**<https://www.journaldev.com/18148/spring-boot-elasticsearch>**

[Download Spring Boot ElasticSearch Project](https://www.journaldev.com/wp-content/uploads/spring/spring-boot-elasticsearch.zip)

[**https://www.journaldev.com/wp-content/uploads/spring/spring-boot-elasticsearch.zip**](https://www.journaldev.com/wp-content/uploads/spring/spring-boot-elasticsearch.zip)

**Spring Boot Elasticsearch**

[SHUBHAM](https://www.journaldev.com/author/shubham) [16 COMMENTS](https://www.journaldev.com/18148/spring-boot-elasticsearch#comments)

**Table of Contents**[[hide](https://www.journaldev.com/18148/spring-boot-elasticsearch)]

* [1 Spring Boot Elasticsearch 6](https://www.journaldev.com/18148/spring-boot-elasticsearch#spring-boot-elasticsearch-6)
* [2 Setting up the project](https://www.journaldev.com/18148/spring-boot-elasticsearch#setting-up-the-project)
  + [2.1 Maven Dependencies](https://www.journaldev.com/18148/spring-boot-elasticsearch#maven-dependencies)
* [3 Elasticsearch Configuration](https://www.journaldev.com/18148/spring-boot-elasticsearch#elasticsearch-configuration)
* [4 Working the app](https://www.journaldev.com/18148/spring-boot-elasticsearch#working-the-app)
  + [4.1 Model for the app](https://www.journaldev.com/18148/spring-boot-elasticsearch#model-for-the-app)
  + [4.2 Defining the Controller](https://www.journaldev.com/18148/spring-boot-elasticsearch#defining-the-controller)
* [5 Defining the APIs](https://www.journaldev.com/18148/spring-boot-elasticsearch#defining-the-apis)
  + [5.1 Get a book with ID](https://www.journaldev.com/18148/spring-boot-elasticsearch#get-a-book-with-id)
  + [5.2 Insert a Book](https://www.journaldev.com/18148/spring-boot-elasticsearch#insert-a-book)
  + [5.3 Update a book](https://www.journaldev.com/18148/spring-boot-elasticsearch#update-a-book)
  + [5.4 Deleting a Book](https://www.journaldev.com/18148/spring-boot-elasticsearch#deleting-a-book)
* [6 Defining the DAO layer](https://www.journaldev.com/18148/spring-boot-elasticsearch#defining-the-dao-layer)
  + [6.1 Defining DAO](https://www.journaldev.com/18148/spring-boot-elasticsearch#defining-dao)
  + [6.2 Insert Query](https://www.journaldev.com/18148/spring-boot-elasticsearch#insert-query)
  + [6.3 Search Query](https://www.journaldev.com/18148/spring-boot-elasticsearch#search-query)
  + [6.4 Deleting Data](https://www.journaldev.com/18148/spring-boot-elasticsearch#deleting-data)
* [7 Running the application](https://www.journaldev.com/18148/spring-boot-elasticsearch#running-the-application)
* [8 Summary](https://www.journaldev.com/18148/spring-boot-elasticsearch#summary)

**Spring Boot Elasticsearch 6**

In this post, we will setup up a sample Spring boot Elasticsearch application. We will use latest version of Elasticsearch i.e. 6.1.x. To interact with the Elasticsearch search engine, we will use **Elasticsearch Rest client**. We are not using Spring Data ElasticSearch because that doesn’t support latest ElasticSearch version i.e 6.x.



**Setting up the project**

**Maven Dependencies**

We will use Maven build system for this project and here are the dependencies we used:

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.9.RELEASE</version>

<relativePath/>

<!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<!-- Elasticsearch Dependencies -->

<dependency>

<groupId>org.elasticsearch</groupId>

<artifactId>elasticsearch</artifactId>

<version>6.1.2</version>

</dependency>

<dependency>

<groupId>org.elasticsearch.client</groupId>

<artifactId>elasticsearch-rest-high-level-client</artifactId>

<version>6.1.2</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

<version>2.8.2</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.8.2</version>

</dependency>

<dependency>

<groupId>org.elasticsearch.client</groupId>

<artifactId>elasticsearch-rest-client</artifactId>

<version>6.1.2</version>

</dependency>

<dependency>

<groupId>org.elasticsearch.client</groupId>

<artifactId>elasticsearch-rest-client-sniffer</artifactId>

<version>6.1.2</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

Make sure to use stable version for [Spring Boot](https://www.journaldev.com/7969/spring-boot-tutorial) from the maven central.

**Elasticsearch Configuration**

Now, we will have to configure ElasticSearch in our application. Let’s d this in two parts. First, we will provide Elasticsearch address in our application.properties file:

spring.data.elasticsearch.cluster-name=elasticsearch

spring.data.elasticsearch.cluster-nodes=elasticsearch

We only provided the Elasticsearch cluster name and node name here and these are actually the default values. Now, it’s time to put these values to use in our Java config class.

package com.journaldev.elasticsearch.config;

import org.apache.http.HttpHost;

import org.elasticsearch.client.RestClient;

import org.elasticsearch.client.RestHighLevelClient;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.beans.factory.config.AbstractFactoryBean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class ElasticSearchConfiguration extends AbstractFactoryBean {

private static final Logger LOG = LoggerFactory.getLogger(ElasticSearchConfiguration.class);

@Value("${spring.data.elasticsearch.cluster-nodes}")

private String clusterNodes;

@Value("${spring.data.elasticsearch.cluster-name}")

private String clusterName;

private RestHighLevelClient restHighLevelClient;

@Override

public void destroy() {

try {

if (restHighLevelClient != null) {

restHighLevelClient.close();

}

} catch (final Exception e) {

LOG.error("Error closing ElasticSearch client: ", e);

}

}

@Override

public Class<RestHighLevelClient> getObjectType() {

return RestHighLevelClient.class;

}

@Override

public boolean isSingleton() {

return false;

}

@Override

public RestHighLevelClient createInstance() {

return buildClient();

}

private RestHighLevelClient buildClient() {

try {

restHighLevelClient = new RestHighLevelClient(

RestClient.builder(

new HttpHost("localhost", 9200, "http"),

new HttpHost("localhost", 9201, "http")));

} catch (Exception e) {

LOG.error(e.getMessage());

}

return restHighLevelClient;

}

}

With this configuration, we ensured that ElasticSearch is able to make a successful connection to its server using the Rest Client API.

**Working the app**

Let’s start putting the working components of the app now.

**Model for the app**

We will be just using a simple model as a User:

package com.journaldev.elasticsearch.model;

import com.fasterxml.[jackson](https://www.journaldev.com/2324/jackson-json-java-parser-api-example-tutorial).annotation.JsonInclude;

@JsonInclude(JsonInclude.Include.NON\_EMPTY)

public class Book {

private String id;

private String title;

private String author;

private float price;

//standard setters and getters

}

We will be making following functionalities and Database interactions in our app:

* Get a book with ID
* Insert a Book
* Update a Book
* Delete a Book

**Defining the Controller**

Let us move to making our Controller:

package com.journaldev.elasticsearch.controller;

import com.journaldev.elasticsearch.model.Book;

import com.journaldev.elasticsearch.dao.BookDao;

import org.springframework.web.bind.annotation.\*;

import java.util.Map;

@RestController

@RequestMapping("/books")

public class BookController {

private BookDao bookDao;

public BookController(BookDao bookDao) {

this.bookDao = bookDao;

}

...

}

We just Autowired the DAO dependency and we will use this next.

**Defining the APIs**

For the functionalities we mentioned, we will now be making APIs and accessing the DAO dependency which will internally use Elasticsearch Rest Client API.

**Get a book with ID**

Let us get a book with ID:

@GetMapping("/{id}")

public Map<String, Object> getBookById(@PathVariable String id){

return bookDao.getBookById(id);

}

**Insert a Book**

Now, let us insert a book now:

@PostMapping

public Book insertBook(@RequestBody Book book) throws Exception {

return bookDao.insertBook(book);

}

**Update a book**

We will be updating a book in this snippet:

@PutMapping("/{id}")

public Map<String, Object> updateBookById(@RequestBody Book book, @PathVariable String id) {

return bookDao.updateBookById(id, book);

}

**Deleting a Book**

Now that we have added sample data into the DB, let’s try to extract some part of it:

@DeleteMapping("/{id}")

public void deleteBookById(@PathVariable String id) {

bookDao.deleteBookById(id);

}

This is the ease Spring Data API offers us but it also has some downsides. We will elaborate this when we Have defined the ElasticsearchTemplate version as well. Let’s get started with that too.

**Defining the DAO layer**

Now, we will actually define the DAL queries which achieves these objectives.

**Defining DAO**

We will start by mentioning the dependencies we need:

package com.journaldev.elasticsearch.dao;

import com.journaldev.elasticsearch.model.Book;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.elasticsearch.ElasticsearchException;

import org.elasticsearch.action.delete.DeleteRequest;

import org.elasticsearch.action.delete.DeleteResponse;

import org.elasticsearch.action.get.GetRequest;

import org.elasticsearch.action.get.GetResponse;

import org.elasticsearch.action.index.IndexRequest;

import org.elasticsearch.action.index.IndexResponse;

import org.elasticsearch.action.update.UpdateRequest;

import org.elasticsearch.action.update.UpdateResponse;

import org.elasticsearch.client.RestHighLevelClient;

import org.elasticsearch.common.xcontent.XContentType;

import org.springframework.stereotype.Repository;

import java.util.HashMap;

import java.util.Map;

import java.util.UUID;

@Repository

public class BookDao {

private final String INDEX = "bookdata";

private final String TYPE = "books";

private RestHighLevelClient restHighLevelClient;

private ObjectMapper objectMapper;

public BookDao( ObjectMapper objectMapper, RestHighLevelClient restHighLevelClient) {

this.objectMapper = objectMapper;

this.restHighLevelClient = restHighLevelClient;

}

...

}

**Insert Query**

We will start by inserting a Book into the ES:

public Book insertBook(Book book){

book.setId(UUID.randomUUID().toString());

Map dataMap = objectMapper.convertValue(book, Map.class);

IndexRequest indexRequest = new IndexRequest(INDEX, TYPE, book.getId())

.source(dataMap);

try {

IndexResponse response = restHighLevelClient.index(indexRequest);

} catch(ElasticsearchException e) {

e.getDetailedMessage();

} catch (java.io.IOException ex){

ex.getLocalizedMessage();

}

return book;

}

Clearly, we need to convert the model data to a Map data structure before we insert it into the ES database.

**Search Query**

We will now search a book with an ID:

public Map<String, Object> getBookById(String id){

GetRequest getRequest = new GetRequest(INDEX, TYPE, id);

GetResponse getResponse = null;

try {

getResponse = restHighLevelClient.get(getRequest);

} catch (java.io.IOException e){

e.getLocalizedMessage();

}

Map<String, Object> sourceAsMap = getResponse.getSourceAsMap();

return sourceAsMap;

}

Here, this is to be noted that data is searched and got back as a Map data structure as well. This is something we need to handle, converting data to Map and forth to make DB transactions.

public Map<String, Object> updateBookById(String id, Book book){

UpdateRequest updateRequest = new UpdateRequest(INDEX, TYPE, id)

.fetchSource(true); // Fetch Object after its update

Map<String, Object> error = new HashMap<>();

error.put("Error", "Unable to update book");

try {

String bookJson = objectMapper.writeValueAsString(book);

updateRequest.doc(bookJson, XContentType.JSON);

UpdateResponse updateResponse = restHighLevelClient.update(updateRequest);

Map<String, Object> sourceAsMap = updateResponse.getGetResult().sourceAsMap();

return sourceAsMap;

}catch (JsonProcessingException e){

e.getMessage();

} catch (java.io.IOException e){

e.getLocalizedMessage();

}

return error;

}

**Deleting Data**

We will now search a book with an ID:

public void deleteBookById(String id) {

DeleteRequest deleteRequest = new DeleteRequest(INDEX, TYPE, id);

try {

DeleteResponse deleteResponse = restHighLevelClient.delete(deleteRequest);

} catch (java.io.IOException e){

e.getLocalizedMessage();

}

}

I think deleting an object was having the easiest query. Let’s try this application now by running it.

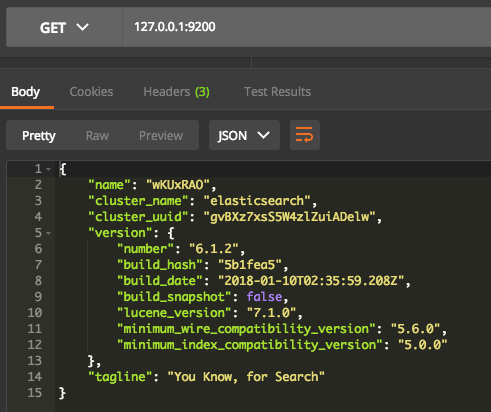
**Running the application**

We can run this app simply by using a single command:

mvn spring-boot:run

Once the app is running, we can try saving a new book by using this API. Before doing that, just confirm that the ES is running by using this API:

127.0.0.1:9200

We will get this response:  
  
Now let’s inserting the data by using following API:

127.0.0.1:8080/books

We use a POST request with this JSON:

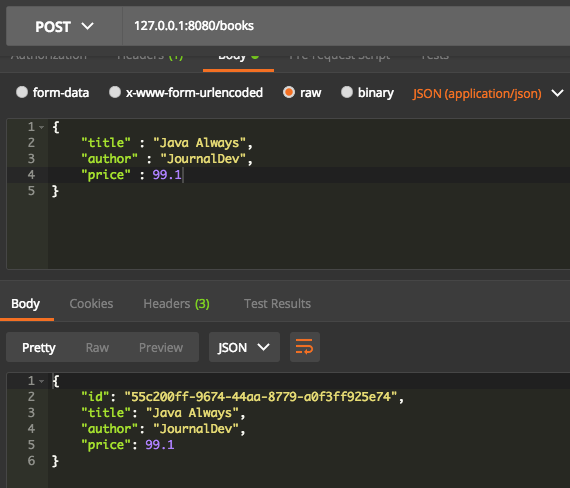
{

"title" : "Java Always",

"author" : "JournalDev",

"price" : 99.1

}

We have the following response:  
  
Let’s try another request to get the book with above ID. We will use GET request on this API:

127.0.0.1:8080/books/55c200ff-9674-44aa-8779-a0f3ff925e74

This is what we get back:  
  
Go on and try more APIs we defined.

**Summary**

In this lesson, we looked at how Elasticsearch can be queried using a Rest Client API it provides. Download the source code here and modify it.

[Download Spring Boot ElasticSearch Project](https://www.journaldev.com/wp-content/uploads/spring/spring-boot-elasticsearch.zip)