

SIG Workshop Elastic Search & Kibana

20th February 2018

Elastic Search [Index]: NoSQL, REST API, distributed, highly available. Great for searching. Well integrated from Log Stash (for gathering and loading data, in real time, for example from log files) and Kibana (for exploring and visualizing data in the Search Index).

In this workshop:

- Get Elastic Search and Kibana up and running in a local Docker based environment
 - Optionally: use cloud based environment
 - Optionally: install directly on your OS or in a VM
- Create an Elastic Search Index through the REST API and experiment with search options
- Leverage Kibana to explore the search index [documents] in a user friendly, visual way and create several visualizations (tag cloud, geographic map, bar chart, ...) on top of the data
- Compare Elastic Search queries with (Oracle) SQL
- Try out the combination of LogStash, Elastic Search and Kibana for gathering, storing, querying and analyzing log files – in a fully hosted environment

Resources

Elastic Documentation:

Sources for this workshop: <https://github.com/lucasjellema/sig-elasticsearch-february-2018>

1. Prepare your environment

We assume a Docker host on which you can start containers.

Alternatives in case you do not have Docker available to you:

- Use an Elastic Search cloud environment (get an almost free trial environment at <https://www.elastic.co/cloud/as-a-service> or a \$0,055 / hour environment at <https://qbox.io>)
- Install Elastic Search and Kibana on your local operating system or inside a Virtual Machine (start at <https://www.elastic.co/downloads/elasticsearch>)

Installation of Elastic Stack on Docker

To run Elastic Search [Index Server] on Docker, all you need to do is run the following command:

```
docker run -d --name elasticsearch -p 9200:9200 -p 9300:9300 elasticsearch
```

This will run the image *elasticsearch* in a container also called *elasticsearch* exposing ports 9200 and 9300 from that container to those same portnumbers on the Docker host.

You can extend the command to:

```
docker run -d --name elasticsearch -p 9200:9200 -p 9300:9300 -v  
/temp/elasticdata:/usr/share/elasticsearch/data elasticsearch
```

Here a volume mapping is added for the folder into which Elastic Search writes its data to a directory on the Docker host called /temp/elasticdata.

From the Windows Host - in case you are running Docker on Windows – you can verify whether ES is running:

```
curl -X GET http://192.168.99.100:9200
```

where the IP address is the address assigned to the Virtual Machine that is the host for the Docker Server.

To start a container for Kibana – associated with the Elastic Search Server we already started, use this Docker command:

```
docker run --name kibana --link elasticsearch:elasticsearch -p 5601:5601 -  
d kibana
```

This command runs a container called *kibana* from the image *kibana* and exposes port 5601 from that container to the Docker host. The container is linked to an existing container called *elasticsearch* that is made available inside the Kibana container as a host with name *elasticsearch*.

When Kibana is started successfully, it can be accessed from your browser, at:

<http://host:5601>

which on my Windows host for the Docker server becomes:

<http://192.168.99.100:5601>

Additional Tooling

Additionally, we will be using Postman – the REST API testing tool: <https://www.getpostman.com/apps> (with downloads for Windows, Mac and Linux).

You may be interested in using the Chrome App ElasticSearch Head that can be installed into Chrome. Browse from: https://chrome.google.com/webstore/detail/elasticsearch-head/ffmkiejjmecolpfloofpjologoblkegm?hl=en-US&utm_source=chrome-ntp-launcher.

The sources used in this workshop can be acquired from GitHub: <https://github.com/lucasjellema/sign-elasticsearch-february-2018>.

2. Quick Take Off – The National Parks data set

In this section, we take a dataset with details on US National Parks, turn it into an Elastic Search Index and explore the data. First we use the REST API to perform a number of searches. Then we turn to Kibana to make these data explorations easier and more fun. Note: from these visual data explorations you may get idea for embedding programmatic queries in your own application, directly against the Elastic Search Index.

The data used in this practice is in the file national-parks-data.json . It was downloaded from the website data.world: <https://data.world/kevinnayar/us-national-parks/workspace/file?filename=data.json> . I had to prepare the data a little in order to be ingestable into an Elastic Search Index:

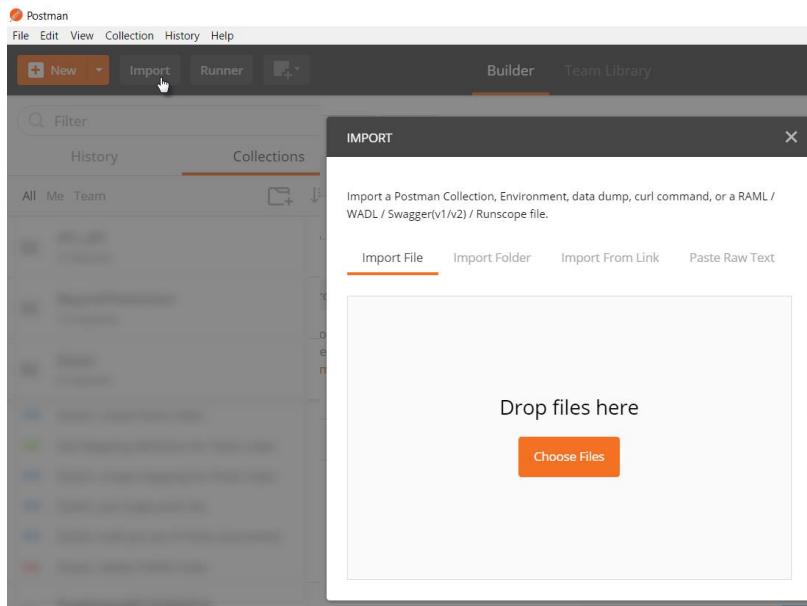
- Change the property names longitude and latitude to lon and lat respectively
- Remove the thousands separator (",") from the fields that represent numbers (visitors, acres, square_km)

In this lab, we will use Postman to interact with Elastic Search and create the index, the mapping definition and load the data in one bulk operation. Subsequently we will query the data, still through REST calls made from Postman.

Establish the Connection to Elastic Search

Start Postman.

Import the Postman collection ElasticSearch-NationalParks from file ElasticSearch-NationalParks.postman_collection.json.



After the import completes, you should see a folder (or collection) called Elastic. It contains definitions for multiple REST API calls to the Elastic Search API.

Elastic

requests

PUT Elastic: create Parks index

GET Get Mapping definition for Parks index

PUT Elastic: create mapping for Parks index

PUT Elastic: put single park doc

PUT Elastic: bulk put set of Parks documents

DEL Elastic: delete PARKS index

These requests refer to the endpoint of the Elastic Search server you are accessing through an environment variable called `ELASTIC_HOME`. Before you can successfully execute these requests, you have to define this variable in an Environment definition in Postman and set its value to whatever the endpoint is in your environment.



Click on Manage Environments. Then select an existing environment or create a new one. In this environment, define the variable `ELASTIC_HOME` and set its value according to your local environment:

The screenshot shows the Postman 'Manage Environments' interface. In the 'Edit Environment' dialog for 'local', there is a table with two columns: 'Key' and 'Value'. The 'ELASTIC_HOME' row is selected, indicated by a red border around both the key and value cells. The 'Value' cell contains the IP address 'http://192.168.99.100'. At the bottom right of the dialog are 'Cancel' and 'Update' buttons.

| Key | Value |
|--|-----------------------|
| <input checked="" type="checkbox"/> ELASTIC_HOME | http://192.168.99.100 |

(in my case, I use Docker Quickstart Terminal on Windows. Elastic Search is running in a container inside the Linux Virtual Machine that was setup by Docker Quickstart Terminal to run the Docker Server. This VM is accessed at IP address 192.168.99.100. The container running Elastic Search exposes ports (9200 and 9300) on the Docker Host. These are therefore accessible from the Windows Host machine – where Postman is running – at this IP address.)

To verify the connection, run the Postman Request called *Elastic: check health of server*. Upon a successful response, you can continue. Otherwise you will first have to find out what is wrong and correct it (is the server running did you use the right name for the Postman Environment variable, is the correct environment selected in Postman at the time of making the request, is the IP address used for the ELASTIC_HOME endpoint correct and can you ping that endpoint successfully from the machine where Postman is running?)

Create the Index and Load the Data

Elastic Search is NoSQL database that allows us to just store JSON document without specifying the structure of those documents in advance. Now that we have a server up and running and we can access it from Postman, nothing is stopping us from loading data into that server and by doing so, creating an index.

Execute the Postman PUT Request *Elastic: bulk put set of Parks documents*. Check the response. It should have response code 200 and contain a logging of all the documents that were created in the index. The data for these documents was in the body of the request. Some 59 documents should be created by this request.

```

1 {
2   "took": 123,
3   "errors": false,
4   "items": [
5     {
6       "create": {
7         "_index": "parks",
8         "_type": "doc",
9         "_id": "6",
10        "_version": 1,
11        "_result": "created",
12      }
13    }
14  ]
15 }

```

While processing the request, Elastic Search has extracted meta-data about the documents. This meta-data is used for creating inverted indexes that in turn are used during queries.

Execute the request called *Get Mapping definition for Parks index*. The response contains the meta-data that was derived by Elastic Search.

```

1 {
2   "parks": {
3     "mappings": {
4       "doc": {
5         "properties": {
6           "area": {
7             "properties": {
8               "acres": {
9                 "type": "text",
10                "fields": {
11                  "keyword": {
12                    "type": "keyword",
13                    "ignore_above": 256
14                  }
15                }
16              }
17            }
18          }
19        }
20      }
21    }
22  }

```

This mapping document tells us about the properties that Elastic Search has identified, the names it will use for these properties and also the type that was assigned to each property. These types determine which operations we can perform on the properties. Numerical properties can be aggregated and used in mathematical operations, but text properties cannot. In the automatically derived mapping document, we can see that properties acres, square_km and visitors have not been identified as numerical. Also the property *date_established_readable* is identified as text – even though we know it to actually contain the textual representation of a date.

Geopoints can be used in geospatial queries. The coordinates property in the national park documents describe a geographical location, but ES has not recognized this property for what it is - but as just a pair of *float* values.

Additionally, some of the text analysis capabilities of Elastic Search depend on the specific language the text is provided in. ES does not derive the language automatically, we have to instruct it when the language is not the default.

In order to fully leverage the querying capabilities of Elastic Search – and the visualization capabilities of Kibana – we should define the mapping definitions for the national park documents ourselves, to help ES do a better job. See the ES documentation on Mapping for more details:

<https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html> .

We will start from fresh with our index.

Execute Postman request *Elastic: delete PARKS index* to delete the *parks* index. Next, execute request *Elastic: create Parks index*. This creates the index – empty, without data or meta-data.

Now we will add the enriched meta-data – with more instructions for Elastic Search. Execute request *Elastic: create mapping for Parks index*.

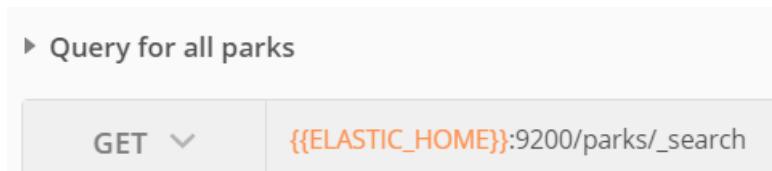
Check the body of this request, and find the following refinements:

- Geopoint type for *coordinates*
- Float type for acres and square_km
- Integer type for visitors
- Date type (plus format) for *date_established_readable*
- Type text and fielddata : true for *description*; also: associated custom analyzer std_english with field description (to remove stopwords from this field; see:
<https://www.elastic.co/guide/en/elasticsearch/reference/current/configuring-analyzers.html> for details)

Execute request *Elastic: bulk put set of Parks documents* to upload all 59 documents to the index. At this point, Elastic Search has treated the properties in the right way, which will now allow us to query the data in a more meaningful, rich way.

Query the Data through the REST API

The most simple and straightforward way to retrieve records from an Elastic Search index is using a simple GET request, as is shown in Postman Request *Query for all parks*. This is a simple GET request to the _search operation on the index:



A screenshot of the Postman interface. The title bar says "Query for all parks". Below it, the method is set to "GET" and the URL is "{{ELASTIC_HOME}}:9200/parks/_search".

This simply returns all National Park Records from the index, with all fields, no scores or sorting.

Sorting, Batching and Focusing

The Postman Request *Query for all parks - sorted on world heritage site and number of visitors* shows how the query result can be in the request. Specifically, it shows how the sorting of the returned records is indicated, how the size (and start point) of the returned set is determined and how the fields returned in the result can be selected:

▶ Query for all parks - sorted on world heritage site and number of visitors

POST {{ELASTIC_HOME}}:9200/parks/_search

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 {  
2   "query": {  
3     "match_all": {}  
4   }  
5   "sort": [  
6     {"world_heritage_site": "desc"},  
7     {"visitors": "asc"}  
8   ]  
9   , "_source": ["title", "states.title", "visitors", "description", "date_established_readable"]  
10  , "size": 40  
11 }
```

Use wildcards in Queries

In Postman request *Query Parks - in a state whose name starts with M* the query is on property states.title. In this case we want to retrieve all parks who are in a state whose title starts with an M. The wild card character * is used, to indicate that the M can be followed by any sequence of characters.

▼ Query Parks - in a state whose name starts with M

Add a description

POST {{ELASTIC_HOME}}:9200/parks/_search

Authorization Headers (1) Body Pre-request Script

form-data x-www-form-urlencoded raw binary

```
1 {  
2   "query": {  
3     "query_string": {  
4       "query": "M*"  
5       , "fields": ["states.title"]  
6     }  
7   }  
8 }  
9 }
```

See the ES for details on wildcards in queries:

<https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-wildcard-query.html>.

Sort records on Geospatial attributes

Elastic Search is aware of geo-spatial properties and leverage that awareness in search operations. One example is seen in Postman request *Query Parks - sort for distance from New York*. Here parks are sorted based on their distance to New York City (whose coordinates are 40.7 (latitude) and -74 (longitude)).

▶ Query Parks - sort for distance from New York

POST {{ELASTIC_HOME}}:9200/parks/_search

Authorization Headers (1) Body **raw** Pre-request Script Te

form-data x-www-form-urlencoded raw binary **JSC**

```
1 {  
2   "query": {  
3     "query_string": {  
4       "query": "mountain"  
5     }  
6   },  
7   "sort": { "_geo_distance": {  
8     "coordinates": "40.7,-74",  
9     "order": "asc",  
10    "unit": "km"  
11  }  
12 }
```

Score records on a simple search term

The Postman Request *Query Parks - that contain Mountain* performs a simple query. It search parks for the word “mountain”. It takes all properties into account. It will calculate a score for all records, based on the presence of contents that match – a little or very much – on that term.

▶ Query Parks - that contain Mountain

POST {{ELASTIC_HOME}}:9200/parks/_search

Authorization Headers (1) Body **raw** Pre-request Script

form-data x-www-form-urlencoded raw binary

```
1 {  
2   "query": {  
3     "query_string": {  
4       "query": "mountain"  
5     }  
6   }  
7 }
```

Combine scored condition with absolute filters

A query can contain both absolute filters conditions – that determine if records can be part of the result set at all – and criteria that enhance the score when matched, but are not a hard prerequisite.

The query in request *Query and Filter Parks - that ideally contain Mountain and are a World Heritage Site for sure* has a combination of these two types of conditions:

▶ Query and Filter Parks - that ideally contain Mountain and are a World Heritage Site for sure

POST {{ELASTIC_HOME}}:9200/parks/_search

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json) ▾

```
1 {  
2   "query": {  
3     "bool": {  
4       "should": {  
5         "match": {  
6           "description": "mountain"  
7         }  
8       },  
9       "filter": {  
10         "term": {  
11           "world_heritage_site": true  
12         }  
13       }  
14     }  
15   }  
16 }
```

This request returns only parks who are a world heritage site. These parks are score based on whether their description contains mountain (a lot).

Highlighting and Query Explanation

Elastic Search search results can be enriched in several ways. One is by explicitly highlighting the terms that were matched and the sentences that contained them. Another is by explaining how the search result was arrived at.

Execute this Postman Request:

▼ Query Parks by description with as much largest and mountain as possible - with highlighting and query execution explanation

Add a description

| | | | | |
|--|--|--------------------------------------|------------------------------|---------------------------|
| POST | {{ELASTIC_HOME}}:9200/parks/_search?explain=true | Params | | |
| Authorization | Headers (1) | Body | | |
| | | Pre-request Script Tests | | |
| <input type="radio"/> form-data | <input type="radio"/> x-www-form-urlencoded | <input checked="" type="radio"/> raw | <input type="radio"/> binary | JSON (application/json) ▾ |
| 1 { 2 "query": { 3 "match": { 4 "description": "largest mountain" 5 } 6 }, 7 "highlight" : { 8 "fields" : { 9 "description" : {} 10 } 11 } | | | | |

The results will contain both the result highlighting and the query explanation. Remove the URL query parameter *explain* to return to a simpler result set.

Aggregation

The simplest aggregation is calculation of totals across all documents.

Use Query Total Park Visitors Count to perform such a grand total style query to get the sum of all visitors across all parks.

▼ Query Total Park Visitors Count

Add a description

POST {{ELASTIC_HOME}}:9200/parks/_search?explain=true

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary **JSON (application/json)**

```

1 [{}]
2   "size": 0,
3   "aggs": {
4     "visitor_count": {
5       "sum": {
6         "field": "visitors"
7       }
8     }
9   }
10 ]

```

Usually when we do aggregations, we like to make use of grouping – or bucketing in Elastic Search terminology. Request Query Park Area aggregate (sum) area per state performs an aggregation with grouping. The sum of the park area in square_km is calculated per state. The outer aggs component defines the buckets (unique values of states.title) and the sub-aggregation *state_park_area* describes the aggregation (sum of the field *area.square_km* value) within the bucket.

```

1 [
2   "size":0,
3   "aggs": {
4     "by_state": {
5       "terms": {
6         "field": "states.title"
7       },
8       "aggs": {
9         "state_park_area": {
10           "sum": {
11             "field": "area.square_km"
12           }
13         }
14       }
15     }
16   }
17 ]
18

```

The request labeled *Query States by Park Area aggregate (sum) area sorted descending* extends this query with the sort instruction to list the states by combined national park area in descending order.

```

1  {
2      "size": 0,
3      "aggs": {
4          "by_state": {
5              "terms": {
6                  "field": "states.title".
7                      "order": {
8                          "state_park_area": "desc"
9                      }
10                 },
11             "aggs": {
12                 "state_park_area": {
13                     "sum": {
14                         "field": "area.square_km"
15                     }
16                 }
17             }
18         }
19     }
20 }
```

A special type of bucket aggregation can be done for time intervals. We will now construct the query for finding the number of national parks established per decade – between 1860 and 2020. We make use of the `date_histogram` aggregation

(<https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html>).

The `date_histogram` aggregation is configured with the field that contains the date on which the documents are aggregated, the duration of the interval and possibly the limits of the search results. The interval can be expressed using time units

(<https://www.elastic.co/guide/en/elasticsearch/reference/current/common-options.html#time-units>) such as 2h, 9d and 30s and using the keywords year, quarter, month, week. Unfortunately, `10year` is not supported, even though `3652d` is.

▼ Query Parks - Established per decade

Add a description

POST {{ELASTIC_HOME}}:9200/parks/_search?explain=true

Authorization Headers (1) Body ● Pre-request Script Test:

form-data x-www-form-urlencoded raw binary JSON

```

1  {
2      "size": 0,
3      "aggs": {
4          "state_park_area": {
5              "date_histogram": {
6                  "field": "date_established_readable",
7                  "format": "yyyy",
8                  "interval": "3653d",
9                  "extended_bounds": {
10                      "min": "1860",
11                      "max": "2020"
12                  }
13              }
14          }
15      }
16  }

```

Queries through GET requests and URL parameters

Complex queries can be performed using GET requests by stuffing all search instructions in the URL. See the Postman Query *Parks - fuzzy, proximity - with query execution explanation*

`{{ELASTIC_HOME}}:9200/parks/_search?q=description%3Amountain~2+visitors%3A<1000000+!(states.title%3AAIa
ska)&default_operator=AND`

and execute it. It returns: parks that have the string "mountain" in their description - or something that is close to it (our fuzziness allows for up to two character edits distance); we also require (note how the default-operator is explicitly set to AND) the number of visitors to be under 1 Million and these parks are not located in Alaska.

This article describes in a lot of detail how the ES queries through the URL GET interface are constructed:
<https://www.compose.com/articles/using-query-string-queries-in-elasticsearch/> Tip: URL Encoding:
https://www.w3schools.com/tags/ref_urlencode.asp describes how to encode characters such as : (to %3A)

Use Kibana to Explore the Data

Open Kibana – probably at port 5601 at the same (Docker) host that is running Elastic Search. For me the endpoint is:

<http://192.168.99.100:5601>

Kibana has to be connected a to an Elastic Search Index, in order to explore data. When the Kibana container was started, it was already configured to connect to the Elastic Search Server. Now we need to provide the name – or pattern – of an Index on that server.

Type *parks* or *park** to focus Kibana on the National Parks index.

The screenshot shows the Kibana Management interface. On the left, there's a sidebar with icons for Discover, Visualize, Dashboard, Timelion, Dev Tools, and Management. The Management option is selected. The main area is titled 'Management / Kibana' and shows 'Index Patterns' (highlighted), 'Saved Objects', and 'Advanced Settings'. A warning message says: 'Warning: No default index pattern. You must select or create one to continue.' Below this, there's a section titled 'Configure an index pattern' with a sub-section 'Index pattern' (with a red box around it). The input field contains 'parks'. To the right of the input field is a placeholder 'Example: logstash-*'. Below the input field is a dropdown for 'Time Filter field name' set to 'date_established_readable'. There's also a checkbox for 'Use event times to create index names [DEPRECATED]'. At the bottom is a blue 'Create' button.

In the dropdown box for Time Filter field name, select the option *I don't want to use the Time Filter*. This option is primarily useful with data that continuously streams in, like log file entries, IoT events and web site click actions.

Configure an index pattern

In order to use Kibana you must configure at least one index pattern. Index patterns are used to configure fields.

Index pattern advanced options

parks

Patterns allow you to define dynamic index names using * as a wildcard. Example: logstash-*

Time Filter field name refresh fields

date_established_readable

I don't want to use the Time Filter

date_established_readable

Create

Next, click on the Create button:

Configure an index pattern

In order to use Kibana you must configure at least one index pattern. Index patterns are used to configure fields.

Index pattern advanced options

parks

Patterns allow you to define dynamic index names using * as a wildcard. Example: logstash-*

Time Filter field name refresh fields

I don't want to use the Time Filter

Create

Kibana reads the meta-data for the index and configures its own representation, that is shown next:

The screenshot shows the Kibana interface under 'Management / Kibana'. On the left sidebar, the 'Discover' option is highlighted. The main area displays the 'Index Patterns' section for the 'parks' index pattern. A red box highlights the title '★ parks'. Below it, a table lists all fields in the 'parks' index with their core types and various settings like searchability and aggregability.

| name | type | format | searchable | aggregatable | excluded | controls |
|---------------------------|-----------|--------|------------|--------------|----------|----------|
| _id | string | | ✓ | | | |
| _index | string | | ✓ | ✓ | | |
| _score | number | | | | | |
| _source | _source | | | | | |
| _type | string | | ✓ | ✓ | | |
| area.acres | number | | ✓ | ✓ | | |
| area.square_km | number | | ✓ | ✓ | | |
| coordinates | geo_point | | ✓ | ✓ | | |
| date_established_readable | date | | ✓ | ✓ | | |
| date_established_unix | number | | ✓ | ✓ | | |
| description | string | | ✓ | ✓ | | |
| id | string | | ✓ | | | |
| id.keyword | string | | ✓ | ✓ | | |
| image.attrition | string | | ✓ | | | |
| image.attrition_keyword | string | | ✓ | ✓ | | |

Click on the option *Discover* in the upper left hand corner.

This brings a page where you search, filter, explore the data in the index, using the same search facilities we have seen in the REST APIs.

The screenshot shows the Kibana 'Discover' page for the 'parks' index pattern. The search bar at the top contains the query 'Search... (e.g. status:200 AND extension:PHP)'. The results table shows 58 hits, with the first few entries expanded to show detailed document snippets. The left sidebar includes a 'Selected Fields' section with '_source' selected, and an 'Available Fields' section listing other fields like '_id', '_index', '_score', '_type', etc.

| id | source |
|----|---|
| 1 | area.acres: 1,201,647.03 area.square_km: 4,862.9 coordinates: { "lat": 36.06, "lon": -112.14 } date_established_readable: February 26th 1919, 01:00:00.000 date_established_unix: -1,604,599,200 description: The Grand Canyon, carved by the mighty Colorado River, is 277 miles (446 km) long, up to 1 mile (1.6 km) deep, an up to 15 miles (24 km) wide. Millions of years of erosion have exposed the multicolored layers of the Colorado Plateau in mesas and canyon walls, visible from both the north and south rims, or from a number of trails that descend into the canyon itself. image.url: grand-canyon.jpg image.attrition: PixelBay/88KD image.attrition_url: https://pixabay.com/en/users/BKD-1006949/ nps_link: https://www.nps.gov/grca/index.htm states: { "id": "state_arizona", "title": "Arizona" } |
| 2 | area.acres: 1,508,968.1 area.square_km: 6,106.6 coordinates: { "lat": 25.32, "lon": -80.93 } date_established_readable: May 30th 1934, 02:00:00.000 date_established_unix: -1,123,182,000 description: The Everglades are the largest tropical wilderness in the United States. This mangrove and tropical rainforest ecosystem and marine estuary is home to 36 protected species, including the Florida panther, American crocodile, and West Indian manatee. Some areas have been drained and developed; restoration projects aim to restore the ecology. image.url: everglades.jpg image.attrition: PixelBay/85Keeze image.attrition_url: https://pixabay.com/en/users/skeezee-272447/ nps_link: https://www.nps.gov/ever/index.htm states: { "id": "state_florida", "title": "Florida" } title: Everglade id: park_everglades.jpg image.attrition: PixelBay/8Broensis image.attrition_url: https://pixabay.com/en/users/Broensis-5213623/ nps_link: https://www.nps.gov/grsm/index.htm |
| 3 | area.acres: 522,426.88 area.square_km: 2,114.2 coordinates: { "lat": 35.68, "lon": -83.51 } date_established_readable: June 15th 1934, 02:00:00.000 date_established_unix: -1,121,799,600 description: The Great Smoky Mountains, part of the Appalachian Mountains, span a wide range of elevations, making them home to over 400 vertebrate species, 100 tree species, and 5000 plant species. Hiking is the park's main attraction, with over 800 miles (1,300 km) of trails, including 70 miles (113 km) of the Appalachian Trail. Other activities include fishing, horseback riding, and touring nearly 80 historic structures. image.url: great-smoky-mountains.jpg image.attrition: PixelBay/8Broensis image.attrition_url: https://pixabay.com/en/users/Broensis-5213623/ nps_link: https://www.nps.gov/grsm/index.htm |
| 4 | area.acres: 2,219,790.71 area.square_km: 8,983.2 coordinates: { "lat": 44.6, "lon": -110.5 } date_established_readable: March 1st 1872, 01:00:00.000 date_established_unix: -3,087,396,000 description: Situated on the Yellowstone Caldera, the park has an expansive network of geothermal areas including boiling mud pots, vividly colored hot springs such as Grand Prismatic Spring, and regularly erupting geysers, the best-known being Old Faithful. The yellow-hued Grand Canyon of the Yellowstone River contains several high waterfalls, while four mountain ranges traverse the park. More than 60 mammal species including gray wolves, grizzly bears, black bears, lynxes, bison, and elk, make this park one of the best wildlife viewing spots in the country. image.url: yellowstone.jpg image.attrition: PixelBay/85Keeze image.attrition_url: https://pixabay.com/en/users/skeezee-272447/ nps_link: https://www.nps.gov/yell/index.htm |
| 5 | area.acres: 3,223,383.43 area.square_km: 13,044.6 coordinates: { "lat": 58.0, "lon": -137 } date_established_readable: December 2nd 1980, 01:00:00.000 date_established_unix: 344,584,800 description: Glacier Bay contains tidewater glaciers, mountains, fjords, and a temperate rainforest, and is home to large populations of grizzly bears, mountain goats, whales, seals, and eagles. When discovered in 1794 by George Vancouver, the entire bay was covered by ice, but the glaciers have since receded more than 65 miles (105 km). image.url: glacier-bay.jpg image.attrition: PixelBay/85Keeze image.attrition_url: https://pixabay.com/en/users/skeezee-272447/ nps_link: https://www.nps.gov/glba/index.htm |

For example: type the following search string in the search bar and press enter or click on the magnifying class:

states.title:"Alaska" AND visitors:[500000 TO 600000] AND description:"mountain"

The screenshot shows the Kibana interface with a search bar at the top containing the query: "states.title:'Alaska' AND visitors:[500000 TO 600000] AND description:'mountain'". The results table has columns: title, states, visitors, and description. Two entries are shown:

| | title | states | visitors | description |
|-------------|---|---------|--|-------------|
| Glacier Bay | { "id": "state_ak", "title": "Alaska" } | 520,171 | Glacier Bay contains tidewater glaciers, mountains, fjords, and a temperate rainforest, and is home to large populations of grizzly bears, mountain goats, whales, seals, and eagles. When discovered in 1794 by George Vancouver, the entire bay was covered by ice, but the glaciers have since receded more than 65 miles (105 km). | |
| Denali | { "id": "state_ak", "title": "Alaska" } | 587,412 | Centered on Denali, the tallest mountain in North America, Denali is serviced by a single road leading to Wonder Lake. Denali and other peaks of the Alaska Range are covered with long glaciers and boreal forest. Wildlife includes grizzly bears, Dall sheep, caribou, and gray wolves. | |

The results of executing this query are shown.

Note: feel free to experiment with the query – by changing the search conditions and the properties that are searched on.

For example: find all parks that are within 1000 km from San Francisco and that have forest or trees mentioned in their description:

The screenshot shows the Kibana interface with a search bar at the top containing the query: "forest trees". The results table has columns: title, states, visitors, description, and coordinates. Three entries are shown:

| | title | states | visitors | description | coordinates |
|----------------------------------|---|-----------|--|----------------------------------|-------------|
| Redwood National and State Parks | { "id": "state_ca", "title": "California" } | 536,297 | This park and the co-managed state parks protect almost half of all remaining coastal redwoods, the tallest trees on earth. There are three large river systems in this very seismically active area, and 37 miles (60 km) of protected coastline reveal tide pools and seastacks. The prairie, estuary, coast, river, and forest ecosystems contain a wide variety of animal and plant species. | { "lat": 41.3, "lon": -124 } | |
| Joshua Tree National Park | { "id": "state_ca", "title": "California" } | 2,505,286 | Covering large areas of the Colorado and Mojave Deserts and the Little San Bernardino Mountains, this desert landscape is populated by vast stands of Joshua trees. Large changes in elevation reveal various contrasting environments including bleached sand dunes, dry lakes, rugged mountains, and maze-like clusters of monzogranite monoliths. | { "lat": 33.79, "lon": -115.9 } | |
| Sequoia National Park | { "id": "state_ca", "title": "California" } | 1,254,688 | This park protects the Giant Forest, which boasts some of the world's largest trees, the General Sherman being the largest measured tree in the park. Other features include over 240 caves, a long segment of the Sierra Nevada including the tallest mountain in the contiguous United States, and Moro Rock, a large granite dome. | { "lat": 36.43, "lon": -118.66 } | |

Note: the geospatial query used here:

```
{
  "query": {
    "bool": {
      "must": {
        "match_all": {}
      },
      "filter": {
        "geo_distance": {
          "distance": "1000km",
          "coordinates": {
            "lat": 37.7,
            "lon": -122.2
          }
        }
      }
    }
  }
}
```

```

    "lon": -122.4
  }
}
}
}
}

```

3 hits

forest trees

query: {"bool":{"must":{"match_all":{}}, "filter":{"geo_distance":{"distance":"1000km", "coordinates":{"lat":37.7, "lon":-122.4}}}}}}

Edit filter

Filter

```

1 {
2   "query": {
3     "bool": {
4       "must": {
5         "match_all": {}
6       },
7       "filter": {
8         "geo_distance": {
9           "distance": "1000km",
10          "coordinates": {
11            "lat": 37.7,
12            "lon": -122.4
13          }
14        }
15      }
16    }
17  }
18}

```

Filters are built using the [Elasticsearch Query DSL](#).

Label

Within 1000 km from Sa

Description

This park and the three large river sea stacks. The pr...

Covering large areas vast stands of Jos rugged mountains, ...

This park protects n the park. Other United States, and

Edit Query DSL

Cancel **Save**

"title":

Note: you can use the option *Share* in the upper right hand corner to get a URL that when clicked will navigate the browser immediately to the Kibana data discovery page with this same query executed (and possibly different results depending on what data was added, changed and deleted from the index)

The screenshot shows the Kibana interface with the 'Discover' tab selected. In the top right, the 'Share' button is highlighted. A tooltip for 'Share Snapshot' explains that snapshot URLs encode the current state of the search in the URL itself. Below it, a 'Link' section shows the URL [{http://192.168.99.100:5601/app/kibana#/discover?_g=\(q=states.title:Alaska AND visitors:\[500000 TO 600000\] AND description:mountain\)}](http://192.168.99.100:5601/app/kibana#/discover?_g=(q=states.title:Alaska AND visitors:[500000 TO 600000] AND description:mountain)). A note says we recommend sharing shortened snapshot URLs for maximum compatibility. A 'Uses lucene query syntax' link is also present.

Visualizations

One of the very powerful capabilities of Kibana is the creation of data visualizations. Kibana understands Elastic Search and the meta data for the Elastic Search Indexes. It can use this understanding to help very quickly create relevant visualizations, such as a map for geospatial data, a timeline for date and time related data and various type of charts for numerical, clustered data.

Click on Visualize in the left column. Then click on the Plus icon to create a new visualization.

The screenshot shows the Kibana interface with the 'Visualize' tab selected. On the right, there is a list of existing visualizations: 'Name' (New parks per lustrum) and 'Type' (Vertical Bar). A blue plus icon is located at the top right of this list, indicating where to click to add a new visualization.

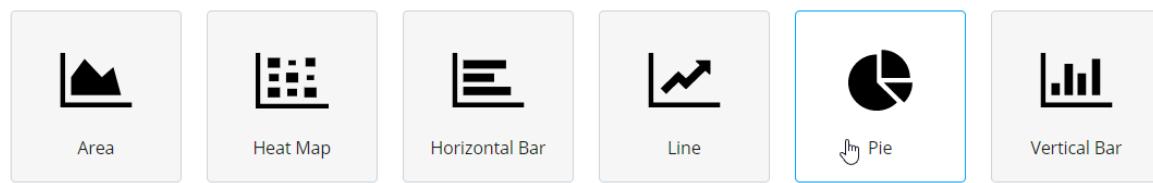
Nested Pie chart (aka Sunburst) – Comparing size of Parks by and within States

Select the Pie Chart visualization type:

Select visualization type

Search visualization types...

Basic Charts



Data



Maps

Select the index *parts* as the data source for the visualization:

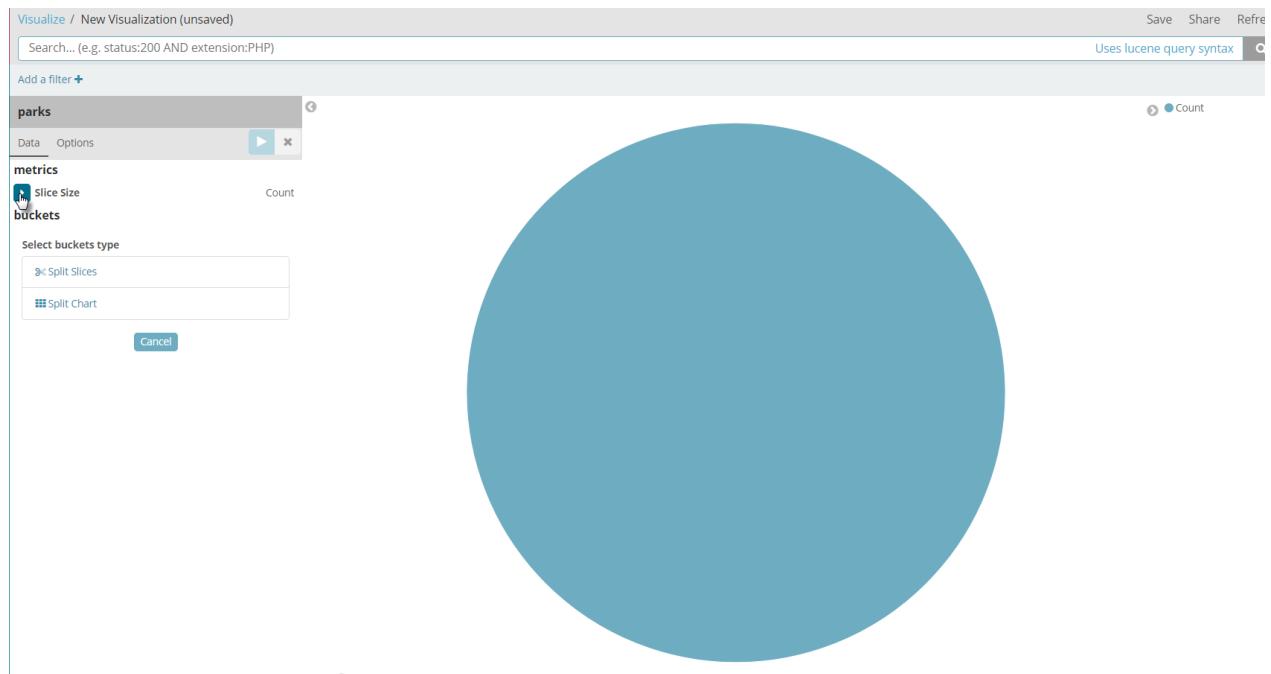
From a New Search, Select Index

A screenshot of a search interface. At the top left is a search bar with the placeholder "Filter...". To its right is the text "1 of 1". Below the search bar is a table with one row. The first column contains the word "Name" with an upward arrow indicating it's sorted. The second column contains the word "parks" with a small hand cursor icon pointing at it.

Or, From a Saved Search

A screenshot of a saved search interface. At the top left is a search bar with the placeholder "Saved Searches Filter...". To its right is the text "0-0 of 0" and a link "Manage saved searches". Below the search bar is a table with one column labeled "Name" with an upward arrow. The table is empty, displaying the message "No matching saved searches found."

Click on the expand icon for Slice Size – to define the pie segments:



Specify Sum as the type of aggregation, select field area.acres and set Area as the custom label.

Add a filter +

parks

Data Options ▶ ×

metrics

Slice Size

Aggregation

Sum ▾

Field

area.acres ▾

Custom Label

Area

◀ Advanced

Next, click on Split Slices

◀ Advanced

buckets

Select buckets type

Split Slices

Split Chart

Cancel

Time to define the clustering – what are the slices aggregated on.

We will have to levels of slices. The first is parks grouped by state. Select *Terms* as the type of aggregation. Select *states.title.keyword* as the field and metric: *Area* for order by. Set the size to 10 (meaning: include the 10 largest slices) and type *State* as the custom label:

parks

Data Options Run X

Sum
Field: area.acres
Custom Label: Area

Advanced

buckets

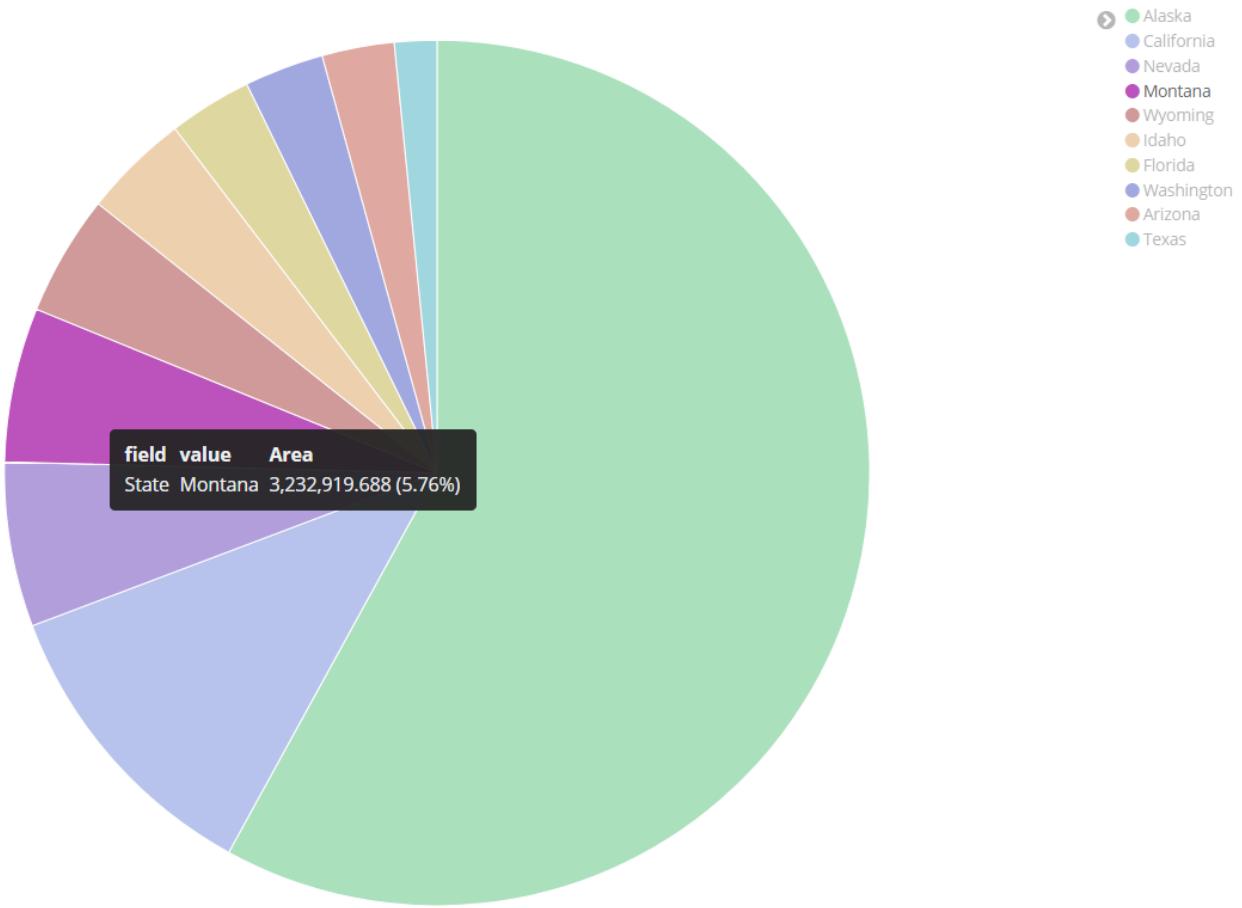
Split Slices X

Aggregation: Terms
Field: states.title.keyword
Order By: metric: Area
Order: Descending Size: 10
Custom Label: State

Advanced Add sub-buckets

Click on the run icon in the top right hand corner of this panel. This will render the Pie Chart based on these definitions for all records in the *parks* index – since we did not define a search condition.

This is what the chart looks like:



Now we will add a second level.

Click on the button *Add sub-buckets*:

[Advanced](#)
Add sub-buckets

Now define the slices for this second level – the level of the park [within the state].

Click on Split Slices:

Order Size

Descending ▼ 10

Custom Label

State

◀ Advanced

Select buckets type

✖ Split Slices

Split Chart

Cancel

The screenshot shows a search interface with various configuration options. At the top, there are fields for 'Order' set to 'Descending' and 'Size' set to '10'. Below this is a 'Custom Label' field containing 'State'. An 'Advanced' button is available. A modal window titled 'Select buckets type' is open, displaying two options: '✖ Split Slices' (which has a hand cursor icon over it) and 'Split Chart'. A 'Cancel' button is located at the bottom of the modal.

and define these settings:

- Aggregation [Type]: terms
- Field: title.keyword
- Order by: metric: Area
- Size: 7
- Custom Label: Park

parks

Data Options  

Order by

metric: Area

Order **Size**

Descending ▾ 10

Custom Label

State

 Advanced

Split Slices

Sub Aggregation

Terms

Field

title.keyword

Order By

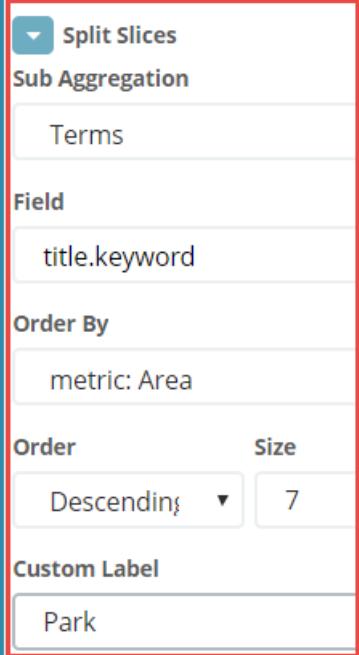
metric: Area

Order **Size**

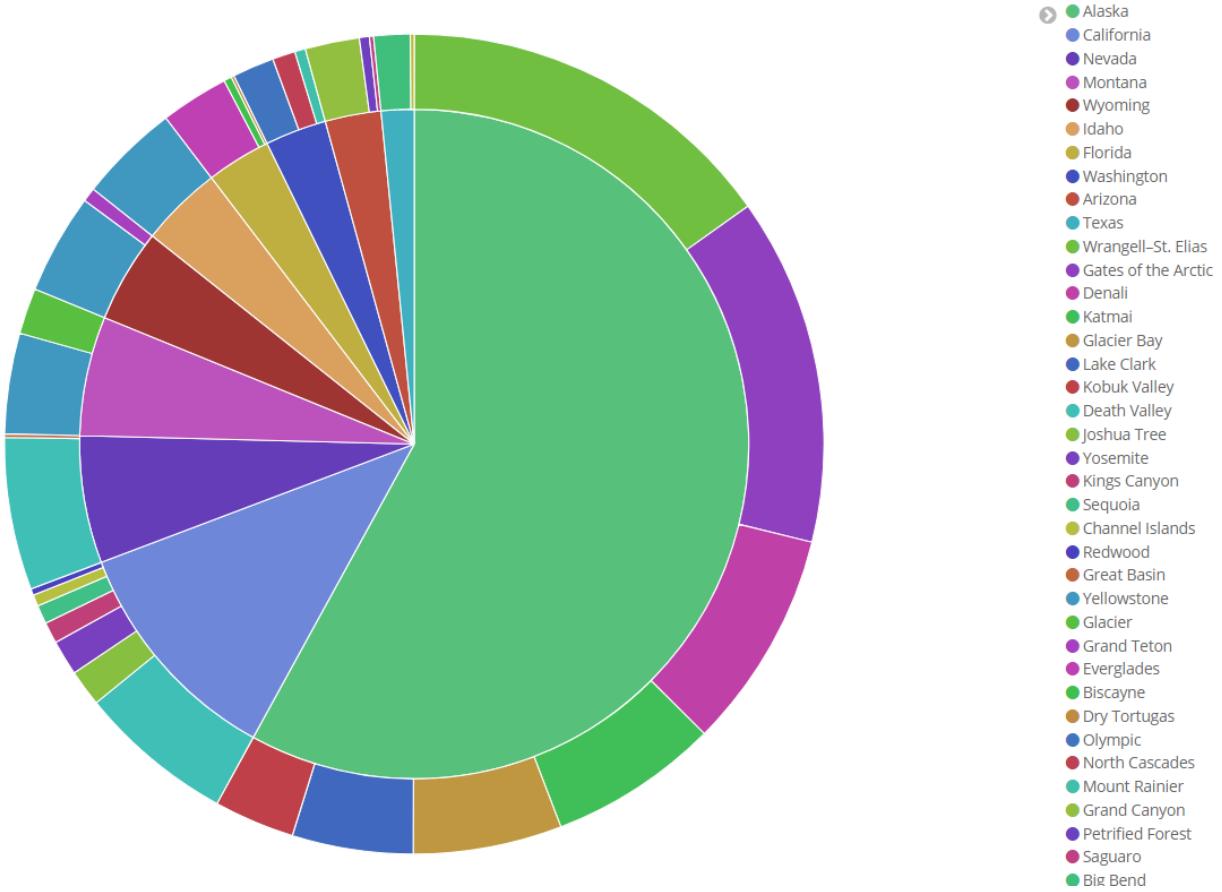
Descending ▾ 7

Custom Label

Park

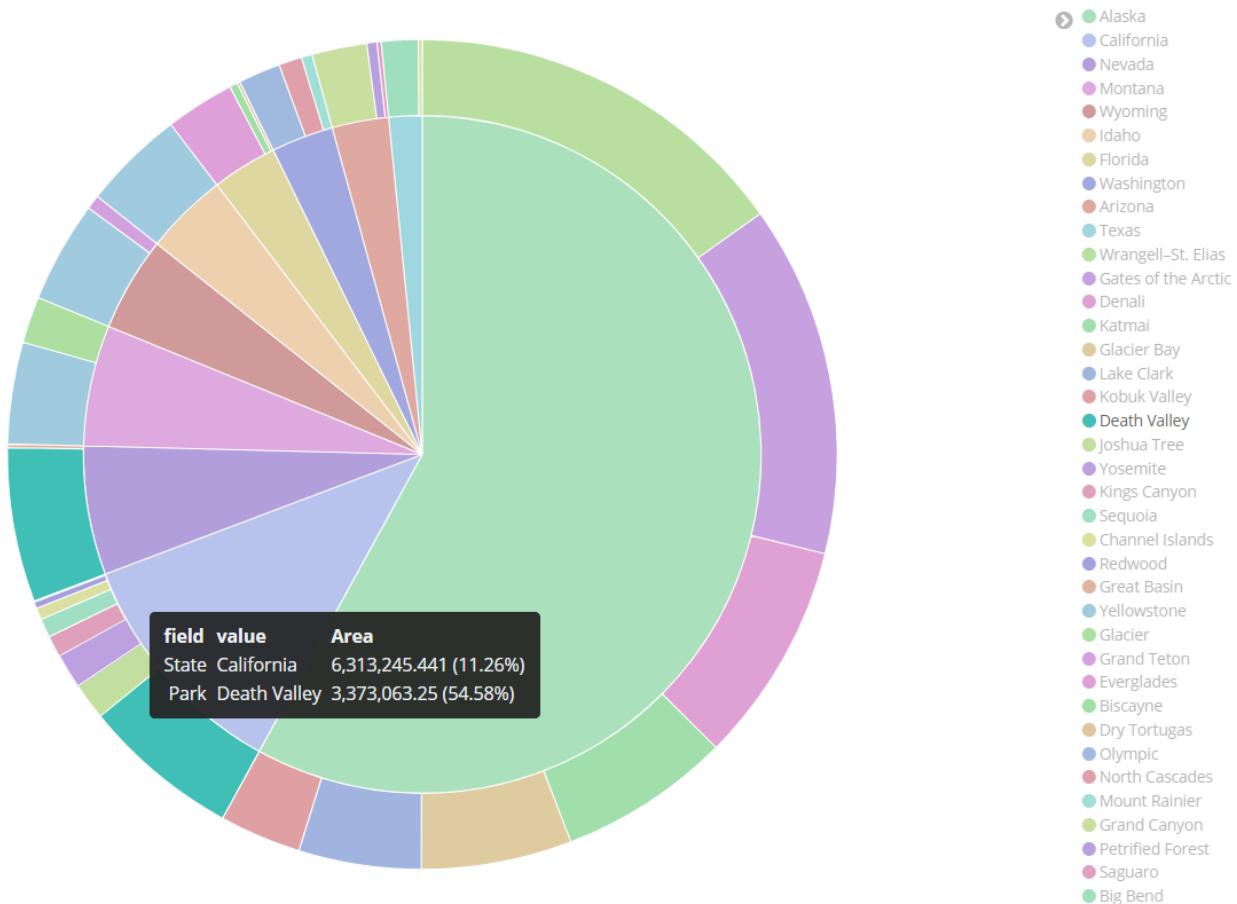


Then click on the run button to render the chart based on these extended definitions:

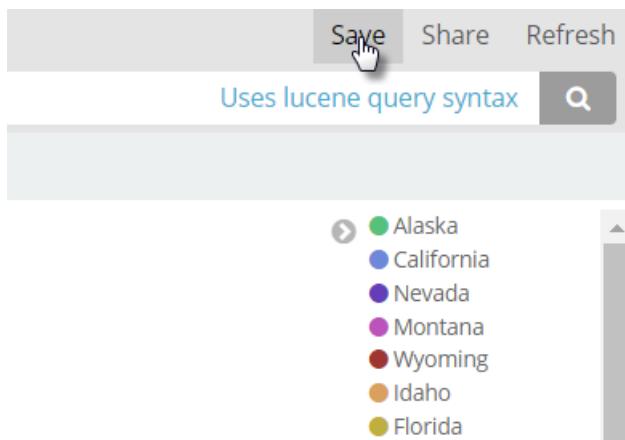


The second ring shows the sizes of the individual parks and tells us how the total area of a state is distributed over its parks.

In case you are wondering what the biggest park in California is – the next figure will tell you.



Click on the Save option in the upper right hand corner. Save the Visualization as *Park Area Pie*.



Tagcloud with States

Our objective: show in a tag cloud the names of the states with the biggest national parks areas – using the tag sizes to indicate the relative area sizes.

Create a new visualization, of type Tag Cloud.

Select visualization type

Search visualization types...

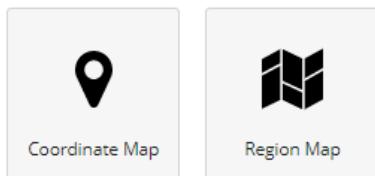
Basic Charts



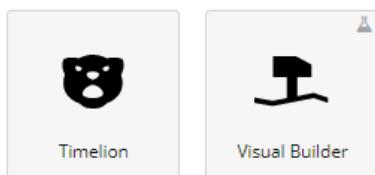
Data



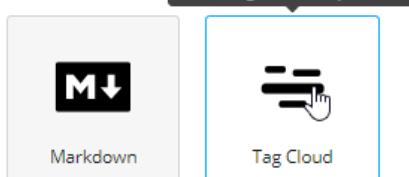
Maps



Time Series



Other



Select the *parks* index as the data source.

Define the tag size:

parks

Data Options  

metrics

 Tag Size

Aggregation

Sum 

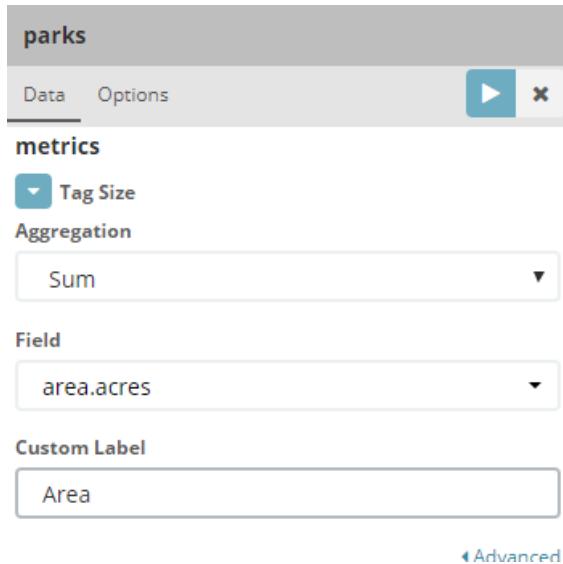
Field

area.acres 

Custom Label

Area

[Advanced](#)



Next define the *buckets* (the aggregation grouping).

buckets

 Tags

Aggregation

Terms 

Field

states.title.keyword 

Order By

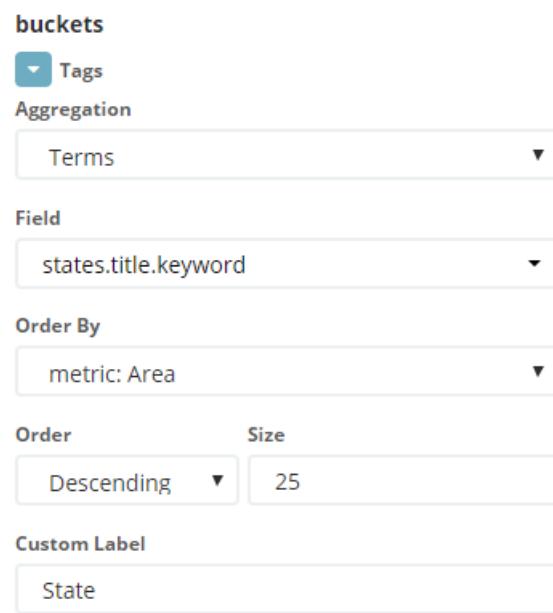
metric: Area 

Order **Size**

Descending  25

Custom Label

State



If you open the *Advanced* section under *buckets*, you can specify an include (whitelist) or exclude (blacklist) set of tags for the cloud.

Click on the run (or render) icon to display the tag cloud:



The options panel allows you to fiddle with the font size, the text orientation and scaling algorithm.

Visualize / New Visualization (unsaved)

Search... (e.g. status:200 AND extension:PHP)

Add a filter +

parks

Data Options

Text Scale

linear

Orientations

multiple

Font Size

19px 92px

Show Label

Area - State

Save this visualization as State Tag Cloud.

Tagcloud with terms from description

Create a new visualization. Choose Tag Cloud as the type. And *parks* as the index to fetch the data from. Define the metrics – Count as Aggregation type – and buckets – Terms for Aggregation, description for field, Order By: the metric [Number of Occurrences] and Descending. Important: use the Exclude property to filter out keywords that are not in the Stoplist for standard English, but that are meaningless for our purposes. For example:

also|than|km|over|has|its|more|from|which|including|well|other|covered|area

The screenshot shows the configuration interface for a Tag Cloud visualization named "parks".

Metrics: Tag Size, Count (Aggregation).

Buckets: Tags, Terms (Aggregation), description (Field).

Order By: metric: Number of occurrences (Descending, Size 18).

Custom Label: Keywords.

Exclude: A red box highlights the "Exclude" section containing the following terms: also|than|km|over|has|its|more|from|w

Include: This section is currently empty.

Press the Run icon to render the tag cloud:



Number of occurrences - Keywords

Note: by clicking on tags, you add them as filter. This means that the tag cloud is recreated - based only on those national parks that have the selected terms in their description. Here I have clicked on *river* and subsequently on *bison*.

Save the visualization as *Descriptive Tag Cloud*.

Map with all national parks

The park records have geopoints associated with them that describe their geographical location. Let's visualize the parks on a real geographical map.

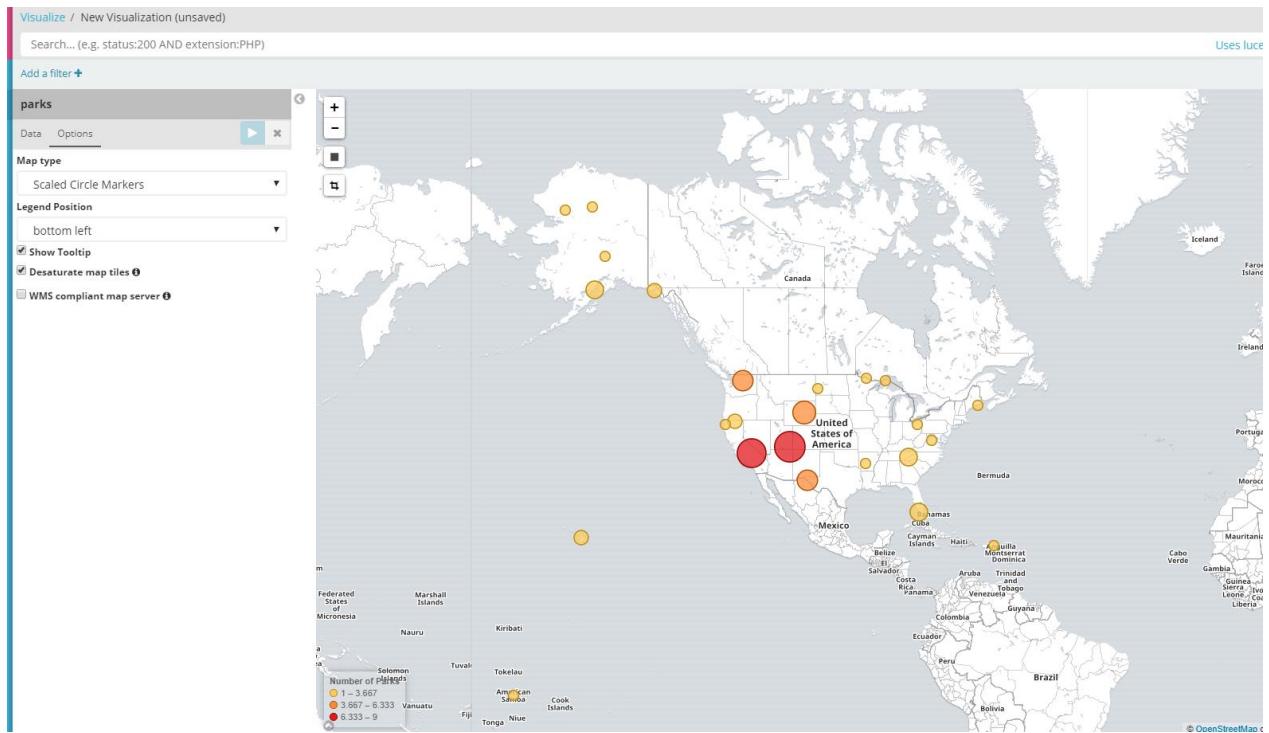
Create a new visualization. Choose Coordinate Map as the type. And *parks* as the index to fetch the data from.

Configure the visualization like this:

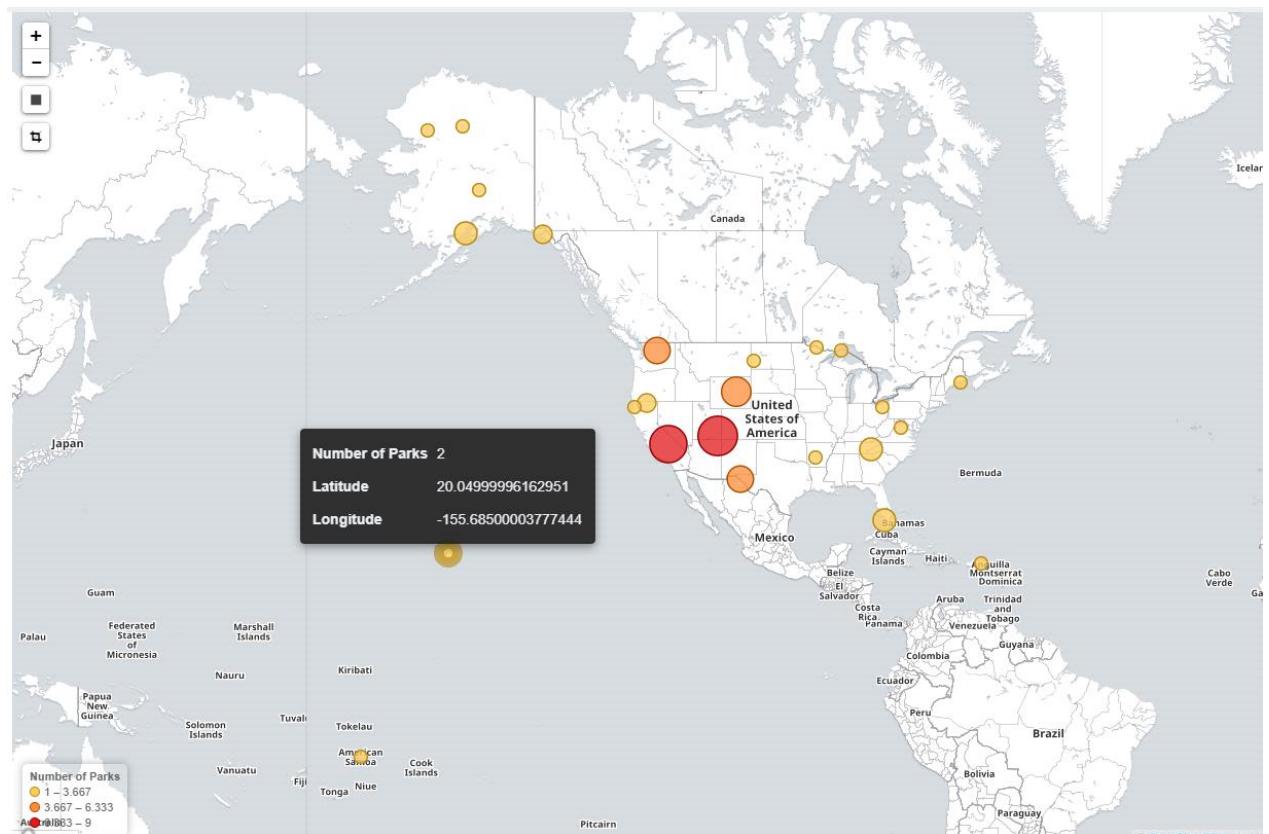
- Metrics
 - Aggregation: Count
 - Custom Label: Number of Parks
- Buckets
 - Aggregation: Geohash (preselected)
 - Field: Coordinates (the only one in the index of type Geopoint)
 - Custom Label: National Park

The screenshot shows the configuration interface for a 'Coordinate Map' visualization. At the top, there's a 'Add a filter +' button. Below it, the 'parks' index is selected. The interface is divided into sections: 'metrics' and 'buckets'. In the 'metrics' section, 'Value' is selected under 'Aggregation' (with 'Count' chosen). In the 'buckets' section, 'Geo Coordinates' is selected under 'Field' (with 'coordinates' chosen). There are also advanced settings like 'Change precision on map zoom' and 'Place markers off grid (use geocentroid)' which are checked. A 'Custom Label' field contains 'Number of Parks'. At the bottom, there's an 'Advanced' link.

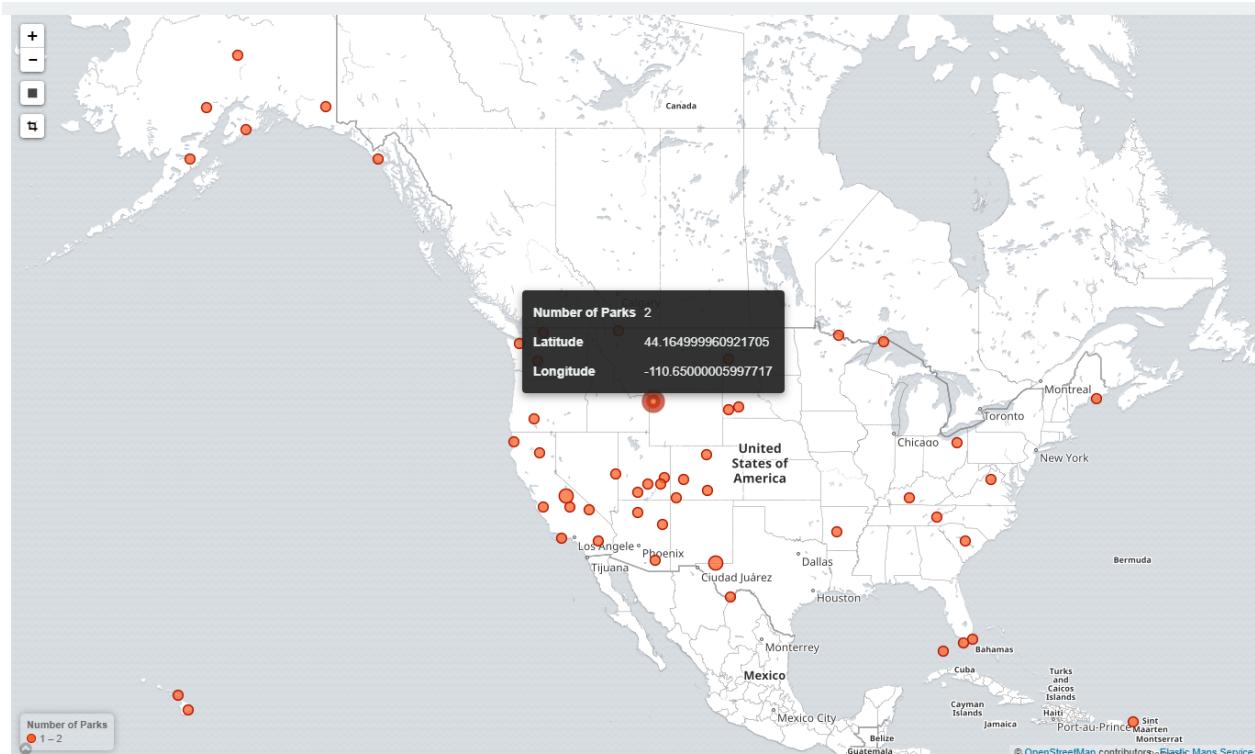
Click on the run icon to render the map.



The size and colors of the markers indicate the number of parks in the vicinity. When the mouse hovers over the markers, details are shown:



When you zoom in, the area over which aggregation of the park count takes place is reduced in size. Only two clusters of two “twin-parks” remain:



Save this visualization as *Map of Parks*.

Date Histogram with History of Park Establishment

Some of the national parks are quite old while others have been established fairly recently. We will now take a look at when the parks were formally opened – to see in which time frame in last 150 years there were ups and downs in this process.

Create a new visualization. Select Vertical Bars as the visualization type. Select *parks* as the data source.

Select visualization type

Search visualization types...

Basic Charts

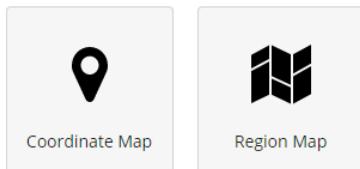


Assign a continuous variable
to each axis

Data



Maps



Click on X-axis as bucket type:

parks

Data Metrics & Axes Panel Settings

metrics

Y-Axis Count Add metrics

buckets

Select buckets type

X-axis

Split Series

Split Chart

Cancel

Select Data Histogram for Aggregation:

metrics

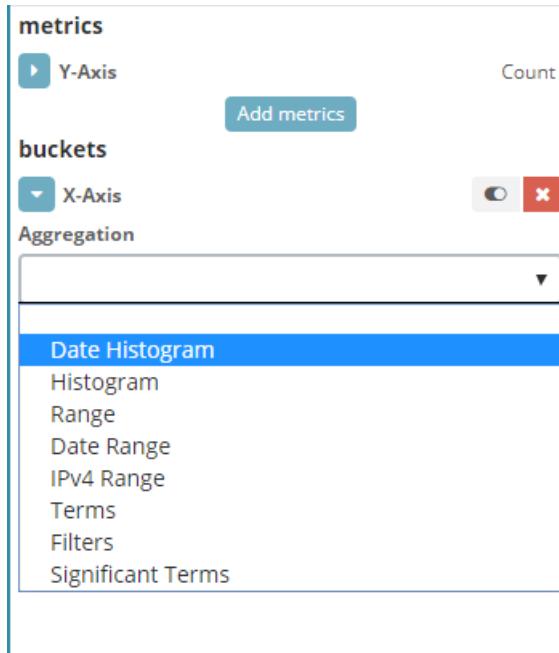
Y-Axis Count Add metrics

buckets

X-Axis

Aggregation

Date Histogram
Histogram
Range
Date Range
IPv4 Range
Terms
Filters
Significant Terms



Specify these settings for Field, Interval and Custom Label:

parks

Data Metrics & Axes Panel Settings

metrics

Y-Axis Count Add metrics

buckets

X-Axis

Aggregation

Date Histogram

Field

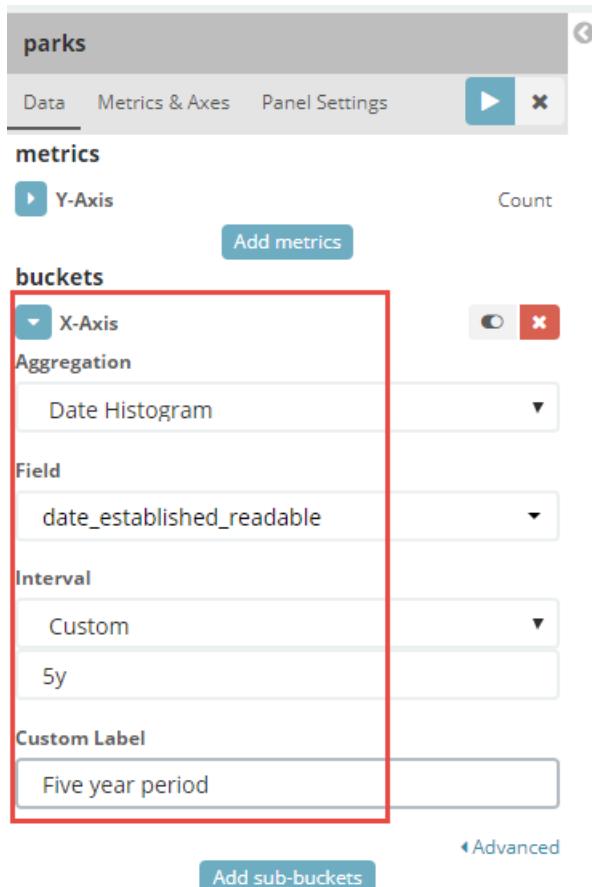
date_established_readable

Interval

Custom
5y

Custom Label

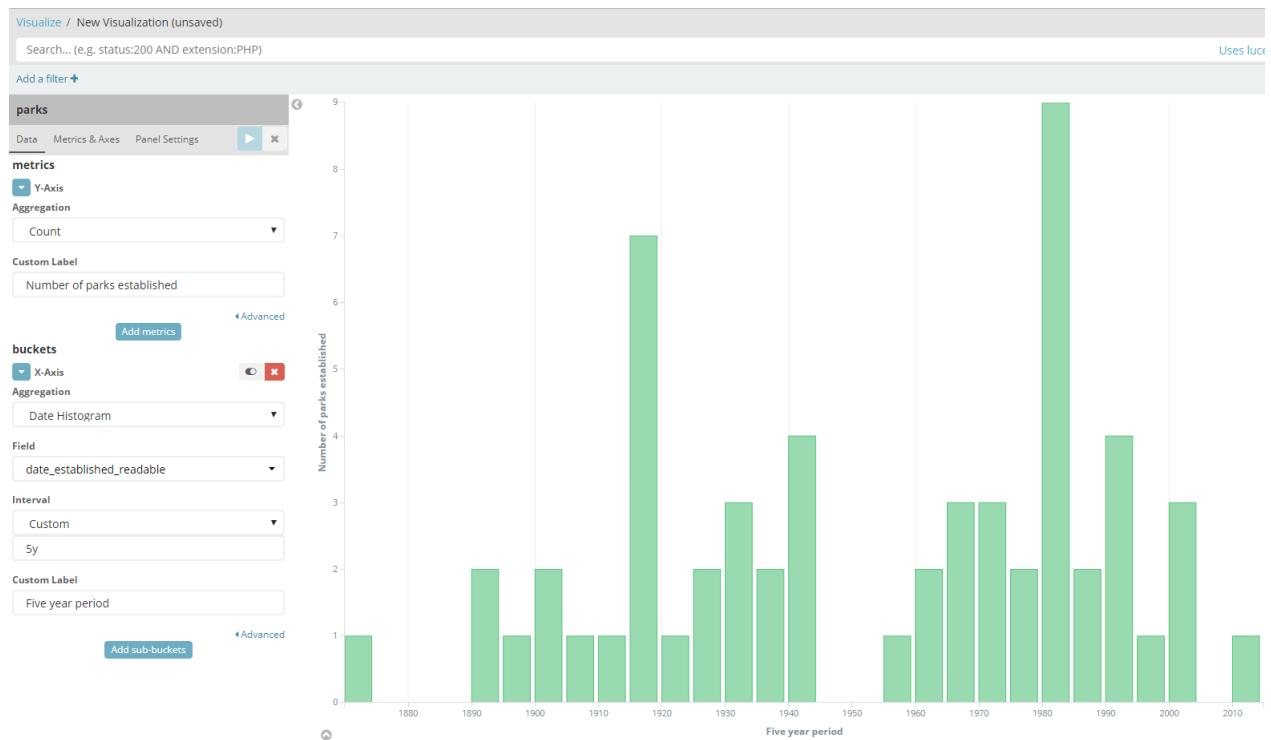
Five year period



Define the Y-axis metric:

The screenshot shows the Kibana interface with the 'metrics' panel open. The 'Y-Axis' section is selected, showing 'Count' as the aggregation type. A 'Custom Label' field contains 'Number of parks established'. Below the panel are 'Advanced' and 'Add metrics' buttons.

And press the run icon:



The gaps are clear – especially the one between 1945-1955. The peaks are somewhat remarkable, especially the highest one.

Let's visualize in which states these parks were established. Click on Add sub-buckets and specify the Split-Series properties as shown here.

◀ Advanced

Split Series

Sub Aggregation

Terms

Field

states.title.keyword

Order By

metric: Number of parks established

Order **Size**

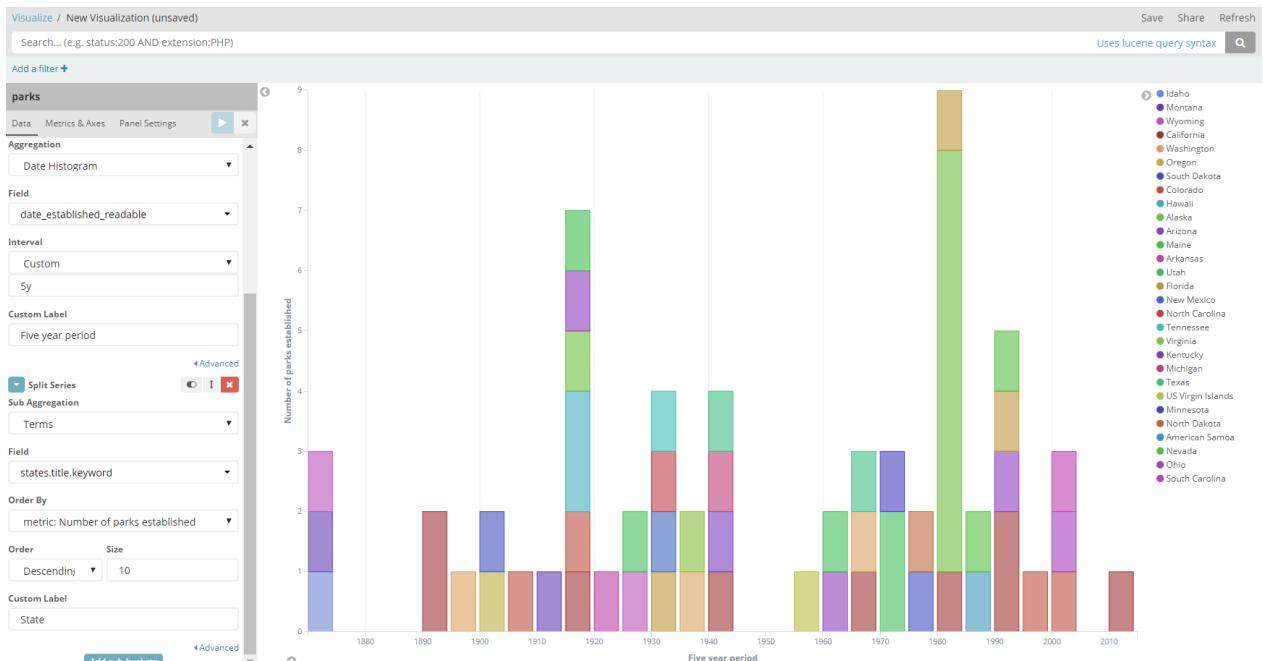
Descending ▾ 10

Custom Label

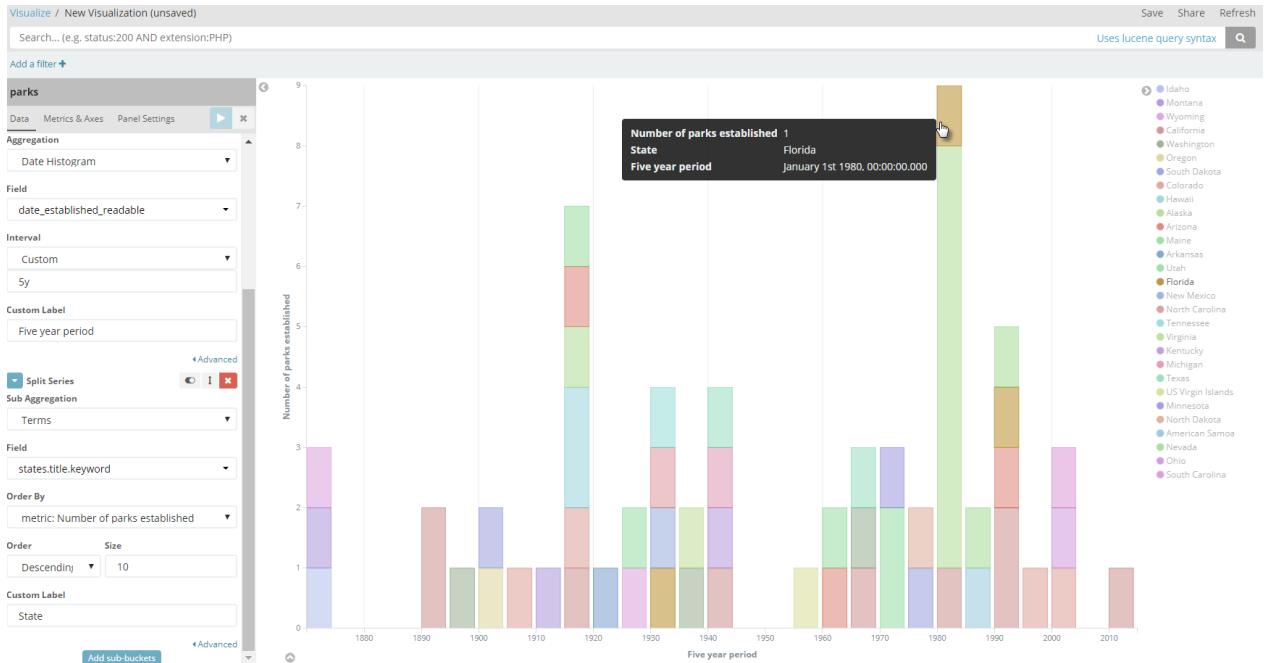
State

Add sub-buckets ▶ Advanced

Run the visualization again:



Hover over a bar segment to learn more about it:



Click on it to turn it into a filter. Here, we click on a segment for Florida.

Visualize / New Visualization (unsaved)

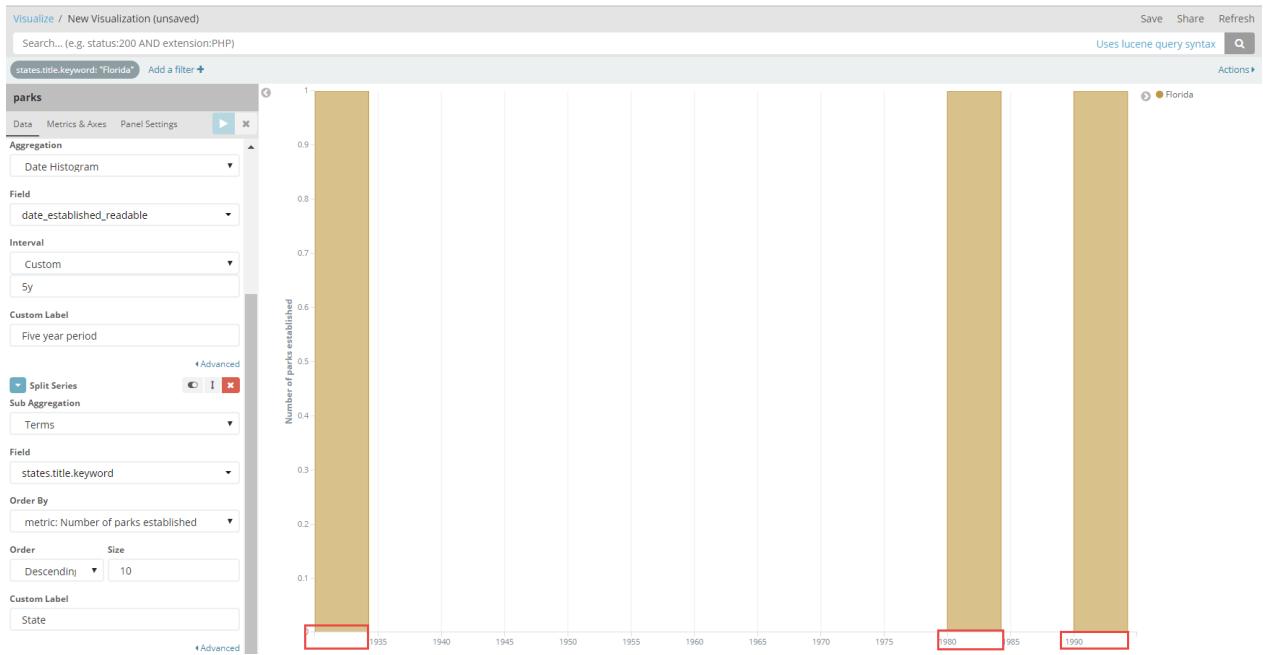
Search... (e.g. status:200 AND extension:PHP)

Apply these filters? date_established_readable: January 1st 1980, 00:00:00.000 to January 1st 1985, 00:00:00.000 states.title.keyword: Florida

Add a filter +

parks

A filter is created for this time period and this state. Uncheck the time related filter condition. Then click on Apply Now to filter the source data on the state of Florida. This allows us to quickly zoom in on the creation of the national parks in this particular state:



Save the visualization as *Parks Creation History*.

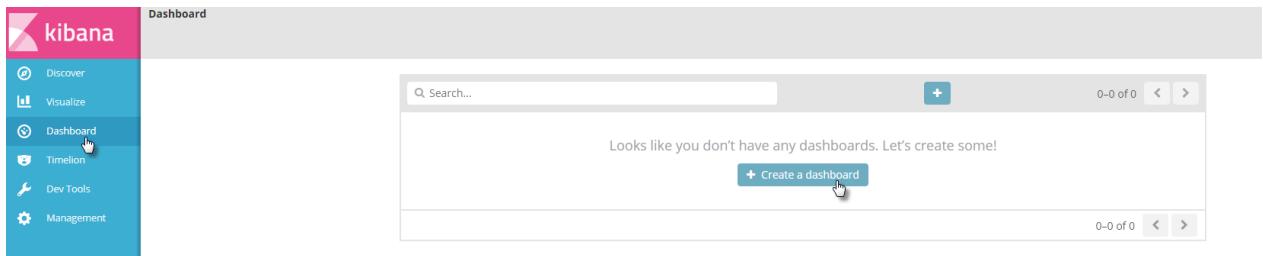
Dashboard

Click on Dashboard.

Click on Create a dashboard.



Click on



A list of Visualizations is presented. Click on *Map of Parks* to add the map as first visualization to the Dashboard:

Dashboard / Editing New Dashboard

Add Panels

Visualization Saved Search

Q visualizations Filter...

Name ▲

Descriptive Tag Cloud

Map of Parks

Park Area Pie

Parks Creation History

State Tag Cloud

Search... (e.g. status:200 AND extension:PHP)

Uses lucene query syntax Q

Save Cancel Add Options Share < ⌂ Last 15 minutes >

This dashboard is empty. Let's fill it up!

Click the **Add** button in the menu bar above to add a visualization to the dashboard.
If you haven't set up any visualizations yet, visit the [Visualize app](#) to create your first visualization.

The result:

Dashboard / Editing New Dashboard (unsaved)

Add Panels

Visualization Saved Search

Q Visualizations Filter...

Name ▲

Descriptive Tag Cloud

Map of Parks

Park Area Pie

Parks Creation History

State Tag Cloud

Search... (e.g. status:200 AND extension:PHP)

Uses lucene query syntax Q

Save Cancel Add Options Share < ⌂ Last 15 minutes >

Add a filter +

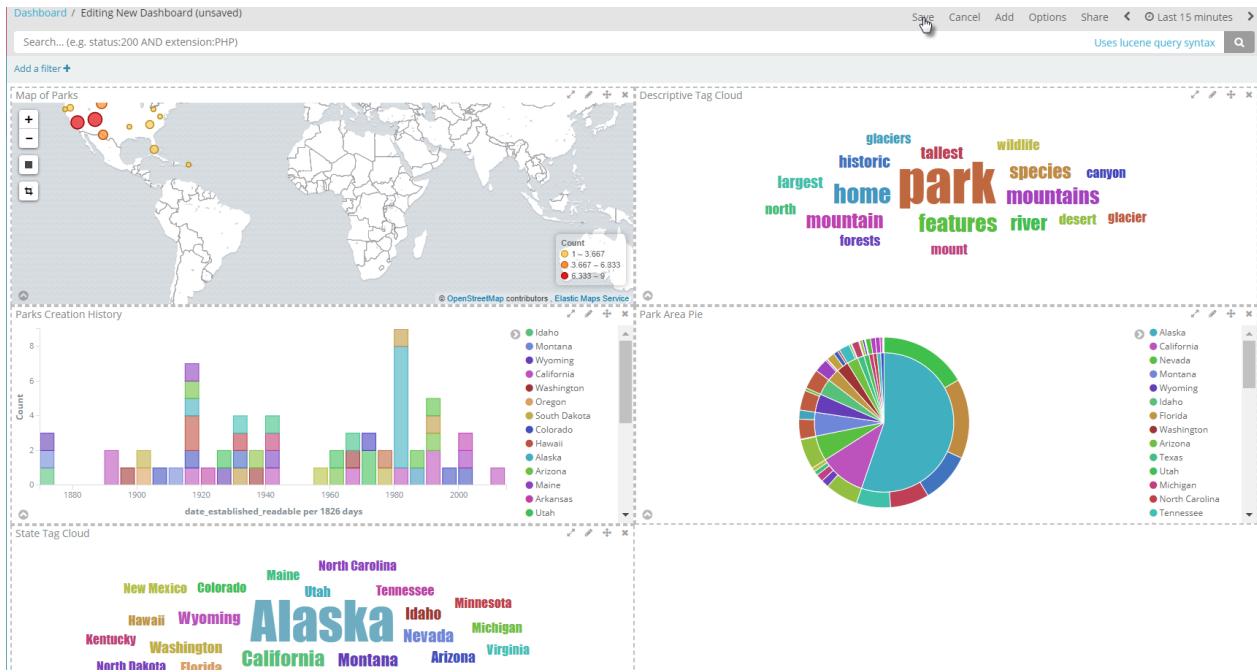
Map of Parks

Count

- 1 - 3.667
- 3.667 - 6.333
- 6.333 - 9

© OpenStreetMap contributors · Elastic Maps Service

Click on the button **Add new Visualization** and add all visualizations to the dashboard.

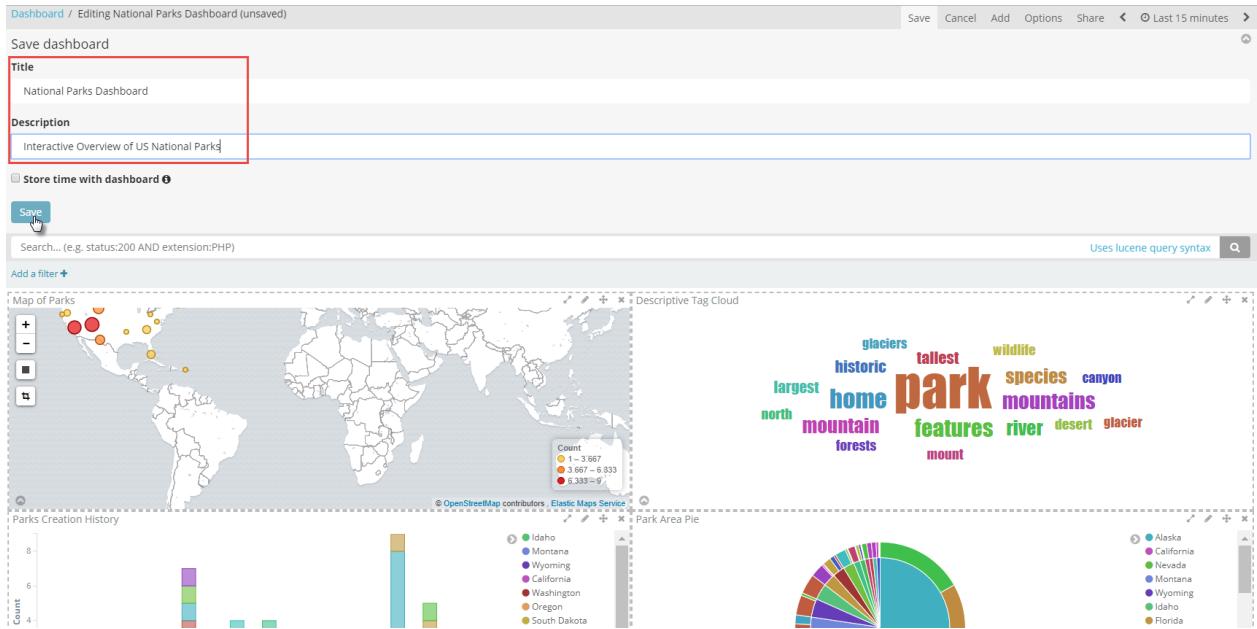


Then click on Save.

Provide name: *National Parks Dashboard*

And description: *Interactive Overview of US National Parks*

Then click on Save



The dashboard is saved:

The screenshot shows the Kibana interface with a sidebar on the left containing links for Discover, Visualize, Dashboard, Timelion, Dev Tools, and Management. The main area features a search bar and a map titled "Map of Parks" showing the locations of national parks in North America. A red box highlights the top navigation bar which displays the message "Dashboard: Saved Dashboard as 'National Parks Dashboard'".

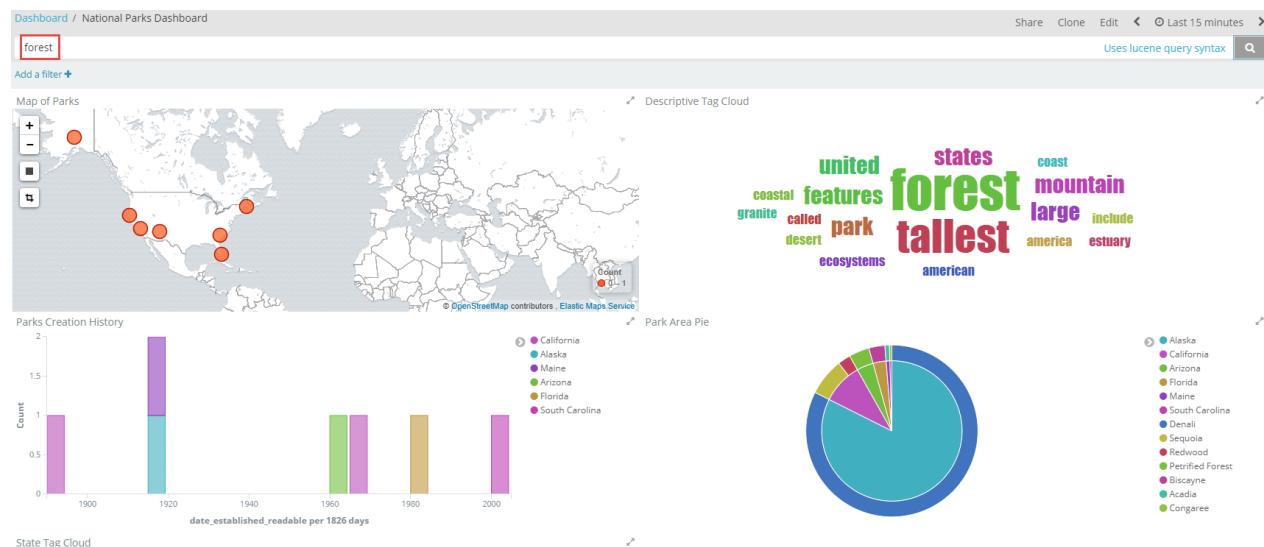
This dashboard can now be shared – both as snapshot in time and as a live dashboard (dynamically updated with fresh data). A shared dashboard can be embedded in an IFRAME in web applications.

A powerful feature in the Dashboard is the search filter.

The screenshot shows the Kibana dashboard with a search bar containing the term "forest". A red box highlights the search bar. To the right, there is a "Uses lucene query syntax" link and a magnifying glass icon. The rest of the interface is similar to the first screenshot, showing the map and search bar.

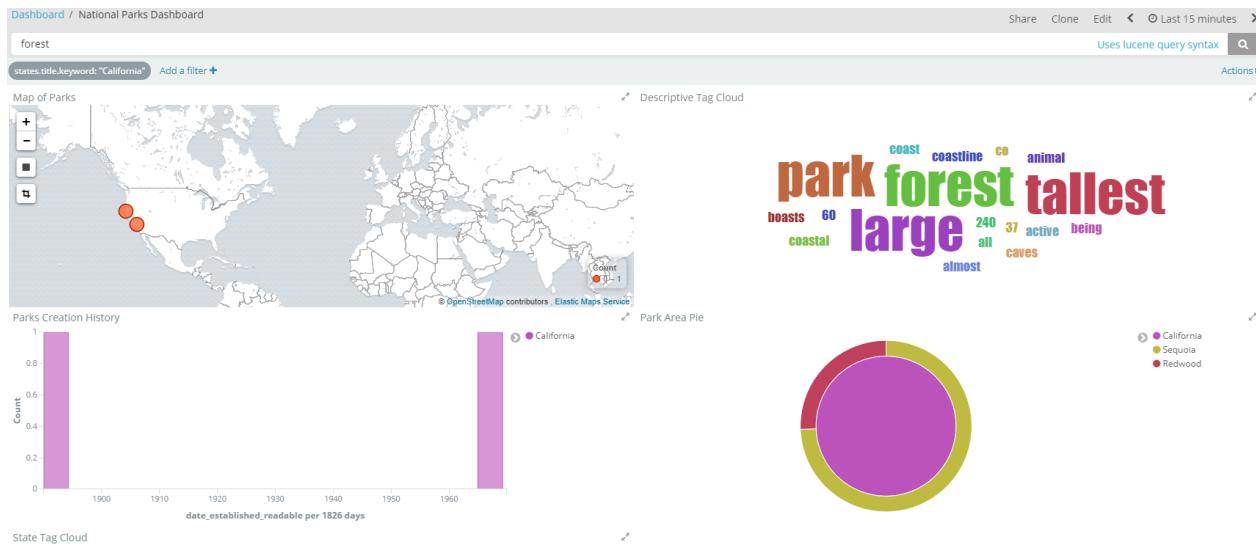
Users can use the search filter to slice and dice the data from the *parks* index – and in doing so filter the data for all visualizations.

For example, type *forest* in the search bar. This results in all visualizations only showing data for national parks with forest in their definition:



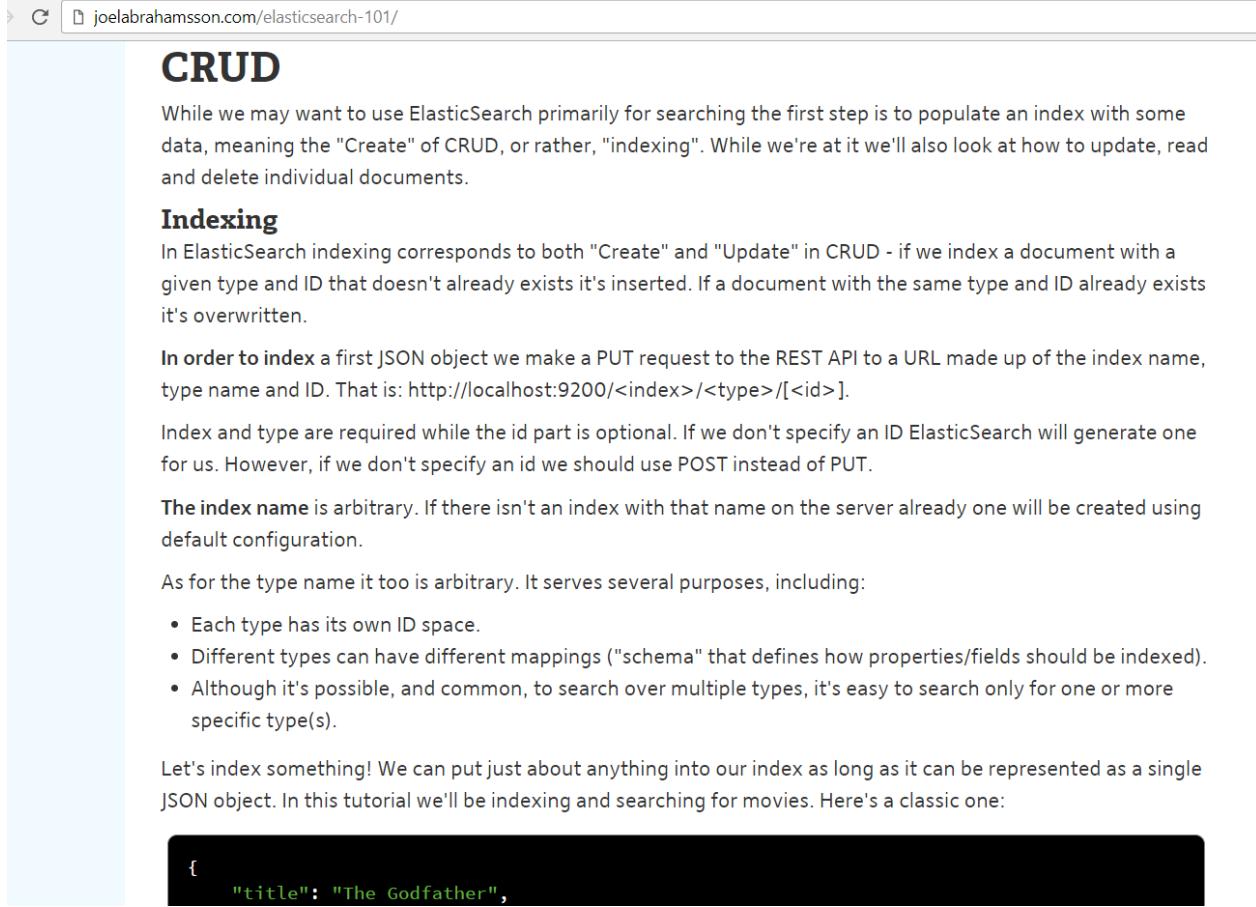
Maine **Alaska**
California Arizona Florida
South Carolina

Additionally, clicking on pie chart slices, tag cloud tags, stack segments in the bar chart will result in new filters in the query. For example clicking on the tag California will result in only the parks in California being available to all visualizations.



3. Getting Started with Elastic Search

A(nother) good way to get started with Elastic Search Index – creating an index, creating some data, performing some queries - is through this tutorial: <http://joelabrahamsson.com/elasticsearch-101/> (even though it is already quite old – the basic principles are all still valid). Go to the heading CRUD to start your interaction with Elastic Search:



The screenshot shows a web browser window with the URL joelabrahamsson.com/elasticsearch-101/ in the address bar. The page content is titled "CRUD". It contains the following text and code examples:

While we may want to use ElasticSearch primarily for searching the first step is to populate an index with some data, meaning the "Create" of CRUD, or rather, "indexing". While we're at it we'll also look at how to update, read and delete individual documents.

Indexing

In ElasticSearch indexing corresponds to both "Create" and "Update" in CRUD - if we index a document with a given type and ID that doesn't already exists it's inserted. If a document with the same type and ID already exists it's overwritten.

In order to index a first JSON object we make a PUT request to the REST API to a URL made up of the index name, type name and ID. That is: `http://localhost:9200/<index>/<type>/[<id>]`.

Index and type are required while the id part is optional. If we don't specify an ID ElasticSearch will generate one for us. However, if we don't specify an id we should use POST instead of PUT.

The index name is arbitrary. If there isn't an index with that name on the server already one will be created using default configuration.

As for the type name it too is arbitrary. It serves several purposes, including:

- Each type has its own ID space.
- Different types can have different mappings ("schema" that defines how properties/fields should be indexed).
- Although it's possible, and common, to search over multiple types, it's easy to search only for one or more specific type(s).

Let's index something! We can put just about anything into our index as long as it can be represented as a single JSON object. In this tutorial we'll be indexing and searching for movies. Here's a classic one:

```
{  
  "title": "The Godfather",  
  "year": 1972,  
  "director": "Francis Ford Coppola",  
  "actors": ["Marlon Brando", "Al Pacino", "Robert De Niro"],  
  "genre": "Drama",  
  "plot": "A Sicilian American crime family in New York City in the early 1940s.",  
  "rating": 9.2  
}
```

4. Getting Started with Kibana

The website of the Elastic Company provides a great tutorial for getting started with Kibana. This tutorial shows you how to:

- Load a sample data set into Elasticsearch
- Define an index pattern in Kibana
- Explore the sample data with Discover
- Set up *visualizations* of the sample data
- Assemble visualizations into a Dashboard

Access this tutorial at: <https://www.elastic.co/guide/en/kibana/current/getting-started.html>

Note: Kibana visualizations can be embedded in web applications very easily. See this tutorial for the instruction: <https://www.elastic.co/blog/kibana-4-video-tutorials-part-4> (it is nothing more than embedded a snippet of HTML)

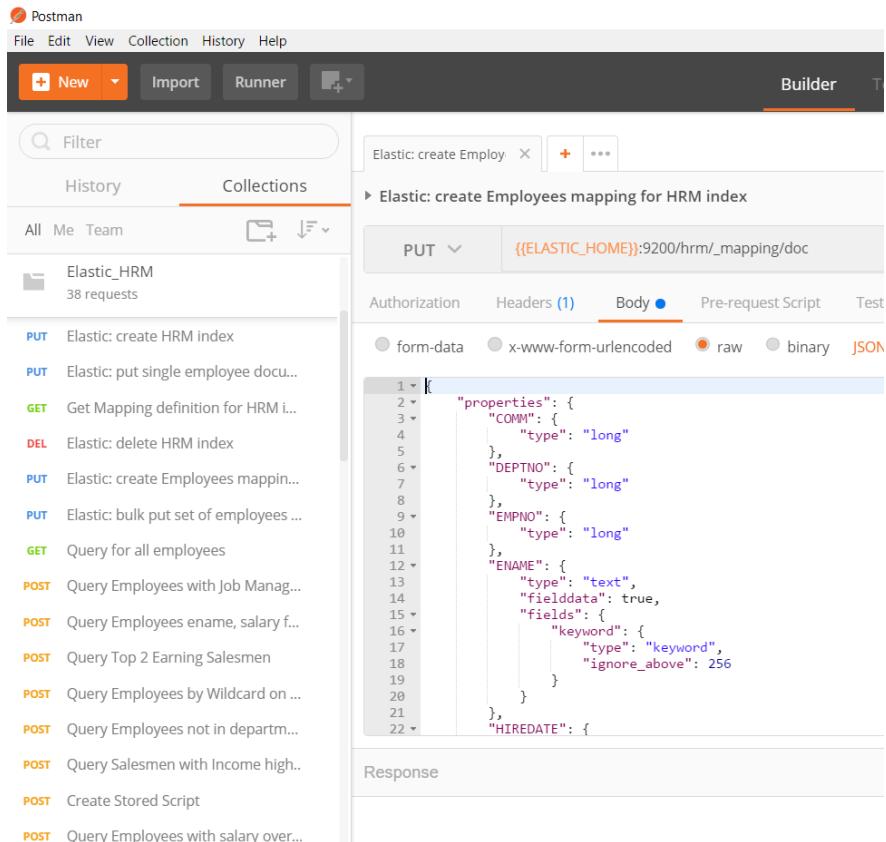
5. Comparing Elastic Search with (Oracle) SQL

Elastic Search is a NoSQL database for storing and especially for querying data. To learn what can be done with Elastic Search – in particular if you come from a relational database background – it can be useful to compare operations in a SQL database with Elastic Search. In this practice you will:

- Create a HRM index and load data for employees and departments (basically the data from EMP and DEPT in the SCOTT sample schema)
- Aa

Open the Powerpoint file *elasticsearch-search-agg-vs-oracle-sql.pptx* – or the corresponding PDF document. It contains over two dozen slides with a SQL query and the corresponding search against Elastic Search.

Open the Postman collection *Elastic_HRM.postman_collection.json* (in directory scott-emp-dept). This collection contains all requests corresponding to the slides in the presentation.



```
1 "properties": {  
2     "COMM": {  
3         "type": "long"  
4     },  
5     "DEPTNO": {  
6         "type": "long"  
7     },  
8     "EMPNO": {  
9         "type": "long"  
10    },  
11    "ENAME": {  
12        "type": "text",  
13        "fielddata": true,  
14        "fields": {  
15            "keyword": {  
16                "type": "keyword",  
17                "ignore_above": 256  
18            }  
19        }  
20    },  
21    "HIREDATE": {  
22        "type": "date"  
23    }  
24}
```

Start by executing *Elastic: create HRM Index*. Then: *Elastic: create Employees mapping for HRM index*. And use *Elastic: bulk put set of employees documents* to load the Employee data into the index.

You can now send the subsequent Query requests.

- GET** Query for all employees
- POST** Query Employees with Job Manager
- POST** Query Employees ename, salary for Salesmen ordered by salary desc
- POST** Query Top 2 Earning Salesmen
- POST** Query Employees by Wildcard on ENAME
- POST** Query Employees not in department 10, sorted by ename
- POST** Query Salesmen with Income higher than X
- POST** Create Stored Script
- POST** Query Employees with salary over salarycap

With *Elastic: add startdate* you can add a property of type date. With the request *Update all employees - set startdate* you take care of setting the value for this property for all documents.

Next, execute these requests to try out aggregations:

- POST** Query Name, Start Year and Month for all employees
- POST** Query Grand Aggregates for Employees
- POST** Query Aggregates per Department for Employees
- POST** Query Aggregates per Department and Hireyear for Employees (with a ...)

Execute request *Elastic: add Department to Mapping* to create the mapping definition for the department properties and run *Update all employees - set Department properties* to set definitions for all these employees.

Execute these requests:

- POST** Query Employees including Department details
- POST** Query Employees working in New York
- POST** Query Employees per Department
- POST** Query Aggregates - Facet Aggregation
- POST** Query Departments with Employee King

With the request *Elastic: add geolocation to Department to Mapping*, the department is extended with a geolocation property and with *Update all employees - set Department geolocation*, the location coordinates are set.

Then execute these geospatial queries:

POST Query Employees who work close to Washington

POST Query Employees with their distance from Washington

Finally, execute *Elastic: add mapping Biography to Employees* to create the mapping for the biography mapping and *Elastic: bulk update employees - set biography* to set some biographies.

These requests are the tip of the iceberg when it comes to leveraging the text search capabilities of Elastic Search.

POST Query Employees who lead

POST Query Employees for MANAGER

6. Explore LogStash + Elastic Search + Kibana in fully hosted environment

To get a feel for processing, storing, analyzing and exploring log file entries – one of the most prominent use cases of the Elastic Stack – you could take a look at a tutorial that provides a handson experience in a hosted environment.

The screenshot shows a Katacoda-based tutorial for setting up an ELK stack. The top navigation bar includes the Katacoda logo, a search bar, and links for "More Information about Katacoda" and "Sign In". The main content area has a title "Deploy ELK stack and aggregate container logs" and a progress indicator "Step 3 of 7".

Step 3 - Start Logstash

The next stage is to launch Logstash. Logstash processes data from any source and normalises it for storing. Logstash supports custom plugins and store data in multiple different backends.

Configuration

Logstash's configuration is stored in a file provided when the instance starts. You can view the config file using `cat logstash.conf`.

The file has three sections. The first section defines how it will receive data, in this case via Syslog requests on port 5000. The second section transforms the data and breaks it into parts of storing. The final section defines where to store the transformed data. In this case `stdout` and the Elasticsearch instance.

Task 1

The first task is to create a data container storing our configuration file. More details on [Data Containers](#) is covered in our [scenario](#).

```
docker create -v /config --name logstash_config busybox; docker cp logstash.conf logstash_config:/config/
```

Task 2

The second task is to launch the Logstash instance using the Data Container with `volumes-from`.

```
docker run -d \
-p 5000:5000 \
-v logstash_config:/config/
```

A terminal window on the right side shows the command history and the output of the Docker run command:

```
> docker run -d \
> -p 5000:5000 \
> -v logstash_config:/config/
```

The terminal also displays the log output of the Logstash container, which includes configuration details and log entries.

Access this tutorial at KataKoda - <https://www.katacoda.com/courses/docker-production/launch-elk-aggregate-container-logs> .

6. Client Libraries for Programmatic Interaction with Elastic Search

Any technology capable of making HTTP REST calls and manipulating JSON can use the generic REST APIs of Elastic Search to interact with any index. Additionally, a number of client libraries is available, that make this interaction easier - higher level, more native and therefore easier and more productive.

An example is the Java High Level REST Client (see:

<https://www.elastic.co/guide/en/elasticsearch/client/java-rest/current/java-rest-high.html>). This is a Java library that you can use in your own application (and that will still perform REST calls under the hood). See this tutorial for a concrete example: <https://qbox.io/blog/rest-calls-new-java-elasticsearch-client-tutorial> .

Client libraries are available for other programming languages as well, such as:

- Python: <https://elasticsearch-py.readthedocs.io/en/master/> (low level) and <https://elasticsearch-dsl.readthedocs.io/en/latest/> (high level).
- PHP: <https://www.elastic.co/guide/en/elasticsearch/client/php-api/current/index.html>
- JavaScript Node (server side)/Browser - <https://github.com/elastic/elasticsearch-js> ; see this tutorial for a quick start introduction: <https://qbox.io/blog/integrating-elasticsearch-into-node-js-application>
- Ruby (and Rails): <https://github.com/elastic/elasticsearch-rails>
- .Net: <https://github.com/elastic/elasticsearch-net>

Introduction to the Node Client Library for Elastic

See the sources on GitHub (<https://github.com/lucasjellema/sig-elasticsearch-february-2018>), specifically in directory *elastic-node-app*.

Check in package.json: the dependency on NPM module *elasticsearch*. See:

<https://github.com/elastic/elasticsearch-js> for the documentation for this module.

Then open file index.js. The salient parts:

- Connecting to the Elastic Search Server
- Executing (complex) queries – with multiple query and filter clauses - and processing the responses

Next, open data-manipulation.js. This file shows how to

- Add data to the index
- Add data in bulk to the index
- Modify documents
- Remove documents from the index

Nice read on interacting with Elastic Search from Node: <https://www.compose.com/articles/getting-started-with-elasticsearch-and-node/>