

ELASTICSEARCH DOCUMENTATION

DOWNLOAD

Elasticsearch can be run on your own hardware or using hosted Elasticsearch Service on Elastic Cloud, which is available on AWS and GCP: <https://www.elastic.co/cloud>

For the purpose of this project, we used our own hardware.

Elasticsearch is provided in the following package formats:

zip/tar.gz	The zip and tar.gz packages are suitable for installation on any system and are the easiest choice for getting started with Elasticsearch on most systems.
deb	The deb package is suitable for Debian, Ubuntu, and other Debian-based systems. Debian packages may be downloaded from the Elasticsearch website or from our Debian repository.
rpm	The rpm package is suitable for installation on Red Hat, Centos, SLES, OpenSuSE and other RPM-based systems. RPMs may be downloaded from the Elasticsearch website or from our RPM repository.
msi	[beta]This functionality is in beta and is subject to change. The design and code is less mature than official GA features and is being provided as-is with no warranties. Beta features are not subject to the support SLA of official GA features. The msi package is suitable for installation on Windows 64-bit systems with at least .NET 4.5 framework installed, and is the easiest choice for getting started with Elasticsearch on Windows. MSIs may be downloaded from the Elasticsearch website.
docker	Images are available for running Elasticsearch as Docker containers. They may be downloaded from the Elastic Docker Registry.

We download deb file on an Ubuntu operating system.

INSTALL

**For the purpose of this project, we try to install Elastic Search on a Ubuntu machine.
We open the terminal and run the following commands in the given order:**

- `sudo apt install curl`

Curl is a command line tool and library for transferring data with URLs

It is free and open-source : <https://github.com/curl/curl>

- `wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-6.7.0.deb`

This command (wget) is used to download debian package from elastic search website

- `wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-6.7.0.deb.sha512`
- `shasum -a 512 -c elasticsearch-6.7.0.deb.sha512`
- `sudo dpkg -i elasticsearch-6.7.0.deb`

These commands will manually download from the website and install

The command also compares the SHA (Secure Hash Algorithm 1) of the downloaded Debian package and the published checksum, which should output :

`elasticsearch-{version}.deb: OK.`

To configure Elasticsearch to start automatically when the system boots up, run the following commands:

```
sudo /bin/systemctl daemon-reload
sudo /bin/systemctl enable elasticsearch.service
```

Elasticsearch can be started and stopped as follows:

Start the service : `sudo systemctl start elasticsearch.service`

Stop the service : `sudo systemctl stop elasticsearch.service`

VERIFY

After you start the service, the default port is usually: 9200 and You can test that your Elasticsearch node is running by sending an HTTP request to port 9200 on localhost

GET /

Command: curl <http://localhost:9200/> on your terminal

```
kmahesh2@vm18-58:~/Desktop$ curl http://localhost:9200/
{
  "name" : "p03q09S",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "BUIVJFS9Q-aRtl3jPHPGnA",
  "version" : {
    "number" : "6.7.0",
    "build_flavor" : "default",
    "build_type" : "deb",
    "build_hash" : "8453f77",
    "build_date" : "2019-03-21T15:32:29.844721Z",
    "build_snapshot" : false,
    "lucene_version" : "7.7.0",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

Your result should look like:

```
{
  "name" : "Cp8oag6",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "AT69_T_DTp-1qgIJlatQqA",
  "version" : {
    "number" : "6.7.1",
    "build_flavor" : "default",
    "build_type" : "zip",
    "build_hash" : "f27399d",
    "build_date" : "2016-03-30T09:51:41.449Z",
    "build_snapshot" : false,
    "lucene_version" : "7.7.0",
    "minimum_wire_compatibility_version" : "1.2.3",
    "minimum_index_compatibility_version" : "1.2.3"
  },
  "tagline" : "You Know, for Search"
}
```

BASIC QUERIES

Index API :

The index API adds or updates a typed JSON document in a specific index, making it searchable.

The following example inserts the JSON document into the "twitter" index, under a type called `_doc` with an id of 1:

```
PUT twitter/_doc/1
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

In curl, we can run the following command from the terminal :

```
curl -X PUT "localhost:9200/twitter/_doc/1" -H 'Content-Type: application/json' -d'
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}'
```



```
curl -X PUT "localhost:9200/twitter/_doc/1" -H 'Content-Type: application/json' -d'
n' -d'
>
> {
>   "user" : "kimchy",
>   "post_date" : "2009-11-15T14:12:12",
>   "message" : "trying out Elasticsearch"
> }
>
> '
> {"_index":"twitter","_type":"_doc","_id":"1","_version":2,"result":"updated",
> "_shards":{"total":2,"successful":1,"failed":0},"_seq_no":1,"_primary_term":1}
]kmahesh2@vm18-58:~/Desktop$
```

The result of the above index operation is:

```
{
  "_shards" : {
    "total" : 2,
    "failed" : 0,
    "successful" : 2
  },
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "result" : "created"
}
```

The `_shards` header provides information about the replication process of the index operation:

Total : Indicates how many shard copies (primary and replica shards) the index operation should be executed on.

Successful: Indicates the number of shard copies the index operation succeeded on.

Failed: An array that contains replication-related errors in the case an index operation failed on a replica shard.

The index operation is successful in the case successful is at least 1.

Operation Type

The index operation also accepts an `op_type` that can be used to force a create operation, allowing for "put-if-absent" behavior. When create is used, the index operation will fail if a document by that id already exists in the index.

Here is an example of using the `op_type` parameter:

```
PUT twitter/_doc/1?op_type=create
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

OR

```
PUT twitter/_doc/1/_create
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

Automatic ID Generation

The index operation can be executed without specifying the id. In such a case, an id will be generated automatically. In addition, the `op_type` will automatically be set to create.

Here is an example (note the POST used instead of PUT):

```
POST twitter/_doc/
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

Get API

The get API allows to get a typed JSON document from the index based on its id. The following example gets a JSON document from an index called twitter, under a type called _doc, with id valued 0:

GET twitter/_doc/0

Command : curl -X GET "localhost:9200/twitter/_doc/0"

```
kmaresh2@vm18-58:~/Desktop$ curl -X GET "localhost:9200/twitter/_doc/0"
{"_index":"twitter","_type":"_doc","_id":"0","found":false}
```

The result of the above get operation is:

```
{
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "0",
  "_version" : 1,
  "_seq_no" : 10,
  "_primary_term" : 1,
  "found": true,
  "_source" : {
    "user" : "kimchy",
    "date" : "2009-11-15T14:12:12",
    "likes": 0,
    "message" : "trying out Elasticsearch"
  }
}
```

Delete API

The delete API allows to delete a typed JSON document from a specific index based on its id.

The following example deletes the JSON document from an index called twitter, under a type called _doc, with id 1:

DELETE /twitter/_doc/1

Command: curl -X DELETE "localhost:9200/twitter/_doc/1"

```
kmaresh2@vm18-58:~/Desktop$ curl -X DELETE "localhost:9200/twitter/_doc/1"
{"_index":"twitter","_type":"_doc","_id":"1","_version":3,"result":"deleted",
"_shards":{"total":2,"successful":1,"failed":0},"_seq_no":3,"_primary_term":1}
kmaresh2@vm18-58:~/Desktop$
```

The result of the above delete operation is:

```
{
  "_shards" : {
    "total" : 2,
    "failed" : 0,
    "successful" : 2
  },
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 2,
  "_primary_term" : 1,
  "_seq_no" : 5,
  "result" : "deleted"
}
```


Delete By Query API

The simplest usage of `_delete_by_query` just performs a deletion on every document that matches a query. Here is the API:

POST `twitter/_delete_by_query`

```
{
  "query": {
    "match": {
      "message": "some message"
    }
  }
}
```

Command:

```
curl -X POST "localhost:9200/twitter/_delete_by_query" -H 'Content-Type: application/json' -d'
  {
    "query": {
      "match": {
        "message": "some message"
      }
    }
  }'
```

The query must be passed as a value to the query key, in the same way as the Search API. You can also use the `q` parameter in the same way as the search API.

```

kmaresh2@vm18-58:~/Desktop$ curl -X POST "localhost:9200/twitter/_delete_by_query" -H 'Content-Type: application/json' -d'
>
> {
>   "query": {
>     "match": {
>       "message": "some message"
>     }
>   }
> }
> '
{"took":9,"timed_out":false,"total":0,"deleted":0,"batches":0,"version_conflicts":0,"noops":0,"retries":{"bulk":0,"search":0},"throttled_millis":0,"requests_per_second":-1.0,"throttled_until_millis":0,"failures":[]}kmaresh2@vm18-58:~/Desktop$
kmaresh2@vm18-58:~/Desktop$ █

```

That will return something like this:

```

{
  "took" : 147,
  "timed_out": false,
  "deleted": 119,
  "batches": 1,
  "version_conflicts": 0,
  "noops": 0,
  "retries": {
    "bulk": 0,
    "search": 0
  },
  "throttled_millis": 0,
  "requests_per_second": -1.0,
  "throttled_until_millis": 0,
  "total": 119,
  "failures" : [ ]
}

```

Update API

The update API allows updating a document based on a script provided. The operation gets the document (collocated with the shard) from the index, runs the script (with optional script language and parameters), and indexes back the result (also allows to delete, or ignore the operation). It uses versioning to make sure no updates have happened during the "get" and "reindex".

Note, this operation still means full reindex of the document, it just removes some network roundtrips and reduces chances of version conflicts between the get and the index. The `_source` field needs to be enabled for this feature to work.


For example, let's index a simple doc:

PUT test/_doc/1

```
{
  "counter" : 1,
  "tags" : ["red"]
}
```

Command :

```
curl -X PUT "localhost:9200/test/_doc/1" -H 'Content-Type: application/json' -d'
{
  "counter" : 1,
  "tags" : ["red"]
}'
```



```
kmahesh2@vm18-58:~/Desktop$ curl -X PUT "localhost:9200/test/_doc/1" -H 'Content-Type: application/json' -d'
>
> {
>   "counter" : 1,
>   "tags" : ["red"]
> }'
{"_index":"test","_type":"_doc","_id":"1","_version":8,"result":"updated","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":7,"_primary_term":1}kmahesh2@vm18-58:~/Desktop$
```

PART B - 3

STEPS TO RUN BASIC QUERIES USING PYTHON ON ELASTICSEARCH

Steps to do before proceeding:

1. Make sure you have Elastic Search Up and Running.
2. Data flow is on, try running basic curl queries to see if JSON objects are present in the output.
3. Elastic-search is installed(if not just run `sudo pip install elasticsearch`)

In the directory where the python is installed, open a command line and enter:

```
python3 filename.py
```

This will open a user prompt and ask for user input.

The user will have 5 query choices:

- Match_all Query
- Term Query
- Match Query
- Prefix Query
- Fuzzy Query

The user needs to enter an integer value (1-5) for each of the queries as follows :

1. Match_all Query
2. Term Query
3. Match Query
4. Prefix Query
5. Fuzzy Query

Depending upon the query chosen, the user will be asked for another input.

Before prompting for another input, if there is a problem with ElasticSearch setup, out the program will print on the console `"Connection failed"`

The user is asked another input, for ex: for a fuzzy query, if the user enters 'sennis' then with default fuzziness our program will return a JSON object response with Text field of tweet with related words like 'tennis' etc.

Similarly, for other queries, the response JSON object will have Text Field of the tweet corresponding to a user query.