Intro to Elasticsearch and Kibana

Table of contents

Elasticsearch and Kibana

Elasticsearch

Elasticsearch is a distributed, RESTful search and analytics engine capable of addressing a growing number of use cases. As the heart of the Elastic Stack, it centrally stores your data for lightning fast search, fine-tuned relevancy, and powerful analytics that scale with ease.

Kibana

Kibana is a free and open user interface that lets you visualize your Elasticsearch data and navigate the Elastic Stack. Do anything from tracking query load to understanding the way requests flow through your apps.

Downloading Elasticsearch and Kibana (macOS/Linux and Windows)

Step 1: The following links will take you to the download pages for Elasticsearch and Kibana.

- https://www.elastic.co/downloads/elasticsearch
- https://www.elastic.co/downloads/kibana

Step 2: Relocate downloaded Elasticsearch and unzip Elasticsearch.
Step 3: Relocate downloaded Kibana and unzip Kibana.
Step 4: Start the Elasticsearch/Kibana server and ensure that everything is working properly

- Go to the command prompt (Terminal).
- Go to the installation folder where you have unzipped the file.
- Run "bin\elasticsearch. bat"("bin/elasticsearch" On Linux/macOs).
- Elasticsearch has now started.
- Run "\bin\kibana. bat"("bin/kibana" On Linux/macOs).
- Kibana has now started.
- Open your browser and go to "https://localhost:5601/app/dev_tools#console"

Who uses Elasticsearch?

SAMSUNG docker shopify

UBER Adobe twilio tinder

Udemy instacart

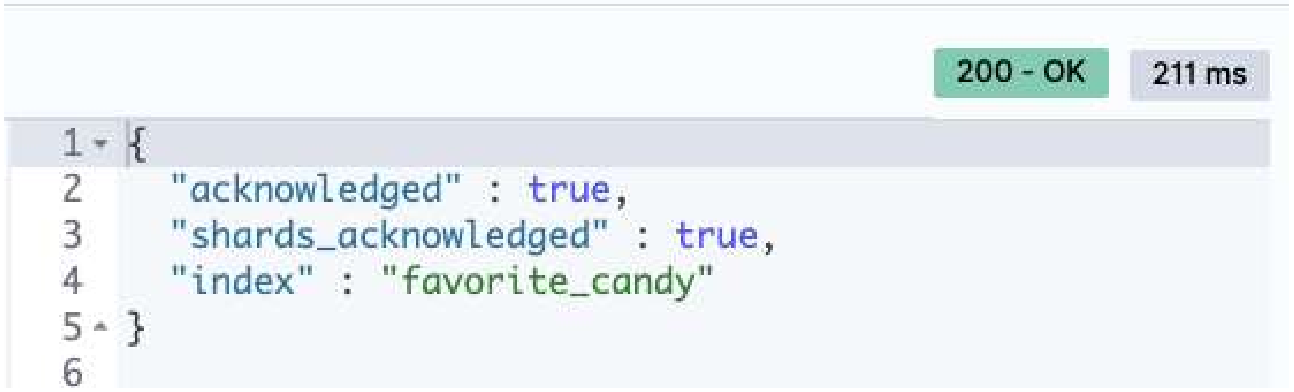
Create an index

Syntax:

PUT Name-of-the-Index

Example:

PUT favorite_candy



```
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "favorite_candy"
}

```

Index a document

When indexing a document, both HTTP verbs POST or PUT can be used.

- *Use POST when you want Elasticsearch to autogenerate an id for your document.*
- *Use PUT when you want to assign a specific id to your document(i.e. if your document has a natural identifier - purchase order number, patient id, & etc). For more detailed explanation, check out this documentation from Elastic*

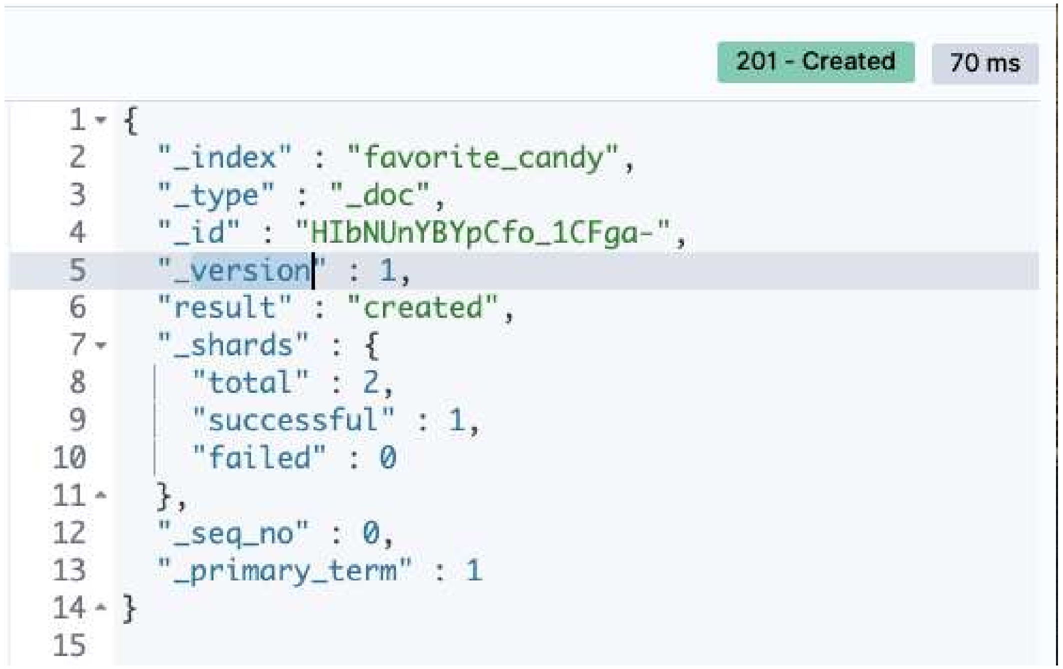
Index a document using POST

Syntax:

```
POST Name-of-the-Index/_doc
{
 "field": "value"
}
```

Example:

```
POST favorite_candy/_doc
{
 "first_name": "Lisa",
 "candy": "Sour Skittles"
}
```



```
{
  "_index" : "favorite_candy",
  "_type" : "_doc",
  "_id" : "HIbNUnYBYpCfo_1CFga-",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

Index a document using PUT

Syntax:

```
PUT Name-of-the-Index/_doc/id-you-want-to-assign-to-this-document
{
 "field": "value"
}
```

Example:

```
PUT favorite_candy/_doc/1
{
 "first_name": "John",
 "candy": "Starburst"
}
```

Create Endpoint

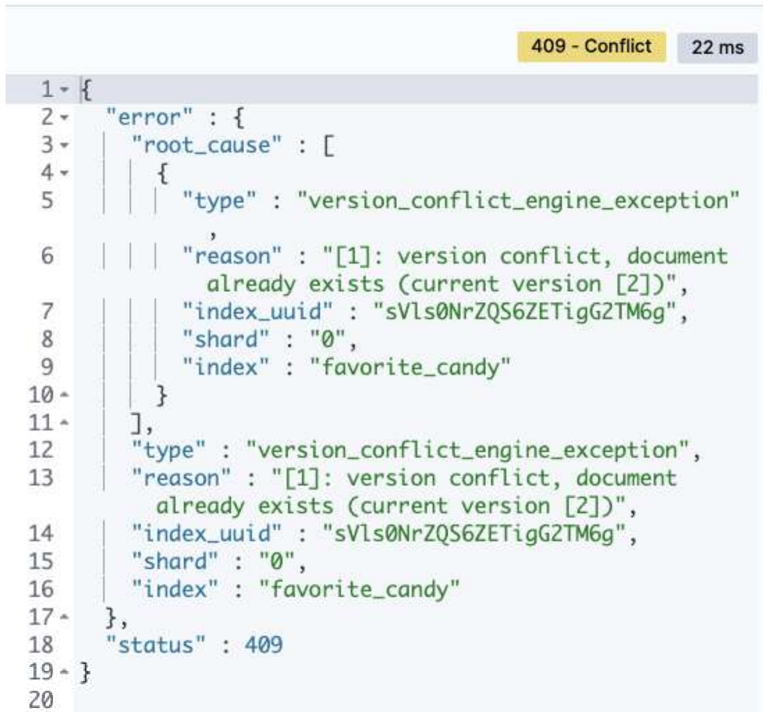
When you index a document using an id that already exists, the existing document is overwritten by the new document. If you do not want a existing document to be overwritten, you can use the _create endpoint! With the _create Endpoint, no indexing will occur and you will get a 409 error message

Syntax:

```
PUT Name-of-the-Index/_create/id
{
 "field": "value"
}
```

Example:

```
PUT favorite_candy/_create/1
{
 "first_name": "Finn",
 "candy": "Jolly Ranchers"
}
```



409 - Conflict 22 ms

```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "version_conflict_engine_exception"
          ,
        "reason" : "[1]: version conflict, document
          already exists (current version [2])",
        "index_uuid" : "sVls0NrZQS6ZETigG2TM6g",
        "shard" : "0",
        "index" : "favorite_candy"
      }
    ],
    "type" : "version_conflict_engine_exception",
    "reason" : "[1]: version conflict, document
      already exists (current version [2])",
    "index_uuid" : "sVls0NrZQS6ZETigG2TM6g",
    "shard" : "0",
    "index" : "favorite_candy"
  },
  "status" : 409
}
```

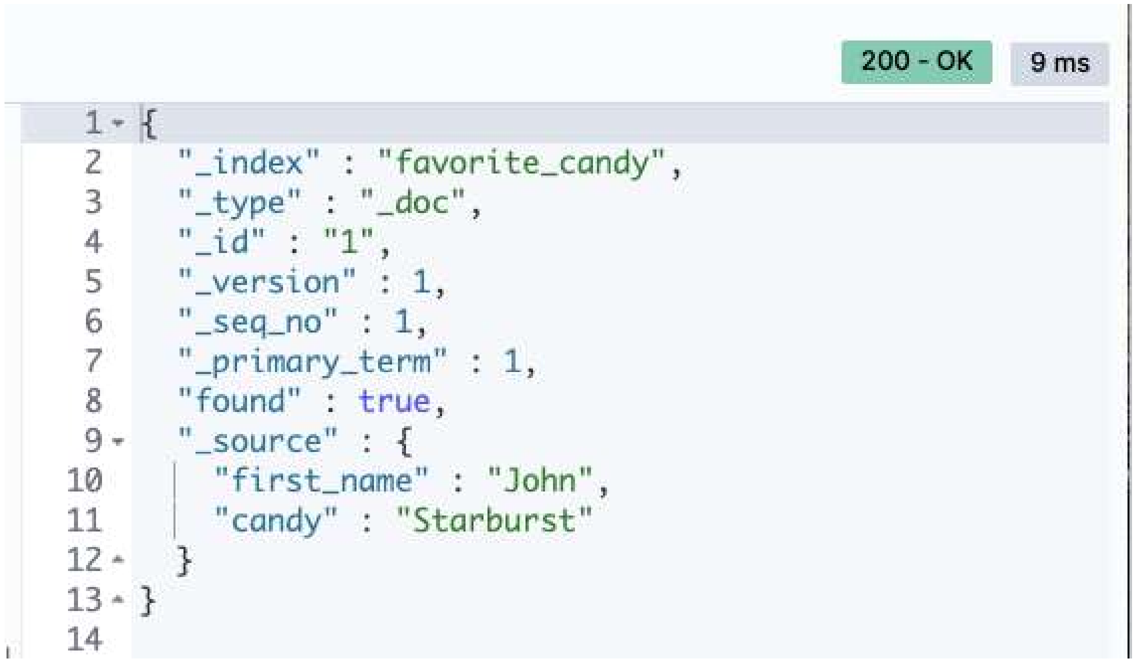
Read a document

Syntax:

GET Name-of-the-Index/_doc/id

Example:

GET favorite_candy/_doc/1



```
{
  "_index" : "favorite_candy",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 1,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "first_name" : "John",
    "candy" : "Starburst"
  }
}
```

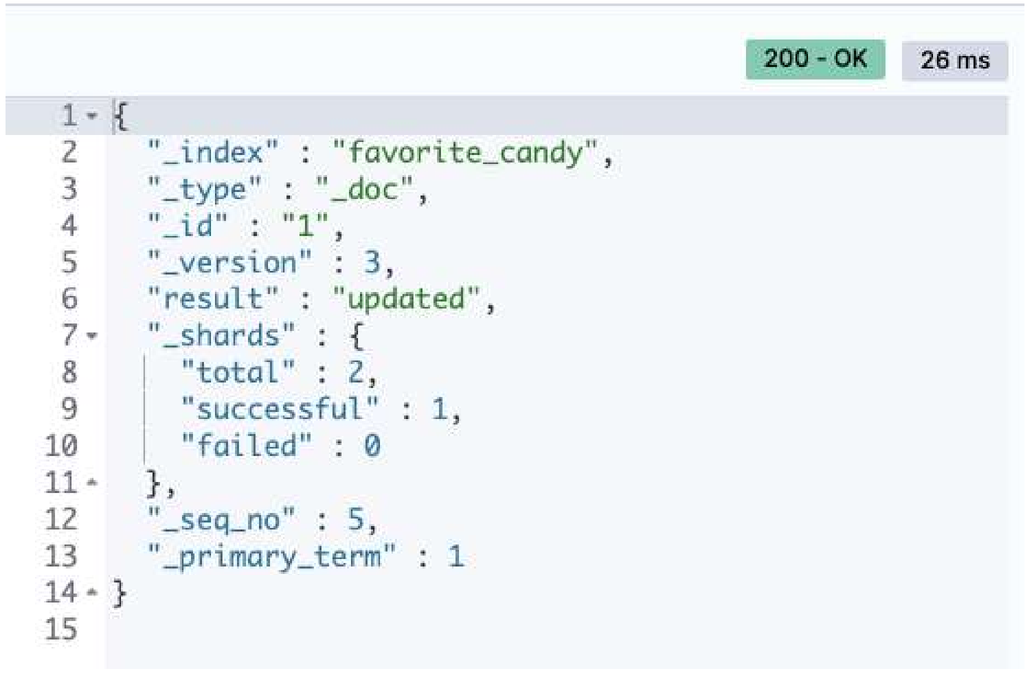
Update a document

Syntax:

```
POST Name-of-the-Index/_update/id
{
 "doc": {
  "field1": "value",
  "field2": "value",
 }
}
```

Example:

```
POST favorite_candy/_update/1
{
 "doc": {
  "candy": "M&M's"
 }
}
```



```
{
  "_index" : "favorite_candy",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 3,
  "result" : "updated",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 5,
  "_primary_term" : 1
}
```

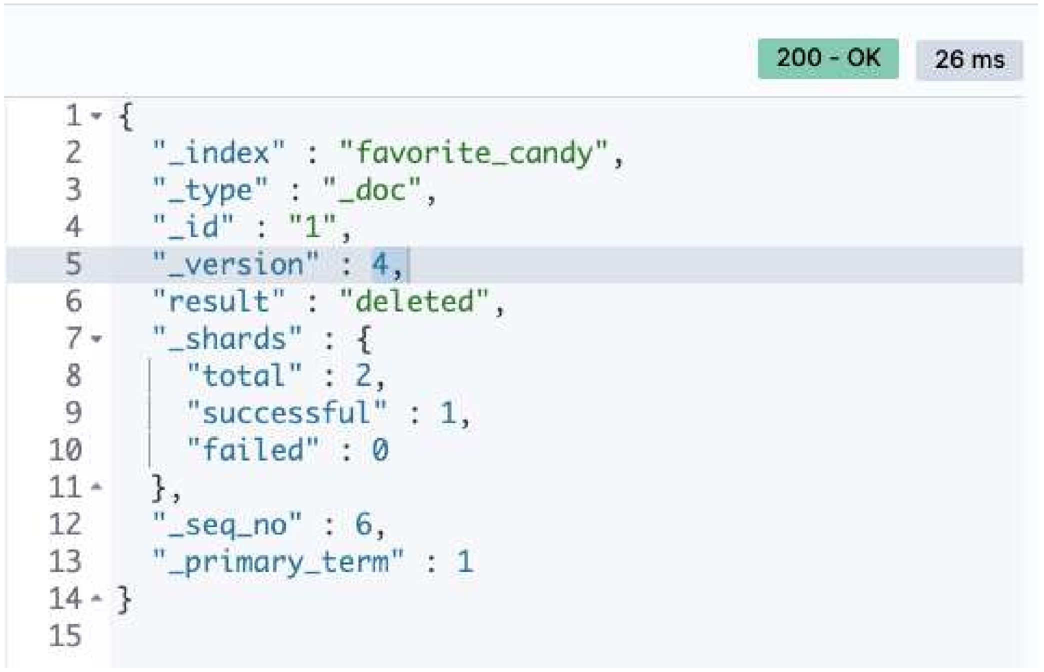
Delete a document

Syntax:

DELETE Name-of-the-Index/_doc/id

Example:

DELETE favorite_candy/_doc/1



```
{
  "_index" : "favorite_candy",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 4,
  "result" : "deleted",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 6,
  "_primary_term" : 1
}
```

Searching for search terms

The match query is a standard query for performing a full text search. This query retrieves documents that contain the search terms. It uses "OR" logic by default, meaning that it will retrieve documents that contain any one of the search terms. The order and the proximity in which the search terms are found(i.e. phrases) are not taken into account

Syntax:

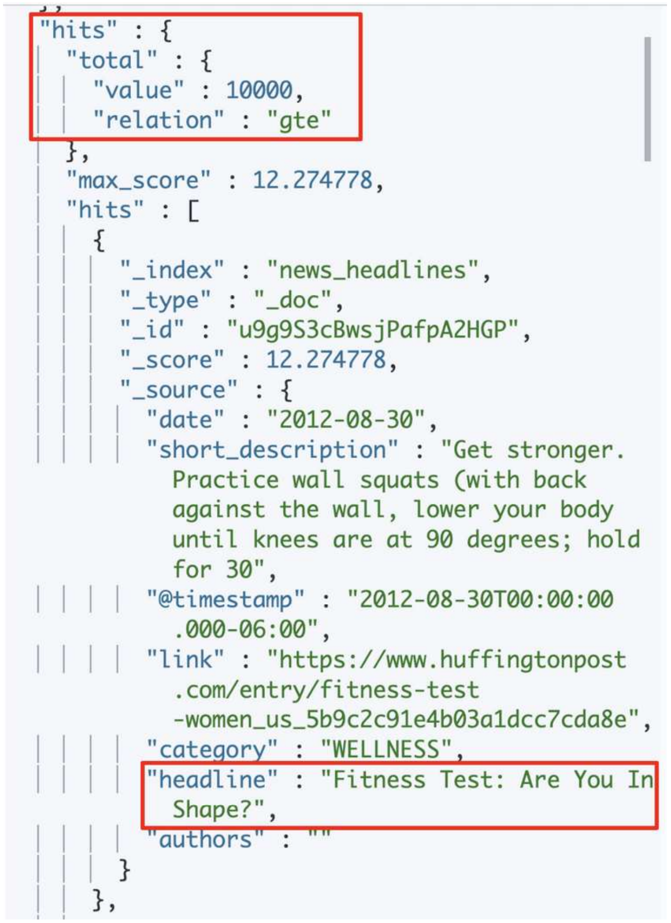
```
GET Enter_name_of_index_here/_search
{
 "query": {
  "match": {
   "Specify the field you want to search": {
    "query": "Enter search terms"
   }
  }
 }
}
```

Searching for search terms

What happens when you use the match query to search for phrases?
Let's search for articles about Ed Sheeran's song "Shape of you" using the match query.

Example:

```
GET news_headlines/_search
{
 "query": {
  "match": {
   "headline": {
    "query": "Shape of you"
   }
  }
 }
}
```



```
"hits" : {
  "total" : {
    "value" : 10000,
    "relation" : "gte"
  },
  "max_score" : 12.274778,
  "hits" : [
    {
      "_index" : "news_headlines",
      "_type" : "_doc",
      "_id" : "u9g9S3cBwsjPafpA2HGP",
      "_score" : 12.274778,
      "_source" : {
        "date" : "2012-08-30",
        "short_description" : "Get stronger.
          Practice wall squats (with back
          against the wall, lower your body
          until knees are at 90 degrees; hold
          for 30",
        "@timestamp" : "2012-08-30T00:00:00
          .000-06:00",
        "link" : "https://www.huffingtonpost
          .com/entry/fitness-test
          -women_us_5b9c2c91e4b03a1dcc7cda8e",
        "category" : "WELLNESS",
        "headline" : "Fitness Test: Are You In
          Shape?",
        "authors" : ""
      }
    },
```

Searching for a phrase

If the order and the proximity in which the search terms are found(i.e. phrases) are important in determining the relevance of your search, you use the match_phrase query.

Syntax:

```
GET Enter_name_of_index_here/_search
{
 "query": {
  "match_phrase": {
   "Specify the field you want to search": {
    "query": "Enter search terms"
   }
  }
 }
}
```

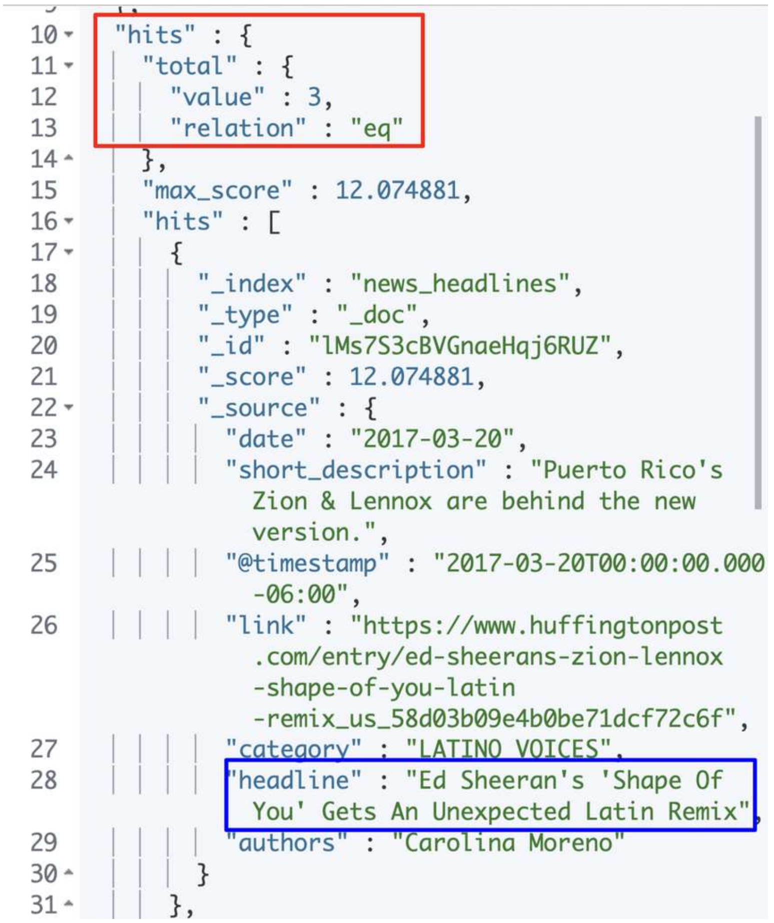
Searching for a phrase

Example:

```
GET news_headlines/_search
{
 "query": {
  "match_phrase": {
   "headline": {
    "query": "Shape of You"
   }
  }
 }
}
```

When the match_phrase parameter is used, all hits must meet the following criteria:

1. *the search terms "Shape", "of", and "you" must appear in the field headline .*
2. *the terms must appear in that order.*
3. *the terms must appear next to each other.*



```
"hits" : {
  "total" : {
    "value" : 3,
    "relation" : "eq"
  },
  "max_score" : 12.074881,
  "hits" : [
    {
      "_index" : "news_headlines",
      "_type" : "_doc",
      "_id" : "lMs7S3cBVGnaeHqj6RUZ",
      "_score" : 12.074881,
      "_source" : {
        "date" : "2017-03-20",
        "short_description" : "Puerto Rico's
          Zion & Lennox are behind the new
          version.",
        "@timestamp" : "2017-03-20T00:00:00.000
          -06:00",
        "link" : "https://www.huffingtonpost
          .com/entry/ed-sheerans-zion-lennox
          -shape-of-you-latin
          -remix_us_58d03b09e4b0be71dcf72c6f",
        "category" : "LATINO VOICES",
        "headline" : "Ed Sheeran's 'Shape Of
          You' Gets An Unexpected Latin Remix",
        "authors" : "Carolina Moreno"
      }
    },
```

Running a match query against multiple fields

To accommodate these contexts, you can write a multi_match query, which searches for terms in multiple fields.
The multi_match query runs a match query on multiple fields and calculates a score for each field. Then, it assigns the highest score among the fields to the document.
This score will determine the ranking of the document within the search results.

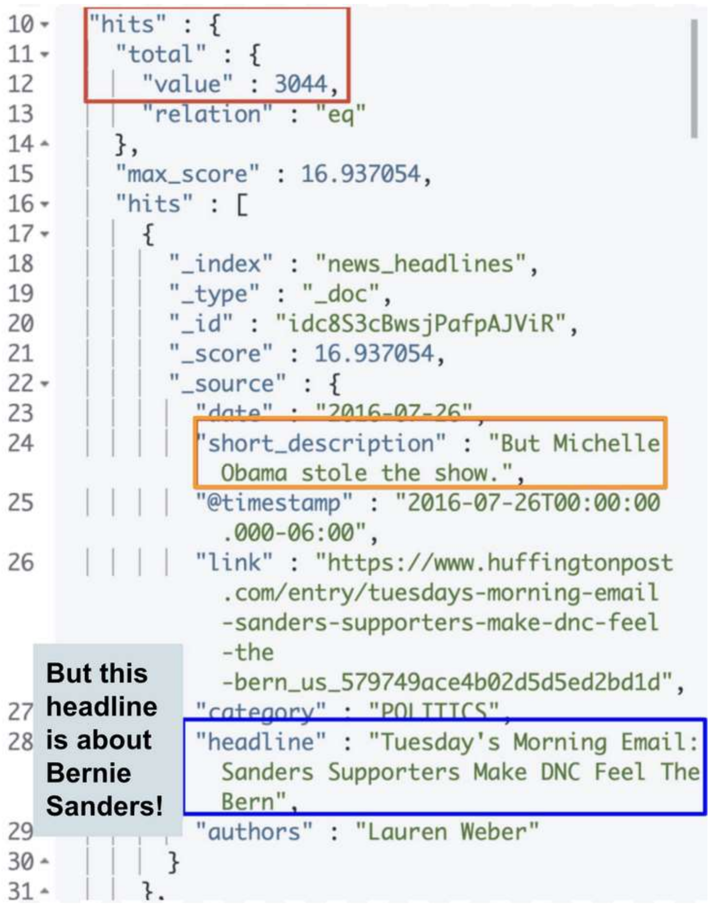
Syntax:

```
GET Enter_the_name_of_the_index_here/_search
{
 "query": {
  "multi_match": {
   "query": "Enter search terms here",
   "fields": [
    "List the field you want to search over",
    "List the field you want to search over",
    "List the field you want to search over"
   ]
  }
 }
}
```

Running a match query against multiple fields

Example:

```
GET news_headlines/_search
{
 "query": {
  "multi_match": {
   "query": "Michelle Obama",
   "fields": [
    "headline",
    "short_description",
    "authors"
   ]
  }
 }
}
```



```
"hits" : {
  "total" : {
    "value" : 3044,
    "relation" : "eq"
  },
  "max_score" : 16.937054,
  "hits" : [
    {
      "_index" : "news_headlines",
      "_type" : "_doc",
      "_id" : "idc8S3cBwsjPafpAJViR",
      "_score" : 16.937054,
      "_source" : {
        "date" : "2016-07-26",
        "short_description" : "But Michelle
          Obama stole the show.",
        "@timestamp" : "2016-07-26T00:00:00
          .000-06:00",
        "link" : "https://www.huffingtonpost
          .com/entry/tuesdays-morning-email
          -sanders-supporters-make-dnc-feel
          -the
          -bern_us_579749ace4b02d5d5ed2bd1d",
        "category" : "POLITICS",
        "headline" : "Tuesday's Morning Email:
          Sanders Supporters Make DNC Feel The
          Bern",
        "authors" : "Lauren Weber"
      }
    },
```

But this headline is about Bernie Sanders!

Per-field boosting

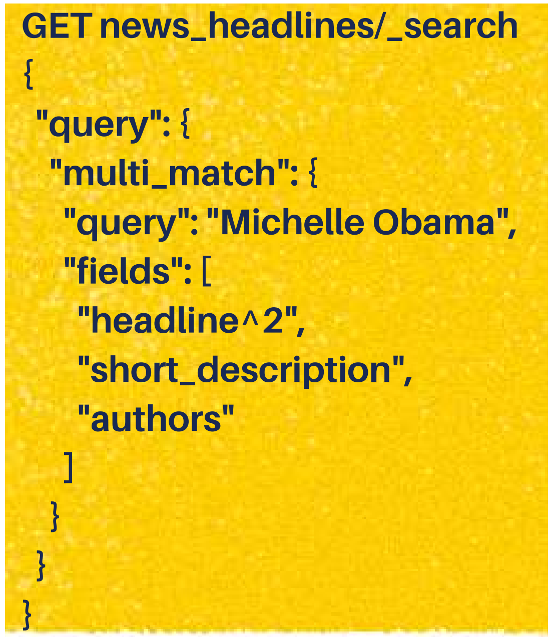
To improve the precision of your search, you can designate one field to carry more weight than the others.
This can be done by boosting the score of the field headline(per-field boosting). This is notated by adding a carat(^) symbol and number 2 to the desired field as shown below.

Syntax:

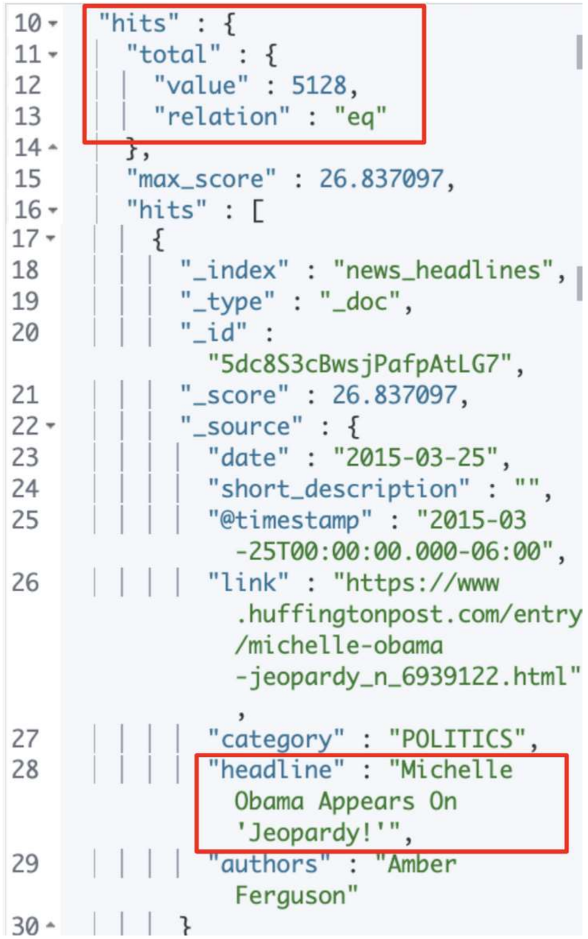
```
GET Enter_the_name_of_the_index_here/_search
{
 "query": {
  "multi_match": {
   "query": "Enter search terms",
   "fields": [
    "List field you want to boost^2",
    "List field you want to search over",
    "List field you want to search over"
   ]
  }
 }
}
```

Per-field boosting

Example:



```
GET news_headlines/_search
{
 "query": {
  "multi_match": {
   "query": "Michelle Obama",
   "fields": [
    "headline^2",
    "short_description",
    "authors"
   ]
  }
 }
}
```

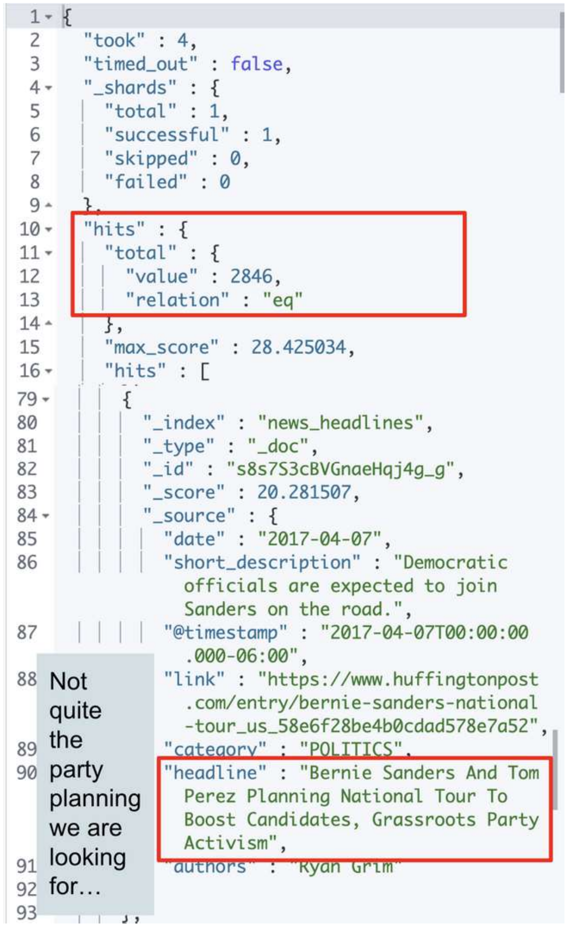


```
"hits" : {
  "total" : {
    "value" : 5128,
    "relation" : "eq"
  },
  "max_score" : 26.837097,
  "hits" : [
    {
      "_index" : "news_headlines",
      "_type" : "_doc",
      "_id" :
        "5dc8S3cBwsjPafpAtLG7",
      "_score" : 26.837097,
      "_source" : {
        "date" : "2015-03-25",
        "short_description" : "",
        "@timestamp" : "2015-03
          -25T00:00:00.000-06:00",
        "link" : "https://www
          .huffingtonpost.com/entry
          /michelle-obama
          -jeopardy_n_6939122.html"
          ,
        "category" : "POLITICS",
        "headline" : "Michelle
          Obama Appears On
          'Jeopardy!'",
        "authors" : "Amber
          Ferguson"
      }
```

What happens when you use the multi_match query to search for a phrase?

Example:

```
GET news_headlines/_search
{
 "query": {
  "multi_match": {
   "query": "party planning",
   "fields": [
    "headline^2",
    "short_description"
   ]
  }
 }
}
```



```
{
  "took" : 4,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 2846,
      "relation" : "eq"
    },
    "max_score" : 28.425034,
    "hits" : [
      {
        "_index" : "news_headlines",
        "_type" : "_doc",
        "_id" : "s8s7S3cBVGnaeHqj4g_g",
        "_score" : 20.281507,
        "_source" : {
          "date" : "2017-04-07",
          "short_description" : "Democratic
            officials are expected to join
            Sanders on the road.",
          "@timestamp" : "2017-04-07T00:00:00
            .000-06:00",
          "link" : "https://www.huffingtonpost
            .com/entry/bernie-sanders-national
            -tour_us_58e6f28be4b0cdad578e7a52",
          "category" : "POLITICS",
          "headline" : "Bernie Sanders And Tom
            Perez Planning National Tour To
            Boost Candidates, Grassroots Party
            Activism",
          "authors" : "Ryan Grim"
        }
      },
```

Not quite the party planning we are looking for…

Improving precision with phrase type match

You can improve the precision of a multi_match query by adding the "type":"phrase" to the query. The phrase type performs a match_phrase query on each field and calculates a score for each field. Then, it assigns the highest score among the fields to the document

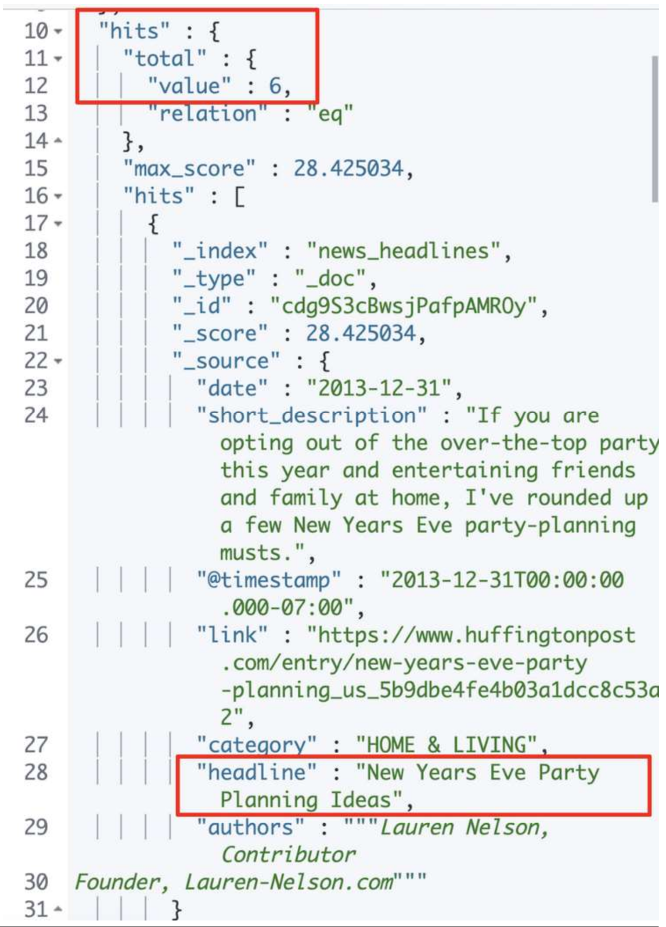
Syntax:

```
GET Enter_the_name_of_the_index_here/_search
{
 "query": {
  "multi_match": {
   "query": "Enter search phrase",
   "fields": [
    "List field you want to boost^2",
    "List field you want to search over",
    "List field you want to search over"
   ],
   "type": "phrase"
  }
 }
}
```

Improving precision with phrase type match

Example:

```
GET news_headlines/_search
{
 "query": {
  "multi_match": {
   "query": "party planning",
   "fields": [
    "headline^2",
    "short_description"
   ],
   "type": "phrase"
  }
 }
}
```



```
"hits" : {
  "total" : {
    "value" : 6,
    "relation" : "eq"
  },
  "max_score" : 28.425034,
  "hits" : [
    {
      "_index" : "news_headlines",
      "_type" : "_doc",
      "_id" : "cdg9S3cBwsjPafpAMROy",
      "_score" : 28.425034,
      "_source" : {
        "date" : "2013-12-31",
        "short_description" : "If you are
          opting out of the over-the-top party
          this year and entertaining friends
          and family at home, I've rounded up
          a few New Years Eve party-planning
          musts.",
        "@timestamp" : "2013-12-31T00:00:00
          .000-07:00",
        "link" : "https://www.huffingtonpost
          .com/entry/new-years-eve-party
          -planning_us_5b9dbe4fe4b03a1dcc8c53a
          2",
        "category" : "HOME & LIVING",
        "headline" : "New Years Eve Party
          Planning Ideas",
        "authors" : """Lauren Nelson,
          Contributor
Founder, Lauren-Nelson.com"""
      }
```

Bool Query

The bool query retrieves documents matching boolean combinations of other queries.
With the bool query, you can combine multiple queries into one request and further specify boolean clauses to narrow down your search results. There are four clauses to choose from:

1. *must*
2. *must_not*
3. *should*
4. *filter*

You can build combinations of one or more of these clauses.
Each clause can contain one or multiple queries that specify the criteria of each clause.
These clauses are optional and can be mixed and matched to cater to your use case.
The order in which they appear does not matter either!

Bool Query

Syntax:

```
GET name_of_index/_search
{
 "query": {
     "bool": {
          "must": [{One or more queries can be specified here. A document MUST match all of these queries to
be considered as a hit.}],
           "must_not": [{A document must NOT match any of the queries specified here. It it does, it is excluded
from the search results.}],
           "should": [{A document does not have to match any queries specified here. However, it if it does
match, this document is given a higher score.}],
           "filter": [{These filters(queries) place documents in either yes or no category. Ones that fall into the
yes category are included in the hits. }]
   }
  }
}
```

Aggregations Request

Syntax:

```
GET Enter_name_of_the_index_here/_search
{
 "aggs": {
  "Name your aggregations here": {
   "Specify the aggregation type here": {
    "field": "Name the field you want to aggregate on here"
   }
  }
 }
}
```

Metric Aggregations

Metric aggregations are used to compute numeric values based on your dataset. It can be used to calculate the values of sum,min, max, avg, unique count(cardinality) and etc.
Compute the sum of all unit prices in the index

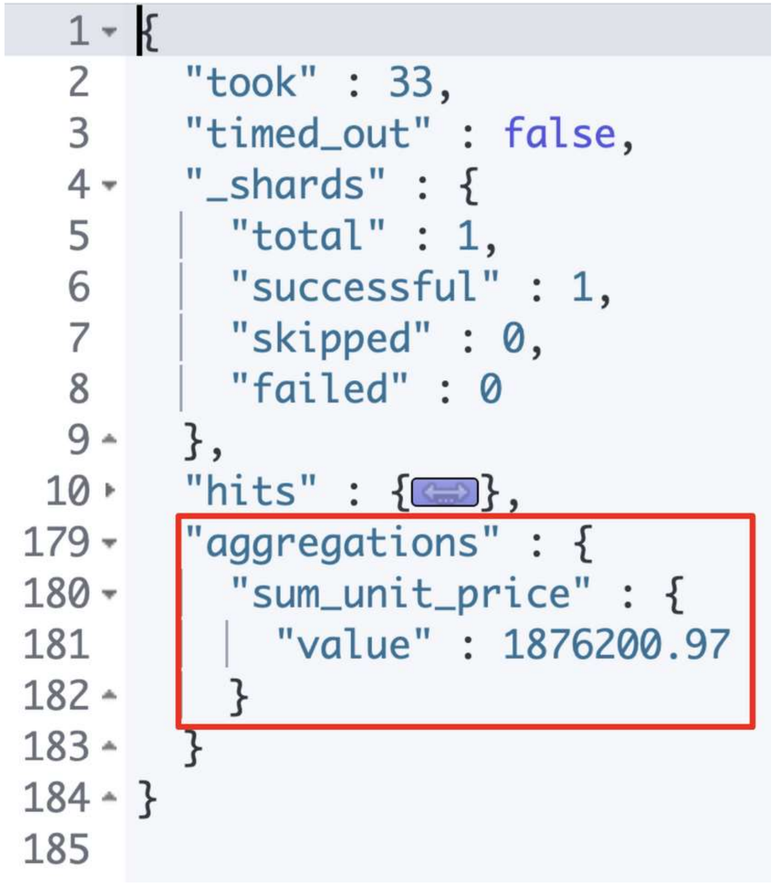
Syntax:

```
GET Enter_name_of_the_index_here/_search
{
 "aggs": {
  "Name your aggregations here": {
   "sum": {
    "field": "Name the field you want to aggregate on here"
   }
  }
 }
}
```

Metric Aggregations

Example:

```
GET ecommerce_data/_search
{
 "aggs": {
  "sum_unit_price": {
   "sum": {
    "field": "UnitPrice"
   }
  }
 }
}
```



```
{
  "took" : 33,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {↔},
  "aggregations" : {
    "sum_unit_price" : {
      "value" : 1876200.97
    }
  }
}
```

Metric Aggregations

Stats Aggregation: Compute the count, min, max, avg, sum in one go

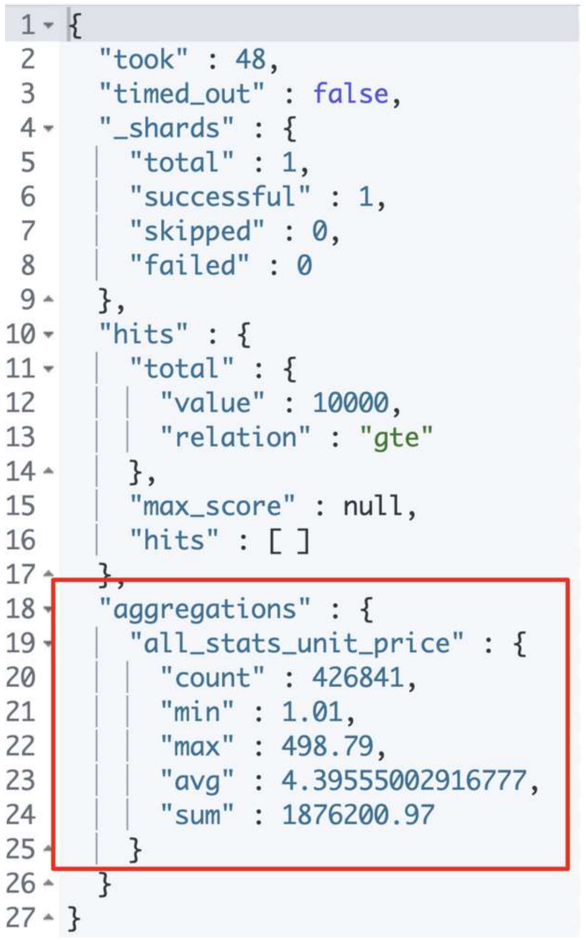
Syntax:

```
GET Enter_name_of_the_index_here/_search
{
 "size": 0,
 "aggs": {
  "Name your aggregations here": {
   "stats": {
    "field": "Name the field you want to aggregate on here"
   }
  }
 }
}
```

Metric Aggregations

Example:

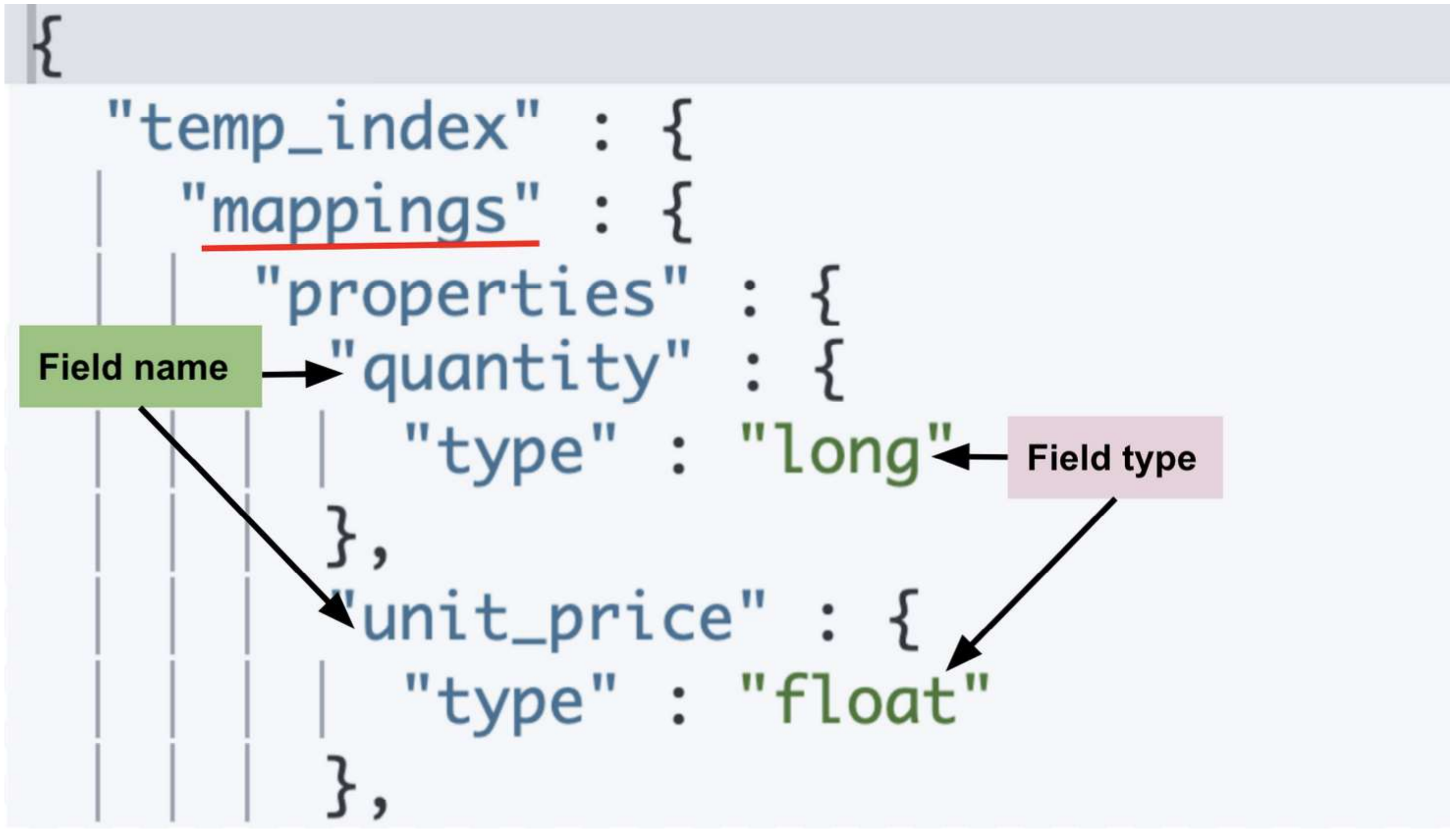
```
GET ecommerce_data/_search
{
 "size": 0,
 "aggs": {
  "all_stats_unit_price": {
   "stats": {
    "field": "UnitPrice"
   }
  }
 }
}
```



```
{
  "took" : 48,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 10000,
      "relation" : "gte"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "all_stats_unit_price" : {
      "count" : 426841,
      "min" : 1.01,
      "max" : 498.79,
      "avg" : 4.39555002916777,
      "sum" : 1876200.97
    }
  }
}
```

Mapping Explained

Mapping determines how a document and its fields are indexed and stored by defining the type of each field.



```
{
  "temp_index" : {
    "mappings" : {
      "properties" : {
        "quantity" : {
          "type" : "long"
        },
        "unit_price" : {
          "type" : "float"
        },
```

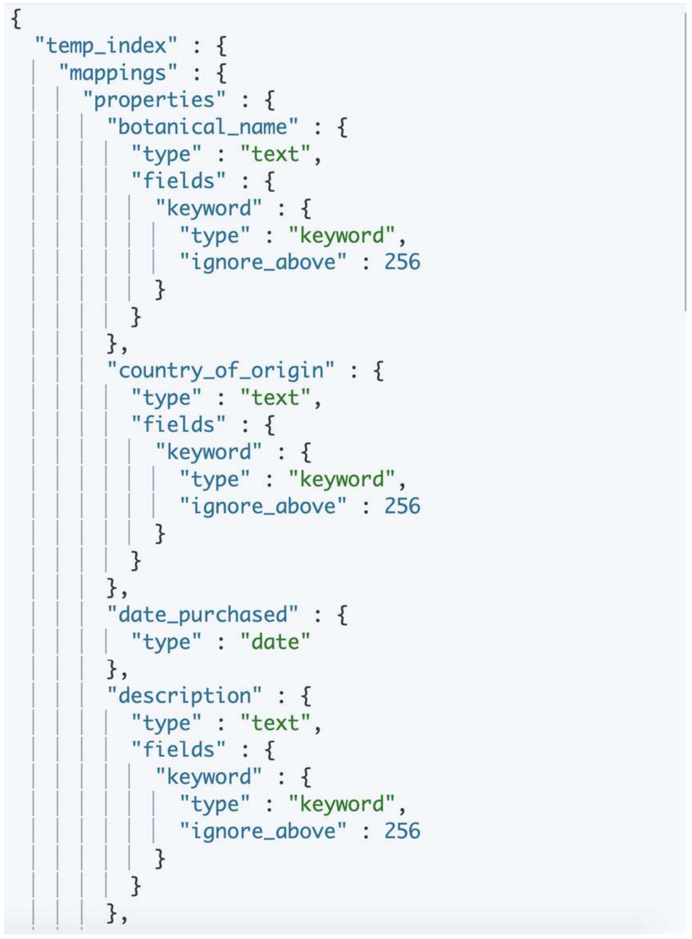
View the Mapping

Syntax:

GET Enter_name_of_the_index_here/_mapping

Example:

GET temp_index/_mapping



```
{
  "temp_index" : {
    "mappings" : {
      "properties" : {
        "botanical_name" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "country_of_origin" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "date_purchased" : {
          "type" : "date"
        },
        "description" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
```

References

- https://www.elastic.co/guide/index.html
- Blog: https://dev.to/lisahjung

For more courses feel free to check our website:
https://sdadclub.tech/courses/courses.html