elasticmagic Documentation

Release 0.1.0-alpha

Alexander Koval <kovalidis@gmail.com>

Feb 09, 2023

Contents

Contents:

CHAPTER 1

Quick Start

First of all create Elasticsearch cluster and index objects:

```
from elasticsearch import Elasticsearch
from elasticmagic import Cluster, Index

es_cluster = Cluster(Elasticsearch())
es_index = Index(es_cluster, 'test')
```

Let's describe elasticsearch document:

```
from elasticmagic import Document, Field
from elasticmagic.types import String, Integer, Float

class ProductDocument(Document):
    __doc_type__ = 'product'

    name = Field(String, fields={
        'sort': Field(
            String, index='no', doc_values=True, analyzer='keyword'
        ),
    })
    status = Field(Integer)
    price = Field(Float)
```

To create or update document mapping just run:

```
es_index.put_mapping(ProductDocument)
```

Try to reindex some documents:

```
from decimal import Decimal

doc1 = ProductDocument(
    name="Lego Ninjago Cole's dragon",
```

(continues on next page)

(continued from previous page)

```
    status=0,
    price=Decimal('10.99'),
)
doc2 = ProductDocument()
doc2.name = 'Lego minifigure'
doc2.status = 1
doc2.price = Decimal('2.50')
result = es_index.add([doc1, doc2])
assert result.errors == False
```

Now we can build query:

```
search_query = (
    es_index.search_query(ProductDocument.name.match('lego'))
    .filter(ProductDocument.status == 0)
    .order_by(ProductDocument.name.sort)
    .limit(20)
)
```

And finally make request and process result:

```
for doc in search_query:
    print('{}: {}'.format(doc._id, doc.name))
```

Let's build a histogram by price:

```
from elasticmagic import agg

search_query = (
    es_index.search_query()
    .filter(ProductDocument.status == 0)
    .aggs({
        'prices': agg.Histogram(ProductDocument.price, interval=20)
    })
    .limit(0)
)

for bucket in search_query.result.get_aggregation('prices').buckets:
    print('{} ({})'.format(bucket.key, bucket.doc_count))
```

CHAPTER 2

Search Query API

class elasticmagic.search.**SearchQuery**(*q=None*, *cluster=None*, *index=None*, *doc_cls=None*, *doc_type=None*, *routing=None*, *preference=None*, *timeout=None*, *search_type=None*, *query_cache=None*, *terminate_after=None*, *scroll=None*, *stats=None*, *track_total_hits=None*, ***kwargs*)

Elasticsearch search query construction object.

SearchQuery object is usually instantiated by calling `Index.search_query()` method.

See `Index.search_query()` for more details.

to_dict(*compiler=None*)

Compiles the query and returns python dictionary that can be serialized to json.

get_result()

Executes current query and returns processed `SearchResult` object. Caches result so subsequent calls with the same search query will return cached value.

count()

Executes current query and returns number of documents matched the query. Uses count api.

exists()

Executes current query optimized for checking that there are at least 1 matching document. This method is an analogue of the old exists search api For Elasticsearch 5.x and more it emulates the exists API using *size=0* and *terminate_after=1*.

explain(*doc_or_id*, *doc_cls=None*, *routing=None*, ***kwargs*)

Computes a score explanation for the current search query and the document.

Returns `result.ExplainResult`

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-explain.html

delete(*conflicts=None*, *refresh=None*, *timeout=None*, *scroll=None*, *scroll_size=None*, *wait_for_completion=None*, *requests_per_second=None*, ***kwargs*)

Deletes all documents that match the query.

Note: As it uses delete by query api the documents that were changed between the time when a snapshot was taken and when the delete request was processed won't be deleted.

aggregations (**args*, ***kwargs*)

Adds aggregations to the search query.

Parameters ***aggs** – dictionaries with aggregations. Can be `None` that cleans up previous aggregations.

After executing the query you can get aggregation result by its name calling `SearchResult.get_aggregation()` method.

```
from elasticmagic import agg

search_query = SearchQuery().aggregations({
    'stars': agg.Terms(PostDocument.stars, size=50, aggs={
        'profit': agg.Sum(PostDocument.profit)})})
```

```
assert search_query.to_dict(Compiler_5_0) == {
    'aggregations': {
        'stars': {'terms': {'field': 'stars', 'size': 50},
            'aggregations': {
                'profit': {'sum': {'field': 'profit'}}}}}}
```

aggs (**args*, ***kwargs*)

A shortcut for the *aggregations()* method

boost_score (**args*, ***kwargs*)

Adds one more level of the function_score query with default sum modes. It is especially useful for complex ordering scenarios.

Parameters ***functions** – See *function_score()*

```
from elasticmagic import Factor, ScriptScore, Script

search_query = (
    SearchQuery(PostDocument.title.match('test'))
    .function_score(
        # Slightly boost hits on post popularity
        Factor(PostDocument.popularity, modifier='sqrt'))
    .boost_score(
        # Display advertized posts higher than any others
        ScriptScore(
            Script(inline='log10(10.0 + doc[cpc_field].value)',
                   params={'cpc_field': PostDocument.adv_cpc}),
            weight=1000,
            filter=PostDocument.adv_cpc > 0))
)
```

```
assert search_query.to_dict(Compiler_5_0) == {
    'query': {
        'function_score': {
            'query': {
                'function_score': {
                    'query': {'match': {'title': 'test'}},
                    'functions': [
```

(continues on next page)

(continued from previous page)

```
                        {'field_value_factor': {
                            'field': 'popularity',
                            'modifier': 'sqrt'}}]}},
            'functions': [
                {'script_score': {'script': {
                     'inline': 'log10(10.0 + doc[cpc_field].value)',
                     'params': {'cpc_field': 'adv_cpc'}}},
                 'filter': {'range': {'adv_cpc': {'gt': 0}}},
                 'weight': 1000}],
            'boost_mode': 'sum',
            'score_mode': 'sum'}}}
```

clone()
Clones this query so you can modify both queries independently.

docvalue_fields(**fields*)
Allows to load doc values fields. Marked in mapping as `doc_values: true` (turned on by default).

Example:

```
search_query = SearchQuery().docvalue_fields(PostDocument.rank)
```

```
assert search_query.to_dict(Compiler_7_0) == {'docvalue_fields': ['rank']}
```

See stored fields and stored fields filtering for more information.

filter(**filters, **kwargs*)
Adds a filters into elasticsearch filter context.

Returns new *SearchQuery* object with applied filters.

```
search_query = SearchQuery().filter(
    PostDocument.status == 'published',
    PostDocument.publish_date >= datetime.date(2015, 1, 1),
)
```

```
assert search_query.to_dict(Compiler_5_0) == {
    'query': {'bool': {'filter': [
        {'term': {'status': 'published'}},
        {'range': {
            'publish_date': {
                'gte': datetime.date(2015, 1, 1)}}}]}}}
```

Filter expression can be a python dictionary object:

```
search_query = SearchQuery().filter({'term': {'status': 'published'}})
```

from_(*offset*)
Sets the offset - the number of hits to skip. Used for pagination.

See from / size

function_score(**args, **kwargs*)
Adds function scores to the search query.

Parameters ***functions** – list of function scores.

```
from elasticmagic import Weight, FieldValueFactor

search_query = (
    SearchQuery(PostDocument.title.match('test'))
    .function_score(
        Weight(2, filter=PostDocument.created_date == 'now/d'),
        FieldValueFactor(
            PostDocument.popularity, factor=1.2, modifier='sqrt'
        )
    )
)
```

```
assert search_query.to_dict(Compiler_5_0) == {
    'query': {
        'function_score': {
            'query': {'match': {'title': 'test'}},
            'functions': [
                {'weight': 2,
                 'filter': {'term': {'created_date': 'now/d'}}},
                {'field_value_factor': {
                    'field': 'popularity',
                    'factor': 1.2,
                     'modifier': 'sqrt'}}]}}}
```

function_score_settings (**function_score_settings*)
Adds or updates function score level. Levels will be inserted before existing.

highlight (*fields=None, type=None, pre_tags=None, post_tags=None, fragment_size=None, number_of_fragments=None, order=None, encoder=None, require_field_match=None, boundary_max_scan=None, highlight_query=None, matched_fields=None, fragment_offset=None, no_match_size=None, phrase_limit=None, **kwargs*)
Highlights search results.

```
from elasticmagic import MultiMatch

search_query = (
    SearchQuery(
        MultiMatch('The quick brown fox',
                   [PostDocument.title, PostDocument.content])
    )
    .highlight([PostDocument.title, PostDocument.content])
)
```

When processing search result you can get hit highlight by calling `Document.get_highlight()`.

See highlighting for details.

limit (*limit*)
Sets size of the maximum amount of hits. Used for pagination.

See from / size

min_score (*min_score*)
Excludes hits with a `_score` less then `min_score`. See min score

offset (*offset*)
Sets the offset - the number of hits to skip. Used for pagination.

See from / size

order_by (**orders*)
Apply sorting criterion to the search query. Corresponds elasticsearch's sort.

```
search_query = SearchQuery().order_by(
    PostDocument.publish_date.desc(),
    PostDocument._score,
)
assert search_query.to_dict(Compiler_5_0) == {
    'sort': [
        {'publish_date': 'desc'},
        '_score'
    ]
}
```

When called with single `None` argument clears any sorting criterion applied before:

```
search_query = SearchQuery().order_by(None)
assert search_query.to_dict(Compiler_5_0) == {}
```

post_filter (**filters*, ***kwargs*)
Adds a filters into elasticsearch post filter context.

All parameters have the same meaning as for `filter()` method.

query (*q*)
Replaces query clause. Elasticsearch's query clause will calculate `_score` for every matching document.

Parameters q – query expression. Existing query can be cancelled by passing `None`.

```
search_query = SearchQuery().query(
    PostDocument.title.match('test', minimum_should_match='100%'))
```

```
assert search_query.to_dict(Compiler_5_0) == {
    'query': {'match': {'title': {
        'query': 'test',
        'minimum_should_match': '100%'}}}}
```

rescore (*rescorer*, *window_size=None*)
Adds a rescorer for the query. See rescoring

script_fields (**args*, ***kwargs*)
Allows to evaluate fields based on scripts.

```
from elasticmagic import Script

search_query = SearchQuery().script_fields(
    rating=Script(
        inline='doc[params.positive_opinions_field].value / '
               'doc[params.total_opinions_field].value * 5',
        params={
            'total_opinions_field': PostDocument.total_opinions,
            'positive_opinions_field': PostDocument.positive_opinions,
        }
    )
)
```

```
expected = {
    'script_fields': {
```

(continues on next page)

(continued from previous page)

```
        'rating': {
            'script': {
                'inline': 'doc[params.positive_opinions_field].value / '
                          'doc[params.total_opinions_field].value * 5',
                'params': {
                    'total_opinions_field': 'total_opinions',
                    'positive_opinions_field': 'positive_opinions'
                }
            }
        }
    }
}

assert search_query.to_dict(Compiler_5_0) == {
    'script_fields': {
        'rating': {
            'script': {
                'inline': 'doc[params.positive_opinions_field].value / '
                          'doc[params.total_opinions_field].value * 5',
                'params': {
                    'total_opinions_field': 'total_opinions',
                    'positive_opinions_field': 'positive_opinions'
                }
            }
        }
    }
}
```

See script fields

size (*limit*)
Sets size of the maximum amount of hits. Used for pagination.

See from / size

slice (*offset*, *limit*)
Applies offset and limit to the query.

source (**fields*, ***kwargs*)
Controls which fields of the document's `_source` field to retrieve.

Parameters

- ***fields** – list of fields which should be returned by elasticsearch. Can be one of the following types:
  - field expression, for example: `PostDocument.title`
  - `str` means field name or glob pattern. For example: `"title"`, `"user.*"`
  - `False` disables retrieving source
  - `True` enables retrieving all source document
  - `None` cancels source filtering applied before
- **include** – list of fields to include
- **exclude** – list of fields to exclude

Example:

```
search_query = SearchQuery().source(PostDocument.title, 'user.*')
```

```
assert search_query.to_dict(Compiler_5_0) == {'_source': ['title', 'user.*']}
```

See source filtering for more information.

stored_fields (**fields*)

Allows to load fields that marked as `store: true`.

Example:

```
search_query = SearchQuery().stored_fields(PostDocument.rank)
```

```
assert search_query.to_dict(Compiler_5_0) == {'stored_fields': ['rank']}
```

See stored fields and stored fields filtering for more information.

suggest (**args, **kwargs*)

Adds suggesters to the query

class elasticmagic.ext.asyncio.search.**AsyncSearchQuery** (*q=None, cluster=None, index=None, doc_cls=None, doc_type=None, routing=None, preference=None, timeout=None, search_type=None, query_cache=None, terminate_after=None, scroll=None, stats=None, track_total_hits=None, **kwargs*)

Asynchronous version of the `SearchQuery`

CHAPTER 3

Aggregations

class elasticmagic.agg.**Min**(*field=None*, *script=None*, ***kwargs*)

A single-value metric aggregation that returns the minimum value among all extracted numeric values. See min agg.

```
search_query = search_query.aggs({
    'min_price': agg.Min(SaleDocument.price)
})
assert search_query.to_dict() == {
    'aggregations': {
        'min_price': {'min': {'field': 'price'}}}}
min_price_agg = search_query.get_result().get_aggregation('min_price')
print(min_price_agg.value)
print(min_price_agg.value_as_string)
```

```
10.0
10.0
```

class elasticmagic.agg.**Max**(*field=None*, *script=None*, ***kwargs*)

A single-value metric aggregation that returns the maximum value among all extracted numeric values. See max agg.

class elasticmagic.agg.**Sum**(*field=None*, *script=None*, ***kwargs*)

A single-value metric aggregation that sums up all extracted numeric values. See sum agg.

```
search_query = search_query.aggs({
    'prices': agg.Sum(SaleDocument.price)
})
assert search_query.to_dict() == {
    'aggregations': {
        'prices': {'sum': {'field': 'price'}}}}
prices_agg = search_query.get_result().get_aggregation('prices')
print(prices_agg.value)
print(prices_agg.value_as_string)
```

```
450.0
450.0
```

class elasticmagic.agg.**Avg**(*field=None*, *script=None*, ***kwargs*)
A single-value metric aggregation that computes average of all extracted numeric values. See avg agg.

class elasticmagic.agg.**ValueCount**(*field=None*, *script=None*, ***kwargs*)
A single-value metric aggregation that counts the number of all extracted values. See value_count agg.

```
from elasticmagic import Script

search_query = search_query.aggs({
    'types_count': agg.ValueCount(script=Script(
        inline='doc[params.field].value',
        params={'field': SaleDocument.type}
    ))
})
assert search_query.to_dict() == {
    'aggregations': {
        'types_count': {
            'value_count': {
                'script': {
                    'inline': 'doc[params.field].value',
                    'params': {'field': 'type'}}}}}}
print(search_query.get_result().get_aggregation('types_count').value)
```

```
7
```

class elasticmagic.agg.**TopHits**(*size=None*, *from_=None*, *sort=None*, *_source=None*, *instance_mapper=None*, ***kwargs*)
A *top_hits* metric aggregation that groups result set by certain fields via a bucket aggregator. See top_hits agg.

```
search_query = search_query.aggs({
    'top_tags': agg.Terms(
        SaleDocument.type, size=3,
        aggs={'top_sales_hits': agg.TopHits(
            size=1,
            sort=SaleDocument.date.desc(),
            _source={
                'includes': [SaleDocument.date, SaleDocument.price]
            }
        )}
    )
})
assert search_query.to_dict() == {
    'aggregations': {
        'top_tags': {
            'terms': {'field': 'type', 'size': 3},
            'aggregations': {
                'top_sales_hits': {
                    'top_hits': {
                        'size': 1,
                        'sort': {'date': 'desc'},
                        '_source': {'includes': ['date', 'price']}}}}}}}
top_tags = search_query.get_result().get_aggregation('top_tags')
for tag_bucket in top_tags.buckets:
    top_hit = tag_bucket.get_aggregation('top_sales_hits').hits[0]
```

(continues on next page)

(continued from previous page)

```
    print(
        '{0.key} ({0.doc_count}) - {1.price}'.format(
            tag_bucket, top_hit
        )
    )
```

```
hat (3) - 200
t-shirt (3) - 175
bag (1) - 150
```

class elasticmagic.agg.**Stats**(*field=None*, *script=None*, ***kwargs*)

A multi-value metrics aggregation that computes stats over all extracted numeric values. See stats agg.

```
search_query = search_query.aggs({
    'grades_stats': agg.Stats(GradeDocument.grade)
})
assert search_query.to_dict() == {
    'aggregations': {
        'grades_stats': {'stats': {'field': 'grade'}}}}
grades_stats = search_query.get_result().get_aggregation('grades_stats')
print('count:', grades_stats.count)
print('min:', grades_stats.min)
print('max:', grades_stats.max)
print('avg:', grades_stats.avg)
print('sum:', grades_stats.sum)
```

```
count: 6
min: 60
max: 98
avg: 78.5
sum: 471
```

class elasticmagic.agg.**ExtendedStats**(*field=None*, *script=None*, ***kwargs*)

A multi-value metrics aggregation that computes stats over all extracted numeric values. See extended_stats agg.

This aggregation is an extended version of the `Stats` aggregation. There are some additional metrics: *sum_of_squares*, *variance*, *std_deviation*.

class elasticmagic.agg.**Percentiles**(*field=None*, *script=None*, *percents=None*, *compression=None*, ***kwargs*)

A multi-value metrics aggregation that calculates percentiles over all extracted numeric values. See percentiles agg.

Note: Percentiles are usually calculated approximately. Elasticsearch calculates them using TDigest algotighm.

```
search_query = search_query.aggs(
    load_time_outlier=agg.Percentiles(field=PageLoadDoc.load_time)
)
assert search_query.to_dict() == {
    'aggregations': {
        'load_time_outlier': {
            'percentiles': {'field': 'load_time'}}}}
load_time_agg = search_query.get_result()            .get_aggregation('load_time_
↪outlier')
```

(continues on next page)

(continued from previous page)

```
for p, v in load_time_agg.values[:-1]:
    print('{:<4} - {}'.format(p, v))
print('99 percentile is: {}'.format(load_time_agg.get_value(99)))
```

```
1.0  - 15.0
5.0  - 20.0
25.0 - 23.0
50.0 - 25.0
75.0 - 29.0
95.0 - 60.0
99 percentile is: 150.0
```

class elasticmagic.agg.**PercentileRanks**(*field=None*, *script=None*, *values=None*, *compression=None*, ***kwargs*)

A multi-value metrics aggregation that calculates percentile ranks over all extracted numeric values. See percentile_ranks agg.

```
search_query = search_query.aggs(
    load_time_outlier=agg.PercentileRanks(
        field=PageLoadDoc.load_time,
        values=[15, 30],
    )
)
assert search_query.to_dict() == {
    'aggregations': {
        'load_time_outlier': {
            'percentile_ranks': {
                'field': 'load_time',
                'values': [15, 30]}}}}
load_time_agg = search_query.get_result()             .get_aggregation('load_time_
↪outlier')
for v, p in load_time_agg.values:
    print('{:<4} - {}'.format(v, p))
print(
    '{}% of values are below 15'.format(load_time_agg.get_percent(15))
)
```

```
15.0 - 92.0
30.0 - 100.0
92.0% of values are below 15
```

class elasticmagic.agg.**Cardinality**(*field=None*, *script=None*, *precision_threshold=None*, *rehash=None*, ***kwargs*)

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

CHAPTER 4

Indices and tables

- genindex

Python Module Index

e

Index

S

T

V