Elastic Search

Summary	In this codelab, we will learn how to create Synthetic Data using Snowflake
URL	https://www.elastic.co/webinars/getting-started-elasticsearch
Category	Web
Status	Published
Author	Nikhil Kohli

Getting Started with ElasticSearch

Getting Started with ElasticSearch

> ***"Elasticsearch is a search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is developed in Java."***

What Is Elasticsearch?

Elasticsearch is a distributed, open source search and analytics engine for all types of data, including textual, numerical, geospatial, structured, and unstructured. Known for its simple REST APIs, distributed nature, speed, and scalability, Elasticsearch is the central component of the Elastic Stack, a set of open source tools for data ingestion, enrichment, storage, analysis, and visualization.

ELK Stack

Commonly referred to as the ELK Stack (after Elasticsearch, Logstash, and Kibana), the Elastic Stack now includes a rich collection of lightweight shipping agents known as Beats for sending data to Elasticsearch.

> *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.*

Logstash is a light-weight, open-source, server-side data processing pipeline that allows you to collect data from a variety of sources, transform it on the fly, and send it to your desired

destination. It is most often used as a data pipeline for Elasticsearch, an open-source analytics and search engine

Kibana is an open source data visualization dashboard for Elasticsearch. It provides visualization capabilities on top of the content indexed on an Elasticsearch cluster

What it is Used For ?

The speed and scalability of Elasticsearch and its ability to index many types of content mean that it can be used for a number of use cases:

- Application search
- Website search
- Enterprise search
- Logging and log analytics
- Infrastructure metrics and container monitoring
- Application performance monitoring
- Geospatial data analysis and visualization
- Security analytics
- Business analytics

Basic Concepts

Basic Concepts

Elasticsearch is made of below entities -

Cluster

A cluster is a collection of one or more nodes that, together, holds the entire data.

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.

An **index** is broken into **shards** in order to distribute them and scale. **Replicas** are copies of the shards. A **node** is a running instance of elastic search which belongs to a **cluster**. A cluster consists of one or more **nodes** which share the same cluster name

---

Installation

Install Postman

[Postman Download]

Install ElasticSearch

Elasticsearch

To download and install Elasticsearch, open a terminal window and use the commands that work with your system

mac:

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

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

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

deb:

```
curl -L -O
https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.7.1-amd64.deb

sudo dpkg -i elasticsearch-7.7.1-amd64.deb
```

```
sudo /etc/init.d/elasticsearch start
```

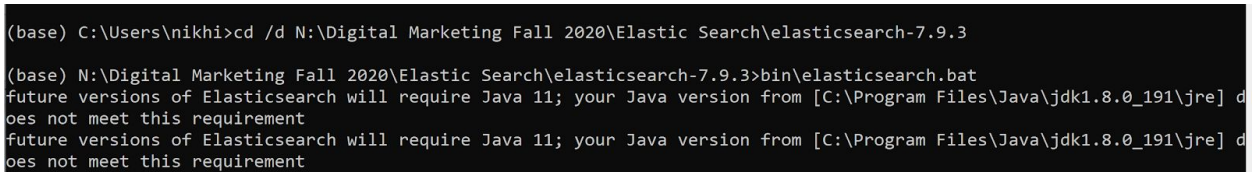
linux:

```
curl -L -O
https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.7.1-li
nux-x86_64.tar.gz
tar -xzvf elasticsearch-7.7.1-linux-x86_64.tar.gz
cd elasticsearch-7.7.1
./bin/elasticsearch
```

win:

1.

```
Download the Elasticsearch 7.7.1 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, for example:
```



Make sure Elasticsearch is up and running

To test that the Elasticsearch daemon is up and running, try sending an HTTP GET request on port 9200.

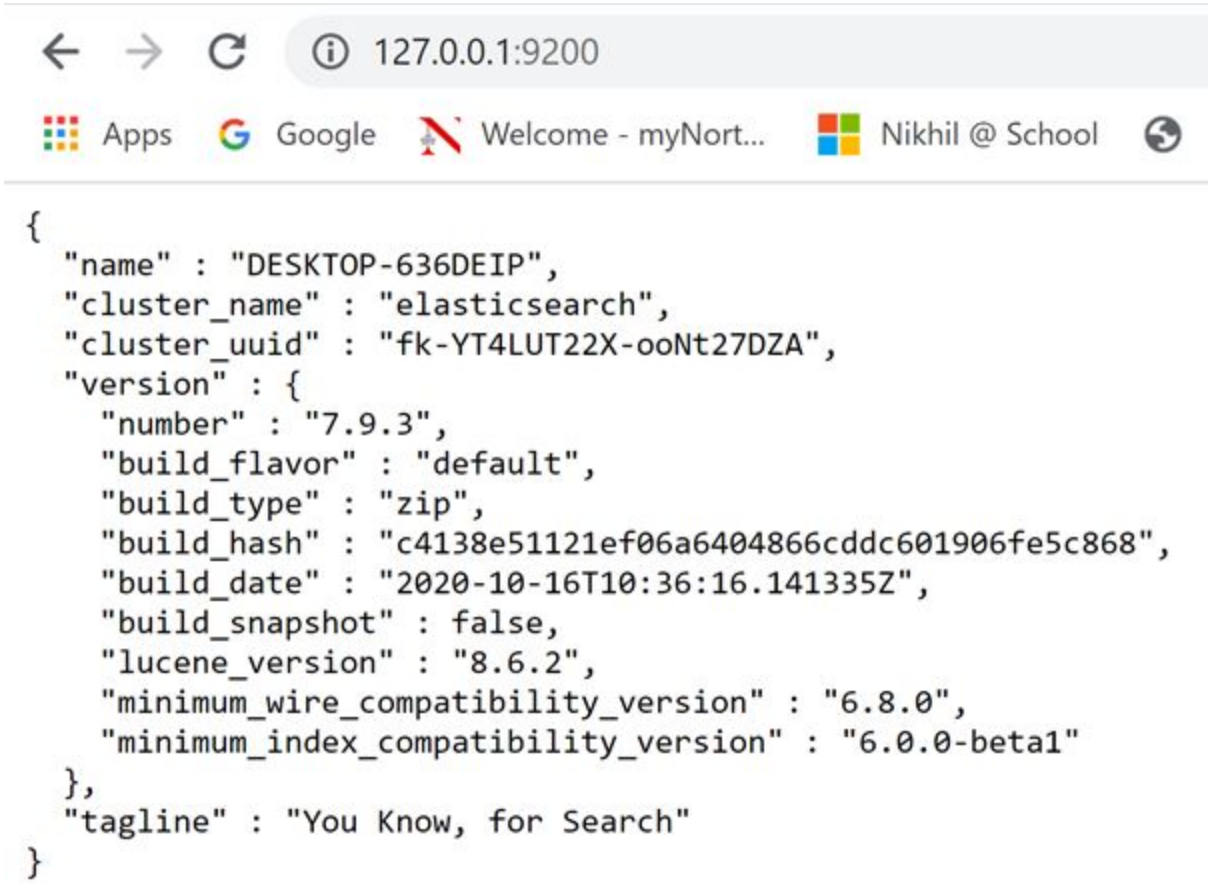
On Anaconda prompt go to

```
cd C:\ProgramFiles\elasticsearch-7.7.1
```

```
Bin\elasticsearch.bat
```

Open http://127.0.0.1:9200

You should see a response similar to this:



Install Kibana (Same way)

Kibana Download

mac:

```
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
```

win:

1.

```
Download the Kibana 7.7.1 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, for example:
```

brew:

```
brew tap elastic/tap

brew install elastic/tap/kibana-full

kibana
```

deb, rpm, or linux:

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

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

cd kibana-7.7.1-linux-x86_64/

./bin/kibana
```

Run Kibana

```
cd C:\Program

Files\kibana-7.7.1-window

s Bin\kibana.bat
```

http://localhost:5601

Using ElasticSearch (CRUD)

CRUD or Indexing

While we may want to use ElasticSearch primarily for searching, the first step is to populate an index with some data, meaning the "Create" of CRUD, or rather, "indexing". While we're at it we'll also look at how to update, read and delete individual documents

Let's create an *index* and add a few *documents* in it. *Index* is equivalent to a 'database' in RDBMS while a *document* resembles a 'row'.

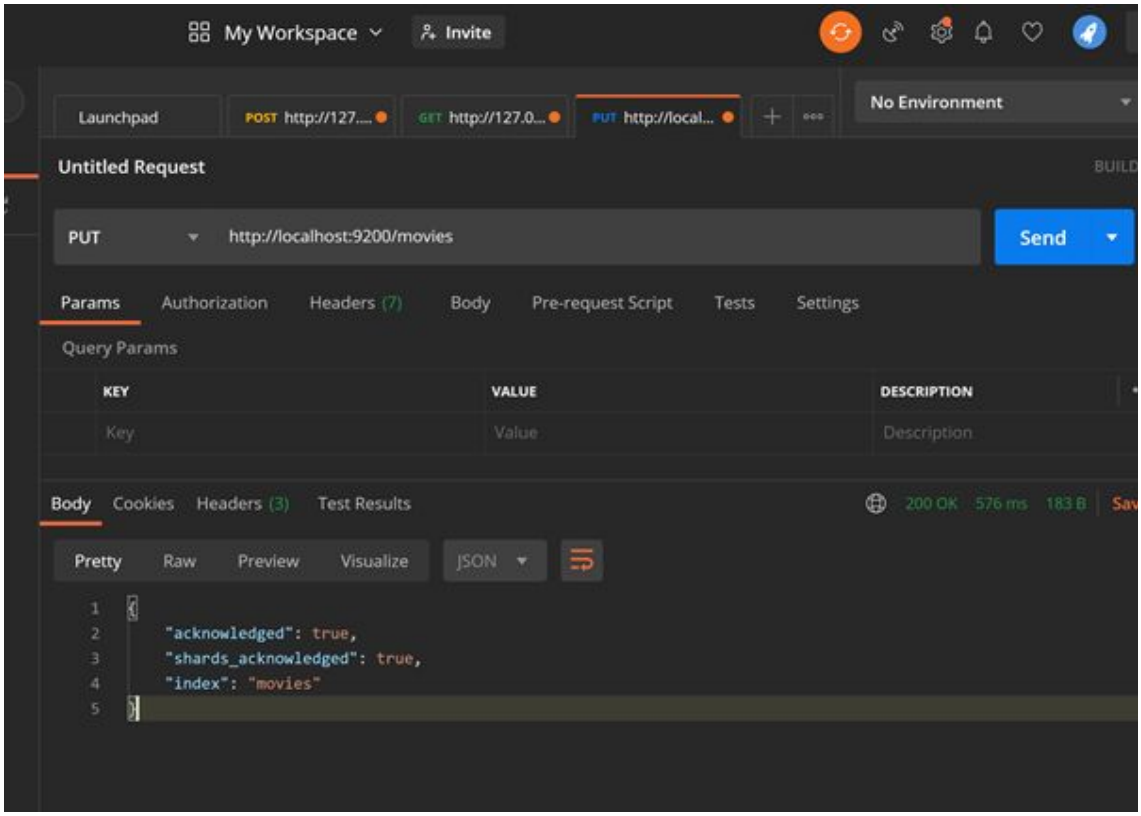
We will create an index named *movies and series with types - movie and netflix*.

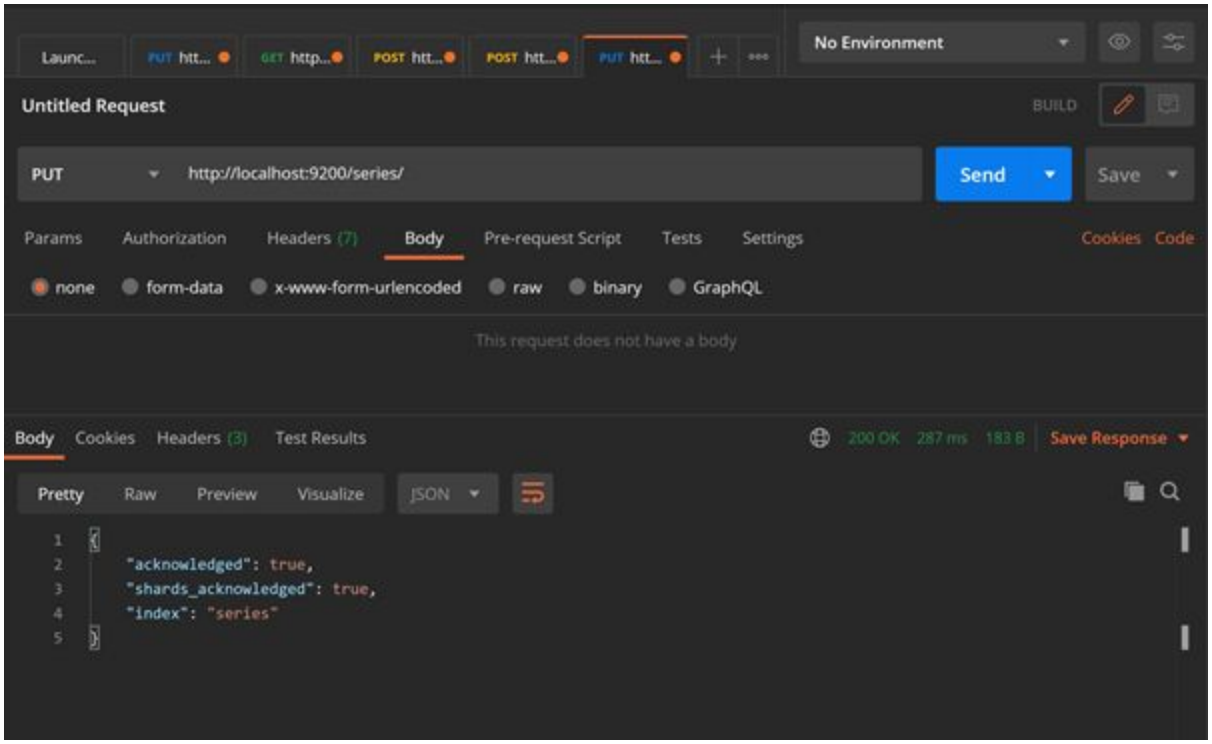
```
Fire a PUT request from Postman
http://localhost:9200/movies

http://localhost:9200/series

or you can use the cURL command
curl -X PUT "http://localhost:9200/movies/"
```

Output

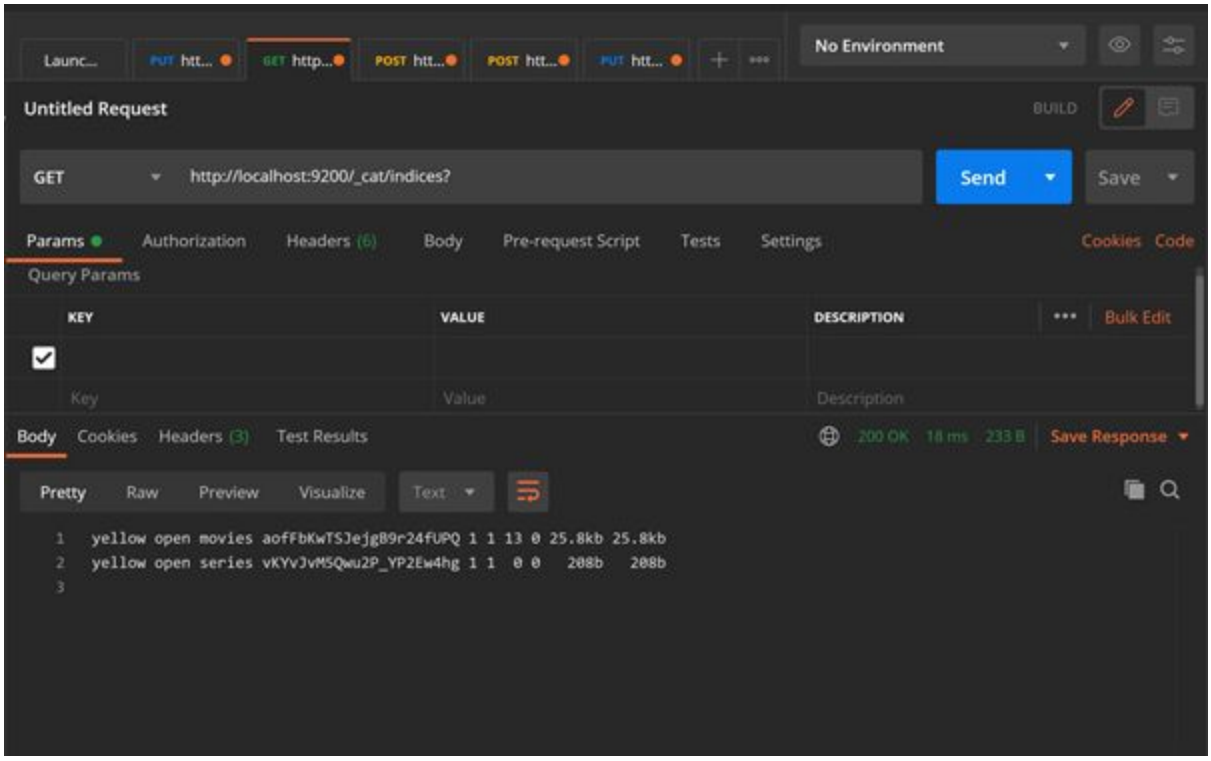

My Workspace
Invite
Launchpad
POST http://127....
GET http://127.0...
PUT http://local...
No Environment
Untitled Request
PUT
http://localhost:9200/movies
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Query Params
KEY
VALUE
DESCRIPTION
Key
Value
Description
Body
Cookies
Headers (3)
Test Results
200 OK
576 ms
183 B
Pretty
Raw
Preview
Visualize
JSON
{
"acknowledged": true,
"shards_acknowledged": true,
"index": "movies"
}



To check all the indexes that have been created invoke a GET request

```
http://localhost:9200/_cat/indices?

or use the cURL command
curl -X GET "http://localhost:9200/_cat/indices"
```

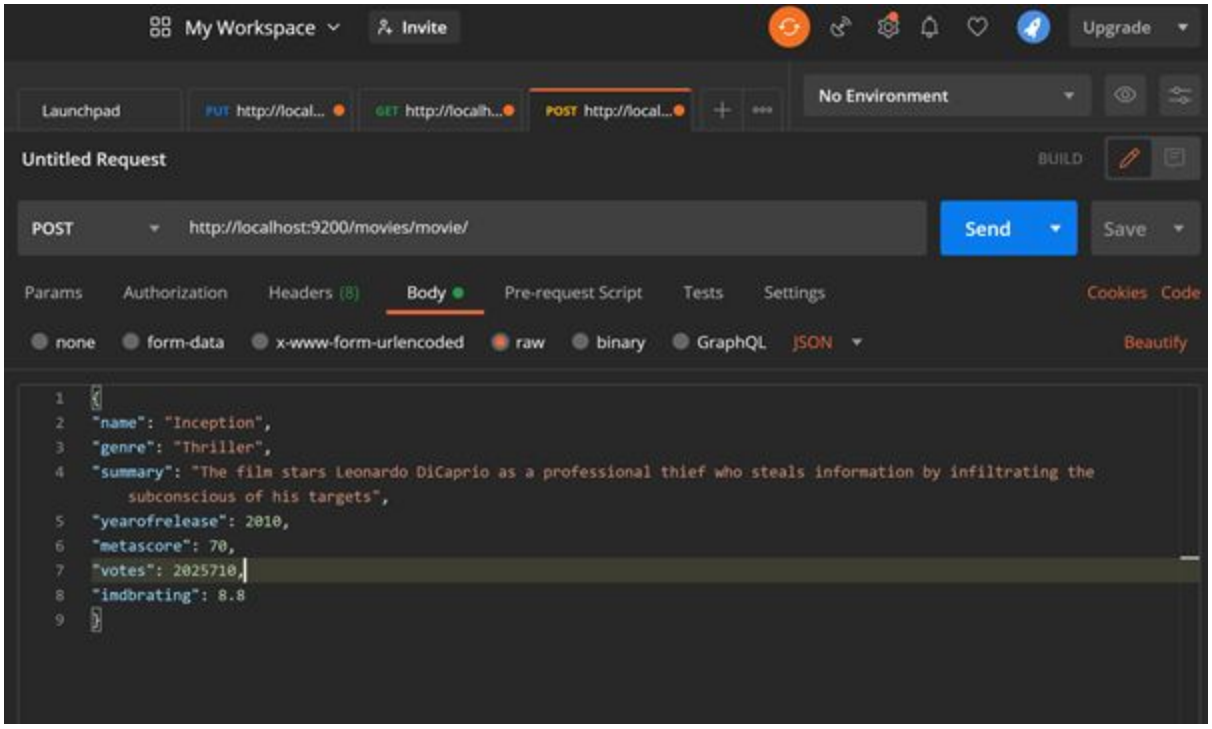


To insert records in the database fire a POST request

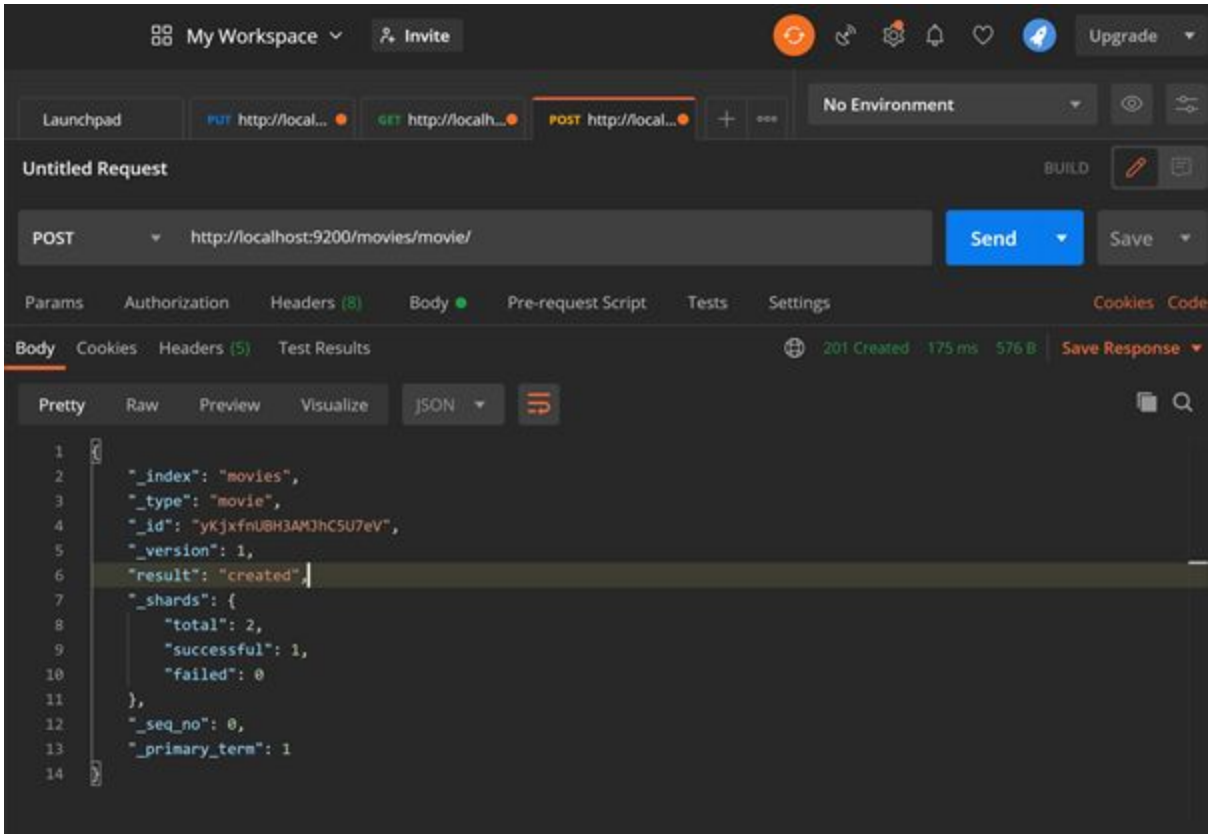
```
http://localhost:9200/movies/movie/
```

and pass the record in JSON format in the body.

Make sure to set the `Content-Type` as `application/json` in the Headers section in the HTTP section.



Output Response

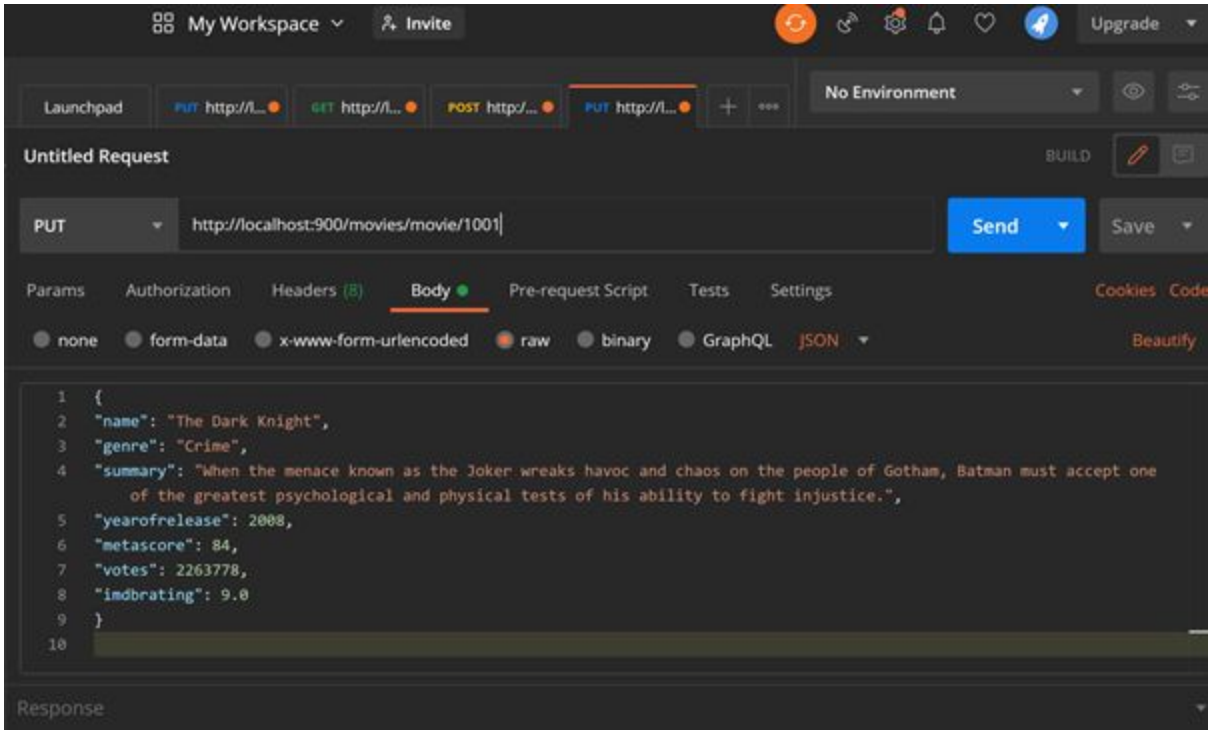


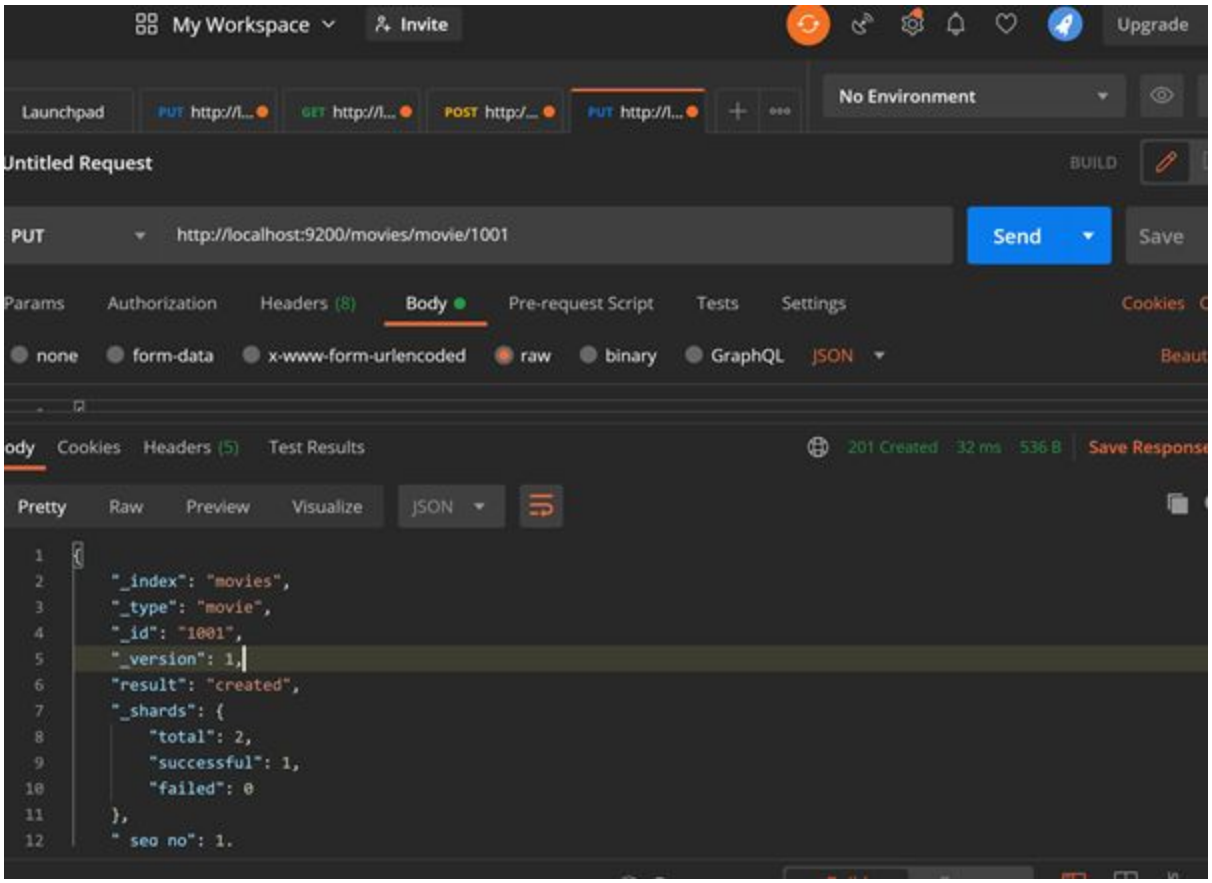
Observe that the `result` field is set to `created`. We can see from the response that elasticsearch assigned an `_id` automatically.

We can specify our own `_id` by invoking a PUT request

```
http://localhost:900/movies/movie/<id>

http://localhost:9200/movies/movie/1001
```

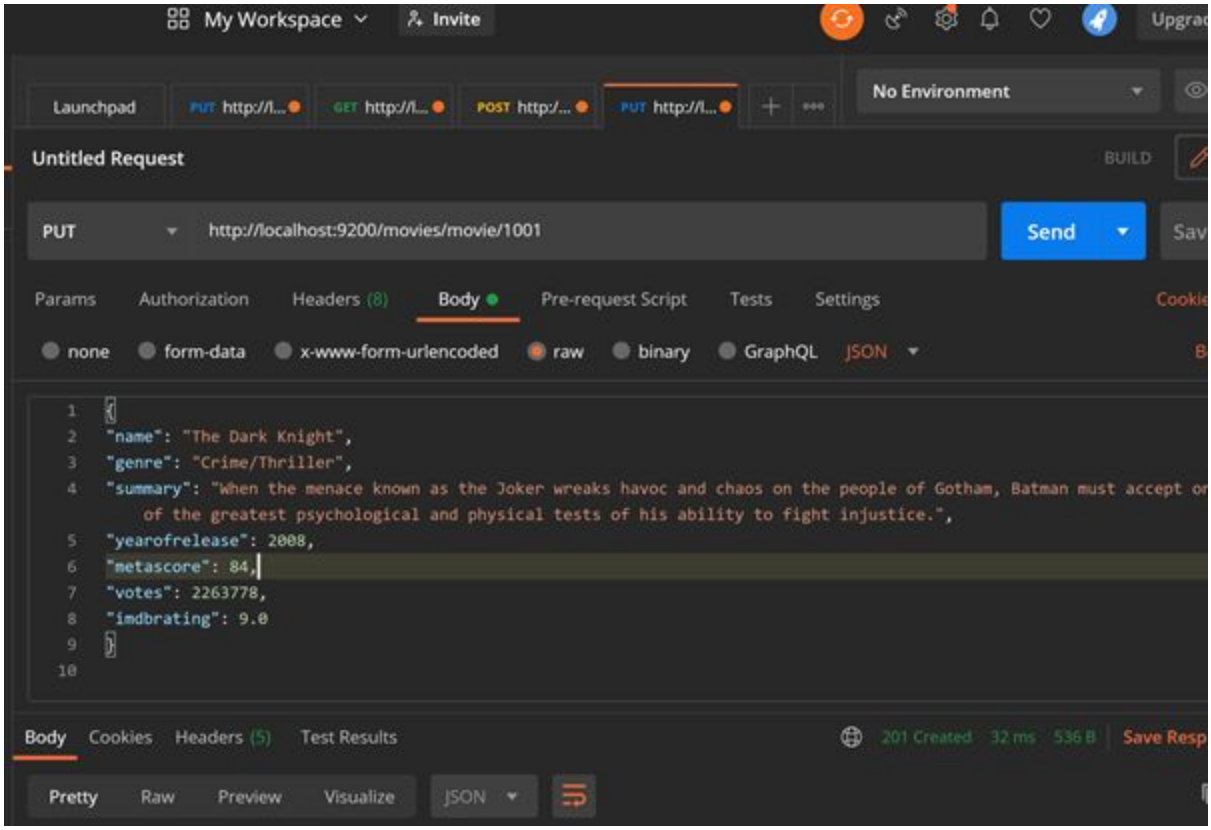

My Workspace
Invite
Upgrade
Launchpad
No Environment
Untitled Request
BUILD
PUT
http://localhost:900/movies/movie/1001
Send
Save
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"name": "The Dark Knight",
"genre": "Crime",
"summary": "When the menace known as the Joker wreaks havoc and chaos on the people of Gotham, Batman must accept one of the greatest psychological and physical tests of his ability to fight injustice.",
"yearofrelease": 2008,
"metascore": 84,
"votes": 2263778,
"imdbrating": 9.0
}
Response


My Workspace
Invite
Upgrade
Launchpad
No Environment
Untitled Request
BUILD
PUT
http://localhost:9200/movies/movie/1001
Send
Save
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Cookies
Headers (5)
Test Results
201 Created
32 ms
536 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"_index": "movies",
"_type": "movie",
"_id": "1001",
"_version": 1,
"result": "created",
"_shards": {
"total": 2,
"successful": 1,
"failed": 0
},
" seq no": 1.

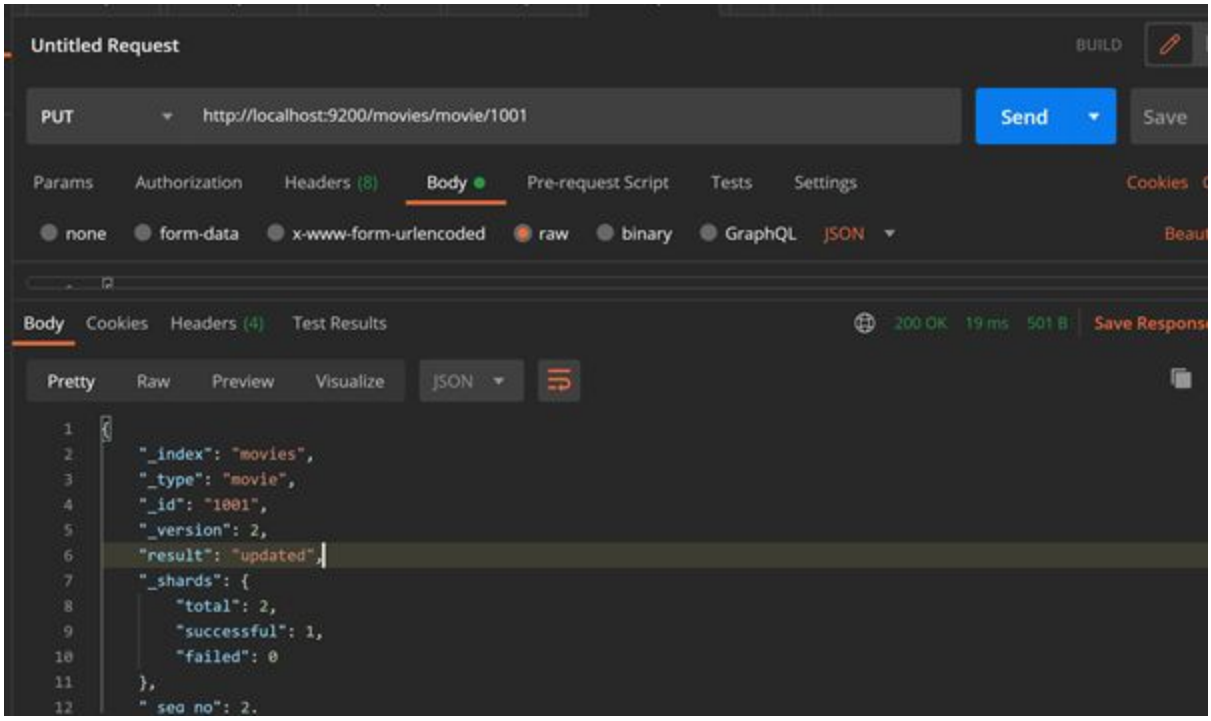
UPDATE & DELETE

To update the record, just change the JSON and fire the same request.

(Put request will update the document, Post request will create another document)

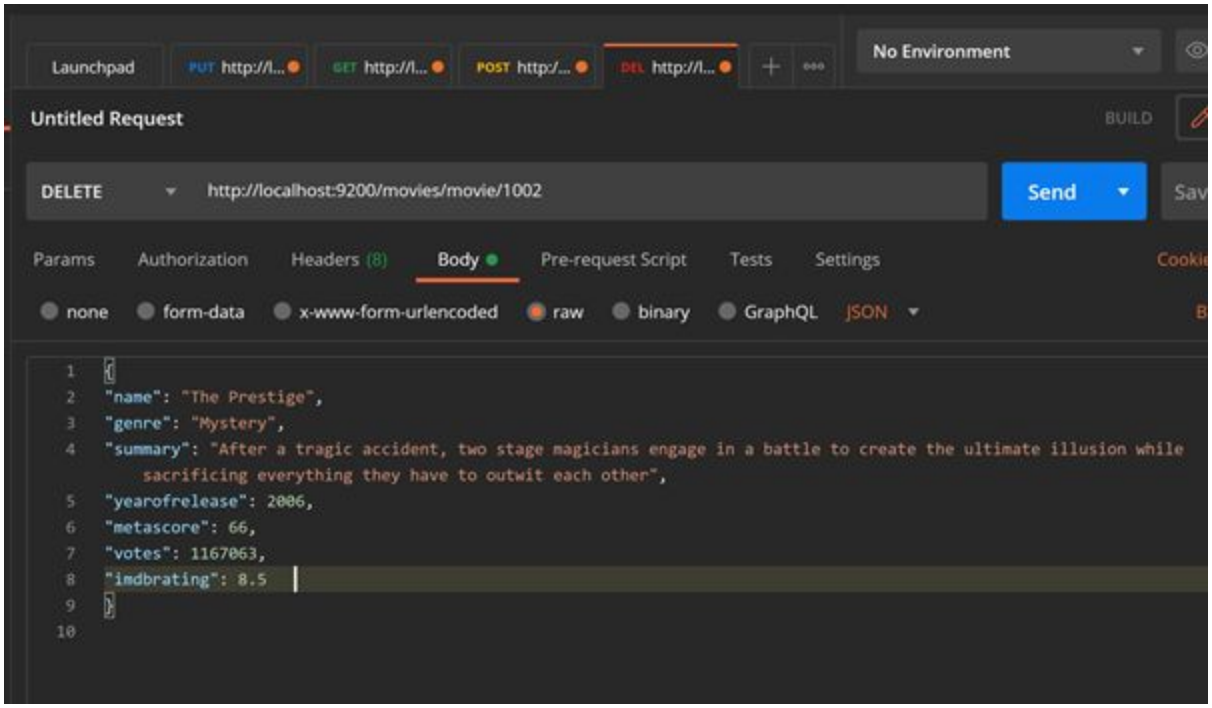


Response

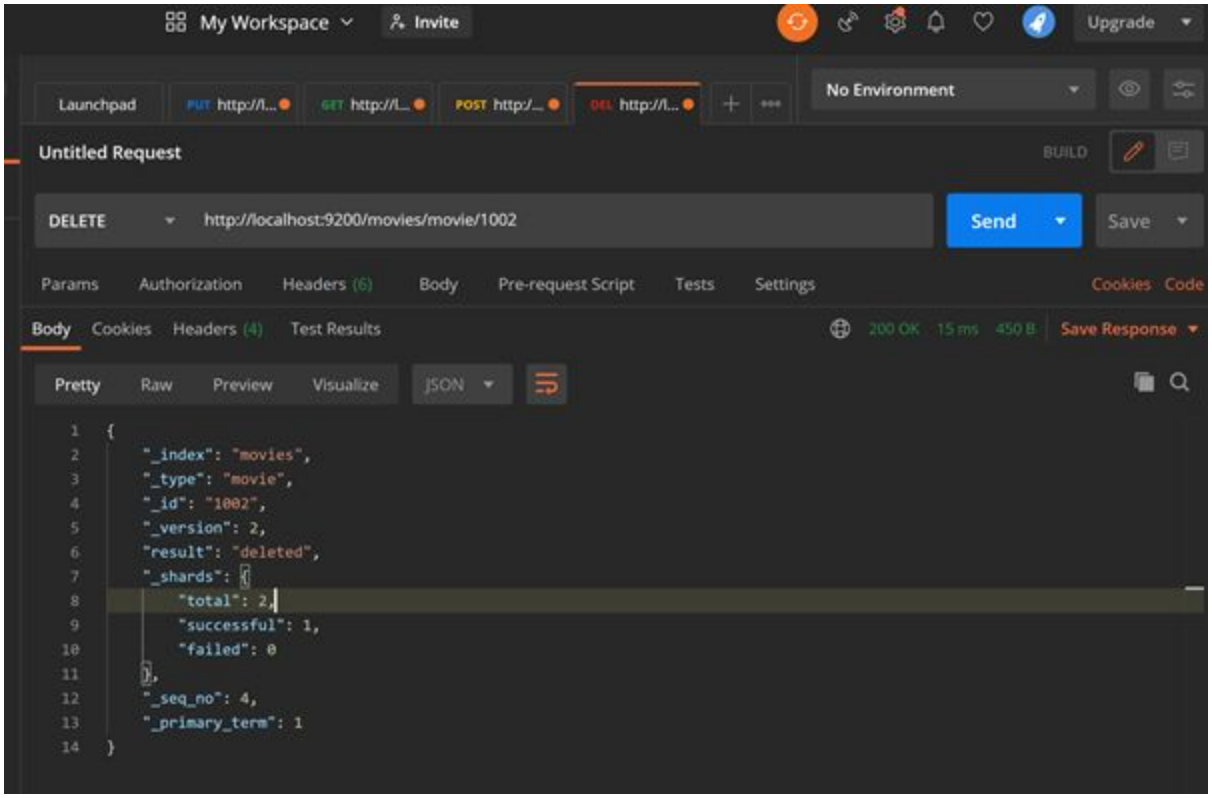


You can delete a particular record by passing the id and invoking the DELETE request

```
http://localhost:9200/movies/movie/1001
```


Launchpad
No Environment
Untitled Request
BUILD
DELETE
http://localhost:9200/movies/movie/1002
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"name": "The Prestige",
"genre": "Mystery",
"summary": "After a tragic accident, two stage magicians engage in a battle to create the ultimate illusion while sacrificing everything they have to outwit each other",
"yearofrelease": 2006,
"metascore": 66,
"votes": 1167063,
"imdbrating": 8.5
}

Response


My Workspace
Invite
Upgrade
Launchpad
No Environment
Untitled Request
BUILD
DELETE
http://localhost:9200/movies/movie/1002
Send
Save
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Code
Body
Cookies
Headers (4)
Test Results
200 OK
15 ms
450 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"_index": "movies",
"_type": "movie",
"_id": "1002",
"_version": 2,
"result": "deleted",
"_shards": {
"total": 2,
"successful": 1,
"failed": 0
},
"_seq_no": 4,
"_primary_term": 1
}

BULK API

Let’s insert few more documents into our index so that we can execute search queries

```
{
"name":"Thor Ragnarok",
"genre":"Action",
"summary":"Thor is imprisoned on the planet Sakaar, and must race against time to return to Asgard and stop Ragnarök, the destruction of his world at the hands of the powerful and ruthless villain Hela",
"yearofrelease":2017,
"metascore":74,
"votes":374270,
"imdbrating":7.9
}
{
"name":"Infinity War",
"genre":"Sci-Fi",
"summary":"The Avengers and their allies must be willing to sacrifice all in an attempt to defeat the powerful Thanos before his blitz of devastation and ruin puts an end to the universe",
"yearofrelease":2018,
"metascore":68,
"votes":450856,
"imdbrating":8.6
}
{
"name":"Christopher Robin",
"genre":"Comedy",
"summary":"A working-class family man, Christopher Robin, encounters his childhood friend Winnie the-Pooh, who helps him to  rediscover the joys of life",
"yearofrelease":2018,
"metascore":60,
"votes":9648,
"imdbrating":7.9
}
```

You can insert the above documents one by one using the POST request as we did earlier or use the bulk API provided by elasticsearch. The bulk API lets you insert multiple documents at once.

We can perform insert, update and delete all in one request with this API

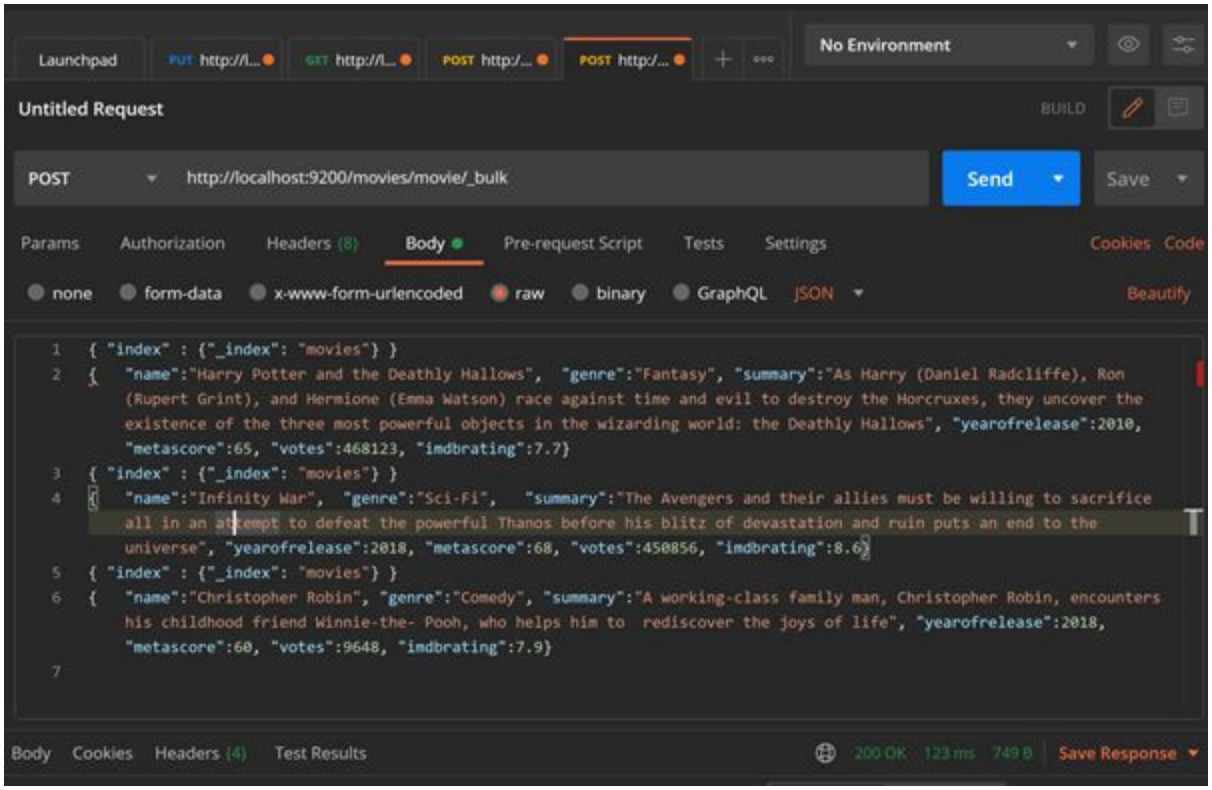
Invoke a POST request

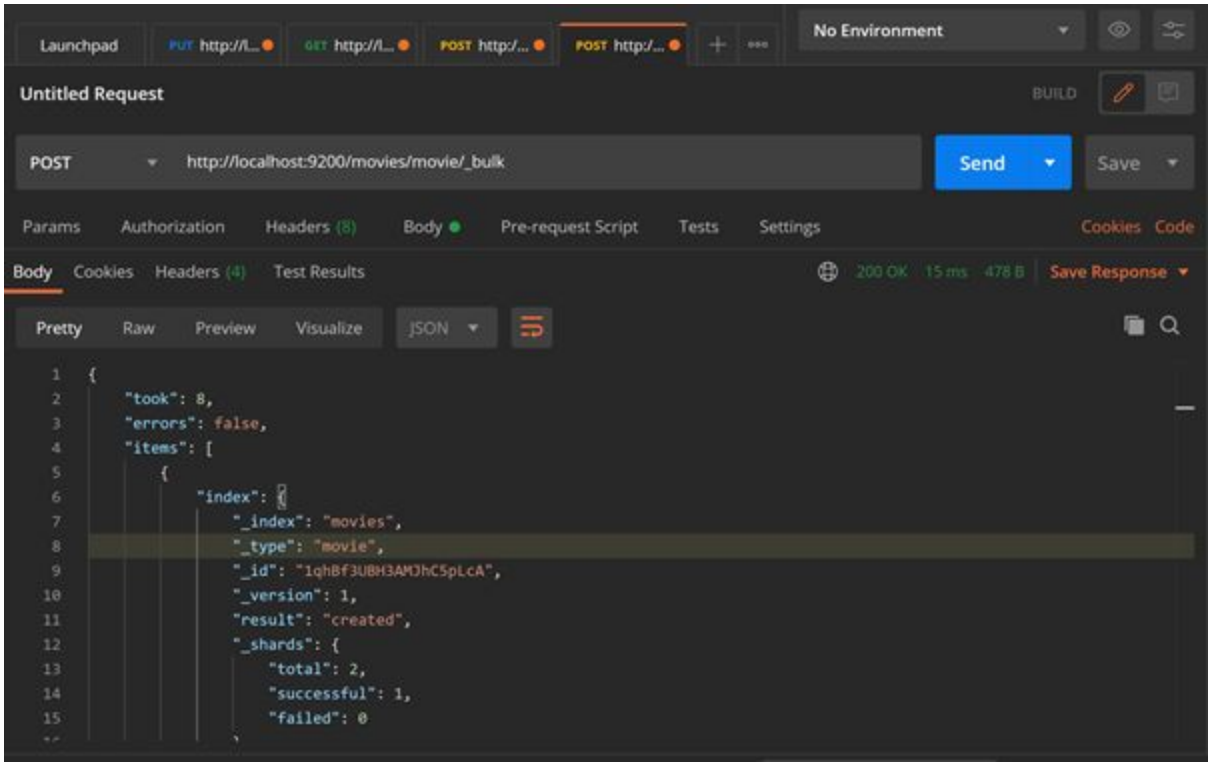
```
http://localhost:9200/movies/movie/_bulk
```

with the request body as given below

```
{ "index" : {"_index": "movies"} }
{     "name":"Harry Potter and the Deathly Hallows",  "genre":"Fantasy",
"summary":"As Harry Ron and Hermione race against time and evil to destroy
the Horcruxes they uncover the existence of the three most powerful objects
in the wizarding world: the Deathly Hallows", "yearofrelease":2010,
"metascore":65, "votes":468123, "imdbrating":7.7}
{ "index" : {"_index": "movies"} }
{     "name":"Infinity War",  "genre":"SciFi",  "summary":"The Avengers and
their allies must be willing to sacrifice all in an attempt to defeat the
powerful Thanos before his blitz of devastation and ruin puts an end to the
universe", "yearofrelease":2018, "metascore":68, "votes":450856,
"imdbrating":8.6}
{ "index" : {"_index": "movies"} }
{     "name":"Christopher Robin",   "genre":"Comedy", "summary":"A working
class family man Christopher Robin encounters his childhood friend Winnie
the Pooh who helps him to rediscover the joys of life",
"yearofrelease":2018, "metascore":60, "votes":9648, "imdbrating":7.9}
```

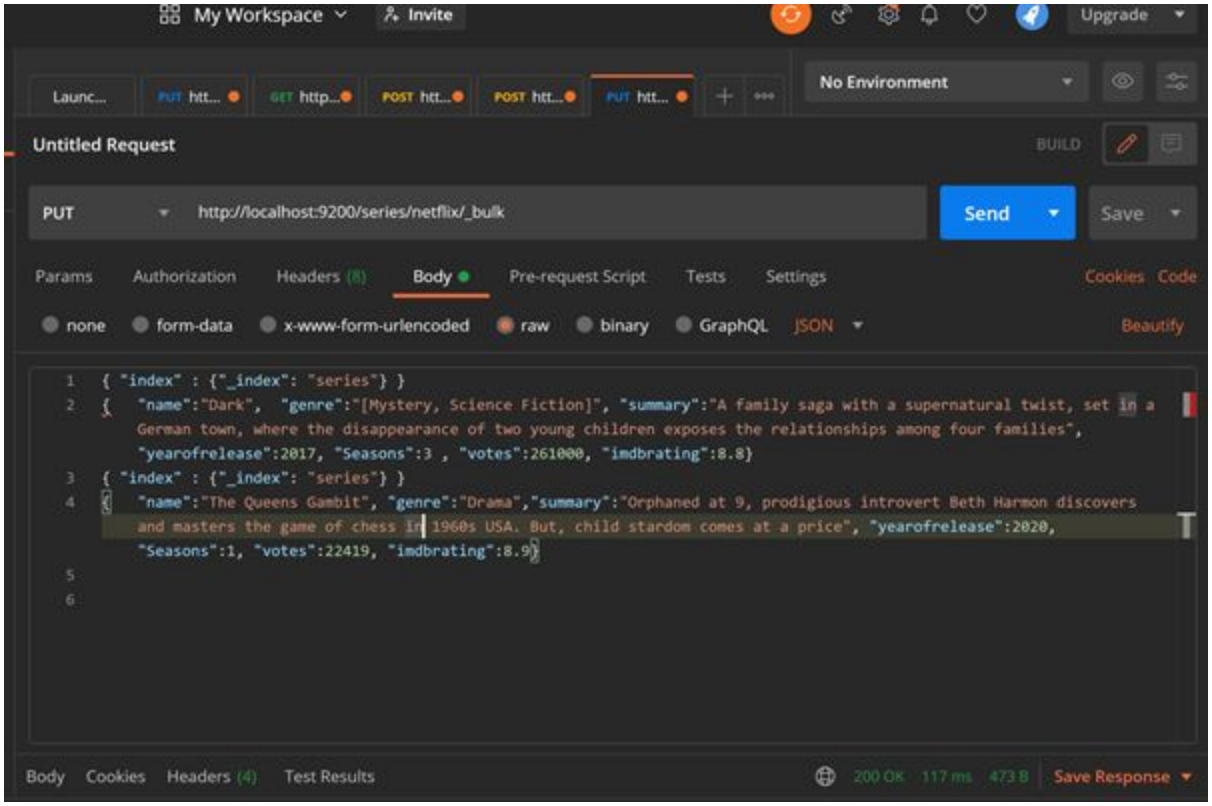
{"index" :{“_index”: “Index name”}} defines the operation on the document. Here it means that we are indexing the document. You can use delete and update as well.
For each record we have to define the operation(index, update or delete) and then the document.


Launchpad
PUT http://l...
GET http://l...
POST http:/...
POST http:/...
No Environment
Untitled Request
BUILD
POST
http://localhost:9200/movies/movie/_bulk
Send
Save
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Code
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{ "index" : {"_index": "movies"} }
{ "name":"Harry Potter and the Deathly Hallows", "genre":"Fantasy", "summary":"As Harry (Daniel Radcliffe), Ron (Rupert Grint), and Hermione (Emma Watson) race against time and evil to destroy the Horcruxes, they uncover the existence of the three most powerful objects in the wizarding world: the Deathly Hallows", "yearofrelease":2010, "metascore":65, "votes":468123, "imdbrating":7.7}
{ "index" : {"_index": "movies"} }
{ "name":"Infinity War", "genre":"Sci-Fi", "summary":"The Avengers and their allies must be willing to sacrifice all in an attempt to defeat the powerful Thanos before his blitz of devastation and ruin puts an end to the universe", "yearofrelease":2018, "metascore":68, "votes":450856, "imdbrating":8.6}
{ "index" : {"_index": "movies"} }
{ "name":"Christopher Robin", "genre":"Comedy", "summary":"A working-class family man, Christopher Robin, encounters his childhood friend Winnie-the- Pooh, who helps him to rediscover the joys of life", "yearofrelease":2018, "metascore":60, "votes":9648, "imdbrating":7.9}
Body
Cookies
Headers (4)
Test Results
200 OK
123 ms
749 B
Save Response


Launchpad
PUT http://l...
GET http://l...
POST http:/...
POST http:/...
No Environment
Untitled Request
BUILD
POST
http://localhost:9200/movies/movie/_bulk
Send
Save
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Code
Body
Cookies
Headers (4)
Test Results
200 OK
15 ms
478 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"took": 8,
"errors": false,
"items": [
{
"index": {
"_index": "movies",
"_type": "movie",
"_id": "1qhBf3UBH3AMJhC5pLcA",
"_version": 1,
"result": "created",
"_shards": {
"total": 2,
"successful": 1,
"failed": 0

Create a new type Netflix in Series index and bulk upload below

```
{ "index" : {"_index": "series"} }
{     "name":"Dark",    "genre":"[Mystery, Science Fiction]", "summary":"A family saga with a supernatural twist, set in a German town, where the disappearance of two young children exposes the relationships among four families", "yearofrelease":2017, "Seasons":3 , "votes":261000, "imdbrating":8.8}
{ "index" : {"_index": "series"} }
{     "name":"The Queens Gambit",   "genre":"Drama","summary":"Orphaned at 9, prodigious introvert Beth Harmon discovers and masters the game of chess in 1960s USA. But, child stardom comes at a price", "yearofrelease":2020, "Seasons":1, "votes":22419, "imdbrating":8.9}
```



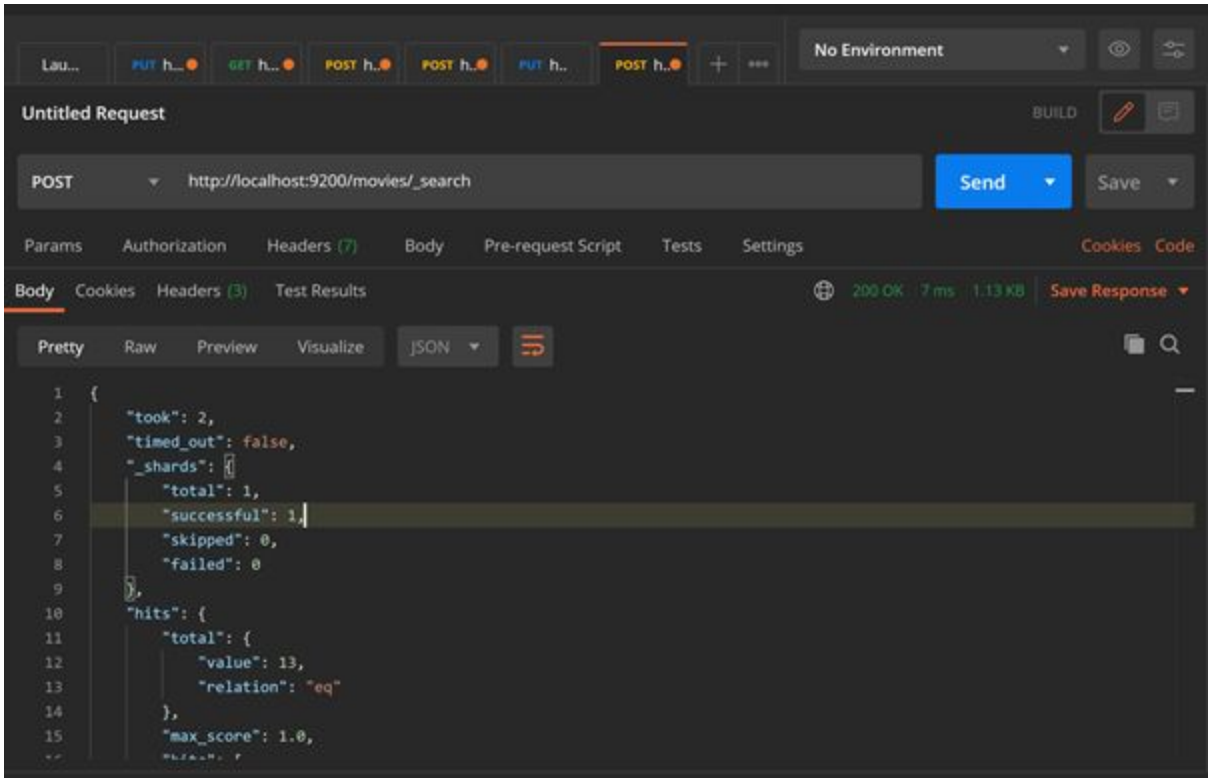
ElasticSearch (Search & Filtering)

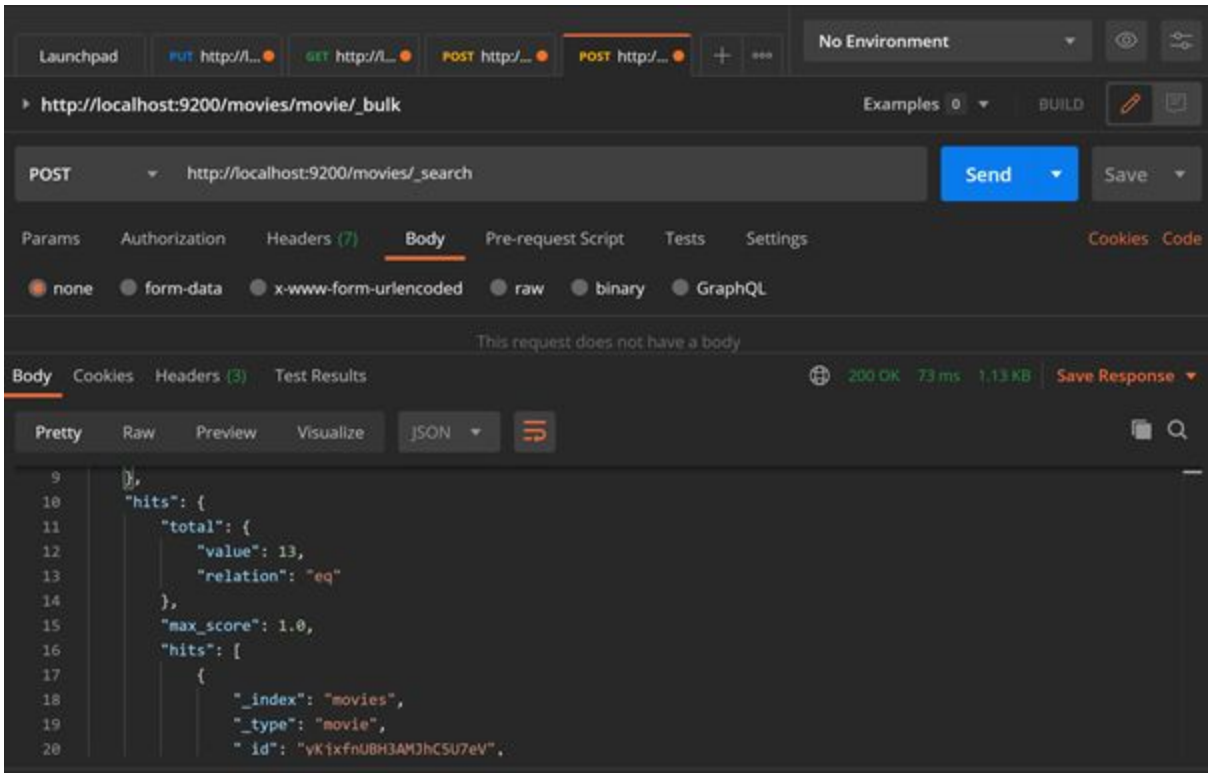
Searching

Elasticsearch is a very fast and powerful search engine. The _search API allows you to execute a search query against the index and get back the results that best match the query.

To get all the documents fire a GET request from the postman client

```
http://localhost:9200/movies/_search
```





Let's understand the response:

`took`

is the time taken by elasticsearch to return the results.

`_shards`

is the number of shards that were searched. Basically, each index is split into multiple shards.

`hits.total`

is the total number of records matched for the search query

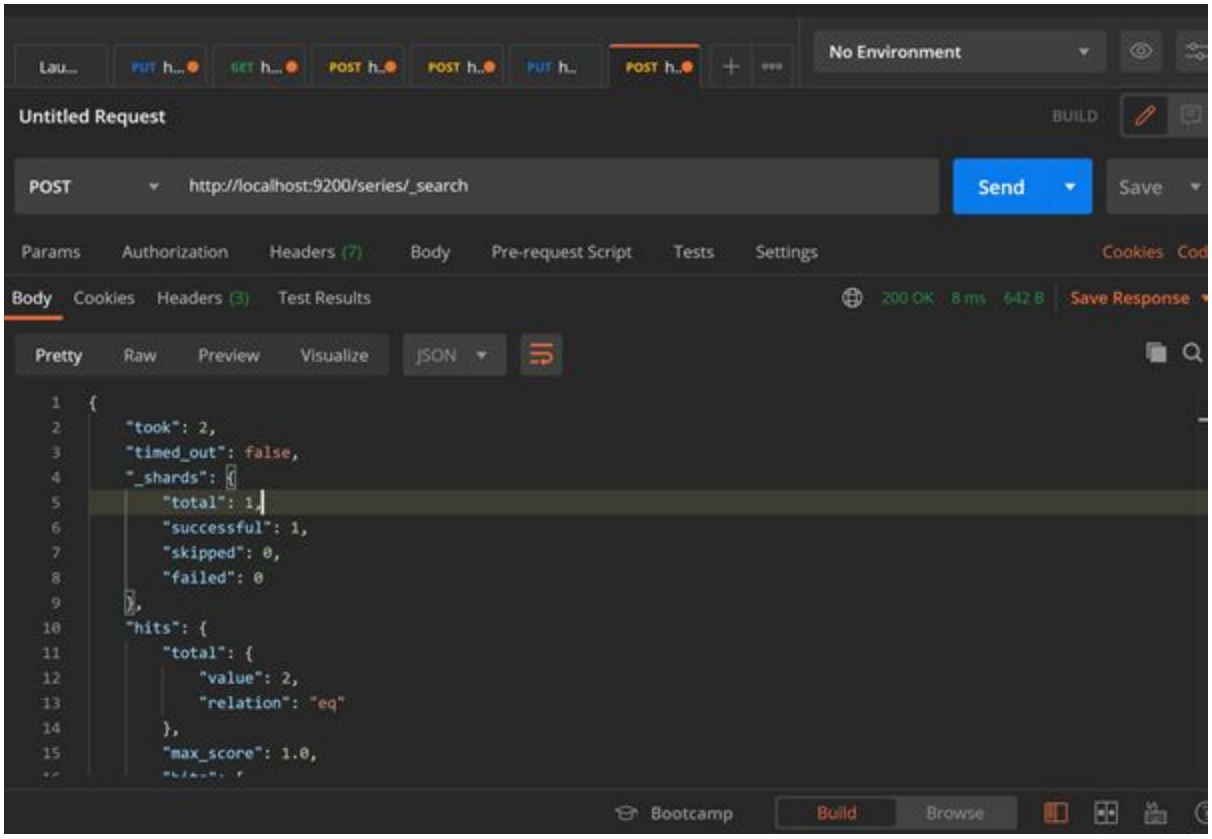
`hits.maxscore`

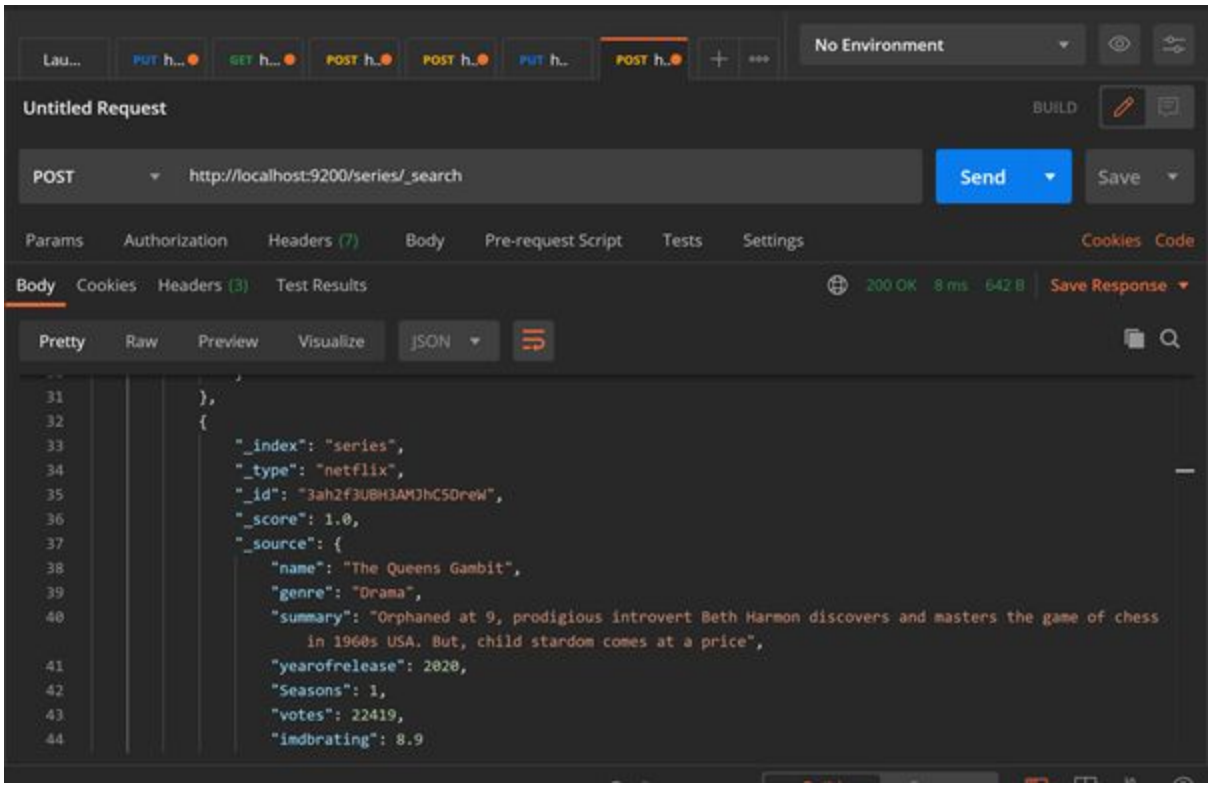
The score of the document that is the most relevant

`hits.hits`

the search results returned by elasticsearch

Search with a different Index - Series

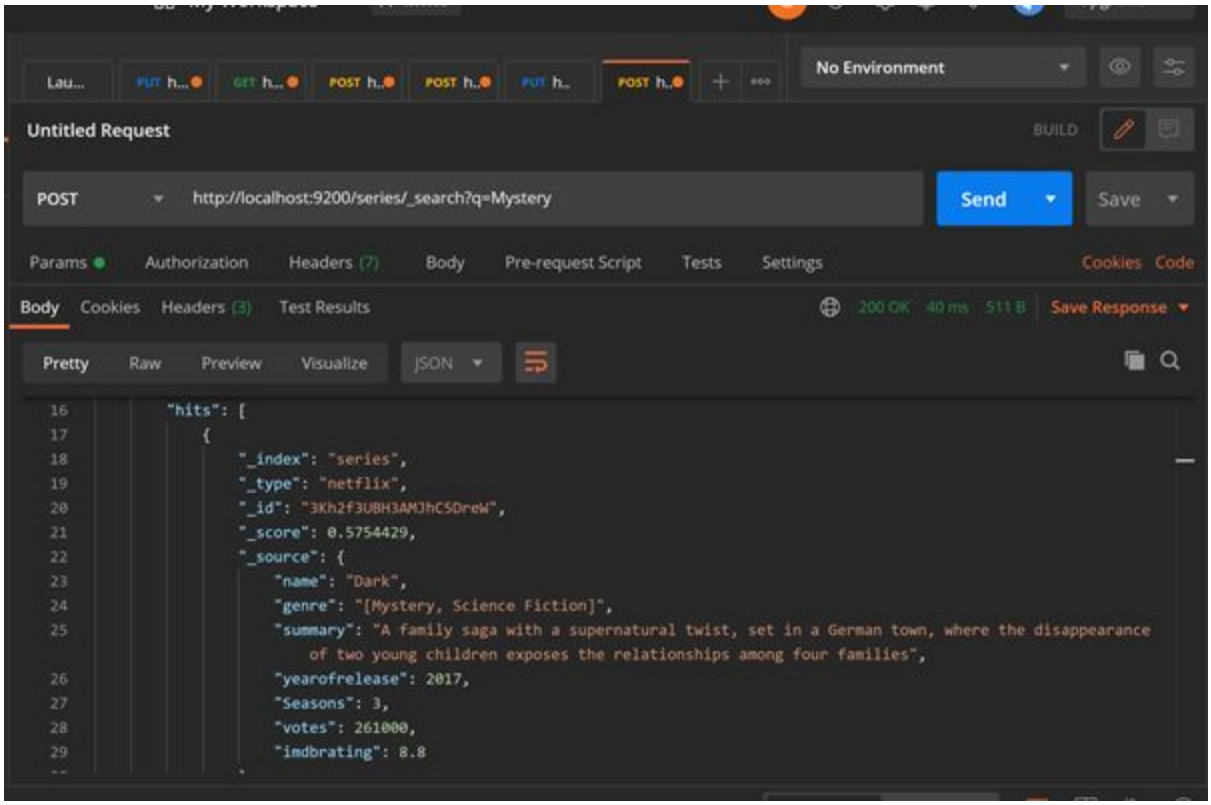

No Environment
Untitled Request
BUILD
POST
http://localhost:9200/series/_search
Send
Save
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies
Headers (3)
Test Results
200 OK
8 ms
642 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"took": 2,
"timed_out": false,
"_shards": {
"total": 1,
"successful": 1,
"skipped": 0,
"failed": 0
},
"hits": {
"total": {
"value": 2,
"relation": "eq"
},
"max_score": 1.0,
Bootcamp
Build
Browse


No Environment
Untitled Request
BUILD
POST
http://localhost:9200/series/_search
Send
Save
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Code
Body
Cookies
Headers (3)
Test Results
200 OK
8 ms
642 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
},
{
"_index": "series",
"_type": "netflix",
"_id": "3ah2f3UBH3AMJhC5DreW",
"_score": 1.0,
"_source": {
"name": "The Queens Gambit",
"genre": "Drama",
"summary": "Orphaned at 9, prodigious introvert Beth Harmon discovers and masters the game of chess in 1960s USA. But, child stardom comes at a price",
"yearofrelease": 2020,
"Seasons": 1,
"votes": 22419,
"imdbrating": 8.9

Search with Query

If you want to look for records that contain the word `Mystery` , use the query

```
http://localhost:9200/series/_search?q=Mystery
```



Elasticsearch will look for the word `Mystery` against all the fields in the document. The query returned 2 records

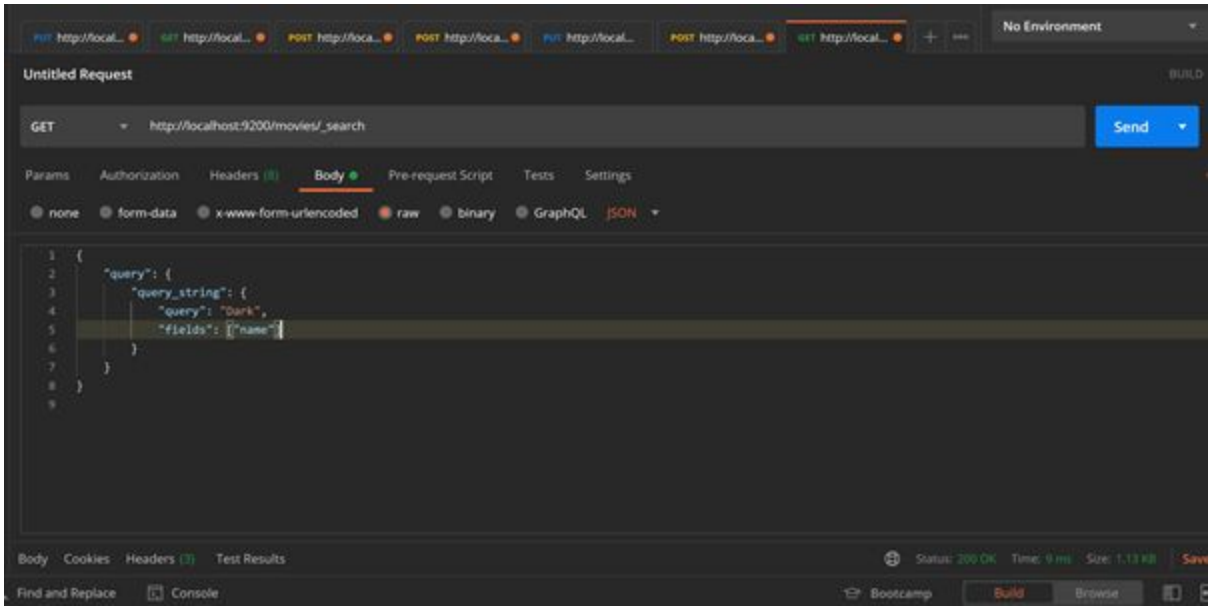
Filter or Drill Down Search with field Name & other attributes -

Include field ‘name’ – query for ‘Dark’

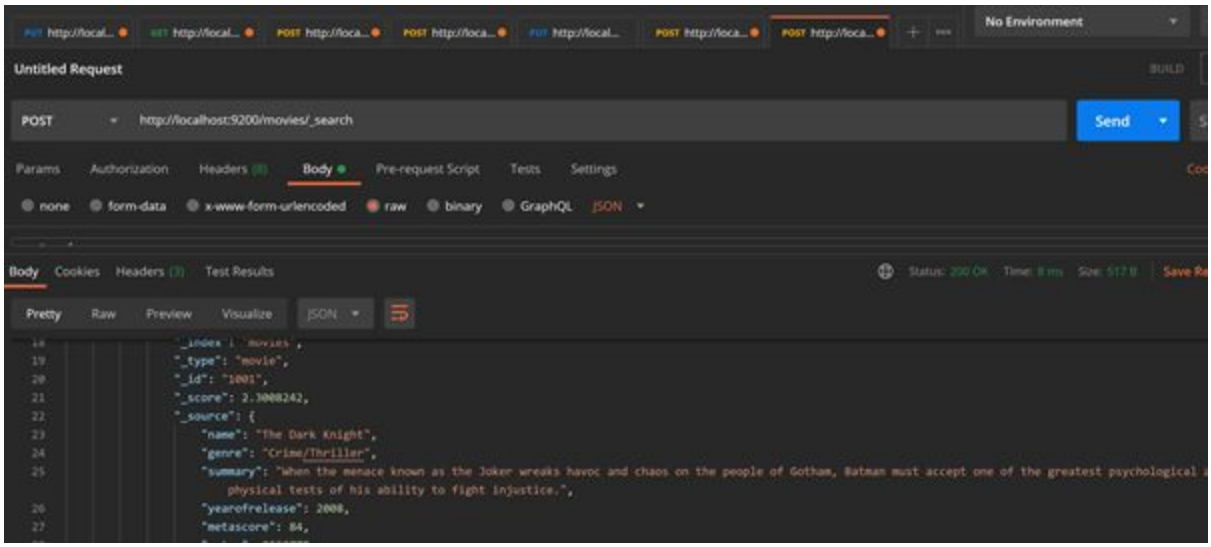
This will find the documents where the name field contains Dark

```
http://localhost:9200/movies/_search
```

```
{
    "query": {
        "query_string": {
            "query": "Dark",
            "fields": ["name"]
        }
    }
}
```



Response



Bool Query to filter on various attributed

If we query using the string ‘Thriller’ or as a genre, we will get 2 results. We can further add a filter to this to drill down the search further to add the year of release to get only 1 desired result.

```
{
    "query": {
        "query_string": {
            "query": "Thriller"
        }
    }
}
```

Query context

A query clause used in query context answers the question "*How well does this document match this query clause?*" Besides deciding whether or not the document matches, the query clause also calculates a `_score` representing how well the document matches, relative to other documents.

Query context is in effect whenever a query clause is passed to a `query` parameter, such as the `query` parameter in the `search` API.

Filter context

In *filter* context, a query clause answers the question "*Does this document match this query clause?*" The answer is a simple Yes or No — no scores are calculated. Filter context is mostly used for filtering structured data, e.g.

- Does this `timestamp` fall into the range 2015 to 2016?
- *Is the `status` field set to `"published"`*?

```
{
  "query": {
    "bool" : {
      "must" : {
```

```
        "term" : { "genre" : "Thriller" }
      },
      "filter": {
        "term" : { "yearofrelease" : 2008 }
      },
      "must_not" : {
        "range" : {
          "age" : { "gte" : 10, "lte" : 20 }
        }
      },
      "should" : [
        { "term" : { "tag" : "wow" } },
        { "term" : { "tag" : "elasticsearch" } }
      ],
      "minimum_should_match" : 1,
      "boost" : 1.0
    }
  }
}
```

Occur	Description
`must`	The clause (query) must appear in matching documents and will contribute to the score.
`filter`	The clause (query) must appear in matching documents. However unlike `must` the score of the query will be ignored. Filter clauses are executed in filter context, meaning that scoring is ignored and clauses are considered for caching.

`should`	The clause (query) should appear in the matching document. If the `bool` query is in a query context and has a `must` or `filter` clause then a document will match the `bool` query even if none of the `should` queries match. In this case these clauses are only used to influence the score. If the `bool` query is in a filter context or has neither `must` or `filter` then at least one of the `should` queries must match a document for it to match the `bool` query. This behavior may be explicitly controlled by setting the `minimum_should_match` parameter.
`must_not`	The clause (query) must not appear in the matching documents. Clauses are executed in filter context meaning that scoring is ignored and clauses are considered for caching. Because scoring is ignored, a score of `0` for all documents is returned.

References

- https://medium.com/@victorsmelopoa/an-introduction-to-elasticsearch-with-kibana-78071db3704
- https://www.udemy.com/course/elasticsearch-7-and- elastic-stack/
- https://www.elastic.co/guide/en/apm/agent/python/master/starlette-support.html
  - https://www.elastic.co/guide/en/elasticsearch/reference/6.8/query-dsl-bool-query.html
- http://joelabrahamsson.com/elasticsearch-101/