# Reactive Spring Data Elasticsearch with Spring Boot

* Data di pubblicazione: 25 marzo 2020
* [**Modifica articolo**](https://www.linkedin.com/post/edit/6647897085234434050)
* [**Visualizza statistiche**](https://www.linkedin.com/pulse/reactive-spring-data-elasticsearch-boot-paolo-de-dominicis/ca/post-analytics/urn:li:linkedInArticle:6647897085234434050/)

**[[](https://www.linkedin.com/in/paolodedominicis/)](https://www.linkedin.com/in/paolodedominicis/)**

**[Stato: online](https://www.linkedin.com/in/paolodedominicis/)**

## [Paolo De Dominicis](https://www.linkedin.com/in/paolodedominicis/)

Software Developer presso Key-Biz S.r.l.

[**1 articolo**](https://www.linkedin.com/in/paolodedominicis/detail/recent-activity/posts/)

The aim of this article is to explain the advantages of the new **Reactive Spring Data Elasticsearch API**, introduced with Spring Data Elasticsearch 3.2, and its configuration and simple usage.

We are aware of the popularity of [Spring Framework](https://spring.io/), spread to the global developing community and companies mostly by [Spring Boot](https://spring.io/projects/spring-boot) and its particular quick configuration and use and its great use in the world of microservices.

On the other hand, [Elasticsearch](https://www.elastic.co/elasticsearch/) is conquering the market of documental databases being one of the most reliable and useful tool, quick (mostly for analytics) and well structured.

[Spring Data](https://spring.io/projects/spring-data) is a Spring Framework's project which provides a interaction and access to data storage and services, like relational, non-relational, graph, document-based or map databases. Its greatness comes from the fact that this parent project provide a familiar and user-friendly implementation of this data interactions, that acts as a layer of abstraction to every single database-specific Spring Data implementation.

This aspect is very important because it simplifies the developer's life: in fact, the knowledge of a certain database is kept one step above thanks to this layer of abstraction. This is also important in a scenario in which the architecture changes and a new database has to be used: in most of the cases, a small lines-of-code modification can be enough in order to let everything fit to the new implementation case.

With **Spring Data Elasticsearch 3.2**release, the support to the Reactive programming paradigm has been introduced (only compatible with Elasticsearch 6.8, [LTS version as you can see in the EOL table here](https://www.elastic.co/support/eol), as stated [here](https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/#preface.versions)): it is an asynchronous paradigm based on data streams and propagation of change and events, and its implementation has been realized through **WebClient**.

Anyway, **there are two ways of using these APIs**: one is the already mentioned Spring Data **Repositories** and the other one (the one we will see together) is the **Client**.

**In this article we will focus on the Client usage**that is a little bit different from the classic Spring Data scenario (the Repositories) and we will see how to get our data persisted in a way that allows to persist each of our beans without coupling it to the Elasticsearch Document model definition and to interact with the wide range of Elasticsearch REST API (so basically having the possibility to make the most of these APIs, going from the creation of an index with some particular configuration to the deletion of it).

In this way, we will see that **this kind of implementation** is not Entity-centric (and so Repository-centric), but mostly **Elasticsearch APIs-centric**: in fact, we are going to use a client that, on a lower level, calls the REST APIs made available by our documental database.

## Reactive Spring Data Elasticsearch Configuration

### 1. Prerequisites:

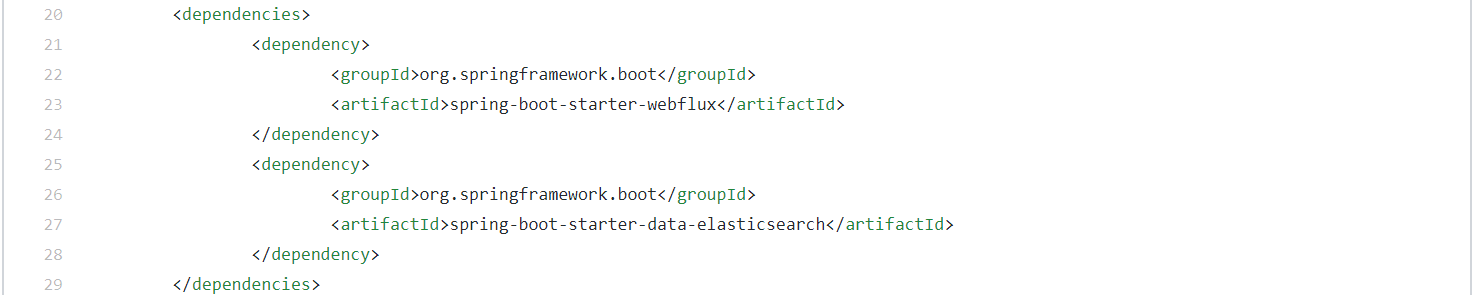
1. Spring Boot (at least 2.2.5.RELEASE) project (quick generation with [Spring Initializr](https://start.spring.io/))
2. Elasticsearch (6.8) instance (suggested a Docker instance, here's the [guide](https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html))

### 2. Dependencies & Configurations

In **application.yml**, add this configuration (click on image to jump to Github)**:**

[](http://github.com/paolodedo/reactive-spring-data-elasticsearch/blob/master/src/main/resources/application.yml)

In **pom.xml**, add these two dependencies (click on image to jump to Github)**:**

[](http://github.com/paolodedo/reactive-spring-data-elasticsearch/blob/master/pom.xml#L20)

Create a Java configuration class, **ElasticsearchConfig.java**(click on image to jump to Github):

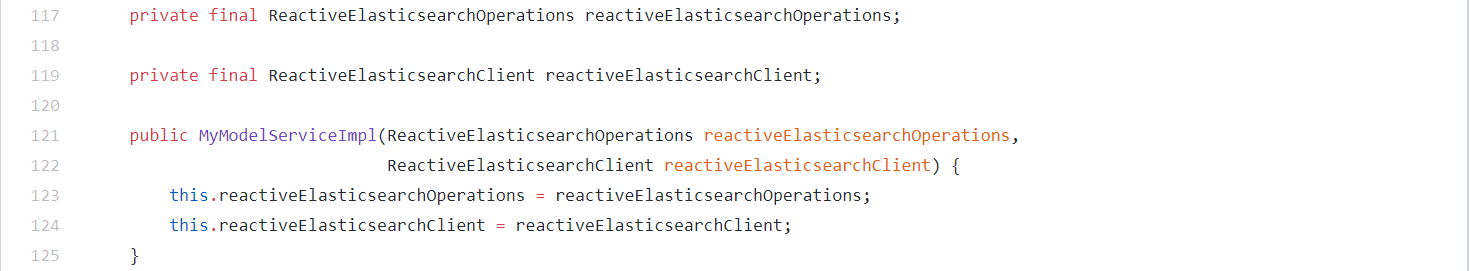
[package it.paolodedo.reactivespringdataelasticsearch.config;  import org.springframework.beans.factory.annotation.Value; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration; import org.springframework.data.elasticsearch.client.ClientConfiguration; import org.springframework.data.elasticsearch.client.reactive.ReactiveElasticsearchClient; import org.springframework.data.elasticsearch.client.reactive.ReactiveRestClients; import org.springframework.data.elasticsearch.core.ReactiveElasticsearchOperations; import org.springframework.data.elasticsearch.core.ReactiveElasticsearchTemplate; import org.springframework.data.elasticsearch.core.convert.ElasticsearchConverter; import org.springframework.data.elasticsearch.core.convert.MappingElasticsearchConverter; import org.springframework.data.elasticsearch.core.mapping.SimpleElasticsearchMappingContext; import org.springframework.web.reactive.function.client.ExchangeStrategies;  @Configuration public class ElasticsearchConfig {   @Bean  public ReactiveElasticsearchClient reactiveElasticsearchClient() {   ClientConfiguration clientConfiguration = ClientConfiguration.builder()    .connectedTo(elassandraHostAndPort)    .withWebClientConfigurer(webClient -> {     ExchangeStrategies exchangeStrategies = ExchangeStrategies.builder()      .codecs(configurer -> configurer.defaultCodecs()       .maxInMemorySize(-1))      .build();     return webClient.mutate().exchangeStrategies(exchangeStrategies).build();    })    .build();    return ReactiveRestClients.create(clientConfiguration);  }   @Bean  public ElasticsearchConverter elasticsearchConverter() {   return new MappingElasticsearchConverter(elasticsearchMappingContext());  }   @Bean  public SimpleElasticsearchMappingContext elasticsearchMappingContext() {   return new SimpleElasticsearchMappingContext();  }   @Bean  public ReactiveElasticsearchOperations reactiveElasticsearchOperations() {   return new ReactiveElasticsearchTemplate(reactiveElasticsearchClient(), elasticsearchConverter());  }   @Value("${spring.data.elasticsearch.client.reactive.endpoints}")  private String elassandraHostAndPort;  }](http://github.com/paolodedo/reactive-spring-data-elasticsearch/blob/master/src/main/java/it/paolodedo/reactivespringdataelasticsearch/config/ElasticsearchConfig.java)

### 3. API Usage

### ****3.1 Dependency Injection of Elasticsearch client interfaces****

Now, let's simply use the Dependency Injection in order to create and use an instance of **ReactiveElasticsearchOperations** and one of **ReactiveElasticsearchClient**interfaces.

To do that, create an interface named **MyModelService** and its service implementation **MyModelServiceImpl** (you will use these in your REST service or whatever) and let's inject the two above mentioned interfaces in our service class. [MyModelServiceImpl class (click on image to jump to Github)]:

[](http://github.com/paolodedo/reactive-spring-data-elasticsearch/blob/master/src/main/java/it/paolodedo/reactivespringdataelasticsearch/service/impl/MyModelServiceImpl.java#L117)

### 3.2 Use of Elasticsearch interfaces by our service and implementation of our service's methods

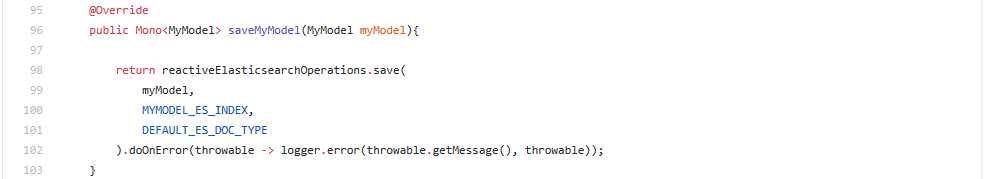
Now that we have instantiated the Elasticsearch client interfaces, we can start using them.

First, we can implement a mechanism to **create an index** with a custom mapping, **only if it is absent**, at the launch of our Spring Boot application (thanks to @PostConstruct annotation). [MyModelServiceImpl class (click on image to jump to Github)]:

[](http://github.com/paolodedo/reactive-spring-data-elasticsearch/blob/master/src/main/java/it/paolodedo/reactivespringdataelasticsearch/service/impl/MyModelServiceImpl.java#L27)

Now it's the time to implement our **CRUDs** operations. [MyModelServiceImpl class (click on image to jump to Github)]:

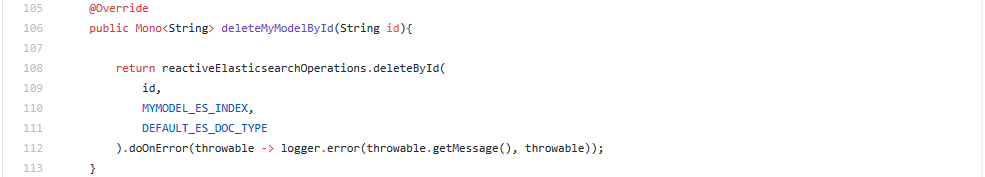
1. CREATE & UPDATE (simply set the ID on the model object you pass)

[](http://github.com/paolodedo/reactive-spring-data-elasticsearch/blob/master/src/main/java/it/paolodedo/reactivespringdataelasticsearch/service/impl/MyModelServiceImpl.java#L95)

2. READ (the findAll, so implemented, lets you filter by a value but you can get the 100% from it with Sort, Bool or Multimatch queries, Aggregations and so on)

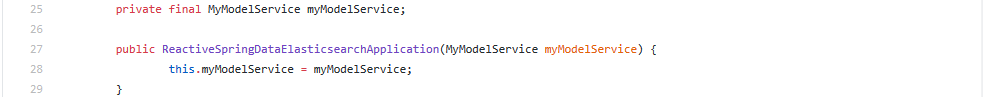
[](http://github.com/paolodedo/reactive-spring-data-elasticsearch/blob/master/src/main/java/it/paolodedo/reactivespringdataelasticsearch/service/impl/MyModelServiceImpl.java#L67)

3. DELETE (you can even implement DELETE by Object, it is the same as by ID)

[](http://github.com/paolodedo/reactive-spring-data-elasticsearch/blob/master/src/main/java/it/paolodedo/reactivespringdataelasticsearch/service/impl/MyModelServiceImpl.java#L105)

### 3.3 Use of our service's CRUD methods

To show you **how to use these methods we have implemented**, I have firstly **injected my MyModelService interface** into my Main Java class (Spring Boot Application class),[ReactiveSpringDataElasticsearchApplication class (click on image to jump to Github)]:

[](http://github.com/paolodedo/reactive-spring-data-elasticsearch/blob/master/src/main/java/it/paolodedo/reactivespringdataelasticsearch/ReactiveSpringDataElasticsearchApplication.java#L25)

then created a simple method launched by the @PostConstruct (I added some delay just to make sure the index is created by the other @PostConstruct annotated method). Note that this implementation is only for test. [ReactiveSpringDataElasticsearchApplication class (click on image to jump to Github)]:

[](http://github.com/paolodedo/reactive-spring-data-elasticsearch/blob/master/src/main/java/it/paolodedo/reactivespringdataelasticsearch/ReactiveSpringDataElasticsearchApplication.java#L29)

**We can see that every method that returns a Mono or Flux instance of something, is executed only when a subscribe() operation is called; thus, the flow is started and we can chain as much operations we want, keeping always in mind that the entire flow is asynchronous and reactive.**

Thank you for reading, I leave the [Github link to the source of this sample project](https://github.com/paolodedo/reactive-spring-data-elasticsearch) and the [link to the official Spring reference documentation](https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/).