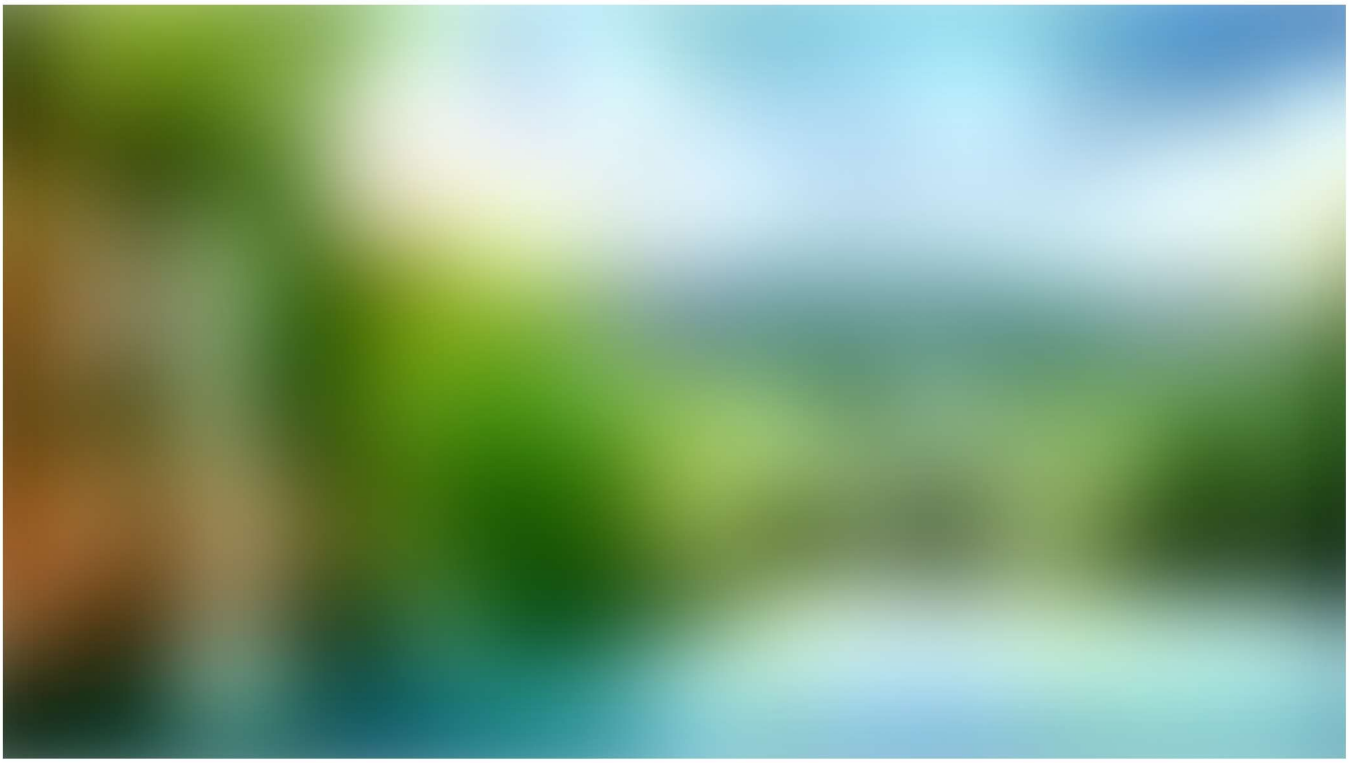
Elasticsearch tutorial for beginners using Python

This tutorial is for the beginners, who wants to learn Elasticsearch from the scratch.

Nitin Panwar
Jan 2, 2019 · 5 min read

In this tutorial i am going to cover all the basic and advance stuff related to the Elasticsearch. So let's get started.



Elasticsearch:-

Elasticsearch is a real-time distributed search and analytics engine. It allows you to explore your data at a speed and at a scale never before possible. It is used for full text search, structured search, analytics and all three in combination. Elastic search is an open source search engine built on top of Apache Lucecne, a full text search engine library.

Installing and running Elasticsearch:

The only requirement for installing Elasticsearch is a recent version of Java. To install Elasticsearch, download and extract the archive file from elastic.co/downlaods/elasticsearch and simply run bin\elasticsearch.bat. Test it in browser @ 'http://localhost:9200'.

Index:-

An index is like a database in traditional database. It is the place to store related documents. To retrive any document we would need three pieces of informantion

1. Index — Database
2. Datatype — Type of the document
3. Id — Id of the document

Let's start the show...

```
# Import Elasticsearch package
from elasticsearch import Elasticsearch
# Connect to the elastic cluster
es=Elasticsearch([{'host':'localhost','port':9200}])
es

<Elasticsearch([{'host': 'localhost', 'port': 9200}])>
```

Elasticsearch is document oriented, meaning that it stores entire object or documents. It not only stores them, but also indexes the content of each document in order to make them searchable. In Elasticsearch you index, search,sort and filter documents.

Elasticsearch uses JSON as the serialisation format for the documents.

Now let's start by indexing the employee documents.

The act of storing data in Elasticsearch is called indexing. An Elasticsearch cluster can contain multiple indices, which in turn contain multiple types. These types hold multiple documents, and each document has multiple fields.

```
e1={
    "first_name":"nitin",
    "last_name":"panwar",
    "age": 27,
    "about": "Love to play cricket",
```

```
    "interests": ['sports','music'],
}

print e1

{'interests': ['sports', 'music'], 'about': 'Love to play cricket',
'first_name': 'nitin', 'last_name': 'panwar', 'age': 27}
```

Inserting a document:

```
#Now let's store this document in Elasticsearch
res = es.index(index='megacorp',doc_type='employee',id=1,body=e1)
```

Simple! There was no need to perform any administrative tasks first, like creating an index or specifying the type of data that each field contains. We could just index a document directly. Elasticsearch ships with defaults for everything, so all the necessary administration tasks were taken care of in the background, using default values.

```
# Let's insert some more documents
e2={
    "first_name" :  "Jane",
    "last_name" :   "Smith",
    "age" :         32,
    "about" :       "I like to collect rock albums",
    "interests":  [ "music" ]
}
e3={
    "first_name" :  "Douglas",
    "last_name" :   "Fir",
    "age" :         35,
    "about":        "I like to build cabinets",
    "interests":  [ "forestry" ]
}

res=es.index(index='megacorp',doc_type='employee',id=2,body=e2)
print res['created']
res=es.index(index='megacorp',doc_type='employee',id=3,body=e3)
print res['created']

False
True
```

Retrieving a Document:

This is easy in Elasticsearch. We simply execute an HTTP GET request and specify the address of the document — the index, type, and ID. Using those three pieces of information, we can return the original JSON document.

```
res=es.get(index='megacorp',doc_type='employee',id=3)
print res

{u'_type': u'employee', u'_source': {u'interests': [u'forestry'],
u'age': 35, u'about': u'I like to build cabinets', u'last_name':
u'Fir', u'first_name': u'Douglas'}, u'_index': u'megacorp',
u'_version': 1, u'found': True, u'_id': u'3'}
```

You will get the actual document in '_source' field

```
print res['_source']

{u'interests': [u'forestry'], u'age': 35, u'about': u'I like to
build cabinets', u'last_name': u'Fir', u'first_name': u'Douglas'}
```

Deleting a document:

```
res=es.delete(index='megacorp',doc_type='employee',id=3)

print res['result']

deleted
```

Now let's validate it in Elasticsearch

```
res= es.search(index='megacorp',body={'query':{'match_all':{}}})
print('Got %d hits:' %res['hits']['total'])

Got 3 hits:
```

Search Lite:

A GET is fairly simple — you get back the document that you ask for. Let's try something a little more advanced, like a simple search!

```
res= es.search(index='megacorp',body={'query':{}})
print res['hits']['hits']

[{u'_score': 1.0, u'_type': u'employee', u'_id': u'4', u'_source':
{u'interests': [u'sports', u'music'], u'age': 27, u'about': u'Love
to play football', u'last_name': u'pafdfd', u'first_name': u'asd'},
u'_index': u'megacorp'}, {u'_score': 1.0, u'_type': u'employee',
u'_id': u'2', u'_source': {u'interests': [u'music'], u'age': 32,
u'about': u'I like to collect rock albums', u'last_name': u'Smith',
u'first_name': u'Jane'}, u'_index': u'megacorp'}, {u'_score': 1.0,
u'_type': u'employee', u'_id': u'1', u'_source': {u'interests':
[u'sports', u'music'], u'age': 27, u'about': u'Love to play
cricket', u'last_name': u'panwar', u'first_name': u'nitin'},
u'_index': u'megacorp'}]
```

Now let's search for the user name who has nitin in his first name.

match operator:

```
res= es.search(index='megacorp',body={'query':{'match':
{'first_name':'nitin'}}})
print res['hits']['hits']

[{u'_score': 0.2876821, u'_type': u'employee', u'_id': u'1',
u'_source': {u'interests': [u'sports', u'music'], u'age': 27,
u'about': u'Love to play cricket', u'last_name': u'panwar',
u'first_name': u'nitin'}, u'_index': u'megacorp'}]
```

bool operator:

bool takes a dictionary containing at least one of must, should, and must_not, each of which takes a list of matches or other further search operators.

```
res= es.search(index='megacorp',body={
        'query':{
            'bool':{
                'must':[{
                        'match':{
                            'first_name':'nitin'
                        }
                    }]
            }
        }
    })

print res['hits']['hits']
```

```
[{u'_score': 0.2876821, u'_type': u'employee', u'_id': u'1',
u'_source': {u'interests': [u'sports', u'music'], u'age': 27,
u'about': u'Love to play cricket', u'last_name': u'panwar',
u'first_name': u'nitin'}, u'_index': u'megacorp'}]
```

Filter operator:

Let's make the search a little more complicated. We still want to find all employees with a first name of nitin, but we want only employees who are older than 30. Our query will change a little to accommodate a filter, which allows us to execute structured searches efficiently:

```
res= es.search(index='megacorp',body={
        'query':{
            'bool':{
                'must':{
                    'match':{
                        'first_name':'nitin'
                    }
                },
                "filter":{
                    "range":{
                        "age":{
                            "gt":25
                        }
                    }
                }
            }
        }
    })

print res['hits']['hits']

[{u'_score': 0.2876821, u'_type': u'employee', u'_id': u'1',
u'_source': {u'interests': [u'sports', u'music'], u'age': 27,
u'about': u'Love to play cricket', u'last_name': u'panwar',
u'first_name': u'nitin'}, u'_index': u'megacorp'}]

res= es.search(index='megacorp',body={
        'query':{
            'bool':{
                'must':{
                    'match':{
                        'first_name':'nitin'
                    }
                },
                "filter":{
                    "range":{
                        "age":{
                            "gt":27
                        }
                    }
```

```
                    }
                }
            }
        })

print res['hits']['hits']

[]
```

Full text search

The searches so far have been simple.Let's try more advanced full text search. Before starting this next type of search let me insert one more documnt.

```
e4={
    "first_name":"asd",
    "last_name":"pafdfd",
    "age": 27,
    "about": "Love to play football",
    "interests": ['sports','music'],
}

res=es.index(index='megacorp',doc_type='employee',id=4,body=e4)
print res['created']

False

res= es.search(index='megacorp',doc_type='employee',body={
        'query':{
            'match':{
                "about":"play cricket"
            }
        }
    })
for hit in res['hits']['hits']:
    print hit['_source']['about']
    print hit['_score']
    print '**********************'

Love to play football
0.7549128
**********************
Love to play cricket
0.5753642
**********************
```

In above example it is returning two records but scores are differnt.

Phrase Search

Finding individual words in a field is all well and good, but sometimes you want to match exact sequence of words of phrases.

```
res= es.search(index='megacorp',doc_type='employee',body={
        'query':{
            'match_phrase':{
                "about":"play cricket"
            }
        }
    })
for hit in res['hits']['hits']:
    print hit['_source']['about']
    print hit['_score']
    print '**********************'

Love to play cricket
0.5753642
**********************
```

Aggregations

Elasticsearch has functionality called aggregations, which allowed you to generate sophisticated analytics over your data. It is similar to Group By in SQL, but much more powerful.

```
res= es.search(index='megacorp',doc_type='employee',body={
        "aggs": {
            "all_interests": {
            "terms": { "field": "interests" }
            }
        }
    })
```

Please comment below if you liked the above article on Elasticseach, it will definitely encourage me to write more or suggest any topic that you want to read further.

Elasticsearch Python Beginner


Download on the
App Store


GET IT ON
Google Play