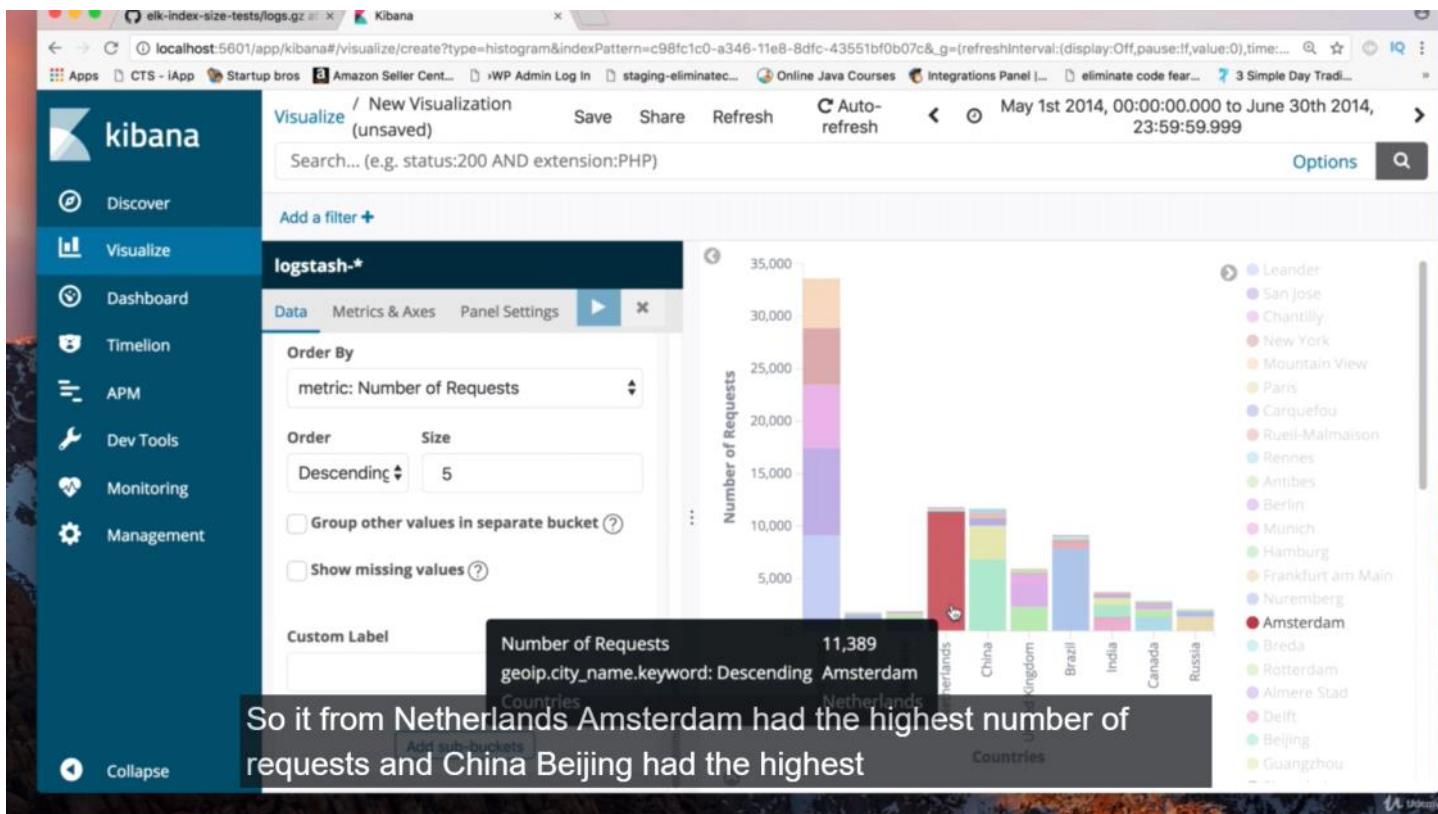
Wednesday, July 10, 2019 10:35 AM

- Once the indexing is complete, the indexes need to be basically identified by an index pattern



- You can use sub-buckets to add multiple levels of visualization

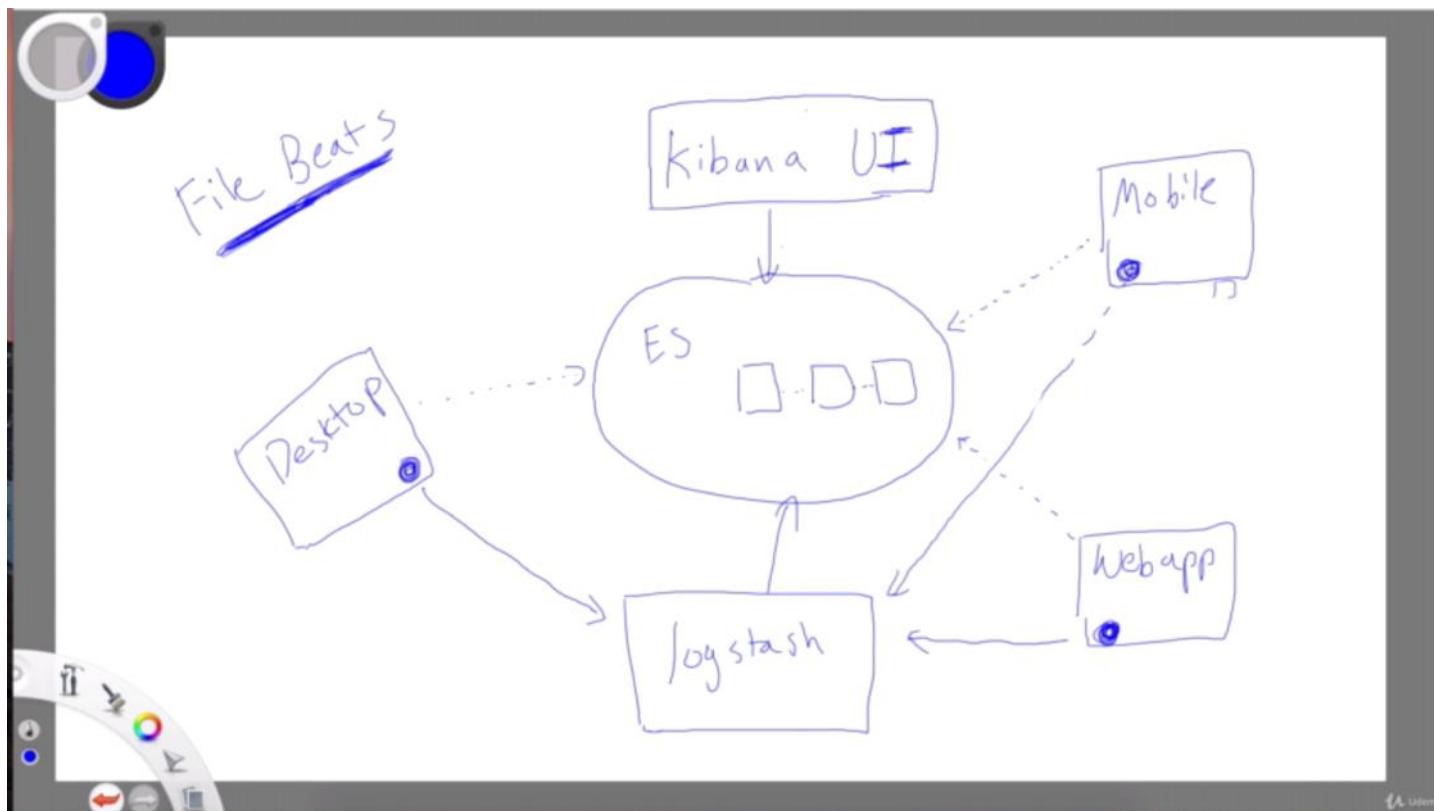


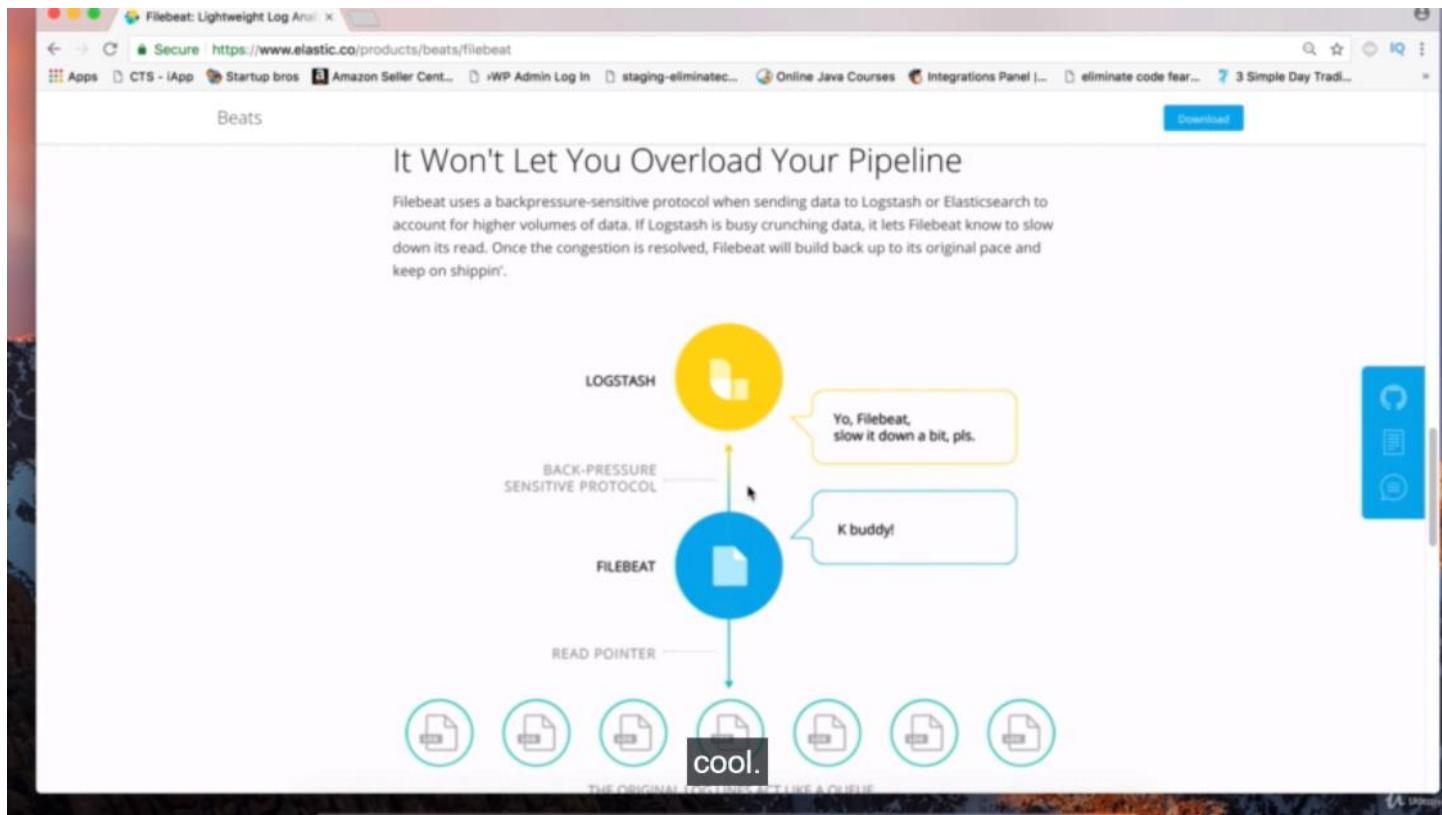


Lecture 19: Architecting the Elastic Stack

Wednesday, July 10, 2019 11:11 AM

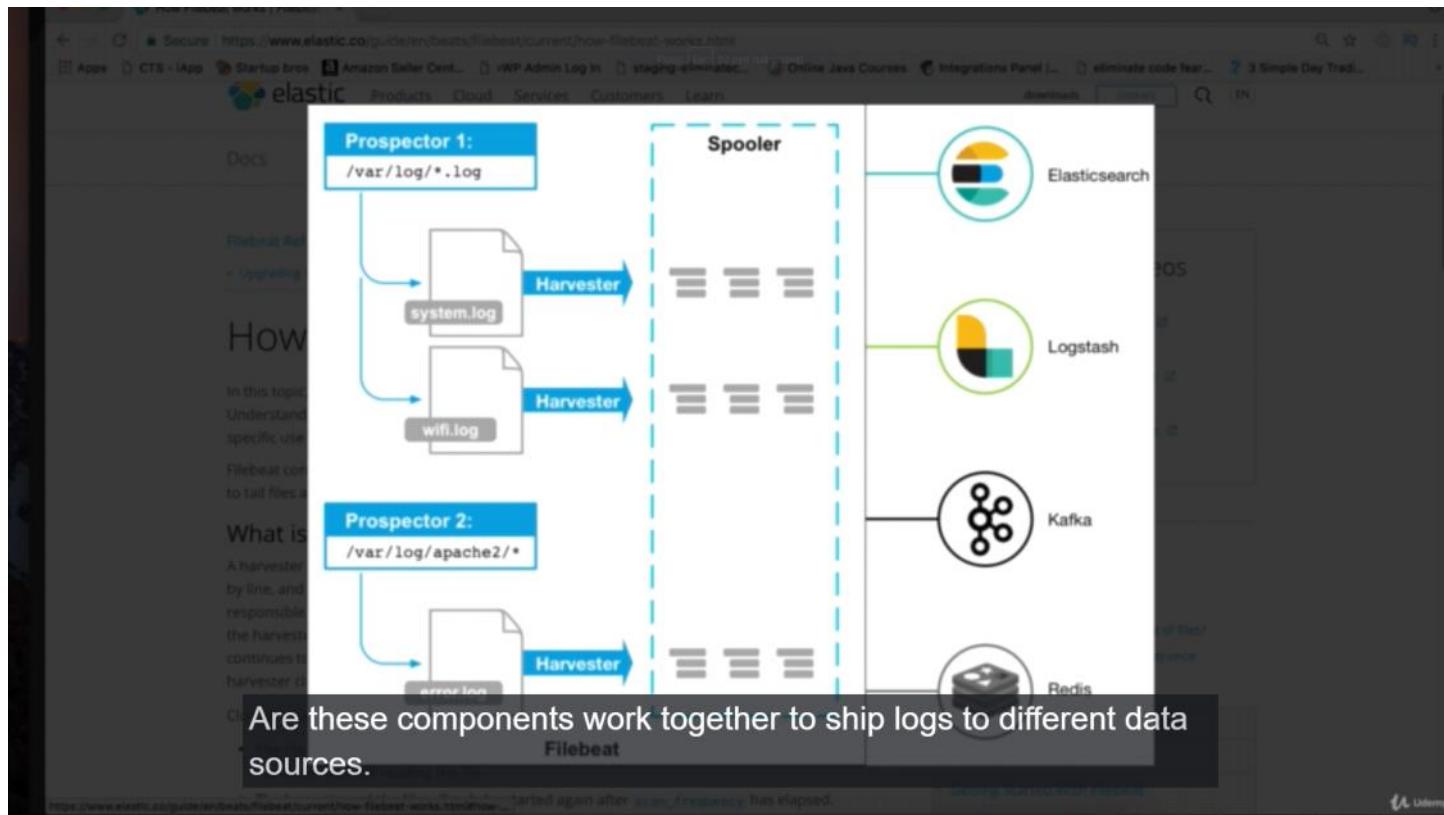
- ES cluster used to interact w/ Kibana UI for query requests, index requests, etc.
- Logstash is used to parse logs and send them to ES
 - Creates a pipeline for parsing and indexing events or logs
 - Enhances data via plugins
- Ex. e-commerce business w/ mobile and web apps
 - Products for sale online
 - Applications have to interact w/ ELK stack for data analytics, insights, etc.
- DO NOT: install Logstash into outward-facing apps (bad architecture b/c resource-intensive)
 - INSTEAD: have a lightweight process that can ship application logs to the Logstash server so that the server is able to parse those logs and index them into ES
- Can utilize SCP servers or rsync to push data over, OR use:
 - BEATS, which are lightweight data servers that can be deployed on your edge apps
- Filebeat allow for shipping logs to Logstash or other tools such as Kafka, MongoDB, MySQL, etc.
- Logstash (parsing, indexing) is meant to reside on a central location where logs are stashed and parsed and then sent for indexing into ES (dedicated node)
- Metricbeat is a lightweight shipper that periodically collects metrics from the OS.
- Packetbeat is used for networking
- Heartbeat logs the performance of the application and its operations
- Winlogbeat, Auditbeat, Mongobeat, Amazonbeat all fall within the Beats auxiliary tool ecosystem
- <https://www.elastic.co/products/beats/filebeat>





Lecture 20: Filebeat Installation and Overview

Wednesday, July 10, 2019 12:57 PM



- These components work together to tail files and send event data to the output that you specify
- Summary of how Filebeat software works^

Lecture 21: Using Filebeat with Logstash

Wednesday, July 10, 2019 1:23 PM

- **Ship logs over to Logstash and let Logstash do the indexing**
- <https://www.elastic.co/guide/en/beats/filebeat/current/logstash-output.html>
- <https://notepad.pw/sw891zjs>