Victor S Melo

146 Followers About Follow

A Practical Introduction to Elasticsearch with Kibana

Victor S Melo Jan 19, 2018 · 10 min read

This week, I had my first contact with Elasticsearch and Kibana (honestly, my first contact was yesterday). So, I found a lot of tutorials, but most of them go deep in technical stuffs that weren't really necessary for me right know, making it a bit harder to understand what really matters.

Then i had an idea: build a simple tutorial focusing on a really basic information about Elasticsearch and Kibana, using a simple language to present the starting topics I think everyone should know when starting to learn about it. I gathered the information from

What Is Elasticsearch

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.**

> *This is an 101 practical introduction to Elasticsearch for people who never had contact with it before.*

Elasticsearch is available here.

Definitions and other stuffs

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 an structure based in documents instead of tables and schema.

The great benefits I found about it are speed and scalability. It is implemented in a way to allow querying to be really fast. And about scalability, it can be runned in your laptop or in hundreds of servers with petabytes of data.

Besides speed and scalability, it has high resiliency relating to failures, and it's really flexible relating to data type.

Again, Elasticsearch is very useful for **big data**, making it easy to analyse million of data in almost real time searches. That is the Elasticsearch magic.

But, how can you search all these datas? For that, you use queries.

And about analysis, Elasticsearch lets you understand billions of log lines easily. It provides aggregations which help you zoom out to explore trends and patterns in your data.

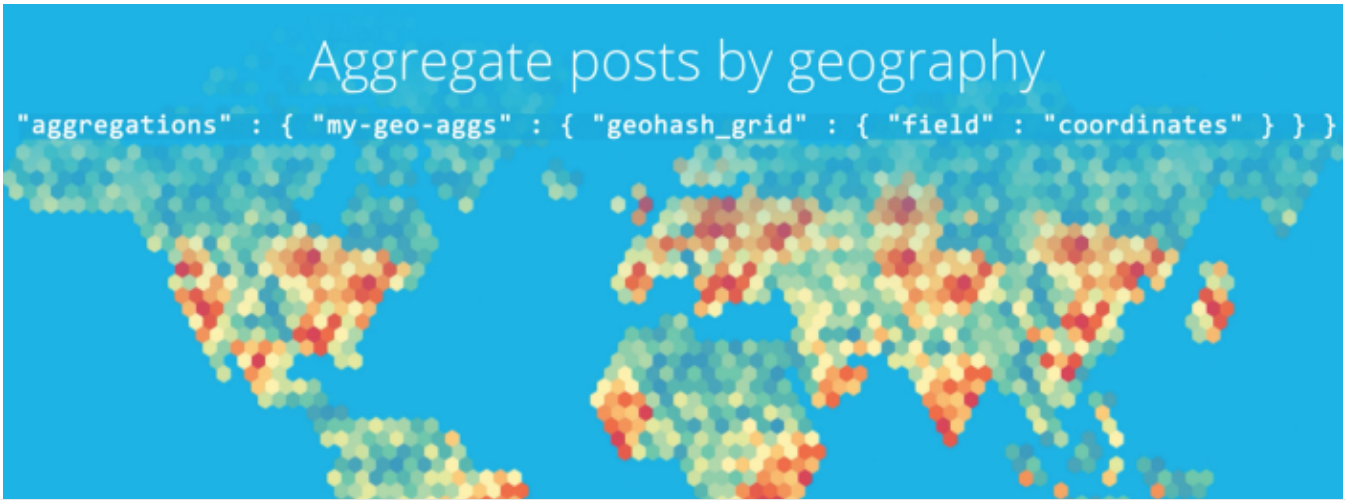
For example, if you have a cloud with 500 nodes, you can analyse the entire infrastructure in a short period of time, importing the logs into Elasticsearch and, based on it's response, you can get to the root cause of an issue in your infrastructure.

Other examples of use:

- Show data with an specific value. For example: show all 23 years old users from the database.

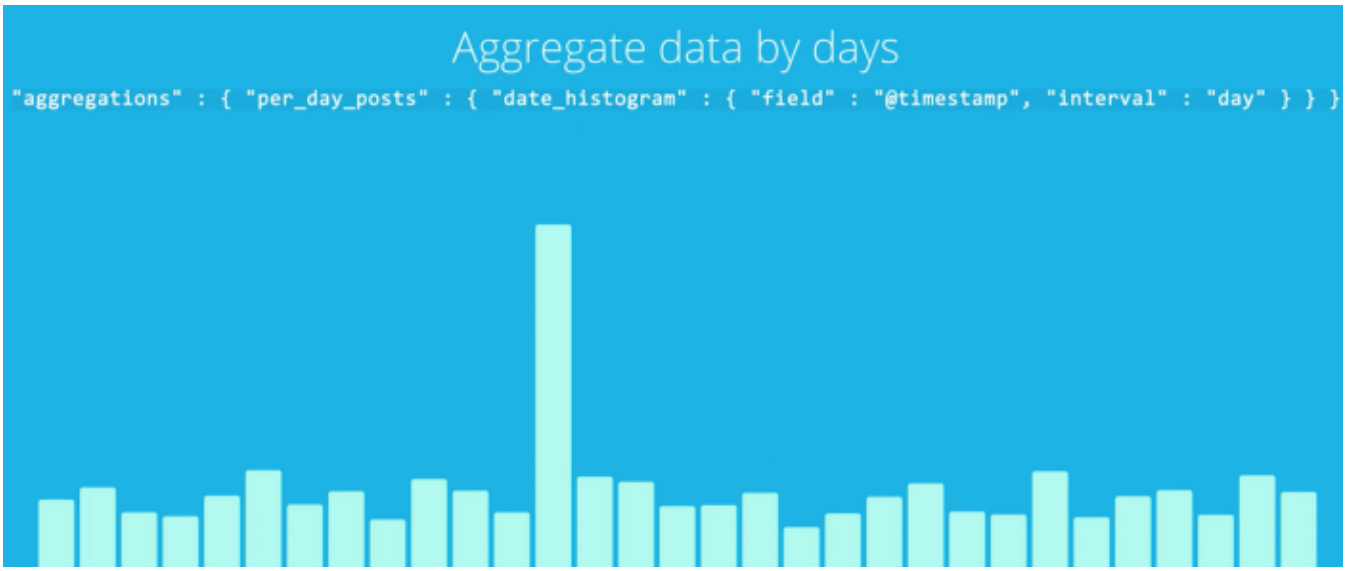
Select data with specific value

- Search data by geography



Search by geography

- Aggregate data by days



Aggregate data by days

Clients

Elasticsearch is so interesting that it is used by Mozilla, GitHub, Stack Exchange, Netflix, and many more users.

Hands On

Now that you know a bit about Elasticsearch, let's go to the practical part of this story.

Installation

To execute Elasticsearch, you must:

1. Install Java: should be version 8 or greater.
2. Install Elasticsearch: You can download it from elastic.co and follow the steps to install in your operational system. If you have MacOS with Homebrew installed, you can install it by simply typing `brew install elasticsearch`

Interface

To use Elasticsearch, you will need an interface. In this story I'll use Kibana, a great web interface to visualize and manipulate the data of Elasticsearch.

You need to download the same version for Elasticsearch and Kibana.

When, in the future, will find yourself needing to develop a software to interact with Elasticsearch, you can use a programming language to interact with it. Some of the programming languages acceptable are:

- Java
- C#
- Python
- JavaScript
- PHP
- Perl
- Ruby

Basic Concepts

Ok! Now you shall already have Elasticsearch and Kibana installed. Before we start using it, let's see some useful concepts about Elasticsearch.

Elasticsearch is made of

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 cluster, stores data and participates in the cluster's indexing and search capabilities.

Index

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 putted 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

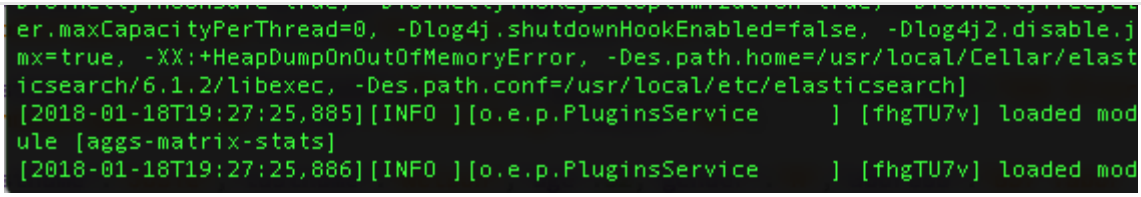
Elasticsearch allows you to make one or more copies of your index's shards which are called replica shards or replica. It provides high availability in case a node fails, and it allows you to scale out your search volume since searches can be executed on all replicas in parallel.

Executing

Ok, now, let's really put our hands on it. Go to the directory where Elasticsearch was installed and execute it through terminal: `$ ./elasticsearch`.

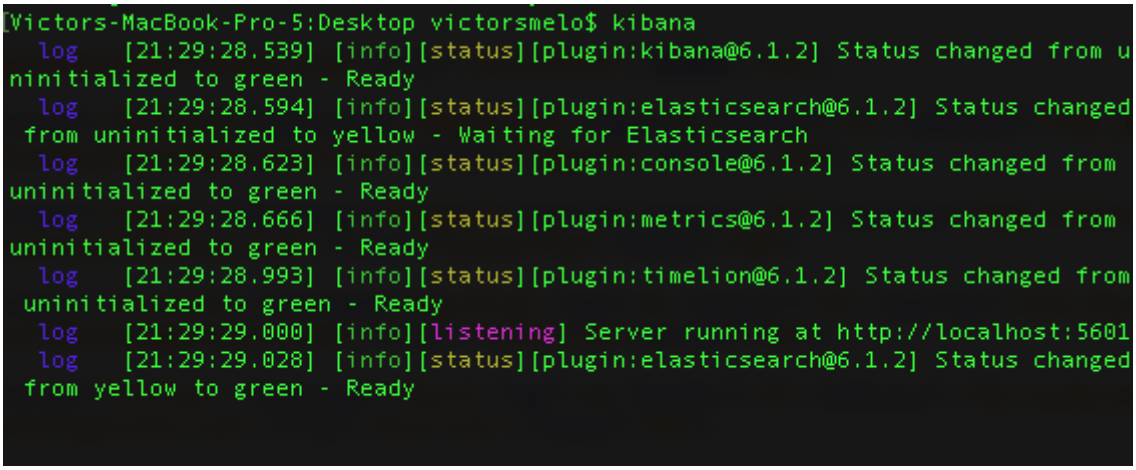
If you have installed it using Homebrew, try to type `elasticsearch` in terminal. It may starts Elasticsearch without needing to go to it's directory.

```
[Victors-MacBook-Pro-5:Desktop victorsmelo$ elasticsearch
[2018-01-18T19:27:24,310][INFO ][o.e.n.Node               ] [] initializing ...
[2018-01-18T19:27:24,440][INFO ][o.e.e.NodeEnvironment    ] [fhgTU7v] using [1]
data paths, mounts [[/ (/dev/disk1s1)]], net usable_space [6.7gb], net total_spa
ce [112.8gb], types [apfs]
[2018-01-18T19:27:24,441][INFO ][o.e.e.NodeEnvironment    ] [fhgTU7v] heap size
[990.7mb], compressed ordinary object pointers [true]
[2018-01-18T19:27:24,557][INFO ][o.e.n.Node               ] node name [fhgTU7v]
derived from node ID [fhgTU7vmTDaHgiM-I4kYYg]; set [node.name] to override
[2018-01-18T19:27:24,557][INFO ][o.e.n.Node               ] version[6.1.2], pid[
25927], build[5b1fea5/2018-01-10T02:35:59.208Z], OS[Mac OS X/10.13.2/x86_64], JV
M[Oracle Corporation/Java HotSpot(TM) 64-Bit Server VM/1.8.0_121/25.121-b13]
```



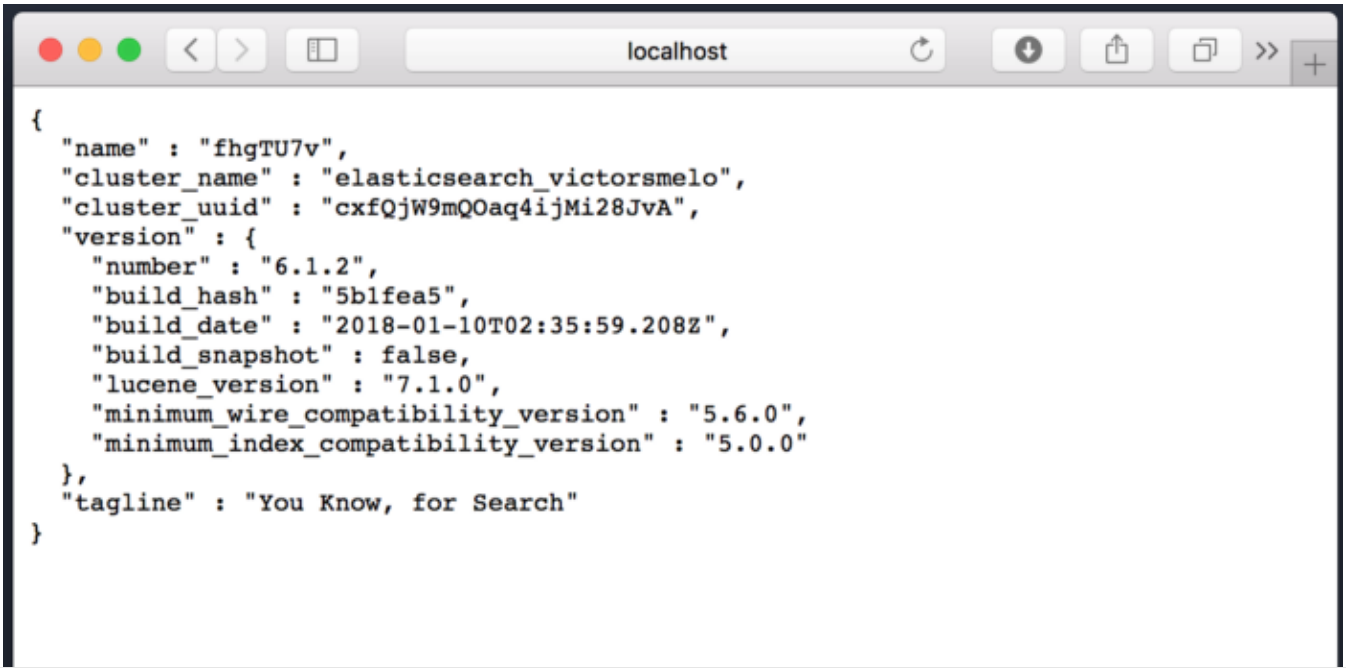
Elasticsearch starting

Now that Elasticsearch is being executed, open another terminal window and execute Kibana, going to it's directory and executing `$ ./kibana` or, with Homebrew, just typing `$ kibana` in terminal.



Kibana Started

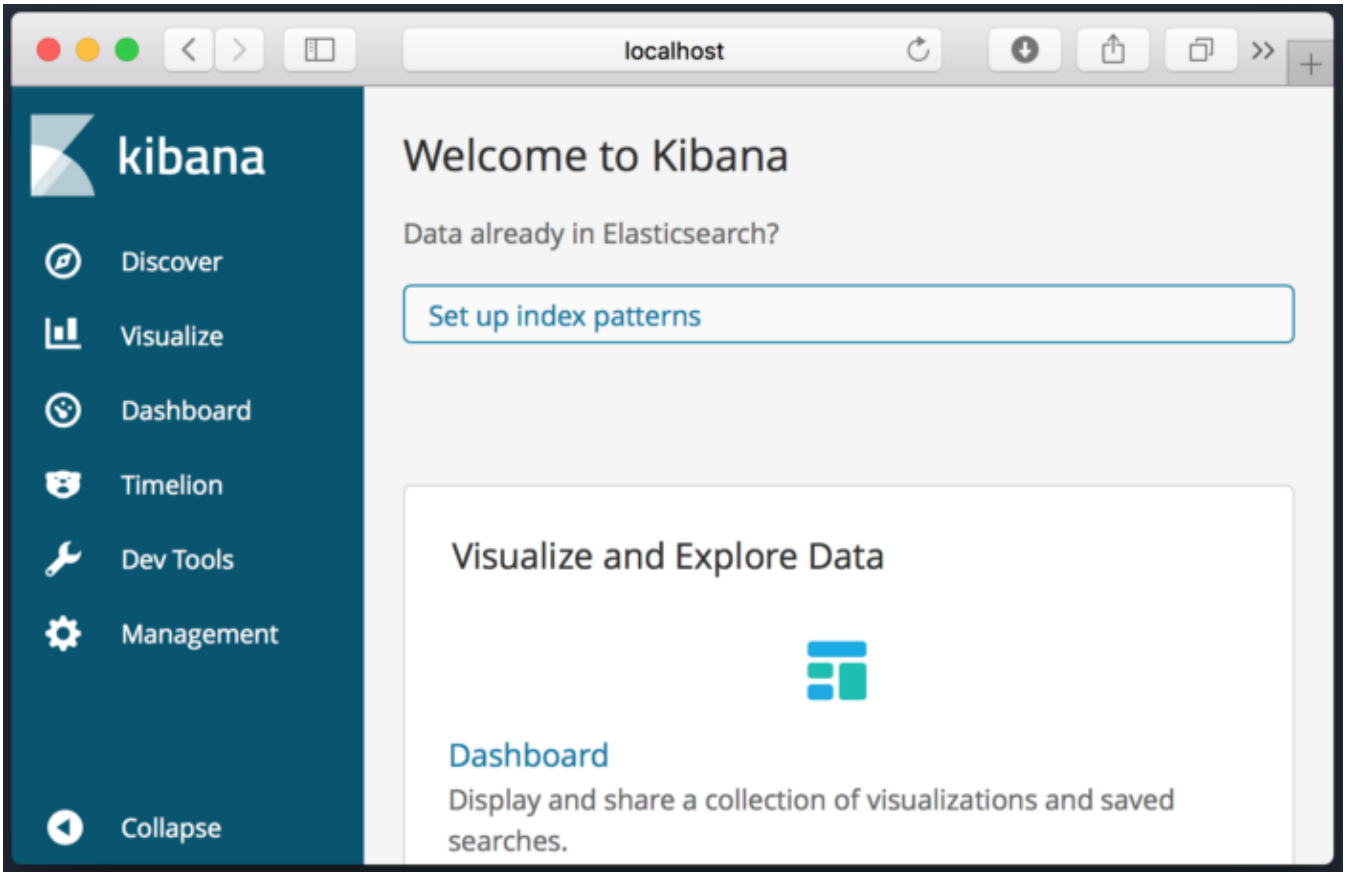
If everything goes right, now you shall have both being executed. To verify it, open a web browser and go to: http://localhost:9200



http://localhost:9200 result indicating Elasticsearch is running

If you see a result similar to the above, this means Elasticsearch is up and running.

To view Kibana interface, go to: `http://localhost:5601`

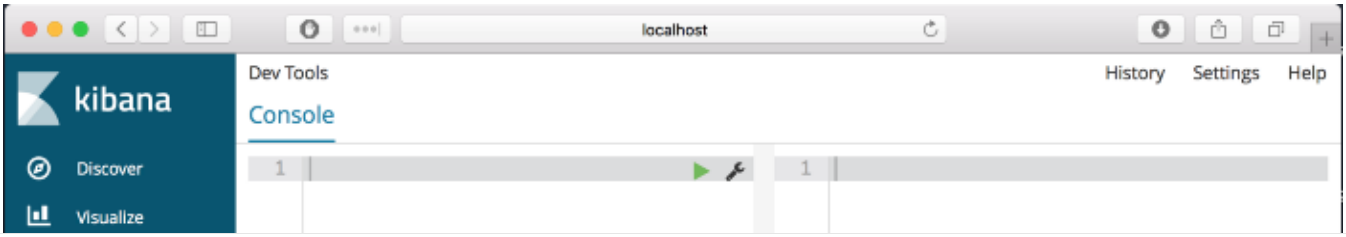


Kibana interface at http://localhost:5601

If everything is shown as above, let" play with some data now.

Commands

In the Kibana interface, select **Dev Tools**, on the left menu. You'll see a left console to type the commands and a right one to see the result.



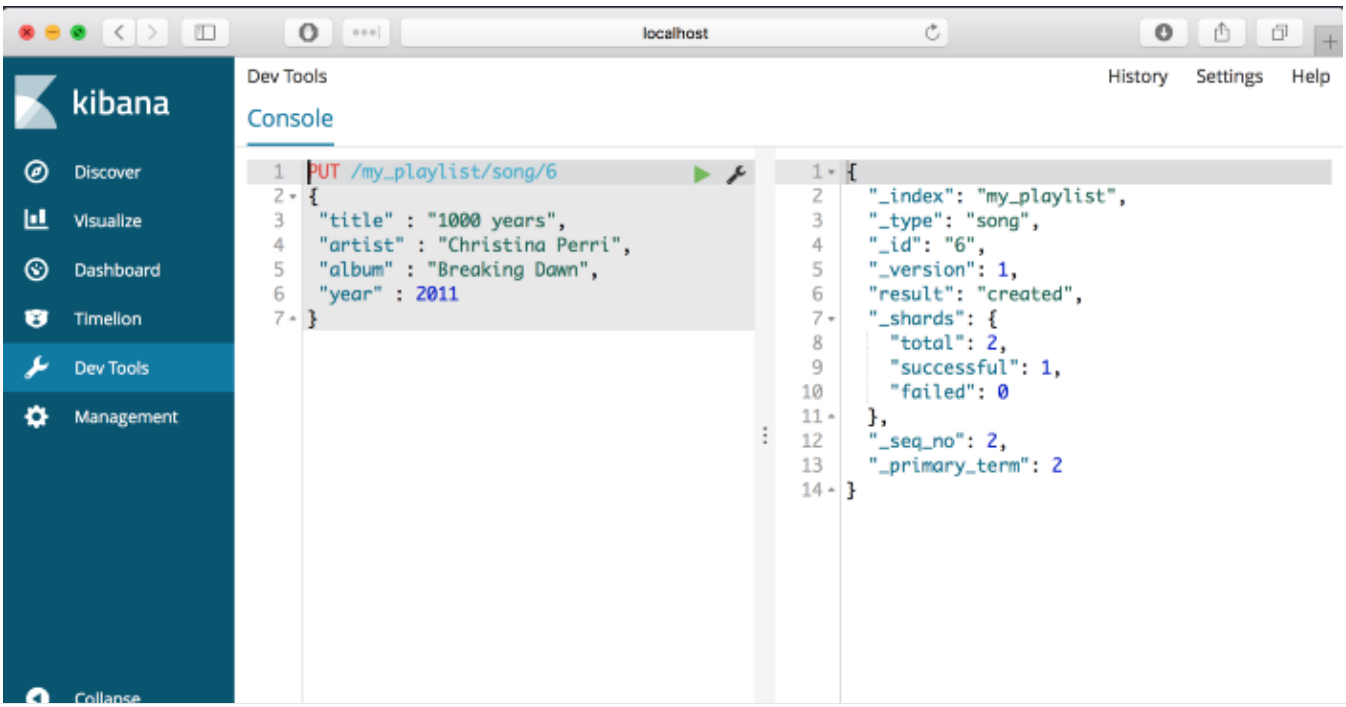
Let's see the commands available to manipulate data.

PUT

PUT command allows you to insert a new document data into Elasticsearch. Type the following code in 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
}
```

The result shall be something like:



have `/my_playlist/song/6`, where:

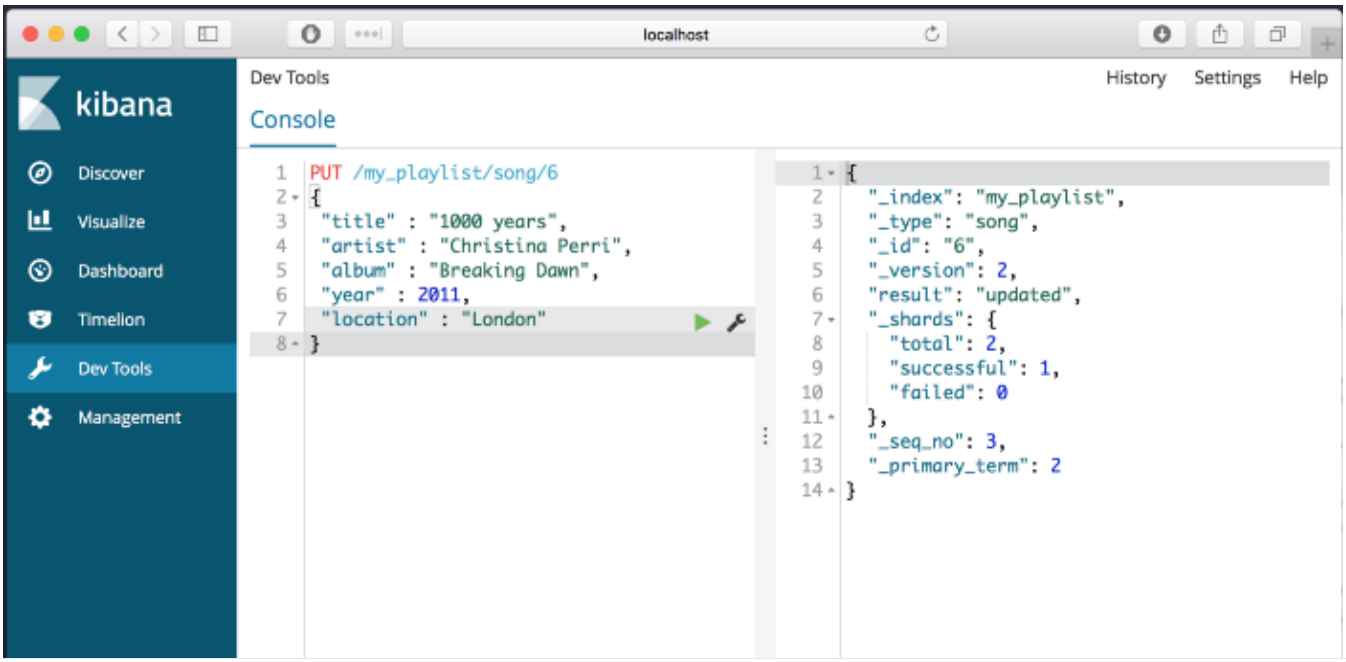
- my_playlist: is the name of index you will insert the data.
- song: is the name of the document to be created.
- 6: id of element instance. In this case, is the song id.

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"
}
```

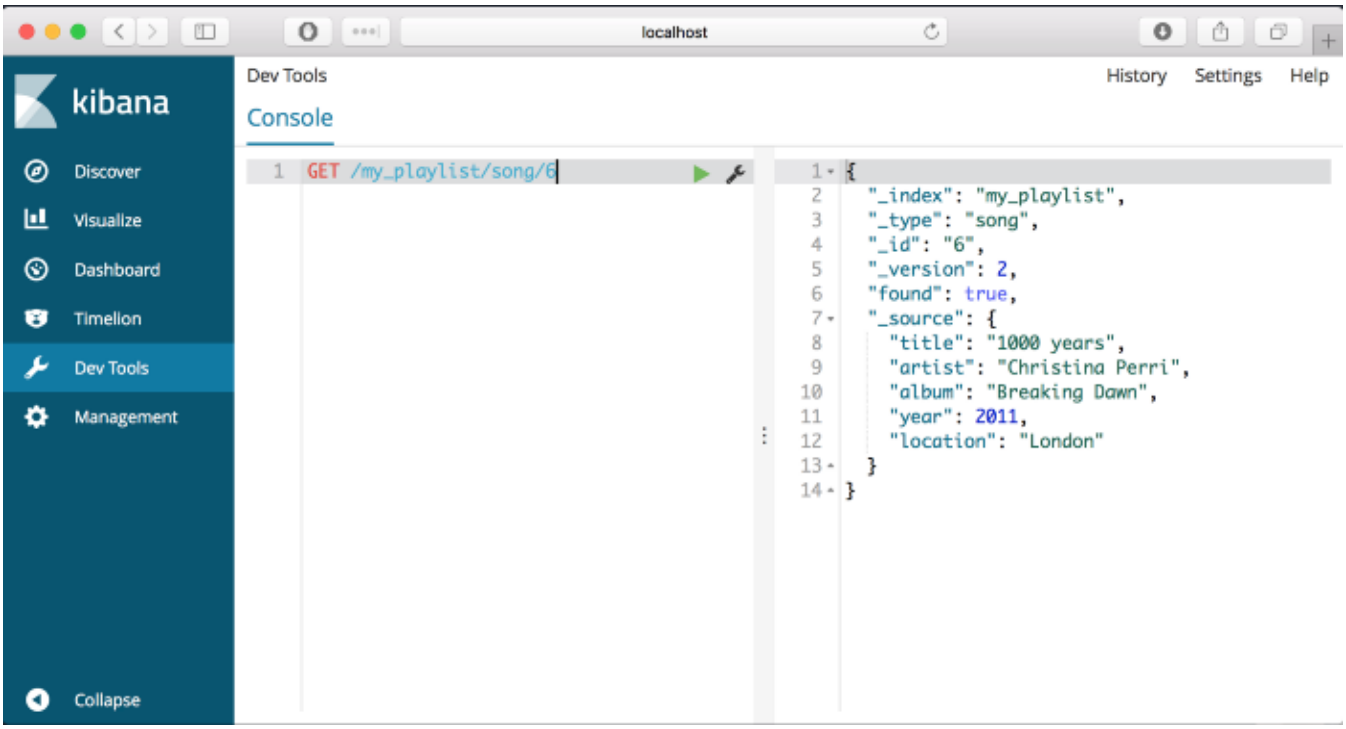
The result shall be



GET

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

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

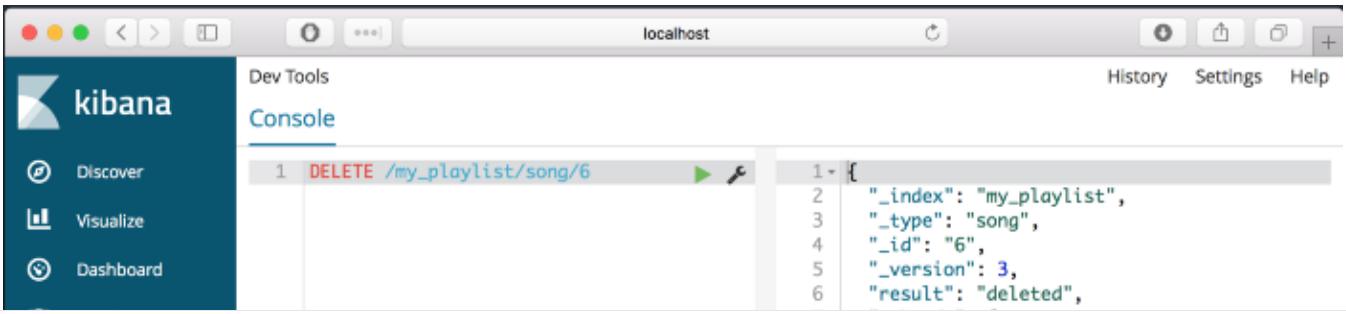


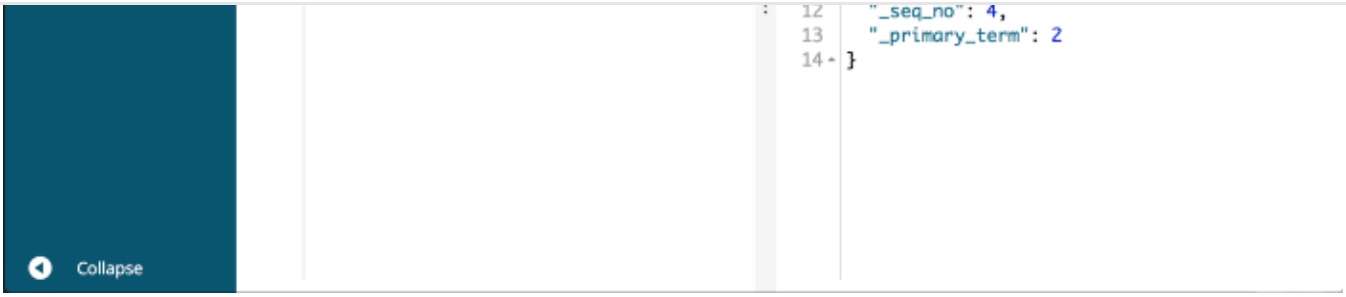
This retrieves the data you have just inserted.

DELETE

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

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





Searching Data

Ok, know you know some commands. But, it was presented in a very simple way. Getting a little deeper about this, you can execute more complex queries.

There are different Search APIs. To keep it simple, I 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.

I will use the accounts JSON file, available here.

To 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:

Simple Examples with URI Search

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.

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)
```

Simple examples using Query DSL

URI may not be the best way to query Elasticsearch. It seems to be preferable to use QueryDSL instead. Below follows a short description and an example about it, but you can read more about QueryDSL here.

Think of the Query DSL as an AST (Abstract Syntax Tree) of queries, consisting of two types of clauses:

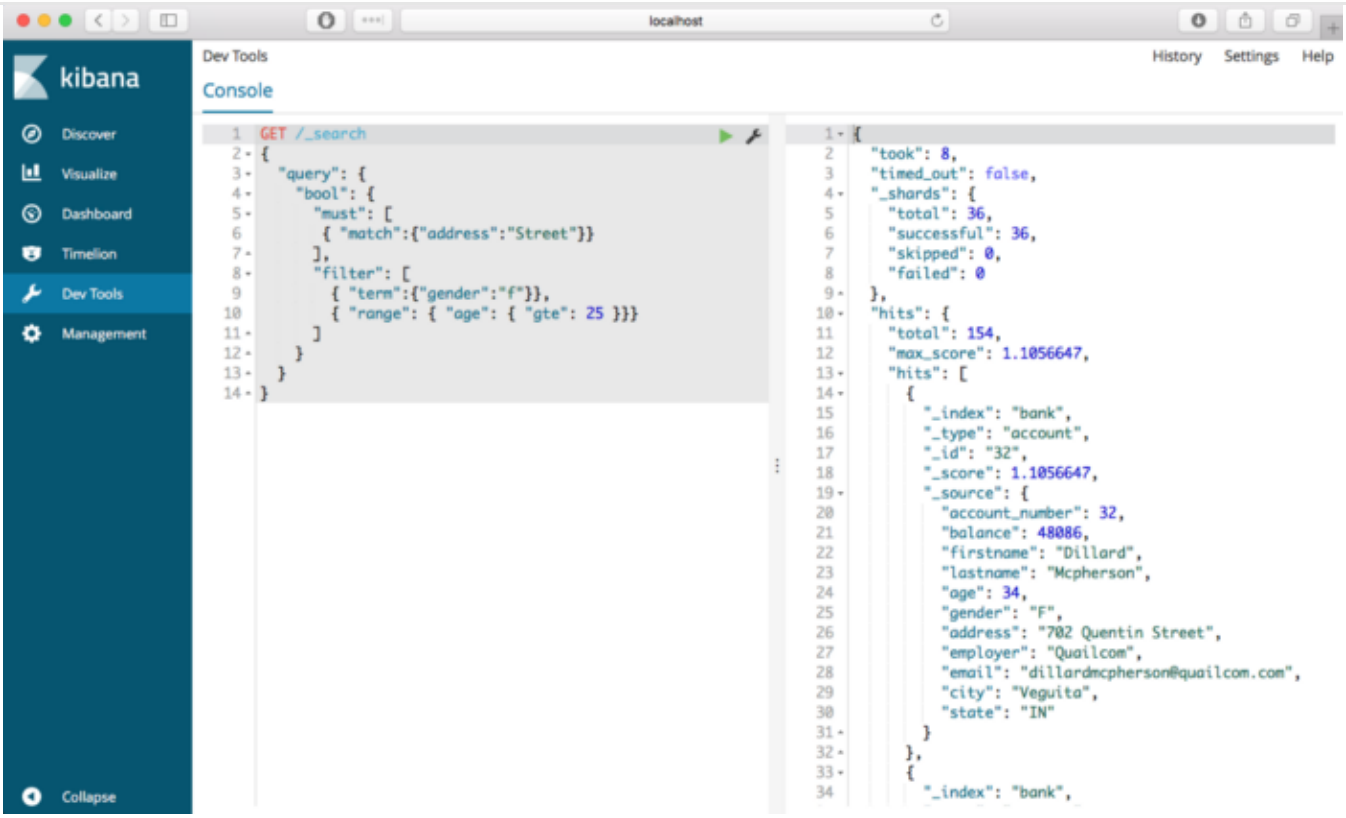
- **Leaf query clause:** it looks for a particular value in a particular field, such as the *match*, *term* or *range* queries.
- **Compound query clause:** it wrap other leaf or compound queries and is used to combine multiple queries in a logical fashion (such as the *bool* or *dis_max* query), or to alter their behaviour (such as the *constant_score* query).

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**




```
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.

documents.

> ***Tip:*** *Use query clauses in query context for conditions which should affect the score of matching documents (i.e. how well does the document match), and use all other query clauses in filter context.*

Now you know a bit about what is Elasticsearch and how to insert, update, delete and search data on it. Kibana has a lot of more features to view the data, including presenting it as different graphics. I recommend you to explore all of them.

There are much more about Elasticsearch. What I've presented here is just a first step to start understanding it, based on how I understood these topics in my first contact with Elasticsearch. After reading this story, you should explore Kibana interface, the Elasticsearch documentation, how to create more complex queries, and so on.

I hope this story helped you having your first contact with Elasticsearch, and that now you can read other tutorials and the documentation with less effort.

Thanks for reading and good luck with your studies :)

Edit:

I leave here my thanks to David Pilato, from the Elasticsearch community forum, who helped me indicating some improvements to this tutorial.

References

- Elasticsearch Reference
- URI Search
- Caelum — Buscas Eficientes com Elasticsearch (it is in portuguese)
- Elasticsearch introduction
- Search APIs

Elasticsearch Kibana Big Data Big Data Analytics Restful Api