

(https://d2u6dc21frj6h.cloudfront.net/d/3eJwljs0OgjAQhF9ls2cXWjwo3Eyl8QE8eWvajSK0JaXgX3x3W7x+M/lmPhiwAbzFODZlqYdO98UY/J11JKucurJlFwvtbQp7Mmn=1)

DZone (/) > Big Data Zone (/big-data-analytics-tutorials-tools-news) > Spring Boot and Elasticsearch Tutorial

# Spring Boot and Elasticsearch Tutorial



(/users/468979/rajesh.bhojwani.html) by **Rajesh Bhojwani** (/users/468979/rajesh.bhojwani.html) 🌟 MVB · Jan. 10, 19 · Big Data Zone

(/big-data-analytics-tutorials-tools-news) · Tutorial

👍 Like (22) 💬 Comment (6) ☆ Save 🐦 Tweet

## Overview

In my last article (<https://dzone.com/articles/elasticsearch-with-spring-boot-application>), I talked about how we can use a Spring-Data-Elasticsearch project to connect with the Elasticsearch engine and perform CRUD operations. However, I did also mention that this project is not updated to be compatible with the latest version of the Elasticsearch (<https://dzone.com/articles/what-is-elasticsearch-and-how-it-can-be-useful>) engine.

So, in this post, I am going to cover how to interact with the latest version of Elasticsearch engine using the Transport Client library. I am going to use Spring Boot as a client application and then add dependencies for other required libraries.

## Pre-Requisites

- JDK 1.8
- Maven
- Elasticsearch engine download 5.x or 6.x ( I will explain the steps how to download)
- Eclipse or VSD as IDE

## Set Up Elasticsearch

**Step 1** - Go to Elastic's official website (<https://www.elastic.co/downloads/past-releases>).

**Step 2** - Select Elasticsearch in the drop down and then version as 5.5.0 and click on the Download button.

**Step 3** - It will give you options if you want to download as Zip, TAR, or RPM file. I selected the Zip format as using it on windows.

**Step 4** - Unzip the downloaded content and go to the bin folder. There will be a file named `elasticsearch.bat`.

**Step 5** - Run this file on Windows OS through command prompt and it will start the Elasticsearch engine for you. Once started, it will start listening to port 9200. So the URL will be `http://localhost:9200/`. Also, port 9300 is exposed as a cluster node. Port 9200 is for REST communication and can be used by Java or any other language and 9300 is for Elasticsearch Cluster Node communication as well, and Java can also connect to this cluster node using the transport protocol.

**Step 6** - Test to verify that it all started properly by launching the URL using the curl command. You can use PowerShell on Windows.

```
1 curl http://localhost:9200/
2
3 StatusCode      : 200
4 StatusDescription : OK
5 Content         : {
6     "name" : "ggyBmti",
7     "cluster_name" : "elasticsearch",
8     "cluster_uuid" : "Bp6EeKIoQNgj0iV5sHtWg",
9     "version" : {
10        "number" : "5.5.0",
11        "build_hash" : "260387d",
12        "build_date" : "2017-...
13 RawContent      : HTTP/1.1 200 OK
```



Content-Length: 327  
Content-Type: application/json; charset=UTF-8

(/users/login.html)

(/search)

**REFCARDZ** (/refcardz) **RESEARCH** (/research) **WEBINARS** (/webinars) **ZONES** ▾

```
18      "name" : "ggvRmti",
19      "cluster_name" : "elasticsearch",
20      "cluster_uuid" : "Bp6EeKIoQNgGj0iV5sHtWg",
21      "versi...
22 Forms      : {}
23 Headers    : {[Content-Length, 327], [Content-Type, application/json; charset=UTF-8]}
24 Images     : {}
25 InputFields : {}
26 Links      : {}
27 ParsedHtml : mshtml.HTMLDocumentClass
28 RawContentLength : 327
```

Once the search engine starts up, let's try to test some of the REST APIs it provides to interact with the engine.

Launch `http://localhost:9200/users/employee/1` using Postman or curl with the POST method. Input should be in JSON format.

```
1 {
2   "userId" : "1",
3   "name" : "Rajesh",
4   "userSettings" : {
5     "gender" : "male",
6     "occupation" : "CA",
7     "hobby" : "chess"
8   }
9 }
```

The response will show that it has created *"users"* as an index name, *"employee"* as a type, and that a document has been created with an id of *"1"*.

```
1 {
2   "_index": "users",
3   "_type": "employee",
4   "_id": "1",
5   "_version": 3,
6   "result": "updated",
7   "_shards": {
8     "total": 2,
9     "successful": 1,
10    "failed": 0
11  },
12   "created": false
13 }
```

Now, to view the content of the document, we can call another REST API

endpoint `http://localhost:9200/users/employee/1` using the GET method. This will result in the below document information:


```
1 {
2   "_index": "users",
3   "_type": "employee",
4   "_id": "1",
5   "_version": 3,
6   "found": true,
7   "_source": {
8     "userId": "1",
9     "name": "Rajesh",
10    "userSettings": {
11      "gender": "male",
12      "occupation": "CA",
13      "hobby": "chess"
14    }
15  }
16 }
```

Now, if we want to search a document by any particular field, you need to add `_search` as a path in the REST API URL. To launch this, run the following command: `curl -XGET 'http://localhost:9200/users/employee/_search'`

This will search for all the documents with an index of *"users"* and type of *"employee."* Now, if you want to search for a particular field, you need to add some query match criteria as part of your JSON.

```
1 curl -XGET 'http://localhost:9200/users/employee/_search'
2 -H 'Content-Type: application/json' -d
3 '{ "query": { "match": { "name" : "Rajesh" } } }'
```

This will search for a document that has the field 'name' set as 'Rajesh.'

[REFCARDZ \(/refcardz\)](#) [RESEARCH \(/research\)](#) [WEBINARS \(/webinars\)](#) [ZONES](#) 

Now, we have seen how Elasticsearch REST APIs work to perform create, retrieve, and other operations for the documents; let's try to understand how we can connect to Elasticsearch engine to our application code. We can either make calls to these REST APIs directly from the code or we can use the transport client provided by Elasticsearch. Let's develop a Spring Boot application to showcase all the CRUD operations.

## Developing a Spring Boot Application

### Maven Dependencies

Other than Spring Boot jars, we need Elasticsearch, a transport client, and log4j jars.

```

1 <dependencies>
2 <dependency>
3 <groupId>org.springframework.boot</groupId>
4 <artifactId>spring-boot-starter-web</artifactId>
5 </dependency>
6 <dependency>
7 <groupId>org.springframework.boot</groupId>
8 <artifactId>spring-boot-starter-test</artifactId>
9 <scope>test</scope>
10 </dependency>
11 <dependency>
12 <groupId>org.elasticsearch</groupId>
13 <artifactId>elasticsearch</artifactId>
14 </dependency>
15 <dependency>
16 <groupId>org.elasticsearch.client</groupId>
17 <artifactId>transport</artifactId>
18 <version>5.0.0</version>
19 </dependency>
20
21 <dependency>
22 <groupId>org.apache.logging.log4j</groupId>
23 <artifactId>log4j-api</artifactId>
24 <version>2.7</version>
25 </dependency>
26 <dependency>
27 <groupId>org.apache.logging.log4j</groupId>
28 <artifactId>log4j-core</artifactId>
29 <version>2.7</version>
30 </dependency>
31 <dependency>
32 <groupId>org.apache.logging.log4j</groupId>
33 <artifactId>log4j-web</artifactId>
34 <version>2.7</version>
35 </dependency>
36 </dependencies>

```

### Configuration

As we will be using a transport client to connect to the Elasticsearch engine, we need to give URL path for the cluster node of the engine. so I have put properties in an application.properties file for the *host* and *port* of the URL.

```

1 # Local Elasticsearch config
2 elasticsearch.host=localhost
3 elasticsearch.port=9300
4
5
6 # App config
7 server.port=8102
8 spring.application.name=BootElastic

```

### Domain

Create a domain class named `User`. The JSON input will be mapped to this `User` object. This will be used to create user document associated with the index and type.

```

1 public class User {
2     private String userId;
3     private String name;
4     private Date creationDate = new Date();
5     private Map<String, String> userSettings = new HashMap<>();
6     -- getter/setter methods
7 }

```

A Java configuration file has been created to create a Transport client which connects to the Elasticsearch cluster node. It is also loading the value of the host and port from the environment configured through the *application.properties* file.

```

1 @Configuration
2 public class config{
3
4     @Value("${elasticsearch.host:localhost}")
5     public String host;
6     @Value("${elasticsearch.port:9300}")
7     public int port;
8
9     public String getHost() {
10    return host;
11 }
12
13
14 public int getPort() {
15    return port;
16 }
17
18 @Bean
19 public Client client(){
20     TransportClient client = null;
21     try{
22         System.out.println("host:"+ host+"port:"+port);
23         client = new PreBuiltTransportClient(Settings.EMPTY)
24             .addTransportAddress(new InetSocketAddress(InetAddress.getByName(host), port));
25     } catch (UnknownHostException e) {
26         e.printStackTrace();
27     }
28     return client;
29 }
30 }
```

## Controller

UserController is created to showcase the below features:

1. Create an index named "users" and type "employee." It will create a document for storing user information. The id for a document can be passed as JSON input or, if it is not passed, Elasticsearch will generate its own id. The client has a method called `prepareIndex()` which builds the document object and store against the index and type. This method is a POST method call where User information will be passed as JSON.

```

1 @Autowired
2 Client client;
3 @PostMapping("/create")
4 public String create(@RequestBody User user) throws IOException {
5
6     IndexResponse response = client.prepareIndex("users", "employee", user.getUserId())
7         .setSource(jsonBuilder()
8             .startObject()
9             .field("name", user.getName())
10            .field("userSettings", user.getUserSettings())
11            .endObject()
12        )
13        .get();
14     System.out.println("response id:"+response.getId());
15     return response.getResult().toString();
16 }
```

2. View the user information based on tge "id" passed. The client has a `prepareGet()` method to retrieve information based on index, type, and id. It will return the User Information in JSON format.

```

1 @GetMapping("/view/{id}")
2 public Map<String, Object> view(@PathVariable final String id) {
3     GetResponse getResponse = client.prepareGet("users", "employee", id).get();
4     return getResponse.getSource();
5 }
```

3. View the user information based on a field name. I have used `matchQuery()` here to search by the "name" field and return User information. However, there are many different types of `query()` available with the `QueryBuilders` class. For



example use `rangeQuery()` to search a field value in a particular range, like ages between 10 to 20 years. There is a `wildcardQuery()` method to search a field with a wildcard. `termQuery()` is also available. You can play with all of these based on your needs.

[REFCARDZ \(/refcardz\)](#) [RESEARCH \(/research\)](#) [WEBINARS \(/webinars\)](#) [ZONES](#) ▾

```

1  @GetMapping("/view/name/{field}")
2  public Map<String, Object> searchByName(@PathVariable final String field) {
3      Map<String, Object> map = null;
4      SearchResponse response = client.prepareSearch("users")
5          .setTypes("employee")
6          .setSearchType(SearchType.QUERY_AND_FETCH)
7          .setQuery(QueryBuilders.matchQuery("name", field))
8          .get();
9      ;
10     List<SearchHit> searchHits = Arrays.asList(response.getHits().getHits());
11     map = searchHits.get(0).getSource();
12     return map;
13 }
14

```

4. Update the document by searching it with `Id` and replace the field value. The client has a method called `update()`. It accepts `UpdateRequest` as input which builds the update query.

```

1  @GetMapping("/update/{id}")
2  public String update(@PathVariable final String id) throws IOException {
3
4      UpdateRequest updateRequest = new UpdateRequest();
5      updateRequest.index("users")
6          .type("employee")
7          .id(id)
8          .doc(jsonBuilder()
9              .startObject()
10             .field("name", "Rajesh")
11             .endObject());
12
13     try {
14         UpdateResponse updateResponse = client.update(updateRequest).get();
15         System.out.println(updateResponse.status());
16         return updateResponse.status().toString();
17     } catch (InterruptedException | ExecutionException e) {
18         System.out.println(e);
19     }
20     return "Exception";
21 }
22

```

5. The last method is to showcase how to delete a document of an index and type. The client does have a `prepareDelete()` method which accepts the index, type, and id to delete the document.

```

1  @GetMapping("/delete/{id}")
2  public String delete(@PathVariable final String id) {
3      DeleteResponse deleteResponse = client.prepareDelete("users", "employee", id).get();
4      return deleteResponse.getResult().toString();
5  }
6

```

The full code has been put on GitHub (<https://github.com/RajeshBhojwani/spring-boot-standalone-elasticsearch.git>).

## Build the Application

Run the `mvn clean install` command to build the jar file.

## Start Application

Run the `java -jar target/standalone-elasticsearch-0.0.1-SNAPSHOT.jar` command to start the Spring Boot application.

## Test Application

The application will be running on the `http://localhost:8102` URL. Now let's test a couple of the use cases we talked about above.

1. Test for creating a document.

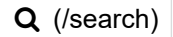
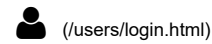
Launch `http://localhost:8102/rest/users/create` as a POST method either through curl or Postman.

Input:

```

1  {
2      "userId": "1",
3      "name": "Sumit",
4      "userSettings": {

```



**REFCARDZ** (/refcardz) **RESEARCH** (/research) **WEBINARS** (/webinars) **ZONES** ▾

You will see the response showing "CREATED."

2. To test if the document has been created, let's test the view functionality.

Launch <http://localhost:8102/rest/users/view/1> with the GET method.

In response, you will see the User information for *id* with a value of "1."

```
1 {
2   "userSettings": {
3     "occupation": "CA",
4     "gender": "male",
5     "hobby": "chess"
6   },
7   "name": "Rajesh"
8 }
```

3. You can view User information via the name field as well by

launching <http://localhost:8102/rest/users/view/name/Rajesh>. This is passing "Rajesh" as the "name" field value.

Similarly, update and delete features can be tested by launching <http://localhost:8102/rest/users/update/1> and <http://localhost:8102/rest/users/delete/1>.

These are all the features I have played around with. As mentioned above, there are many different types of queries you can explore with the Elasticsearch transport client.

Topics: [ELASTICSEARCH](#), [BIG DATA](#), [ELASTICSEARCH TUTORIAL FOR BEGINNERS](#), [SPRING BOOT TUTORIAL](#)

Opinions expressed by DZone contributors are their own.

## Popular on DZone

- [The Evolution of K8s Worker Nodes-CRI-O \(/articles/evolution-of-k8s-worker-nodes-cri-o?fromrel=true\)](/articles/evolution-of-k8s-worker-nodes-cri-o?fromrel=true)
- [Why and When to Opt for a Multicloud Strategy? \(/articles/why-and-when-to-opt-for-a-multicloud-strategy?fromrel=true\)](/articles/why-and-when-to-opt-for-a-multicloud-strategy?fromrel=true)
- [7 Ways To Ensure Employees Are Really Working When Working From Home \(/articles/7-ways-to-ensure-employees-are-really-working-when?fromrel=true\)](/articles/7-ways-to-ensure-employees-are-really-working-when?fromrel=true)
- [Using Multiple Datasources With SpringBoot Application \(/articles/using-multiple-datasources-with-springboot-applica?fromrel=true\)](/articles/using-multiple-datasources-with-springboot-applica?fromrel=true)

### ABOUT US

[About DZone \(/pages/about\)](/pages/about)  
[Send feedback \(mailto:support@dzone.com\)](mailto:support@dzone.com)  
[Careers \(https://devada.com/careers/\)](https://devada.com/careers/)  
[Sitemap \(/sitemap\)](/sitemap)

### ADVERTISE

[Advertise with DZone \(/pages/advertise\)](/pages/advertise)  
 +1 (919) 238-7100 (tel:+19192387100)

### CONTRIBUTE ON DZONE

[Article Submission Guidelines \(/articles/dzones-article-submission-guidelines\)](/articles/dzones-article-submission-guidelines)  
[MVB Program \(/pages/mvb\)](/pages/mvb)  
[Become a Contributor \(/pages/contribute\)](/pages/contribute)  
[Visit the Writers' Zone \(/writers-zone\)](/writers-zone)

### LEGAL

[Terms of Service \(/pages/tos\)](/pages/tos)  
[Privacy Policy \(/pages/privacy\)](/pages/privacy)

### CONTACT US

600 Park Offices Drive  
 Suite 300

