
Elastic Search

Release 1.0.0

Burhan Çelebi

Jan 02, 2022

CONTENTS

1	Installation	1
1.1	Download and Install ElasticSearch	1
1.2	Install package via composer :	1
1.3	Lumen	1
1.4	Laravel	1
2	Getting Started	3
3	Security/Authentication	5
4	Index	7
5	Nested	11
6	Sort	13
7	Multi Match	15
8	Document	19
9	Aggregation	21
10	Boolean Query	25
11	Geo Shape	27
12	Geo Point	29
13	Match	31
14	Range	33
15	Match Phrase	35
16	Suggest	37
17	Query	41
18	Other	43
18.1	Get Map Function	43
18.2	Search Function	43

INSTALLATION

1.1 Download and Install ElasticSearch

<https://www.elastic.co/downloads/elasticsearch?latest>

1.2 Install package via composer :

```
composer require burhancelebi/elasticsearch
```

1.3 Lumen

Add provider to your bootstrap/app.php

```
$app->register(ElasticSearch\ElasticSearchServiceProvider::class);
```

1.4 Laravel

Add provider namespace to your config/app.php providers array

```
ElasticSearch\ElasticSearchServiceProvider::class,
```


GETTING STARTED

Add this namespace to your class and start searching

```
use Elasticsearch\ElasticSearch;
```

Multi Match example:

```
1 <?php
2
3 use Elasticsearch\ElasticSearch;
4
5 public function multiMatch()
6 {
7     $elastic = new Elasticsearch();
8
9     $search = $elastic->multiMatch()
10         ->index('test')
11         ->text('Șero')
12         ->fuzziness(1)
13         ->fields(['text^3', 'name'])
14         ->search();
15
16     return $search;
17 }
```


SECURITY/AUTHENTICATION

You can connect to Elastic using Basic Authentication

```
1 <?php
2
3 use Elasticsearch\ElasticSearch;
4
5 class ExampleController extends Controller
6 {
7     private $elastic;
8
9     public function __construct()
10    {
11        $this->elastic = new ElasticSearch();
12
13        $this->elastic->client()
14            ->basicAuth('user', 'password');
15    }
16
17    public function nested()
18    {
19        $nested = $this->elastic
20            ->nested()
21            ->index('my-index')
22            ->path('user')
23            ->params('size', 1)
24            ->addQuery('elastic.bool', function() {
25                return $this
26                    ->must([
27                        [
28                            'match' => ['user.first' => '$ero']
29                        ]
30                    ])
31                    ->getMap();
32            })
33            ->search();
34
35        return $nested;
36    }
37 }
```

You can set cloud id, hosts and api key

```
1 <?php
2
3 use Elasticsearch\ElasticSearch;
4
5 class ExampleController extends Controller
6 {
7     private $elastic;
8
9     public function __construct()
10    {
11        $this->elastic = new ElasticSearch();
12
13        $hosts = [
14            'http://localhost:9200'
15        ];
16
17        $this->elastic->client()
18            ->basicAuth('user', 'password')
19            ->hosts($hosts)
20            ->cloudId('cloud-id')
21            ->apiKey('enter-your-api-key')
22            ->sslVerification('path/to/cacert.pem');
23    }
24
25    public function multiMatch()
26    {
27
28        $search = $this->elastic->multiMatch()
29            ->index('my-index')
30            ->text('Şero')
31            ->fuzziness(1)
32            ->fields(['text^3', 'name'])
33            ->search();
34
35        return $search;
36    }
37 }
```

INDEX

Index Information

```
1 <?php
2
3 $elastic = new Elasticsearch();
4
5 $result = $elastic->query()->getIndices(); // It returns all indices information
6
7 // You can get specific index information.
8
9 $result = $elastic->query(['pointsort,location'])->getIndices();
10
11 return $result;
```

Create Index

```
1 <?php
2
3 $elastic = new Elasticsearch();
4 $index = $elastic->index()
5           ->name('my-index')
6           ->create();
7
8 return $index;
```

Delete Index

```
1 <?php
2
3 $index = $elastic->index()
4           ->name('my-index')
5           ->delete();
```

Settings

```
1 <?php
2
3 $index = $elastic->index()
4           ->name('my-index')
5           ->settings([
6               'number_of_shards' => 3,
```

(continues on next page)

(continued from previous page)

```
7         'number_of_replicas' => 2
8     })
9     ->create();
```

Mappings

```
1 <?php
2
3 $index = $elastic->index()
4     ->name('my-index')
5     ->mappings(function(){ // Closure $mappings
6         return $this->suggest([
7             'type' => 'completion',
8         ])
9         ->property('title', [
10             'type' => 'text'
11         ])
12         ->getMap();
13     })
14     ->create();
15
16 return $index;
```

Check if an index exists

```
1 <?php
2
3 $exists = $elastic->index()
4     ->exists([
5         'index' => 'my-index'
6     ]);
7
8 return $exists;
```

Get index settings

```
1 <?php
2
3 $settings = $elastic->index()
4     ->getSettings([
5         'index' => 'my-index'
6     ]);
7
8 return $settings;
```

Get index mappings

```
1 <?php
2
3 $map = $elastic->index()
4     ->map([
5         'index' => 'my-index'
6     ]);
```

(continues on next page)

(continued from previous page)

```
7
8 return $map;
```

Update index mappings

```
1 <?php
2
3     $index = $elastic->index()
4         ->name('my-index')
5         ->mappings(function(){
6             return $this
7                 ->body([
8                     'properties' => [
9                         'location.keyword' => [
10                             'type' => 'geo_shape',
11                         ]
12                     ]
13                 ])
14         ->getMap();
15     })
16     ->updateMap();
17
18 return $index;
```


NESTED

The nested type is a specialised version of the object data type that allows arrays of objects to be indexed in a way that they can be queried independently of each other.

Using nested fields for arrays of objects

Firstly we need to set mapping to use:

```
1 <?php
2
3 $index = $this->elastic
4     ->index()
5     ->name('my-index')
6     ->mappings(function(){
7         return $this
8             ->property('user', [
9                 'type' => 'nested'
10            ])
11     ->getMap();
12
13     })
14     ->create();
```

Save a Nested data:

```
1 <?php
2
3 $save = $this->elastic->document()
4     ->index('my-index')
5     ->body([
6         'group' => 'fans',
7         'user' => [
8             [
9                 'first' => 'John',
10                'last'  => 'Smith'
11            ],
12            [
13                'first' => 'Alice',
14                'Last'  => 'White'
15            ]
16        ]
17    ])
18    ->save();
```

Searching Nested data:

```
1 <?php
2
3 $nested = $this->elastic
4     ->nested()
5     ->index('my-index')
6     ->path('user')
7     ->params('size', 2)
8     ->addQuery('elastic.bool', function() {
9         return $this
10             ->must([
11                 [
12                     'match' => ['user.first' => 'Alice']
13                 ]
14             ])
15             ->getMap();
16     })
17     ->search();
```

SORT

Sort people by age

```
1 <?php
2
3 $sort = $elastic->sort()
4         ->index('people')
5         ->field('age')
6         ->order('desc') // you can change to "asc"
7         ->search();
8
9 return $sort;
```


MULTI MATCH

The multi_match query builds on the match query to allow multi-field queries:

```
1 <?php
2
3 $search = $elastic->multiMatch()
4     ->index('my-index')
5     ->text('Şêro')
6     ->fields(['name^3'])
7     ->search();
8
9 // If you want to set limit :
10 $search = $elastic->multiMatch()
11     ->index('my-index')
12     ->size(10) // returns max 10 data
13     ->text('Şêro')
14     ->fields(['name^3'])
15     ->search();
16
17 // add fuzziness :
18 $search = $elastic->multiMatch()
19     ->index('my-index')
20     ->size(10)
21     ->fuzziness(1) // Ideal value is 1 and 2
22     ->text('Şêro')
23     ->fields(['name^3'])
24     ->search();
25
26 // Merge with sort query :
27 $search = $elastic->multiMatch()
28     ->index('my-index')
29     ->size(10)
30     ->fuzziness(1)
31     ->text('Şêro')
32     ->fields(['name^3'])
33     ->mergeWith('elastic.sort', function(){
34         return $this
35             ->field('age')
36             ->order('desc')
37             ->getMap();
38     })
```

(continues on next page)

(continued from previous page)

```

39         ->search();
40
41 // Merge with aggregations
42
43 $search = $elastic->multiMatch()
44     ->index('my-index')
45     ->text('Şero')
46     ->fuzziness(1)
47     ->fields(['name^3'])
48     ->mergeWith('elastic.aggs', function(){
49         return $this
50             ->name('my-aggs')
51             ->terms([
52                 'field' => 'age'
53             ])
54             ->subAggs(function(){
55                 return $this
56                     ->name('sub-aggs')
57                     ->scope([
58                         'avg' => [
59                             'field' => 'age'
60                         ]
61                     ])
62                     ->getMap();
63             })
64             ->getMap();
65     })
66     ->search();
67
68 // Merge with highlight query :
69 $search = $elastic->multiMatch()
70     ->index('my-index')
71     ->size(10)
72     ->fuzziness(1)
73     ->text('Şero')
74     ->fields(['name^3'])
75     ->mergeWith('elastic.sort', function(){
76         return $this
77             ->field('age')
78             ->order('desc')
79             ->getMap();
80     })
81     ->mergeWith('elastic.highlight', function(){
82         return $this
83             ->content('name', '<div>', '</div>')
84             ->getMap();
85     })
86     ->search();
87
88 // Other useful functions
89 $search = $elastic->multiMatch()
90     ->index('my-index')

```

(continues on next page)

(continued from previous page)

```
91     ->size(10)
92     ->fuzziness(1)
93     ->text('Şero')
94     ->fields(['name^3'])
95     ->analyzer('standard')
96     ->tieBreaker(0.3)
97     ->operator('and')
98     ->type('most_fields')
99     ->search();
```


DOCUMENT

Add a document to ElasticSearch

```
1 <?php
2
3 $elastic = new ElasticSearch();
4
5 $index = $elastic->document()
6     ->index('my-index')
7     ->body([
8         'text' => 'What is your name',
9         'location' => [
10             'type' => 'point',
11             'coordinates' => [50.30, 20.34],
12         ]
13     ])
14     ->save();
15
16 // You can merge with suggest
17 $index = $elastic->document()
18     ->index('my-index')
19     ->body([
20         'text' => 'My name is $êro',
21         'location' => [
22             'type' => 'point',
23             'coordinates' => [50.30, 20.34],
24         ]
25     ])
26     ->mergeWith('suggest', function(){
27         return $this
28             ->input([
29                 [
30                     'input' => 'Music Time',
31                     'weight' => 20
32                 ]
33             ])
34             ->getMap();
35     })
36     ->save();
```

Find a Document

```
1 <?php
2
3 $elastic = new ElasticSearch();
4
5 $find = $elastic->find('my-index', 'enter-id')->get();
6
7 return $find;
```

Update a Document

```
1 <?php
2
3 $elastic = new ElasticSearch();
4
5 $update = $elastic->find('my-index', 'Lpkpd53YBR6XiqYGx3M98')->update([
6     'name' => 'Şero'
7 ]);
8
9 return $update;
```

Delete a Document

```
1 <?php
2
3 $elastic = new ElasticSearch();
4
5 $delete = $elastic->find('my-index', 'Lpkpd53YBR6XiqYGx3M98')->delete();
6
7 return $delete;
```

AGGREGATION

An aggregation summarizes your data as metrics, statistics, or other analytics. Aggregations help you answer questions like:

- What's the average load time for my website?
- Who are my most valuable customers based on transaction volume?
- What would be considered a large file on my network?
- How many products are in each product category?

A sample search

```
1 <?php
2
3 $elastic = new Elasticsearch();
4
5 $aggs = $elastic->aggs()
6         ->index('my-index')
7         ->name('my-aggs')
8         ->scope([
9             'avg' => [
10                'field' => 'age'
11            ]
12        ])
13         ->search();
14
15 return $aggs;
```

Terms Aggregation

```
1 <?php
2
3 $elastic = new Elasticsearch();
4
5 $aggs = $elastic->aggs()
6         ->index('my-index')
7         ->name('my-aggs')
8         ->terms([
9             'field' => 'age',
10        ])
11         ->getMap();
```

(continues on next page)

(continued from previous page)

```
12
13 return $aggs;
```

Add custom metadata

```
1 <?php
2
3 $elastic = new ElasticSearch();
4
5 $aggs = $elastic->aggs()
6     ->index('my-index')
7     ->name('my-aggs')
8     ->terms([
9         'field' => 'age',
10    ])
11     ->meta([
12         "my-metadata-field" => "foo"
13    ])
14     ->search();
15
16 return $aggs;
```

Return only aggregation results

By default, searches containing an aggregation return both search hits and aggregation results. To return only aggregation results, set size to 0.

```
1 <?php
2
3 $elastic = new ElasticSearch();
4
5 $aggs = $elastic->aggs()
6     ->index('my-index')
7     ->name('my-avg-aggs')
8     ->size(0) // default 1
9     ->scope([
10         'avg' => [
11             'field' => 'age'
12         ]
13     ])
14     ->search();
```

Run multiple aggregations

You can use multiple aggregations in the same request.

```
1 <?php
2
3 $elastic = new ElasticSearch();
4
5 $aggs = $elastic->aggs()
6     ->index('my-index')
7     ->name('my-aggs')
8     ->scope([
```

(continues on next page)

(continued from previous page)

```

9         'avg' => [
10             'field' => 'age'
11         ]
12     })
13     ->addAggs(function(){
14         return $this
15             ->name('my-aggs-2')
16             ->size(0)
17             ->scope([
18                 'avg' => [
19                     'field' => 'age'
20                 ]
21             ->getMap();
22     })
23     ->addAggs(function(){
24         ...
25     })
26     ->search();
27
28 return $aggs;

```

Run sub-aggregations

Bucket aggregations support bucket or metric sub-aggregations. There is no level or depth limit for nesting sub-aggregations.

```

1 <?php
2
3 $elastic = new Elasticsearch();
4
5 $aggs = $elastic->aggs()
6     ->index('my-index')
7     ->name('my-aggs')
8     ->terms([
9         'field' => 'age',
10    ])
11    ->subAggs(function(){
12        return $this
13            ->name('sub-aggs')
14            ->scope([
15                'avg' => [
16                    'field' => 'age'
17                ]
18            ->getMap();
19    })
20    // you can add multiple sub-aggregations
21    ->subAggs(function(){
22        ...
23    })
24    ->search();
25
26
27 // Or you can use like that:

```

(continues on next page)

(continued from previous page)

```
28
29 $aggs = $elastic->aggs()
30     ->index('my-index')
31     ->name('my-avg-aggs')
32     ->terms([
33         'field' => 'age',
34     ])
35     ->addAggs(function(){
36         return $this
37             ->name('my-avg-aggs-2')
38             ->size(0)
39             ->terms([
40                 'field' => 'age',
41             ])
42             ->subAggs(function(){
43                 return $this
44                     ->name('sub-aggs')
45                     ->scope([
46                         'avg' => [
47                             'field' => 'age'
48                         ]
49                     ])
50                     ->getMap();
51             })
52             ->getMap();
53     })
54     ->getMap();
55
56 return $aggs;
```

BOOLEAN QUERY

A query that matches documents matching boolean combinations of other queries. The bool query maps to Lucene BooleanQuery. It is built using one or more boolean clauses, each clause with a typed occurrence.

```
1 <?php
2
3 $elastic = new Elasticsearch();
4
5 // Search using must
6 $search = $elastic->bool()
7     ->index('my_index')
8     ->must([
9         ['term' => ['name.keyword' => 'Şero']]
10    ])
11     ->search();
12
13 // add filter
14 $search = $elastic->bool()
15     ->index('my_index')
16     ->must([
17         ['term' => ['name.keyword' => 'Şero']]
18    ])
19     ->filter([
20         ['term' => ['tags' => 'production']]
21    ])
22     ->minimum_should_match(1)
23     ->boost(1)
24     ->search();
25
26 // must not
27     ...
28     ->mustNot([
29         ['range' => ['gte' => 10, 'lte' => 20]]
30    ]);
31
32 // should
33     ...
34     ->should([
35         ['term' => ['tags' => 'env1']],
36         ['term' => ['tags' => 'deployed']]
37    ])
```

(continues on next page)

(continued from previous page)

```
38
39 // Merge with SORT
40     ...
41     ->mergeWith('elastic.sort', function(){
42         return $this
43             ->field('age')
44             ->order('asc')
45             ->getMap();
46     });
47
48 // Merge with HIGHLIGHT
49     ...
50     ->mergeWith('elastic.highlight', function(){
51         return $this
52             ->content('name.keyword', '<div>', '</div>')
53             ->getMap();
54     });
55
56 // You can add GEO FILTER
57     ...
58     ->addGeoFilter('geo_shape', function(){
59         return $this
60             ->index('location')
61             ->type('circle')
62             ->text('What is your name')
63             ->coordinates([51.30, 20.34])
64             ->radius('111km')
65             ->getMap();
66     });
67
68 return $search;
```

GEO SHAPE

The `geo_shape` data type facilitates the indexing of and searching with arbitrary geo shapes such as rectangles and polygons. It should be used when either the data being indexed or the queries being executed contain shapes other than just points.

Firstly you have to set index mappings for `geo_shape` if you haven't set:

```
1 <?php
2
3
4 $index = $elastic
5     ->index()
6     ->name('my-index')
7     ->mappings(function(){
8         return $this
9             ->body([
10                 'properties' => [
11                     'location' => [
12                         'type' => 'geo_shape',
13                     ]
14                 ]
15             })
16     ->getMap();
17
18     })
19     ->updateMap();
```

Save a document to search it

```
1 <?php
2
3 $index = $elastic->document()
4     ->index('my-index')
5     ->body([
6         'text' => 'Sêro',
7         'location' => [
8             'type' => 'point',
9             'coordinates' => [-77.30, 38.34],
10         ]
11     ])
12     ->save();
```

Lets do a CIRCLE example:

```
1 <?php
2
3 $geo = $elastic->geoShape()
4         ->index('my-index')
5         ->type('circle')
6         ->text('Sêro')
7         ->coordinates([-77.30, 38.34])
8         ->radius('111km')
9         ->search();
10
11 return $geo;
```

For more information: <https://www.elastic.co/guide/en/elasticsearch/reference/current/geo-shape.html>

GEO POINT

Fields of type `geo_point` accept latitude-longitude pairs

Set the index mappings

```
1 <?php
2
3 $index = $elastic
4     ->index()
5     ->name('my_index')
6     ->mappings(function(){
7         return $this
8             ->body([
9                 'properties' => [
10                     'location' => [
11                         'type' => 'geo_point',
12                     ]
13                 ]
14             })
15     ->getMap();
16 })
17 ->updateMap();
```

Let's add an example data:

```
1 <?php
2
3 $data = $elastic->document()
4     ->index('my-index')
5     ->body([
6         'text' => 'Şêro',
7         'location' => [
8             "lat" => 41.12,
9             "lon" => -71.34
10        ]
11    ])
12    ->save();
```

Searching...

```
1 <?php
2
```

(continues on next page)

(continued from previous page)

```
3 $search = $elastic->geoPoint()
4         ->index('my-index')
5         ->topLeft(42, -72)
6         ->bottomRight(40, -74)
7         ->getMap();
8
9 return $search;
```

You can search with location function

```
1 <?php
2
3 $geo = $elastic->geoPoint()
4         ->index('my-index')
5         ->pointType('geo_distance')
6         ->location([
7             'location' => [
8                 "top_left" => [
9                     "lat" => 42,
10                    "lon" => -72
11                ],
12                "bottom_right" => [
13                    "lat" => 40,
14                    "lon" => -74
15                ]
16            ]
17        ])
18        ->search();
19
20 // you don't need to use pointType(), it will return geo_bounding_box by default
21
22 return $geo;
```

MATCH

Returns documents that match a provided text, number, date or boolean value. The provided text is analyzed before matching.

The match query is the standard query for performing a full-text search, including options for fuzzy matching.

A sample search with Match

```
1 <?php
2
3 $elastic = new Elasticsearch();
4
5 $match = $elastic->match()
6     ->index('my-index')
7     ->field([
8         'name' => [
9             'query' => 'Șero',
10            'fuzziness' => 1
11        ]
12    ])
13    ->search();
14
15 // You can merge with other features:
16
17 $match = $elastic->match()
18     ->index('my-index')
19     ->field([
20         'name' => [
21             'query' => 'Șero',
22             'fuzziness' => 1
23         ]
24     ])
25     ->mergeWith('elastic.sort', function(){
26         return $this->field('age')
27             ->order('desc')
28             ->getMap();
29     })
30     ->mergeWith('elastic.highlight', function(){
31         return $this
32             ->content('name', '<div>', '</div>')
33             ->getMap();
34     })
```

(continues on next page)

(continued from previous page)

```
35         ->search();  
36  
37     return $match;
```

RANGE

Returns documents that contain terms within a provided range.

```
1 <?php
2
3 $elastic = new Elasticsearch();
4
5 $search = $elastic->range()
6           ->index('my-index')
7           ->content([
8               'age' => [
9                   'gte' => 21,
10                  'lte' => 26
11              ]
12          ])
13           ->search();
14
15 return $search;
```


MATCH PHRASE

The `match_phrase` query analyzes the text and creates a phrase query out of the analyzed text.

A sample search

```
1 <?php
2
3 $elastic = new Elasticsearch();
4
5 $phrase = $elastic->matchPhrase()
6           ->index('my-index')
7           ->field('name')
8           ->text('I use Elastic Search')
9           ->addition(['slop' => 1])
10          ->search();
11
12 return $phrase;
```


SUGGEST

Suggests similar looking terms based on a provided text by using a suggester. Parts of the suggest feature are still under development.

Save a suggestion to index

```
1 <?php
2
3 $elastic = new Elasticsearch();
4
5 $suggest = $elastic->suggest()
6             ->index('my-index')
7             ->id(1)
8             ->input([
9                 [
10                    'input' => 'My name is $êro',
11                    'weight' => 25
12                ]
13            ])
14             ->save();
```

Search with suggest

```
1 <?php
2
3 $suggest = $elastic->suggest()
4             ->index('my-index')
5             ->name('my-index-suggest')
6             ->text('My name is ') // it returns "My name is $êro"
7             ->completion([
8                 'field' => 'suggest',
9                 'skip_duplicates' => true
10            ])
11             ->search();
12
13 return $suggest;
```

You can use multiple suggestion

```
1 <?php
2
3 $suggest = $elastic->suggest()
```

(continues on next page)

(continued from previous page)

```

4      ->index('my-index')
5      ->name('my-index-suggest')
6      ->text('My name is ') // it returns "My name is Şêro"
7
8      ->completion([
9          'field' => 'suggest',
10         'skip_duplicates' => true
11     ])
12     ->addSuggest(function(){
13         return $this->index('my-index')
14             ->name('my-index-suggest-2')
15             ->text('Music')
16             ->completion([
17                 'field' => 'suggest',
18                 'skip_duplicates' => true,
19                 'fuzzy' => [
20                     'fuzziness' => 1
21                 ]
22             ])
23     })
24     ->addSuggest(function(){
25         return $this->index('my-index')
26             ->name('my-index-suggest-3')
27             ->text('Music')
28             ->completion([
29                 'field' => 'suggest',
30                 'skip_duplicates' => true
31             ])
32     })
33     ->search();
34
35     return $suggest;
36

```

Other functions you can use for suggestion

```

1  <?php
2
3  $suggest = $elastic->suggest()
4      ->index('my-index')
5      ->name('product-suggestion')
6      ->text('tring out Elasticsearch')
7      ->term(['name' => 'message'])
8      ->input([ "Nevermind", "Nirvana" ])
9      ->weight(34)
10     ->prefix('nir')
11     ->regex('regex')
12     ->completion([
13         'field' => 'suggest',
14         'size' => 10,
15         'contexts' => [
16             'place_type' => [

```

(continues on next page)

(continued from previous page)

```

17         [
18             'context' => 'cafe'
19         ],
20         [
21             'context' => 'restaurants',
22             'boost' => 2
23         ]
24     ]
25 ]
26 })
27 ->contexts([
28     [
29         "name" => "place_type",
30         "type" => "category"
31     ],
32     [
33         "name" => "location",
34         "type" => "geo",
35         "precision" => 4
36     ],
37 ])
38 ->simplePhrase(function(){
39     return $this
40         ->field('name')
41         ->directGenerator([
42             [
43                 'field' => 'name',
44                 'suggest_mode' => 'always'
45             ],
46             [
47                 "field" => "title.reverse",
48                 "suggest_mode" => "always",
49                 "pre_filter" => "reverse",
50                 "post_filter" => "reverse"
51             ]
52         ])
53     ->highlight([
54         'pre_tag' => "<em>",
55         'post_tag' => "</em>"
56     ])
57     ->collate([
58         'source' => [
59             'match' => [
60                 "{{field_name}}" => "{{suggestion}}"
61             ]
62         ]
63     ])
64     ->smoothing([
65         'laplace' => [
66             'alpha' => 0.7
67         ]
68     ])

```

(continues on next page)

(continued from previous page)

```
69         ->size(2)
70         ->getMap();
71     })
72     ->getMap();
```

QUERY

You can use query function for creating, updating, searching etc.

The index aliases API allows aliasing an index with a name, with all APIs automatically converting the alias name to the actual index name.

Display the aliases list

```
1 <?php
2
3 $query = $elastic->query()
4         ->cat()
5         ->aliases();
6
7 return $query;
```

Searching with query function:

```
1 <?php
2
3 $elastic = new Elasticsearch();
4
5 $result = $elastic->query([
6     'index' => 'my-index',
7     'body' => [
8         'query' => [
9             'match' => [
10                'name' => [
11                    'query' => 'Şêro',
12                    'fuzziness' => 1
13                ]
14            ]
15        ]
16    ]
17 ]->search();
18
19 return $result;
```


18.1 Get Map Function

Before using the creating, searching, updating etc. functions, you can display the Elasticsearch map.

For example:

```
1 <?php
2
3 $map = $elastic->index()
4     ->name('my-index')
5     ->mappings(function(){
6         return $this
7             ->body([
8                 'properties' => [
9                     'location.keyword' => [
10                         'type' => 'geo_shape',
11                     ]
12                 ]
13             })
14     ->getMap();
15
16 })
17 ->getMap(); // it returns map
18
19 return $map;
20
21 // You will get something like that:
22 /* {"index":"my-index","body":{"mappings":{"properties":{"location.keyword":{"type":
23     ↪ "geo_shape"}}}}} */
```

18.2 Search Function

This function will trigger Elasticsearch search function, if you add a parameter as array, you will get just those fields.

For example:

```
1 <?php
2
3 $sort = $elastic->sort()
4     ->index('people')
```

(continues on next page)

(continued from previous page)

```
5      ->field('age')
6      ->order('desc') // you can change to "asc"
7      ->search(['hits', 'took']); // spesific parameters
8
9  // You will get hits and took fields
10
11 return $sort;
```