Elastic Search

Release 1.0.0

Burhan Çelebi

Jan 02, 2022

CONTENTS

CHAPTER

ONE

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,
```

CHAPTER

TWO

GETTING STARTED

Add this namespace to your class and start searching

```
use ElasticSearch\ElasticSearch;
```

Multi Match example:

```php
<?php

use ElasticSearch\ElasticSearch;

public function multiMatch()
{
    $elastic = new ElasticSearch();

    $search = $elastic->multiMatch()
                    ->index('test')
                    ->text('Şêro')
                    ->fuzziness(1)
                    ->fields(['text^3', 'name'])
                    ->search();

    return $search;
}
```

CHAPTER

THREE

SECURITY/AUTHENTICATION

You can connect to Elastic using Basic Authentication

```php
<?php

use ElasticSearch\ElasticSearch;

class ExampleController extends Controller
{
    private $elastic;

    public function __construct()
    {
        $this->elastic = new ElasticSearch();

        $this->elastic->client()
                    ->basicAuth('user', 'password');
    }

    public function nested()
    {
        $nested = $this->elastic
                    ->nested()
                    ->index('my-index')
                    ->path('user')
                    ->params('size', 1)
                    ->addQuery('elastic.bool', function() {
                        return $this
                                ->must([
                                    [
                                        'match' => ['user.first' => 'Şero']
                                    ]
                                ])
                                ->getMap();
                    })
                    ->search();

        return $nested;
    }
}
```

You can set cloud id, hosts and api key

```php
<?php

use ElasticSearch\ElasticSearch;

class ExampleController extends Controller
{
    private $elastic;

    public function __construct()
    {
        $this->elastic = new ElasticSearch();

        $hosts = [
            'http://localhost:9200'
        ];

        $this->elastic->client()
                    ->basicAuth('user', 'password')
                    ->hosts($hosts)
                    ->cloudId('cloud-id')
                    ->apiKey('enter-your-api-key')
                        ->sslVerification('path/to/cacert.pem');
    }

    public function multiMatch()
    {

        $search = $this->elastic->multiMatch()
                        ->index('my-index')
                        ->text('Ŝêro')
                        ->fuzziness(1)
                        ->fields(['text^3', 'name'])
                        ->search();

        return $search;
    }
}
```

CHAPTER

FOUR

INDEX

Index Information

```php
<?php

$elastic = new ElasticSearch();

$result = $elastic->query()->getIndices(); // It returns all indices information

// You can get specific index information.

$result = $elastic->query(['pointsort,location'])->getIndices();

return $result;
```

Create Index

```php
<?php

    $elastic = new ElasticSearch();
    $index = $elastic->index()
                    ->name('my-index')
                    ->create();

    return $index;
```

Delete Index

```php
<?php

$index = $elastic->index()
                ->name('my-index')
                ->delete();
```

Settings

```php
<?php

$index = $elastic->index()
               ->name('my-index')
               ->settings([
                   'number_of_shards' => 3,
```

(continues on next page)

(continued from previous page)

```
                'number_of_replicas' => 2
            ])
            ->create();
```

Mappings

```
<?php

$index = $elastic->index()
                ->name('my-index')
                ->mappings(function(){ // Closure $mappings
                    return $this->suggest([
                                'type' => 'completion',
                            ])
                            ->property('title', [
                                'type' => 'text'
                            ])
                            ->getMap();
                })
                ->create();

    return $index;
```

Check if an index exists

```
<?php

$exists = $elastic->index()
                    ->exists([
                        'index' => 'my-index'
                    ]);

    return $exists;
```

Get index settings

```
<?php

$settings = $elastic->index()
                        ->getSettings([
                            'index' => 'my-index'
                        ]);

return $settings;
```

Get index mappings

```
<?php

$map = $elastic->index()
                ->map([
                        'index' => 'my-index'
                  ]);
```

(continues on next page)

(continued from previous page)

```

return $map;
```

Update index mappings

```
<?php

    $index = $elastic->index()
                    ->name('my-index')
                    ->mappings(function(){
                            return $this
                                    ->body([
                                        'properties' => [
                                            'location.keyword' => [
                                                'type' => 'geo_shape',
                                            ]
                                        ]
                                    ])
                                    ->getMap();
                    })
                    ->updateMap();

    return $index;
```

CHAPTER

FIVE

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:

```php
<?php

$index = $this->elastic
                    ->index()
                    ->name('my-index')
                    ->mappings(function(){
                            return $this
                                    ->property('user', [
                                        'type' => 'nested'
                                    ])
                                    ->getMap();
                    })
                    ->create();
```

Save a Nested data:

```php
<?php

$save = $this->elastic->document()
                ->index('my-index')
                ->body([
                    'group' => 'fans',
                    'user' => [
                        [
                            'first' => 'John',
                            'last'  => 'Smith'
                        ],
                        [
                            'first' => 'Alice',
                            'Last' => 'White'
                        ]
                    ]
                ])
                ->save();
```

Searhing Nested data:

```php
<?php

$nested = $this->elastic
                    ->nested()
                    ->index('my-index')
                    ->path('user')
                    ->params('size', 2)
                    ->addQuery('elastic.bool', function() {
                        return $this
                                ->must([
                                    [
                                        'match' => ['user.first' => 'Alice']
                                    ]
                                ])
                                ->getMap();
                    })
                    ->search();
```

CHAPTER

SIX

SORT

Sort people by age

```php
<?php

$sort = $elastic->sort()
                ->index('people')
                ->field('age')
                ->order('desc') // you can change to "asc"
                ->search();

return $sort;
```

CHAPTER

SEVEN

MULTI MATCH

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

```php
<?php

$search = $elastic->multiMatch()
                ->index('my-index')
                ->text('Şêro')
                ->fields(['name^3'])
                ->search();

// If you want to set limit :
$search = $elastic->multiMatch()
                ->index('my-index')
                ->size(10) // returns max 10 data
                ->text('Şêro')
                ->fields(['name^3'])
                ->search();

//  add fuzziness :
$search = $elastic->multiMatch()
                ->index('my-index')
                ->size(10)
                ->fuzziness(1) // Ideal value is 1 and 2
                ->text('Şêro')
                ->fields(['name^3'])
                ->search();

// Merge with sort query :
$search = $elastic->multiMatch()
                ->index('my-index')
                ->size(10)
                ->fuzziness(1)
                ->text('Şêro')
                ->fields(['name^3'])
                ->mergeWith('elastic.sort', function(){
                        return $this
                                ->field('age')
                                ->order('desc')
                                ->getMap();
                    })
```

(continues on next page)

(continued from previous page)

```
                ->search();

// Merge with aggregations

$search = $elastic->multiMatch()
                ->index('my-index')
                ->text('Şêro')
                ->fuzziness(1)
                ->fields(['name^3'])
                ->mergeWith('elastic.aggs', function(){
                  return $this
                            ->name('my-aggs')
                            ->terms([
                                    'field' => 'age'
                            ])
                            ->subAggs(function(){
                                return $this
                                    ->name('sub-aggs')
                                    ->scope([
                                        'avg' => [
                                            'field' => 'age'
                                        ]
                                    ])
                                    ->getMap();
                            })
                            ->getMap();
                })
                ->search();

// Merge with highlight query :
$search = $elastic->multiMatch()
                ->index('my-index')
                ->size(10)
                ->fuzziness(1)
                ->text('Şêro')
                ->fields(['name^3'])
                ->mergeWith('elastic.sort', function(){
                        return $this
                                ->field('age')
                                ->order('desc')
                                ->getMap();
                })
                ->mergeWith('elastic.highlight', function(){
                            return $this
                                ->content('name', '<div>', '</div>')
                                ->getMap();
                })
                ->search();

// Other useful functions
$search = $elastic->multiMatch()
            ->index('my-index')
```

(continues on next page)

(continued from previous page)

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

CHAPTER

EIGHT

DOCUMENT

Add a document to ElasticSearch

```php
<?php

$elastic = new ElasticSearch();

$index = $elastic->document()
                ->index('my-index')
                ->body([
                    'text' => 'What is your name',
                    'location' => [
                        'type' => 'point',
                        'coordinates' => [50.30, 20.34],
                    ]
                ])
                ->save();

// You can merge with suggest
$index = $elastic->document()
                ->index('my-index')
                ->body([
                    'text' => 'My name is Ŝêro',
                    'location' => [
                        'type' => 'point',
                        'coordinates' => [50.30, 20.34],
                    ]
                ])
                ->mergeWith('suggest', function(){
                        return $this
                                ->input([
                                    [
                                        'input' => 'Music Time',
                                        'weight' => 20
                                    ]
                                ])
                                ->getMap();
                    })
                ->save();
```

Find a Document

```php
<?php

$elastic = new ElasticSearch();

$find = $elastic->find('my-index', 'enter-id')->get();

return $find;
```

Update a Document

```php
<?php

$elastic = new ElasticSearch();

$update = $elastic->find('my-index', 'Lpkpd53YBR6XiqYGx3M98')->update([
    'name' => 'Şêro'
]);

return $update;
```

Delete a Document

```php
<?php

$elastic = new ElasticSearch();

$delete = $elastic->find('my-index', 'Lpkpd53YBR6XiqYGx3M98')->delete();

return $delete;
```

CHAPTER

NINE

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

```php
<?php

$elastic = new ElasticSearch();

$aggs = $elastic->aggs()
            ->index('my-index')
            ->name('my-aggs')
            ->scope([
                'avg' => [
                    'field' => 'age'
                ]
            ])
            ->search();

return $aggs;
```

Terms Aggregation

```php
<?php

$elastic = new ElasticSearch();

$aggs = $elastic->aggs()
            ->index('my-index')
            ->name('my-aggs')
            ->terms([
                'field' => 'age',
            ])
            ->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)

```
            'avg' => [
                'field' => 'age'
            ]
        ])
        ->addAggs(function(){
            return $this
                ->name('my-aggs-2')
                ->size(0)
                ->scope([
                    'avg' => [
                        'field' => 'age'
                    ]
                ])->getMap();
        })
        ->addAggs(function(){
            ...
        })
        ->search();

return $aggs;
```

Run sub-aggregations

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

```
<?php

$elastic = new ElasticSearch();

$aggs = $elastic->aggs()
        ->index('my-index')
        ->name('my-aggs')
        ->terms([
            'field' => 'age',
        ])
        ->subAggs(function(){
            return $this
                ->name('sub-aggs')
                ->scope([
                    'avg' => [
                        'field' => 'age'
                    ]
                ])
                ->getMap();
        })
        // you can add multiple sub-aggregations
        ->subAggs(function(){
            ...
        })
        ->search();

// Or you can use like that:
```

(continues on next page)

(continued from previous page)

```php

$aggs = $elastic->aggs()
                ->index('my-index')
                ->name('my-avg-aggs')
                ->terms([
                    'field' => 'age',
                ])
                ->addAggs(function(){
                    return $this
                        ->name('my-avg-aggs-2')
                        ->size(0)
                        ->terms([
                            'field' => 'age',
                        ])
                        ->subAggs(function(){
                            return $this
                                ->name('sub-aggs')
                                ->scope([
                                    'avg' => [
                                        'field' => 'age'
                                    ]
                                ])
                                ->getMap();
                        })
                        ->getMap();
                })
                ->getMap();

return $aggs;
```

CHAPTER

TEN

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.

```php
<?php

$elastic = new ElasticSearch();

// Search using must
$search = $elastic->bool()
                ->index('my_index')
                ->must([
                    ['term' => ['name.keyword' => 'Ŝêro']]
                ])
                ->search();

// add filter
$search = $elastic->bool()
                ->index('my_index')
                ->must([
                    ['term' => ['name.keyword' => 'Ŝêro']]
                ])
                ->filter([
                        ['term' => ['tags' => 'production']]
                ])
                ->minimum_should_match(1)
                ->boost(1)
                ->search();

// must not
                    ...
                    ->mustNot([
                        ['range' => ['gte' => 10, 'lte' => 20]]
                    ]);

// should
                ...
                ->should([
                        ['term' => ['tags' => 'env1']],
                        ['term' => ['tags' => 'deployed']]
                    ])
```

(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;
```

CHAPTER

ELEVEN

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:

```php
<?php


$index = $elastic
                ->index()
                ->name('my-index')
                ->mappings(function(){
                        return $this
                                ->body([
                                    'properties' => [
                                        'location' => [
                                            'type' => 'geo_shape',
                                        ]
                                    ]
                                ])
                                ->getMap();
                })
                ->updateMap();
```

Save a document to search it

```php
<?php

$index = $elastic->document()
                ->index('my-index')
                ->body([
                    'text' => 'Şêro',
                    'location' => [
                        'type' => 'point',
                        'coordinates' => [-77.30, 38.34],
                    ]
                ])
                ->save();
```

Lets do a CIRCLE example:

```php
<?php

$geo = $elastic->geoShape()
                ->index('my-index')
                ->type('circle')
                ->text('Şêro')
                ->coordinates([-77.30, 38.34])
                ->radius('111km')
                ->search();

return $geo;
```

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

CHAPTER

TWELVE

GEO POINT

Fields of type geo_point accept latitude-longitude pairs

Set the index mappings

```php
<?php

$index = $elastic
        ->index()
        ->name('my_index')
        ->mappings(function(){
                return $this
                        ->body([
                            'properties' => [
                                'location' => [
                                    'type' => 'geo_point',
                                ]
                            ]
                        ])
                        ->getMap();
        })
        ->updateMap();
```

Let's add an example data:

```php
<?php

$data = $elastic->document()
                ->index('my-index')
                ->body([
                    'text' => 'Şêro',
                    'location' => [
                        "lat" => 41.12,
                        "lon" => -71.34
                    ]
                ])
                ->save();
```

Searching...

```php
<?php

```

(continues on next page)

(continued from previous page)

```php
$search = $elastic->geoPoint()
                  ->index('my-index')
                  ->topLeft(42, -72)
                  ->bottomRight(40, -74)
                  ->getMap();

return $search;
```

You can search with location function

```php
<?php

$geo = $elastic->geoPoint()
               ->index('my-index')
               ->pointType('geo_distance')
               ->location([
                   'location' => [
                       "top_left" => [
                           "lat" => 42,
                           "lon" => -72
                       ],
                       "bottom_right" => [
                           "lat" => 40,
                           "lon" => -74
                       ]
                   ]
               ])
               ->search();

// you don't need to use pointType(), it will return geo_bounding_box by default

return $geo;
```

CHAPTER

THIRTEEN

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

```php
<?php

$elastic = new ElasticSearch();

$match = $elastic->match()
                    ->index('my-index')
                    ->field([
                            'name' => [
                                'query' => 'Şêro',
                                'fuzziness' => 1
                            ]
                        ])
                    ->search();

// You can merge with other features:

    $match = $elastic->match()
                    ->index('my-index')
                    ->field([
                            'name' => [
                                'query' => 'Şêro',
                                'fuzziness' => 1
                            ]
                        ])
                    ->mergeWith('elastic.sort', function(){
                        return $this->field('age')
                                    ->order('desc')
                                    ->getMap();
                    })
                    ->mergeWith('elastic.highlight', function(){
                        return $this
                                ->content('name', '<div>', '</div>')
                                ->getMap();
                    })
```

(continues on next page)

(continued from previous page)

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

CHAPTER

FOURTEEN

RANGE

Returns documents that contain terms within a provided range.

```php
<?php

$elastic = new ElasticSearch();

$search = $elastic->range()
                ->index('my-index')
                ->content([
                    'age' => [
                        'gte' => 21,
                        'lte' => 26
                    ]
                ])
                ->search();

return $search;
```

CHAPTER

FIFTEEN

MATCH PHRASE

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

A sample search

```php
<?php

$elastic = new ElasticSearch();

$phrase = $elastic->matchPhrase()
                ->index('my-index')
                ->field('name')
                ->text('I use Elastic Search')
                ->addition(['slop' => 1])
                ->search();

return $phrase;
```

CHAPTER

SIXTEEN

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

```php
<?php

$elastic = new ElasticSearch();

$suggest = $elastic->suggest()
                ->index('my-index')
                ->id(1)
                ->input([
                    [
                        'input' => 'My name is Şêro',
                        'weight' => 25
                    ]
                ])
                ->save();
```

Search with suggest

```php
<?php

$suggest = $elastic->suggest()
             ->index('my-index')
             ->name('my-index-suggest')
             ->text('My name is ') // it returns "My name is Şêro"
             ->completion([
                 'field' => 'suggest',
                 'skip_duplicates' => true
             ])
             ->search();

return $suggest;
```

You can use multiple suggestion

```php
<?php

$suggest = $elastic->suggest()
```

(continues on next page)

(continued from previous page)

```
            ->index('my-index')
            ->name('my-index-suggest')
            ->text('My name is ') // it returns "My name is Ŝêro"
            ->completion([
                'field' => 'suggest',
                'skip_duplicates' => true
            ])
            ->addSuggest(function(){
                return $this->index('my-index')
                        ->name('my-index-suggest-2')
                        ->text('Music')
                        ->completion([
                            'field' => 'suggest',
                            'skip_duplicates' => true,
                            'fuzzy' => [
                                'fuzziness' => 1
                            ]
                        ])
                        ->getMap();
            })
            ->addSuggest(function(){
                return $this->index('my-index')
                            ->name('my-index-suggest-3')
                            ->text('Music')
                            ->completion([
                                'field' => 'suggest',
                                'skip_duplicates' => true
                            ])
                            ->getMap();
                })
            ->search();

return $suggest;
```

Other functions you can use for suggestion

```
<?php

$suggest = $elastic->suggest()
                ->index('my-index')
                ->name('product-suggestion')
                ->text('tring out Elasticsearch')
                ->term(['name' => 'message'])
                ->input([ "Nevermind", "Nirvana" ])
                ->weight(34)
                ->prefix('nir')
                ->regex('regex')
                ->completion([
                    'field' => 'suggest',
                    'size'  => 10,
                    'contexts' => [
                        'place_type' => [
```

(continues on next page)

(continued from previous page)

```
                        [
                            'context' => 'cafe'
                        ],
                        [
                            'context'   => 'restaurants',
                            'boost'     => 2
                        ]
                    ]
                ]
            ])
            ->contexts([
                [
                    "name" => "place_type",
                    "type" => "category"
                ],
                [
                    "name" => "location",
                    "type" => "geo",
                    "precision" => 4
                ],
            ])
            ->simplePhrase(function(){
                return $this
                        ->field('name')
                        ->directGenerator([
                            [
                                'field' => 'name',
                                'suggest_mode'  => 'always'
                            ],
                            [
                                "field" => "title.reverse",
                                "suggest_mode" => "always",
                                "pre_filter" => "reverse",
                                "post_filter" => "reverse"
                            ]
                        ])
                        ->highlight([
                            'pre_tag' => "<em>",
                            'post_tag' => "</em>"
                        ])
                        ->collate([
                            'source' => [
                                'match' => [
                                    "{{field_name}}" => "{{suggestion}}"
                                ]
                            ]
                        ])
                        ->smoothing([
                            'laplace' => [
                                'alpha' => 0.7
                            ]
                        ])
```

(continues on next page)

(continued from previous page)

```
                ->size(2)
                ->getMap();
        })
        ->getMap();
```

CHAPTER

SEVENTEEN

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

```php
<?php

$query = $elastic->query()
                 ->cat()
                 ->aliases();

return $query;
```

Searching with query function:

```php
<?php

$elastic = new ElasticSearch();

$result = $elastic->query([
    'index' => 'my-index',
    'body' => [
        'query' => [
            'match' => [
                'name' => [
                    'query' => 'Şêro',
                    'fuzziness' => 1
                ]
            ]
        ]
    ]
])->search();

return $result;
```

CHAPTER

EIGHTEEN

OTHER

18.1 Get Map Function

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

For example:

```php
<?php

$map = $elastic->index()
                ->name('my-index')
                ->mappings(function(){
                        return $this
                                ->body([
                                    'properties' => [
                                        'location.keyword' => [
                                            'type' => 'geo_shape',
                                        ]
                                    ]
                                ])
                                ->getMap();
                })
                ->getMap(); // it returns map

return $map;

// You will get something like that:
/* {"index":"my-index","body":{"mappings":{"properties":{"location.keyword":{"ty pe":
↪"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:

```php
<?php

$sort = $elastic->sort()
            ->index('people')
```

(continues on next page)

(continued from previous page)

```
            ->field('age')
            ->order('desc') // you can change to "asc"
            ->search(['hits', 'took']); // spesific parameters

// You will get hits and took fields

return $sort;
```