Contents

[Urls 2](#_Toc13332883)

[Installation 3](#_Toc13332884)

[Theory 3](#_Toc13332885)

[General 4](#_Toc13332886)

[Commands 4](#_Toc13332887)

[Docker in windows 6](#_Toc13332888)

[Install elasticsearch with docker 6](#_Toc13332889)

[install elasticsearch in jelastic / cloudjiffy 6](#_Toc13332890)

[Tutorial 6](#_Toc13332891)

[Index 8](#_Toc13332892)

# Urls

1. Implementation of elastic search with node.js

<https://blog.patricktriest.com/text-search-docker-elasticsearch/>

1. Elastic search with everything in docker

<https://blog.patricktriest.com/text-search-docker-elasticsearch/>

1. Elastic search on line book

<https://livebook.manning.com/#!/book/elasticsearch-in-action/chapter-1/18>

1. A series of blogs

<https://medium.com/elasticsearch>

Somebody created this % analysis of sources

1. Elastic Search Official Documentation  ([Elasticsearch Reference](https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html) )  | Helped 5% ...used just for reference

2. [Cross-type joins in Elasticsearch](http://rore.im/posts/elasticsearch-joins/)  | Helped 2%

3. [ELK Stack Archives - LogIt.io](http://blog.logit.io/category/elk-stack/)  | Helped 3%

4. This Playlist Helped 25%

<https://www.youtube.com/watch?time_continue=6&v=60UsHHsKyN4>

Rest  65% of my knowledge came from here

5. [Using Elasticsearch for E-commerce](https://qbox.io/blog/series/using-elasticsearch-for-e-commerce)

6.  [Kibana Analytics](https://qbox.io/blog/series/kibana-analytics)

7. [Getting to Know Elasticsearch](https://qbox.io/blog/series/getting-to-know-elasticsearch)

8. [Elasticsearch Scripting](https://qbox.io/blog/series/elasticsearch-scripting)

9. [Elasticsearch Basics](https://qbox.io/blog/series/elasticsearch-basics)

10. [Machine Learning](https://qbox.io/blog/series/machine-learning)

[Qbox Blog](https://qbox.io/blog)  help me stays up to date

23-06-2019

# Installation

* Install java > 7
* Download and unzip elasticsearch.
* Cd c:\elasticsearch\bin
* Elasticsearch
* Now the elasticsearch is running at localhost:9200
* Install fiddler.

In fiddler through composer you can create requests and in inspectors you can see the responses.

## Theory

**Node**: Single running instance of ES. One physical server or VPS may have many nodes. This is a sigle machine.

**Cluster**: One cluster has logically many nodes. Cluster provides collective indexing capabilities of its member nodes.

**Index**: Its like database. One node has many indices. Index has many types and a type has many documents. After version 7.0 there is only one type in an index.

**Type / mapping**: Deprecated. It is collection of many documents. All documents in a type have same fields. Its like table in database. Each row of table is like one document in a type.

**Document**: Just as there are many rows in database table, there are many documents in a type. It must have on id called UID. Document has fields like rows of database table has columns. All documents in a type have common fields.

**Shard**: This is horizontal partition of index.

**Replicas**: Shard can be replicated which are called replicas.

|  |  |
| --- | --- |
| **Elasticsearch** | **RDBMS** |
| Semi-structured or unorganized data | Structured and organized data |
| Eventual Consistency | Tight Consistency |
| BASE transactions | ACID transactions |
| No Predefined Schema | Data and relationships stored in tables. |
| Index | Database |
| Shard | Partition |
| Type | Table . Only one type allowed in an index |
| Document | Row |
| Field | Column |
| [Mapping](https://mindmajix.com/elasticsearch/introduction-to-mapping) | Schema |
| Everything is indexed | Index |
| Query DSL | SQL |

Act of storing data in ES is called indexing.

One cluster has many indices

One index has many types. [Now only one type after version 7]

One type has many documents

One document has many fields

## General

ES client libraries are available for various languages as JavaScript (node.js), java, .net etc. Three types of queries can be done with ES. 1) Structured 2) Full text search. Results are sorted in relevance, 3) complex, combination of above two. When you start default ES instance you will get a single node having name as that of your computer name, say admin-pc.

## Commands

Create index

put customer?pretty. creates customer index. Show the response in pretty manner if there is any.

Get /\_cat/indices?v. get list of indices.

Put /customer/\_doc/1?pretty

{

"name": "John Doe"

}

In index customer creates a document with id 1 and data as above. Output response is pretty. If customer index is not already there it will be created.

GET /customer/\_doc/1?pretty

From customer index gets the document with id as 1.

DELETE /customer?pretty

Deletes the index customer.

<HTTP Verb> /<Index>/<Endpoint>/<ID>

This is the pattern of REST commands for elasticsearch.

If you try to create a new document if it is already there then ES modifies the old document to new one. You can omit the id while creating new document. In that case ES creates its own id. But you need to give post request instead of put request if you are not supplying the id.

POST /customer/\_doc?pretty

{

"name": "Jane Doe"

}

Above indexes a new document without an id. Note the post request instead of put.

Modifying a document. Actually, it is delete / new under the hood.

POST /customer/\_update/1?pretty

{

"doc": { "name": "Jane Doe" }

}

POST /customer/\_update/1?pretty

{

"doc": { "name": "Jane Doe", "age": 20 }

}

DELETE /customer/\_doc/2?pretty

Deletes document.

There is bulk api available with word \_bulk. Herein you can give several commands one after another. The entire bulk api does not fail if a single command fails.

POST /customer/\_bulk?pretty

{"update":{"\_id":"1"}}

{"doc": { "name": "John Doe becomes Jane Doe" } }

{"delete":{"\_id":"2"}}

By default, search query results return 10 objects. If you do ?size=100 then 100 results will be returned.

## Docker in windows

I installed docker-desktop for windows. It did the job. During login do not give complete email. Just give the username capitalch.

Docker login

Docker run hello-world

## Install elasticsearch with docker

docker pull docker.elastic.co/elasticsearch/elasticsearch:7.2.0

docker run -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node" docker.elastic.co/elasticsearch/elasticsearch:7.2.0

localhost:9200 gives you the result from elasticsearch docker

## install elasticsearch in jelastic / cloudjiffy

1. Install docker as normal. After starting of node you will see errors in log file
2. In /usr/share/elasticsearch/config/elasticsearch.yml add this line:

cluster.initial\_master\_nodes: node18391-elastic.cloudjiffy.net. You can check the node name in the log file itself.

1. You need to attach static IP, otherwise it won’t work. Instead of static IP you can also create an end point in cloudjiffy. Don’t forget to add :9200 at the end. It will work.

## Node.js and elasticsearch

Npm install @elastic/elasticsearch –save

Following code creates and index ‘myindex’

Server.js

const es = require('@elastic/elasticsearch');

const client = new es.Client({

node: 'http://localhost:9200'

})

const dataArray = [

{ "name": "sushant", "address": "12 J.L. Nehru road" },

{ "name": "prashant", "address": "P/161 VIP road, Sch VII M, Kol 700054" },

{ "name": "sanjeev", "address": "2D burdwan Road" },

{ "name": "Niraj", "address": "USA" }

]

const getIndexTemplate = (i) => `{"index":{"\_id":${i + 1}}}`

function doIndex(arr, template) {

const bulk = []

arr.forEach((x, i) => {

const j = JSON.parse(template(i));

bulk.push(j);

bulk.push(x);

})

return bulk;

}

const bulk = doIndex(dataArray, getIndexTemplate);

client.bulk(

{

index: 'myindex',

body: bulk

}, function (err, resp) {

if (err) {

console.log(err);

} else {

console.log('success:', resp);

}

});

Note that for every document you need to provide one meta data before. This is not standard json. The bulk command is given an array of json objects. Actual json data is preceded by {“index”:{“\_id”:some id}}. You can also provide index name in the meta data. But if all data is for the same index then put the “index”:”myindex” in the bulk command itself as I did above.

## Real indexing of JSON data and query

I downloaded nobelprize data from Internet in Json format. Data file is around 375kb. This json file I indexed with above node.js code, which was quite comfortable even with cloudjiffy end point url <http://node18391-elastic.cloudjiffy.net:11039>

Index name at cloud was nobel.

Using postman issued following query

<http://node18391-elastic.cloudjiffy.net:11039/_cat/indices>

This gave a listing of all indices presently only ‘nobel’

Now as per reference document at <https://www.elastic.co/guide/en/elasticsearch/reference/7.2/query-dsl-query-string-query.html> query string query I executed following query:

url: <http://node18391-elastic.cloudjiffy.net:11039/nobel/_search>

Header: Content-Type: application/json

Body: {

"query": {

"query\_string" : {

"query" : "litrature"

}

}

}

I got 0 results because there is spelling mistake of literature in query field. To make fuzzy query I just replaced ‘litrature’ with ‘litrature~’. Then I got 110 results. Of course you need to give ?size=1000 to see all the results otherwise only 10 results will be outputted.

The fuzzy operator ~ corrects spelling mistakes. Since I have not given any fields in above query, all the fields are searched. The sample indexed data in nobel index from the json file is as:

{

"year": "2018",

"category": "physics",

"overallMotivation": "\"for groundbreaking inventions in the field of laser physics\"",

"laureates": [

{

"id": "960",

"firstname": "Arthur",

"surname": "Ashkin",

"motivation": "\"for the optical tweezers and their application to biological systems\"",

"share": "2"

},

{

"id": "961",

"firstname": "G\u00e9rard",

"surname": "Mourou",

"motivation": "\"for their method of generating high-intensity, ultra-short optical pulses\"",

"share": "4"

},

{

"id": "962",

"firstname": "Donna",

"surname": "Strickland",

"motivation": "\"for their method of generating high-intensity, ultra-short optical pulses\"",

"share": "4"

}

]

},

{

"year": "2018",

"category": "chemistry",

"laureates": [

{

"id": "963",

"firstname": "Frances H.",

"surname": "Arnold",

"motivation": "\"for the directed evolution of enzymes\"",

"share": "2"

},

{

"id": "964",

"firstname": "George P.",

"surname": "Smith",

"motivation": "\"for the phage display of peptides and antibodies\"",

"share": "4"

},

{

"id": "965",

"firstname": "Sir Gregory P.",

"surname": "Winter",

"motivation": "\"for the phage display of peptides and antibodies\"",

"share": "4"

}

]

}

Everything worked exceptionally well with cloudjiffy cloud. It is now possible to make several indices with different data sets and thereafter query on those indices at real time.

## Tutorial

Followed this <https://www.elastic.co/blog/a-practical-introduction-to-elasticsearch>

1. Start ES through command line by typing elasticsearch in bin folder. Default url is localhost:9200.
2. Check by typing in browser localhost:9200, or curl localhost:9200 or use fiddler.

You will get result as {

"name" : "ADMIN-PC",

"cluster\_name" : "elasticsearch",

"cluster\_uuid" : "RVQFYNOIR8iviv3DwS6qxA",

"version" : {

"number" : "7.1.1",

"build\_flavor" : "default",

"build\_type" : "zip",

"build\_hash" : "7a013de",

"build\_date" : "2019-05-23T14:04:00.380842Z",

"build\_snapshot" : false,

"lucene\_version" : "8.0.0",

"minimum\_wire\_compatibility\_version" : "6.8.0",

"minimum\_index\_compatibility\_version" : "6.0.0-beta1"

},

"tagline" : "You Know, for Search"

}

1. **Create an index** by name accounts. Create a type person in accounts. Add a document in type person.

Post localhost:9200/accounts/person/1 with body as:

{"name": "John", "lastname": "Doe", "job\_description": "Systems administrator and Linux specialist" }

Since accounts is not there, index accounts will be created. Since type person is not there, it will also be automatically created. Last item in url is id as1 so a document with id = 1and other fields as defined in body will be created.

Note: Header "Content-Type: application/x-www-form-urlencoded" is not supported with elastic search you need to use "Content-Type: application/json", otherwise error will happen. I used curl command

curl -d "@data/data1.json" -X POST http://localhost:9200/accounts/person/1 -H "Content-Type: application/json". This worked fine. In data1.json I placed the above json.

It is most convenient to do it through Postman. Following results are found as response

{

"\_index": "accounts",

"\_type": "person",

"\_id": "1",

"\_version": 4,

"result": "updated",

"\_shards": {

"total": 2,

"successful": 1,

"failed": 0

},

"\_seq\_no": 3,

"\_primary\_term": 1

}

In above \_version = 4, that is because I ran the same command 4 times. Every time it incremented the version by 1 and replaced the document.

1. **To retrieve** the above document, do: <http://localhost:9200/accounts/person/1>
2. **Partial update** of document is possible by \_update. Under the hood ES deletes the old and creates new with updates.

POST localhost:9200/accounts/person/1/\_update { "doc":{ "job\_description" : "Systems administrator and Linux specialist" } }

Note the “doc” property above, which is important for updates. Version is changed after update.

1. Search: GET localhost:9200/\_search?q=john

It returns both the records since both contain john.

1. **Delete a document**: DELETE localhost:9200/accounts/person/1
2. **Delete full index**: DELETE localhost:9200/accounts

## Index

**Index (noun):** It is like a database. It is a place to store documents. Plural of index is indices or indexes.

**Index (verb):** To index a document is to store that document in an index (noun). It’s like insert in database except that the new document will replace the older one.

In databases BTree index is used but in ES inverted index is used.