Elastic Search

Summary	Elastic Search with Kibana
URL	https://www.elastic.co/
Tools	Elastic Search, Kibana, Python

What is Elastic Search

Elasticsearch is an open source search engine highly scalable. It allows you to keep and analyse a great volume of information practically in real time. Elasticsearch works with JSON documents files. Using an internal structure, it can parse your data in almost real time to search for the information you need.

It is very useful when working with big data.

Some technical (but useful) information to know about Elasticsearch are:

- It is a real time distributed and analytics engine.
- It is open source, developed in Java.
- It uses a structure based on documents instead of tables and schema.
- Besides speed and scalability, it has high resiliency relating to failures, and it's really flexible relating to data type.

Terminologies and Definitions:

Query: The language to perform and combine many types of searches like structured, unstructured, geo, metric, etc.

Cluster
A cluster is a collection of one or more nodes that, together, holds the entire data. It provides federated indexing and search capabilities across all nodes and is identified by an unique name (by default it is '/ elasticsearch'/)

Node
A node is a single server which is a part of a cluster, stores data and participates in the cluster's indexing and search capabilities.

Index
An index is a collection of documents with similar characteristics and is identified by a name. This name is used to refer to the index while performing indexing, search, update, and delete operations against the documents in it. In a single cluster, you can define as many indexes as you want.

Document
A document is a basic unit of information which can be indexed. It is expressed in JSON which is an ubiquitous internet data interchange format.

Shards
Elasticsearch provides the ability to subdivide the index into multiple pieces called shards. Each shard is in itself a fully-functional and independent "index" that can be hosted on any node within the cluster. This is useful for the case when an index put in a single node would take more disk space than available. The index then is subdivided between different nodes. Besides, shards allow you to distribute and parallelise operations across shards, increasing the performance.

Replicas
Elasticsearch allows you to make one or more copies of your index's shards which are called replica shards or replicas. It provides high availability in case a node fails, and it allows you to scale out your search volume since searches can be executed on all replicas in parallel.

Installation

To download ElasticSearch and Kibana use the below link:
https://www.elastic.co/downloads/

Mac: Elastic Search

If you have MacOS with Homebrew installed, you can install it by simply typing
brew install elasticsearch

```
curl -L -O
https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.7.1-da
rwin-x86_64.tar.gz

tar -xzvf elasticsearch-7.7.1-darwin-x86_64.tar.gz

cd elasticsearch-7.7.1
./bin/elasticsearch
```

Mac : Kibana

If you have MacOS with Homebrew installed, you can install it by simply typing
brew install kibana

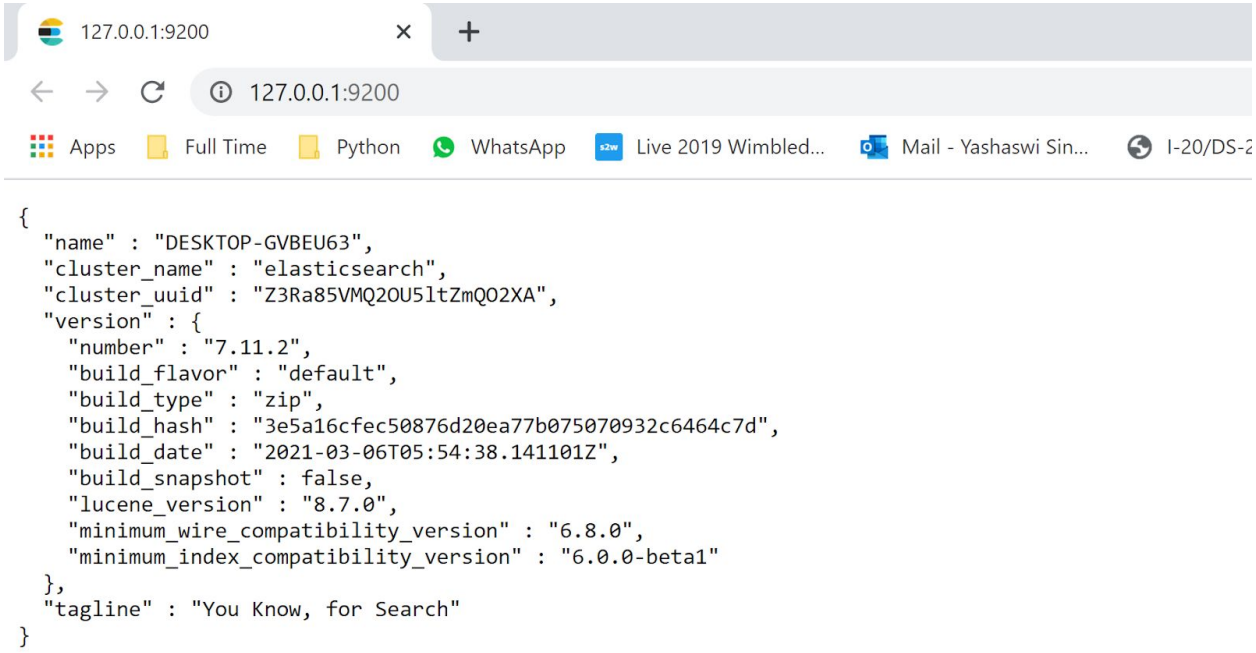
```
curl -L -O
https://artifacts.elastic.co/downloads/kibana/kibana-7.7.1-darwi
n-x86_64.tar.gz

tar xzvf kibana-7.7.1-darwin-x86_64.tar.gz

cd kibana-7.7.1-darwin-x86_64/

./bin/kibana
```

Windows: Elastic Search
Download the Elasticsearch

1. Windows zip file from the Elasticsearch download page.
2. Extract the contents of the zip file to a directory on your computer, for example, C:\Program Files.

3. Open a command prompt as an Administrator and navigate to the directory that contains the extracted files, and run the batch file using command: elasticsearch.bat

To test if it is running, open http://127.0.0.1:9200/
If it is successful, it would show something like this :



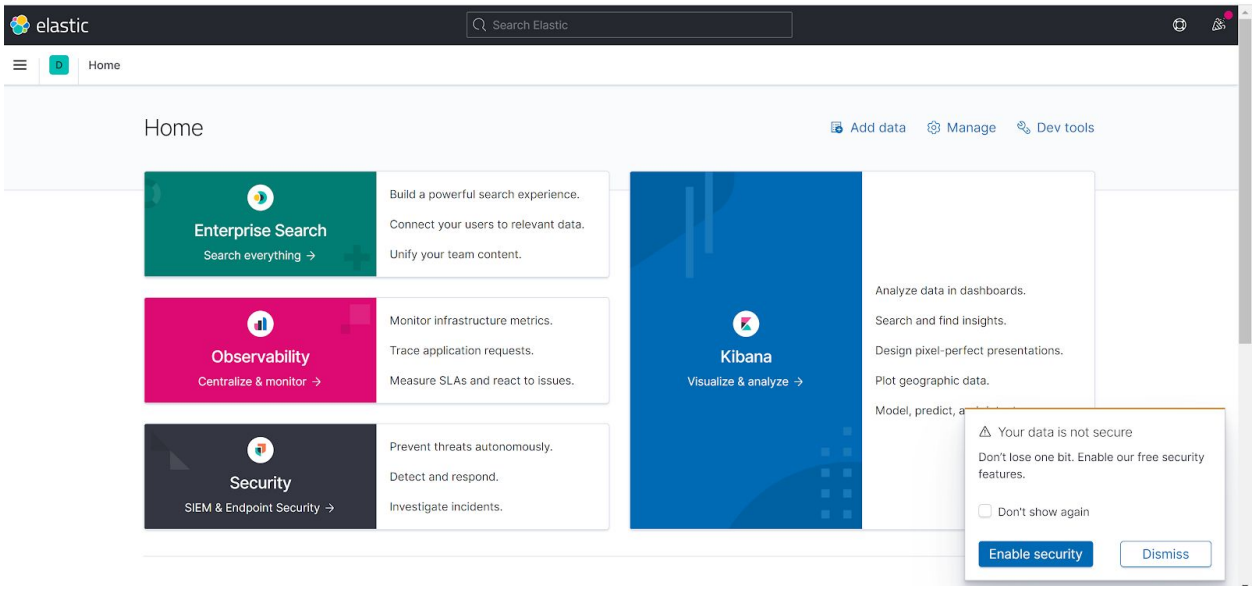
Windows : Kibana

Download Kibana

1. Windows zip file from the Kibana download page.
2. Extract the contents of the zip file to a directory on your computer, for example, C:\Program Files.
3. Open a command prompt as an Administrator and navigate to the directory that contains the extracted files

To test if it is running, open http://localhost:5601/

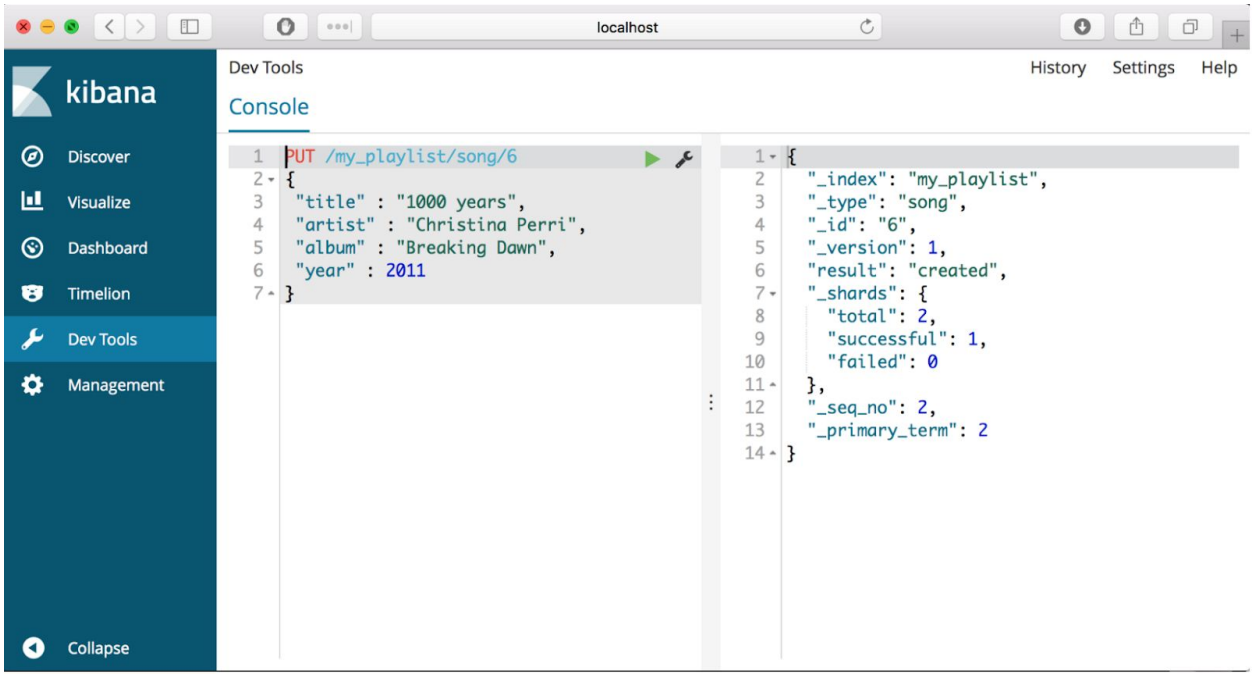
If it is successful, it would show something like this :



PUT

PUT command allows you to insert new document data into Elasticsearch. Type the following code in the console, press the green play button and see the result.

```
PUT /my_playlist/song/6
{
"title" : "1000 years",
"artist" : "Christina Perri",
"album" : "Breaking Dawn",
"year" : 2011
}
```



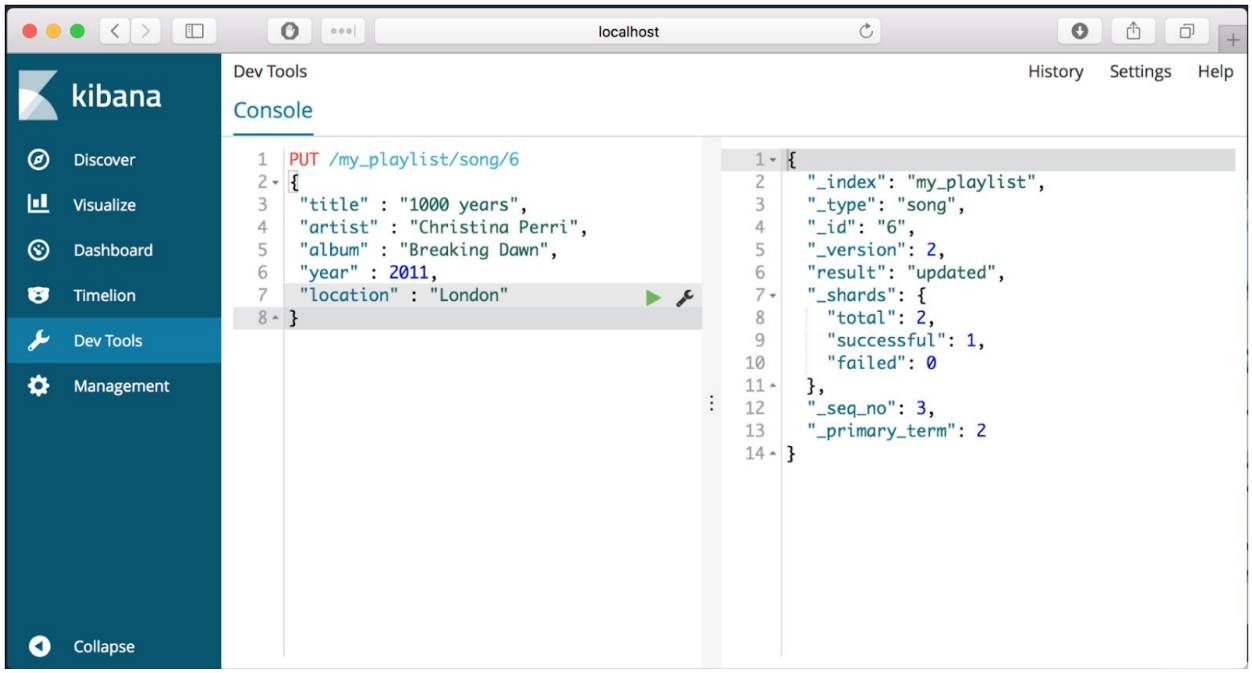
This means you just inserted a document data into Elasticsearch. In this example, we have /my_playlist/song/6, where:

- my_playlist: is the name of index you will insert the data.
- song: is the name of the document to be created.
- 6: id of element instance. In this case, is the song id.

If the index my_playlist did not exist already, it will be created, just as the document song and the id 6.

To UPDATE a value, you use the same PUT command to the same document. For example, if you want to insert a new parameter, location, you could do it in the following way:

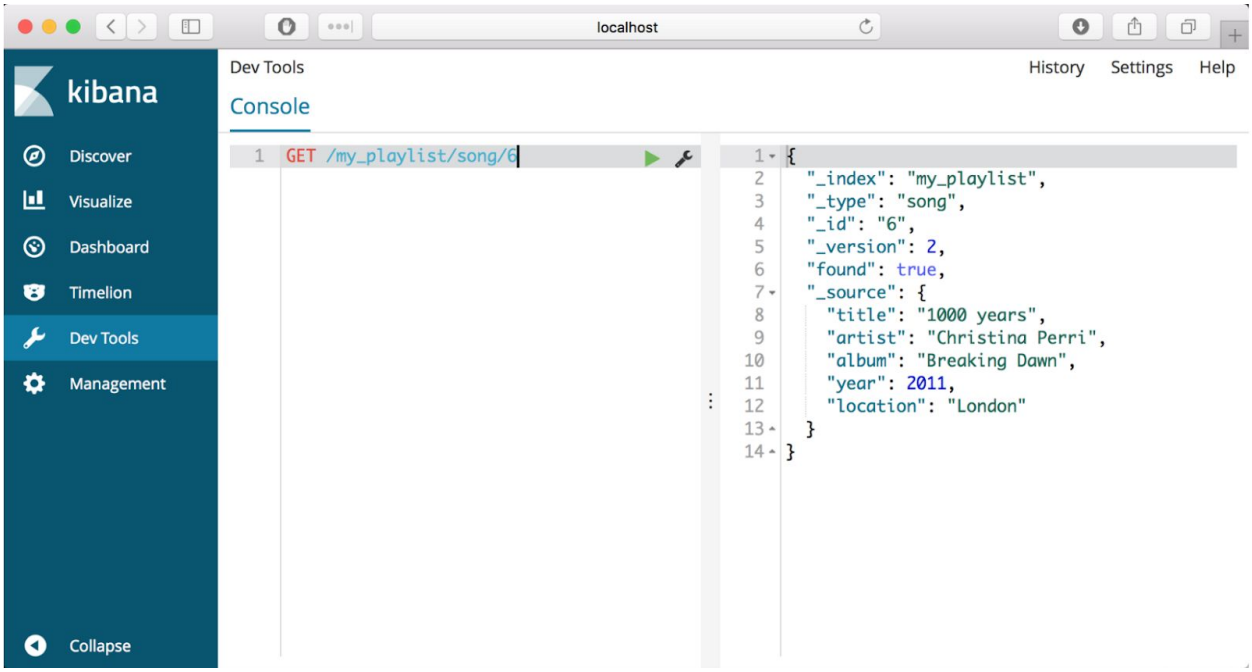
```
PUT /my_playlist/song/6
{
"title" : "1000 years",
"artist" : "Christina Perri",
"album" : "Breaking Dawn",
"year" : 2011,
"location" : "London"
}
```



GET

GET command allows you to retrieve information about your data. Type the following example:

```
GET /my_playlist/song/6
```

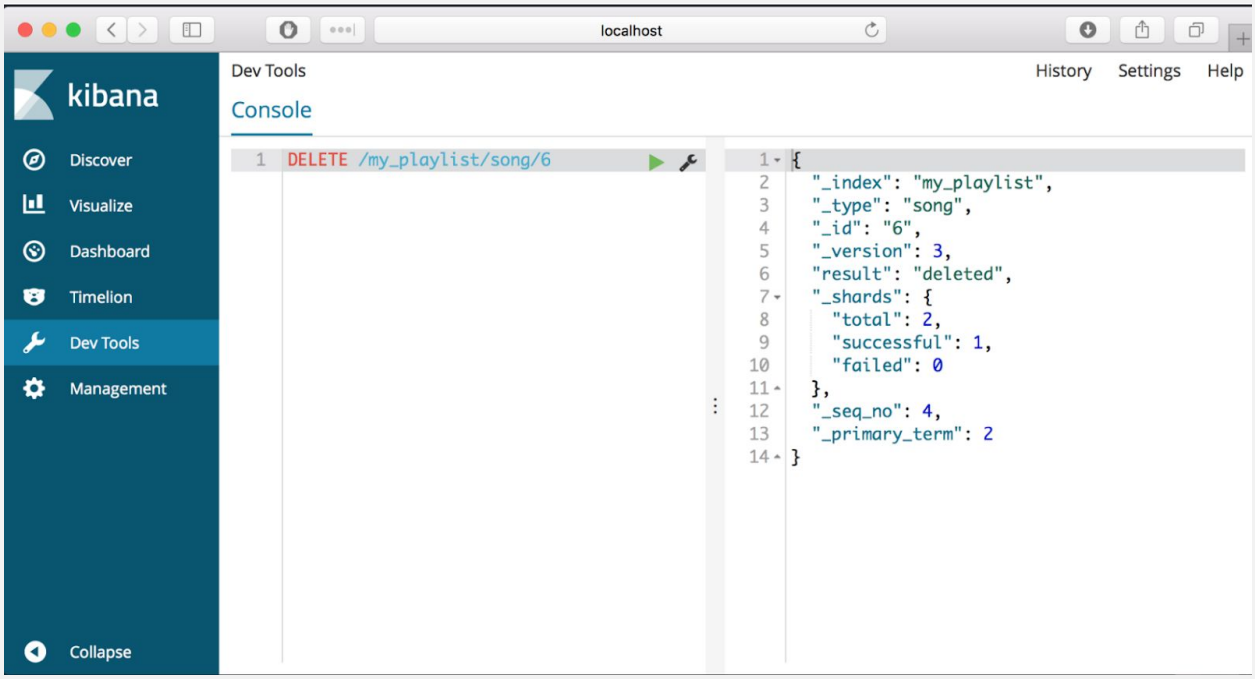


This retrieves the data you have just inserted.

DELETE

To delete a document, you just need to use the following command:

```
DELETE /my_playlist/song/6
```



Searching Data

To keep it simple, we will present just a few and simple examples, but you can find more about it here. The parameter used in the examples is q, representing that the query will be executed through URI.
We will use the accounts JSON file from the Kibana home page, available here.

To bulk load a data set into Elasticsearch, open the terminal, go to the directory the file downloaded is, then execute the following command:

```
curl -H “Content-Type:application/x-ndjson” -XPOST
“localhost:9200/bank/account/_bulk?pretty” --data-binary @accounts.json
```

Now you should have the accounts data into Elasticsearch. You can try the following examples:

Return all accounts from state UT.

```
GET /bank/_search?q=state:UT
```

Return all accounts from UT or CA.

```
GET /bank/_search?q=state:UT OR CA
```

Return all accounts from state TN and from female clients.

```
GET /bank/_search?q=state:TN AND gender:F
```

Return all accounts from people older than 20 years.

```
GET /bank/_search?q=age:>20
```

Return all accounts from people between 20 and 25 years old.

```
GET /bank/_search?q=age:(>=20 AND <=25)
```

Searching Using Query DSL

URI is not the best way to query Elasticsearch. It seems to be preferable to use QueryDSL instead.

Query DSL consists of two clauses :

- **Leaf query clause:** It looks for a particular value in a particular field, such as the match, term or range queries.
- **Compound query clause:** It wraps other leaf or compound queries and is used to combine multiple queries in a logical fashion.

Query clauses behave differently depending on whether they are used in query context or filter context:

- **Query context:** A query clause used in query context answers the question "How well does this document match this query clause?". The answer is a score representing how well the document matches, relative to other documents.
- **Filter context:** A query clause in filter context answers the question "Does this document match this query clause?". The answer is a simple YES or NO.

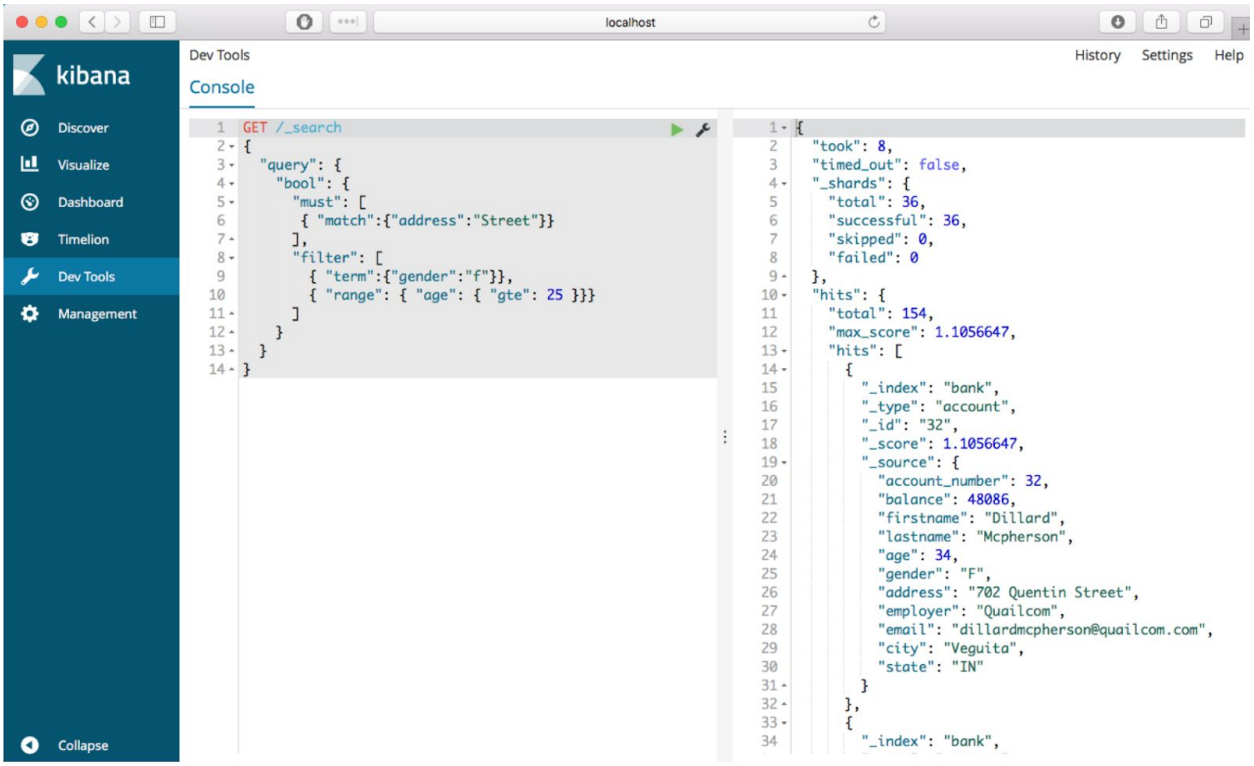
Below is an example of query clause being used in query and filter context in the search API.

This query will match documents where all of the following conditions are met:

- The address field contains the word Street
- The gender field contains the exact word f
- The age field contains a number greater or equal to 25

```
GET /_search
```

```
{
 "query": {                                  //1
   "bool": {                                 //2
     "must": [
      { "match":{"address":"Street"}}        //3
     ],
     "filter": [                             //4
       { "term":{"gender":"f"}},             //5
       { "range": { "age": { "gte": 25 }}}   //6
     ]
   }
 }
}
```



Understanding the code:

//1: The query parameter indicates query context.
//2 and //3: The bool and match clauses are used in query context, which means that they are used to score how well each document matches.
//4: The filter parameter indicates filter context.

//5 and //6: The term and range clauses are used in filter context. They will filter out documents which do not match, but they will not affect the score for matching documents.