

Every document has unique identifier .automatically or provided.

Documents are assigned to indexes.ex (customer data, sales data)

Another word on types

Phew, things sure are changing fast in regards to types! Not only are they being removed (which is great), but to prepare for version 7.x of Elasticsearch, you should use a slightly different syntax than what is used within most of this course as of today. Whenever you specify a type, you should use \_doc as the type name, because this will be the syntax that will be used in version 7.x. So by doing this, you will be future proofing your mappings and queries, ensuring that you won't face major headaches when upgrading Elasticsearch in the future.

At this point, you haven't seen any queries yet, but basically whenever you see me type default as the type, you should write \_doc instead. So when adding a document, for instance, you would see me type:

PUT /users/default/1

{

"name": "Bo Andersen"

}

Then you should type:

PUT /users/\_doc/1

{

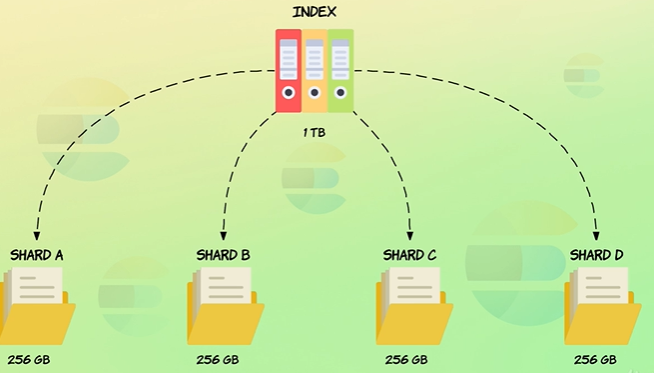
"name": "Bo Andersen"

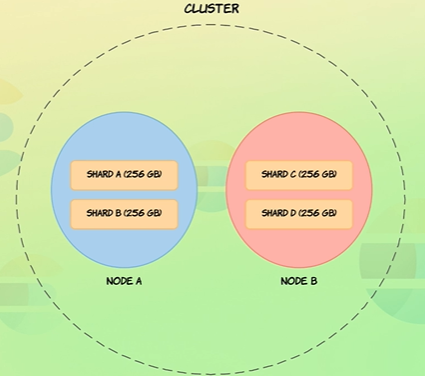
}

This will gradually be updated and reflected within the course, but please understand that re-recording and editing 10+ hours of content takes a very long time, so there will be a period in which you will see both types being specified.

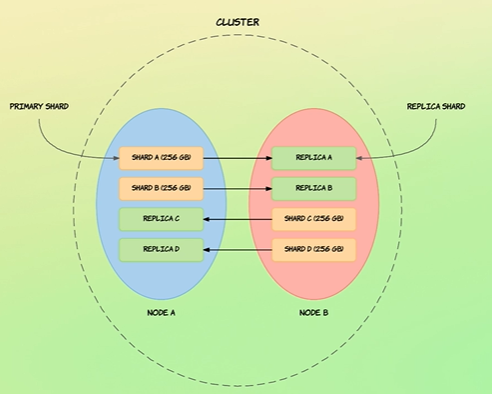
Note that you need to be running at least Elasticsearch 6.2.0 to be able to specify \_doc as the document type.

Thank you for your understanding, and happy searching!



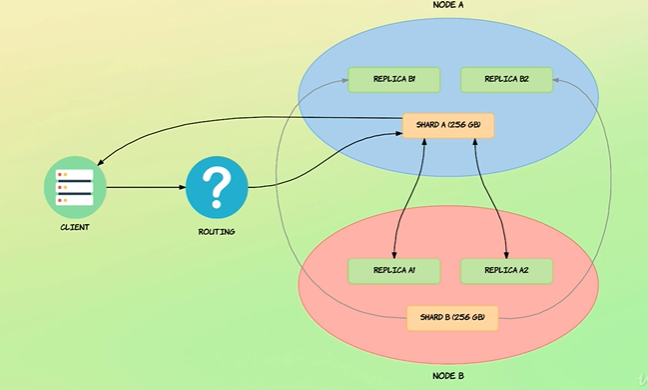


Sharding is a way of dividing index data volume in to multiple parts this enables to distribute data across multiple nodes. Ideally each node will be installed in different vms.

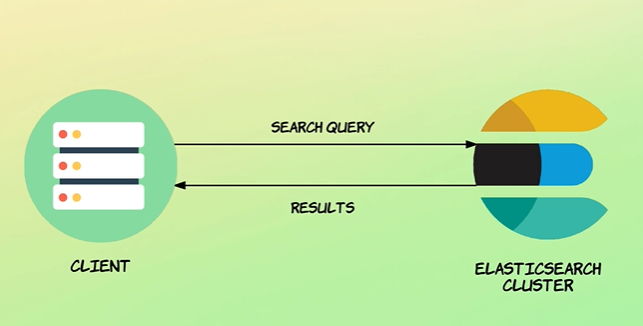


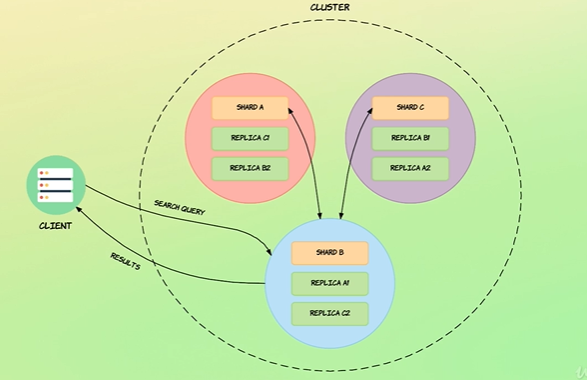
Primary shard gets replicated to replica shard. Write operation happens on primary shard.

Replicas will never be stored on same machine as its own shard. Replica is a copy of shard.

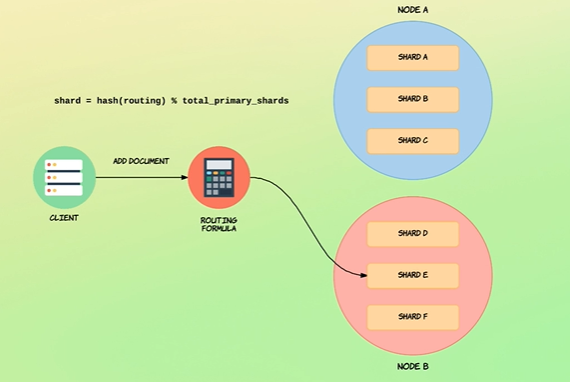


Any update, deletion or addition of data is done always on shard first then replicated in all the replicas.





Co-coordinating receives the request and sends it to other shards or replicas in other nodes and after receiving the result combines and sorts them returns to the client.

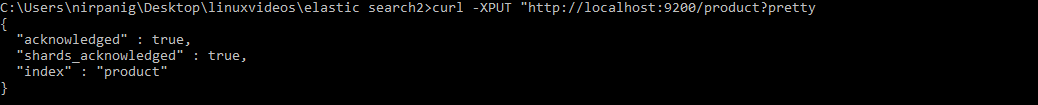


Number of shards cannot be changed once index has been created.

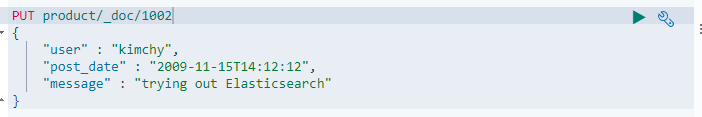
Elasticsearch.yml for configuring

**Creating index**

curl -XPUT "http://localhost:9200/product?pretty



**Creating a document**



curl -XPUT "http://localhost:9200/product/\_doc/1002" -H 'Content-Type: application/json' -d'{ "user" : "kimchy", "post\_date" : "2009-11-15T14:12:12", "message" : "trying out Elasticsearch"}'

**Searching Documents**

GET product/\_search 🡪

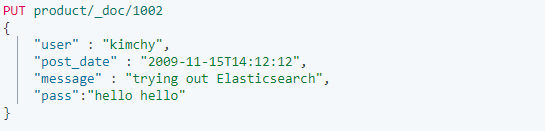
curl -XGET "http://localhost:9200/product/\_search"

GET product/\_doc/1 🡪 getting document based on id in kabana

curl -XGET "<http://localhost:9200/product/_doc/1>?pretty"

**Updating a document**

This way we can add a document or we can replace it with new document having extra field.



curl -XPUT "http://localhost:9200/product/\_doc/1002" -H 'Content-Type: application/json' -d'{ "user" : "kimchy", "post\_date" : "2009-11-15T14:12:12", "message" : "trying out Elasticsearch", "pass":"hello hello"}'



Each time the document is replaced its version number will increase.

**Updating specific fields in a document of elastic search**

In this approach we can update existing field’s value and also we can new fields to the document.

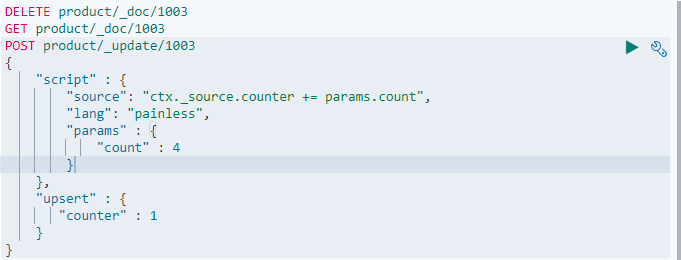


curl -XPOST "http://localhost:9200/product/\_update/1002?pretty" -H 'Content-Type: application/json' -d'{ "doc": { "user": "Jane Doee","price": 180 }}'

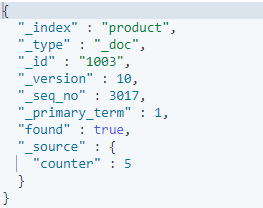
curl -XPOST "http://localhost:9200/product/\_update/1002?pretty" -H 'Content-Type: application/json' -d'{ "script" : "ctx.\_source.price += 5"}'

### Upsert

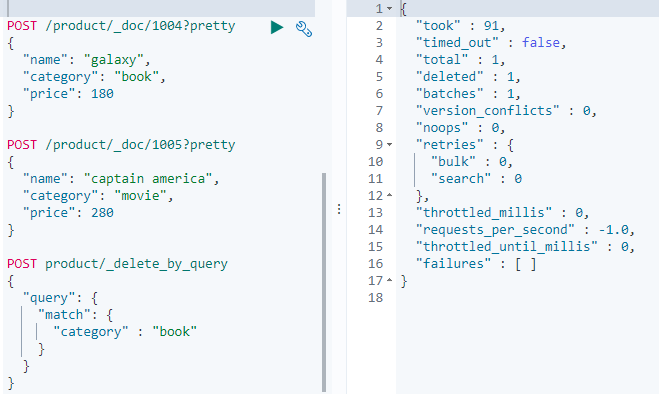
If the document does not already exist, the contents of the upsert element will be inserted as a new document. If the document does exist, then the script will be executed instead:



After running first time and second time the result given below.

**Delete By Query API**



**Deleting Index**



**Bulk API**

Using bulk api we can do multiple operation at the same time ie creating a document, updating multiple document or deleting documents.

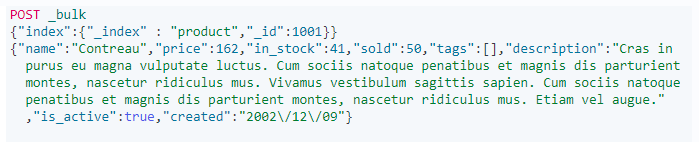


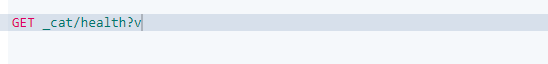
**Uploading bulk document**

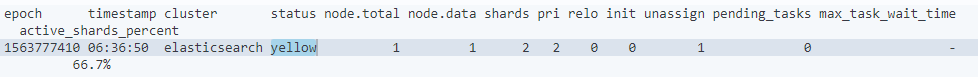


curl -s -H "Content-Type: application/x-ndjson" -XPOST localhost:9200/product/\_bulk?pretty --data-binary "@test-data.json"

Bulk posting example. Where \_index is the index under which we want to create a document.







Get information about cluster

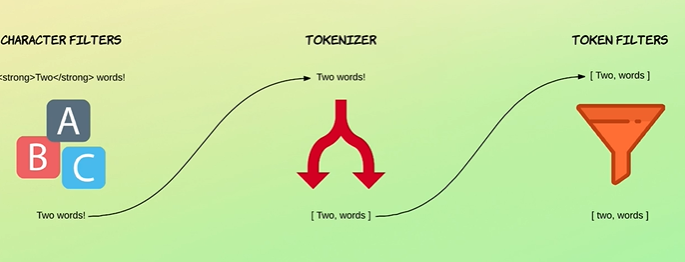
How to add mapping for document and add additional mapping after creating the mapping

Data types in elastic search

Need to write details about meta fields

In elastic search index mapping if dynamic is disabled and we created a document after that we are adding the mapping we will not able to filter it with that field.

We can reindex documents using update\_by\_query? conflicts=proceed



Character filter removes html mark up

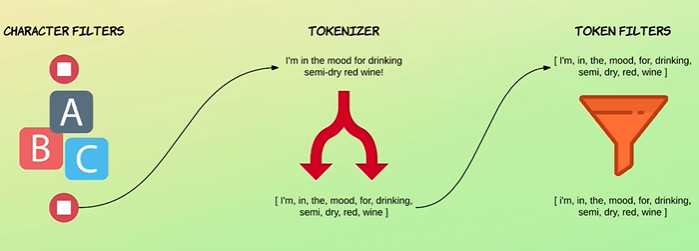
Tokenizer convers input multiple tokens based on white spaces removes commas semicolons hyphens.

Simplest Token filter convers the tokens in to lower case.

Stub removes common words the, an, a etc

Synonym token filter (nice, good)

**Standard Analyzer**

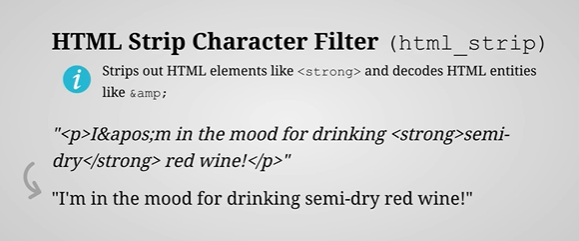


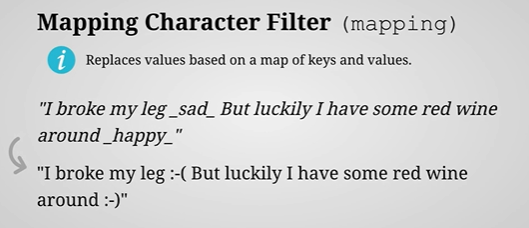
**Inverted Index**

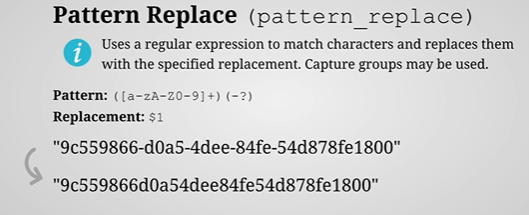
Inverted index is used for full text search. Analyzer is applied to full text fields and the results of this analysis is stored in an inverted index. An inverted is the collection of all the terms in all the documents



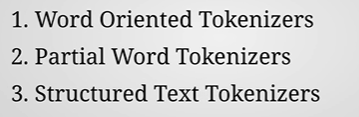
**Character Filter**



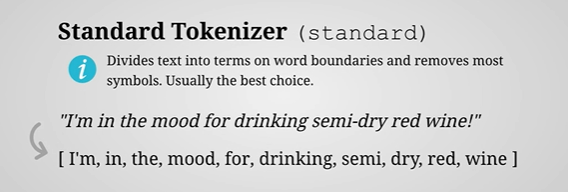


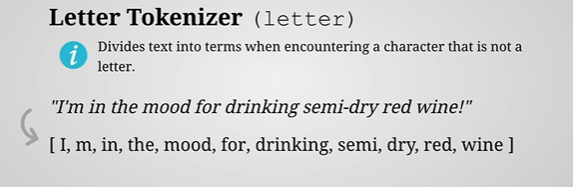


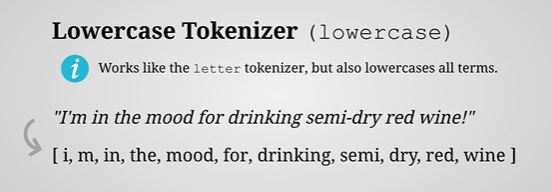
**Tokenizer**

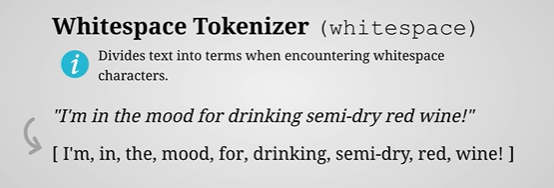








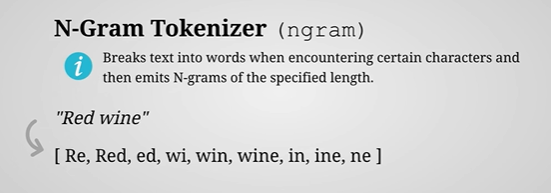


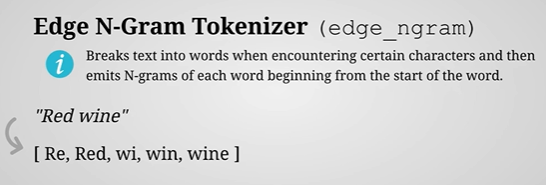




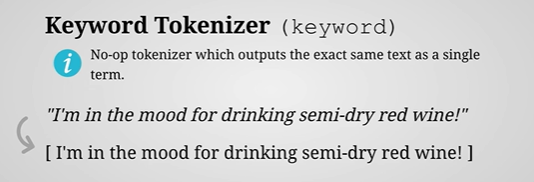
**Partial Word Tokenizer**

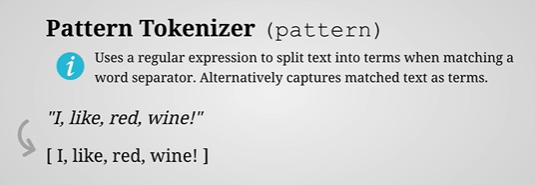


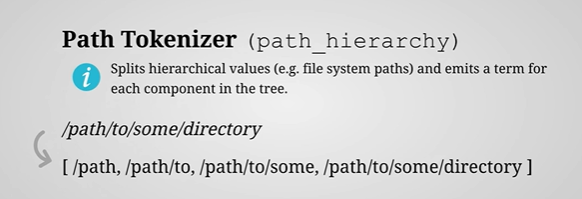






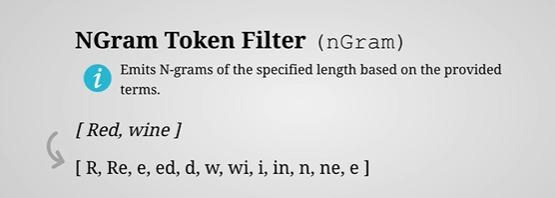


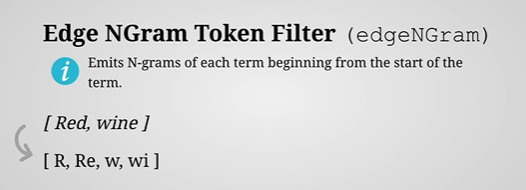




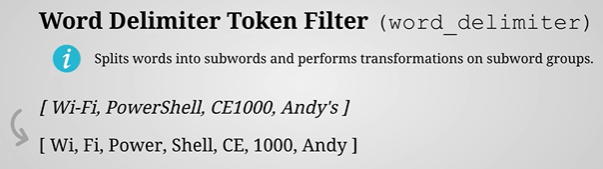
**Token Filters**



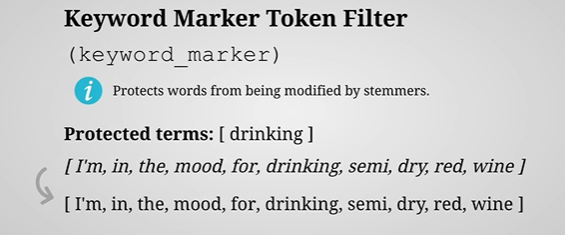








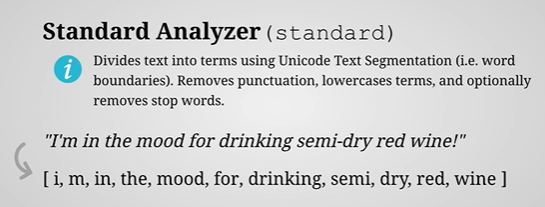


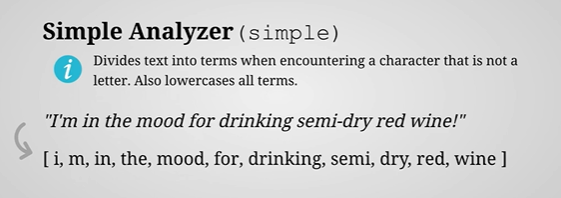


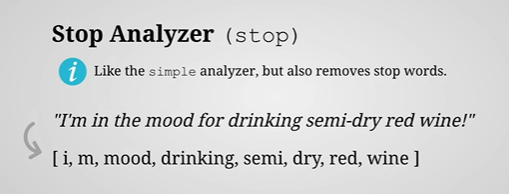


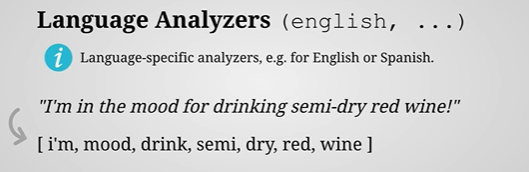


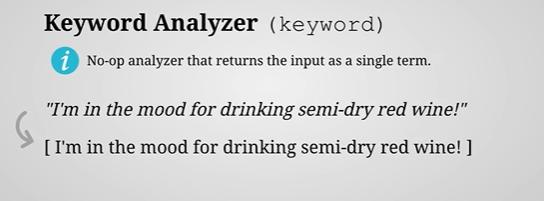
**Analyzers**



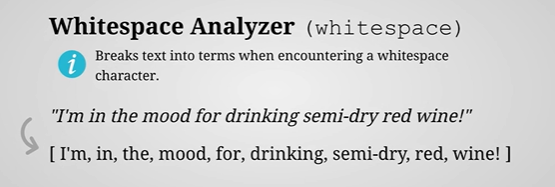




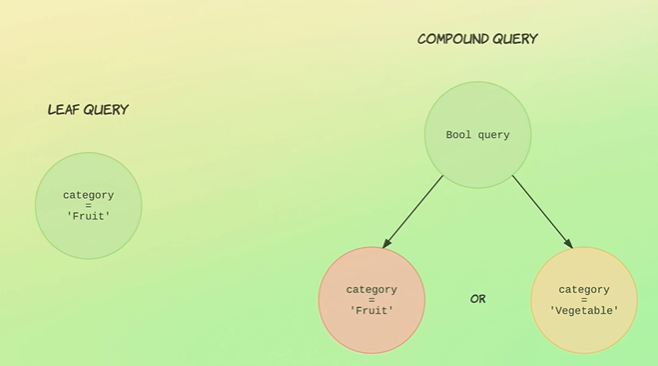








Analyzers can be applied at document level or field level. Field level overrides document level.

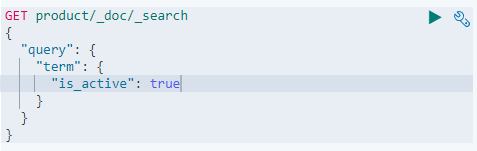


Query context calculates the relevance to how well the document match based on some algorithm. Filter context used whether the document contains the search field and that’s included in the result.

**Term level query and full text query**

Do not use term level query for full text search.

Term level queries are good for matching Boolean, dates, enums, and number.



## **2. Elasticsearch Repositories**

This chapter includes details of the Elasticsearch repository implementation.

### 2.1. Introduction

#### **2.1.1. Spring Namespace**

The Spring Data Elasticsearch module contains a custom namespace allowing definition of repository beans as well as elements for instantiating a ElasticsearchServer .

Using the repositories element looks up Spring Data repositories as described in [Creating Repository Instances](https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/#repositories.create-instances) .

*Example 48. Setting up Elasticsearch repositories using Namespace*

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:elasticsearch="http://www.springframework.org/schema/data/elasticsearch"

xsi:schemaLocation="http://www.springframework.org/schema/beans

https://www.springframework.org/schema/beans/spring-beans-3.1.xsd

http://www.springframework.org/schema/data/elasticsearch

https://www.springframework.org/schema/data/elasticsearch/spring-elasticsearch-1.0.xsd">

<elasticsearch:repositories base-package="com.acme.repositories" />

</beans>

Using the Transport Client or Node Client element registers an instance of Elasticsearch Server in the context.

*Example 49. Transport Client using Namespace*

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:elasticsearch="http://www.springframework.org/schema/data/elasticsearch"

xsi:schemaLocation="http://www.springframework.org/schema/beans

https://www.springframework.org/schema/beans/spring-beans-3.1.xsd

http://www.springframework.org/schema/data/elasticsearch

https://www.springframework.org/schema/data/elasticsearch/spring-elasticsearch-1.0.xsd">

<elasticsearch:transport-client id="client" cluster-nodes="localhost:9300,someip:9300" />

</beans>

*Example 50. Node Client using Namespace*

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:elasticsearch="http://www.springframework.org/schema/data/elasticsearch"

xsi:schemaLocation="http://www.springframework.org/schema/beans

https://www.springframework.org/schema/beans/spring-beans-3.1.xsd

http://www.springframework.org/schema/data/elasticsearch

https://www.springframework.org/schema/data/elasticsearch/spring-elasticsearch-1.0.xsd">

<elasticsearch:node-client id="client" local="true"" />

</beans>

#### **2.1.2. Annotation based configuration**

The Spring Data Elasticsearch repositories support cannot only be activated through an XML namespace but also using an annotation through JavaConfig.

*Example 51. Spring Data Elasticsearch repositories using JavaConfig*

@Configuration

@EnableElasticsearchRepositories(basePackages = "org/springframework/data/elasticsearch/repositories")

static class Config {

@Bean

public ElasticsearchOperations elasticsearchTemplate() {

return new ElasticsearchTemplate(nodeBuilder().local(true).node().client());

}

}

The configuration above sets up an Embedded Elasticsearch Server which is used by the ElasticsearchTemplate . Spring Data Elasticsearch Repositories are activated using the @EnableElasticsearchRepositories annotation, which essentially carries the same attributes as the XML namespace does. If no base package is configured, it will use the one the configuration class resides in.

#### **2.1.3. Elasticsearch Repositores using CDI**

The Spring Data Elasticsearch repositories can also be set up using CDI functionality.

*Example 52. Spring Data Elasticsearch repositories using JavaConfig*

class ElasticsearchTemplateProducer {

@Produces

@ApplicationScoped

public ElasticsearchOperations createElasticsearchTemplate() {

return new ElasticsearchTemplate(nodeBuilder().local(true).node().client());

}

}

class ProductService {

private ProductRepository repository;

public Page<Product> findAvailableBookByName(String name, Pageable pageable) {

return repository.findByAvailableTrueAndNameStartingWith(name, pageable);

}

@Inject

public void setRepository(ProductRepository repository) {

this.repository = repository;

}

}

### 2.2. Query methods

#### **2.2.1. Query lookup strategies**

The Elasticsearch module supports all basic query building feature as String,Abstract,Criteria or have it being derived from the method name.

##### **Declared queries**

Deriving the query from the method name is not always sufficient and/or may result in unreadable method names. In this case one might make either use of @Query annotation (see [Using @Query Annotation](https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/#elasticsearch.query-methods.at-query) ).

#### **2.2.2. Query creation**

Generally the query creation mechanism for Elasticsearch works as described in [Query methods](https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/#repositories.query-methods) . Here’s a short example of what a Elasticsearch query method translates into:

*Example 53. Query creation from method names*

public interface BookRepository extends Repository<Book, String>

{

List<Book> findByNameAndPrice(String name, Integer price);

}

The method name above will be translated into the following Elasticsearch json query

{ "bool" :

{ "must" :

[

{ "field" : {"name" : "?"} },

{ "field" : {"price" : "?"} }

]

}

}

A list of supported keywords for Elasticsearch is shown below.

| *Table 2. Supported keywords inside method names* | | |
| --- | --- | --- |
| **Keyword** | **Sample** | **Elasticsearch Query String** |
| And | findByNameAndPrice | {"bool" : {"must" : [ {"field" : {"name" : "?"}}, {"field" : {"price" : "?"}} ]}} |
| Or | findByNameOrPrice | {"bool" : {"should" : [ {"field" : {"name" : "?"}}, {"field" : {"price" : "?"}} ]}} |
| Is | findByName | {"bool" : {"must" : {"field" : {"name" : "?"}}}} |
| Not | findByNameNot | {"bool" : {"must\_not" : {"field" : {"name" : "?"}}}} |
| Between | findByPriceBetween | {"bool" : {"must" : {"range" : {"price" : {"from" : ?,"to" : ?,"include\_lower" : true,"include\_upper" : true}}}}} |
| LessThanEqual | findByPriceLessThan | {"bool" : {"must" : {"range" : {"price" : {"from" : null,"to" : ?,"include\_lower" : true,"include\_upper" : true}}}}} |
| GreaterThanEqual | findByPriceGreaterThan | {"bool" : {"must" : {"range" : {"price" : {"from" : ?,"to" : null,"include\_lower" : true,"include\_upper" : true}}}}} |
| Before | findByPriceBefore | {"bool" : {"must" : {"range" : {"price" : {"from" : null,"to" : ?,"include\_lower" : true,"include\_upper" : true}}}}} |
| After | findByPriceAfter | {"bool" : {"must" : {"range" : {"price" : {"from" : ?,"to" : null,"include\_lower" : true,"include\_upper" : true}}}}} |
| Like | findByNameLike | {"bool" : {"must" : {"field" : {"name" : {"query" : "?\*","analyze\_wildcard" : true}}}}} |
| StartingWith | findByNameStartingWith | {"bool" : {"must" : {"field" : {"name" : {"query" : "?\*","analyze\_wildcard" : true}}}}} |
| EndingWith | findByNameEndingWith | {"bool" : {"must" : {"field" : {"name" : {"query" : "\*?","analyze\_wildcard" : true}}}}} |
| Contains/Containing | findByNameContaining | {"bool" : {"must" : {"field" : {"name" : {"query" : "**?**","analyze\_wildcard" : true}}}}} |
| In | findByNameIn(Collection<String>names) | {"bool" : {"must" : {"bool" : {"should" : [ {"field" : {"name" : "?"}}, {"field" : {"name" : "?"}} ]}}}} |
| NotIn | findByNameNotIn(Collection<String>names) | {"bool" : {"must\_not" : {"bool" : {"should" : {"field" : {"name" : "?"}}}}}} |
| Near | findByStoreNear | Not Supported Yet ! |
| True | findByAvailableTrue | {"bool" : {"must" : {"field" : {"available" : true}}}} |
| False | findByAvailableFalse | {"bool" : {"must" : {"field" : {"available" : false}}}} |
| OrderBy | findByAvailableTrueOrderByNameDesc | {"sort" : [{ "name" : {"order" : "desc"} }],"bool" : {"must" : {"field" : {"available" : true}}}} |

#### **2.2.3. Using @Query Annotation**

*Example 54. Declare query at the method using the @Query annotation.*

public interface BookRepository extends ElasticsearchRepository<Book, String> {

@Query("{"bool" : {"must" : {"field" : {"name" : "?0"}}}}")

Page<Book> findByName(String name,Pageable pageable);

}

## **3. Miscellaneous Elasticsearch Operation Support**

This chapter covers additional support for Elasticsearch operations that cannot be directly accessed via the repository interface. It is recommended to add those operations as custom implementation as described in [Custom Implementations for Spring Data Repositories](https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/#repositories.custom-implementations) .

### 3.1. Filter Builder

Filter Builder improves query speed.

private ElasticsearchTemplate elasticsearchTemplate;

SearchQuery searchQuery = new NativeSearchQueryBuilder()

.withQuery(matchAllQuery())

.withFilter(boolFilter().must(termFilter("id", documentId)))

.build();

Page<SampleEntity> sampleEntities =

elasticsearchTemplate.queryForPage(searchQuery,SampleEntity.class);

### 3.2. Using Scroll For Big Result Set

Elasticsearch has a scroll API for getting big result set in chunks. ElasticsearchTemplate has startScroll and continueScroll methods that can be used as below.

*Example 55. Using startScroll and continueScroll*

SearchQuery searchQuery = new NativeSearchQueryBuilder()

.withQuery(matchAllQuery())

.withIndices(INDEX\_NAME)

.withTypes(TYPE\_NAME)

.withFields("message")

.withPageable(PageRequest.of(0, 10))

.build();

Page<SampleEntity> scroll = elasticsearchTemplate.startScroll(1000, searchQuery, SampleEntity.class);

String scrollId = ((ScrolledPage) scroll).getScrollId();

List<SampleEntity> sampleEntities = new ArrayList<>();

while (scroll.hasContent()) {

sampleEntities.addAll(scroll.getContent());

scrollId = ((ScrolledPage) scroll).getScrollId();

scroll = elasticsearchTemplate.continueScroll(scrollId, 1000, SampleEntity.class);

}

elasticsearchTemplate.clearScroll(scrollId);

ElasticsearchTemplate additionally has the stream method which wraps the scan and scroll operations into a CloseableIterator.

*Example 56. Using stream*

SearchQuery searchQuery = new NativeSearchQueryBuilder()

.withQuery(matchAllQuery())

.withIndices(INDEX\_NAME)

.withTypes(TYPE\_NAME)

.withFields("message")

.withPageable(PageRequest.of(0, 10))

.build();

CloseableIterator<SampleEntity> stream = elasticsearchTemplate.stream(searchQuery, SampleEntity.class);

List<SampleEntity> sampleEntities = new ArrayList<>();

while (stream.hasNext()) {

sampleEntities.add(stream.next());

}

# **Appendix**

## **Appendix A: Namespace reference**

### The <repositories /> Element

The <repositories /> element triggers the setup of the Spring Data repository infrastructure. The most important attribute is base-package, which defines the package to scan for Spring Data repository interfaces. See “[XML configuration](https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/#repositories.create-instances.spring)”. The following table describes the attributes of the <repositories /> element:

| *Table 3. Attributes* | |
| --- | --- |
| **Name** | **Description** |
| base-package | Defines the package to be scanned for repository interfaces that extend \*Repository (the actual interface is determined by the specific Spring Data module) in auto-detection mode. All packages below the configured package are scanned, too. Wildcards are allowed. |
| repository-impl-postfix | Defines the postfix to autodetect custom repository implementations. Classes whose names end with the configured postfix are considered as candidates. Defaults to Impl. |
| query-lookup-strategy | Determines the strategy to be used to create finder queries. See “[Query Lookup Strategies](https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/#repositories.query-methods.query-lookup-strategies)” for details. Defaults to create-if-not-found. |
| named-queries-location | Defines the location to search for a Properties file containing externally defined queries. |
| consider-nested-repositories | Whether nested repository interface definitions should be considered. Defaults to false. |

## **Appendix B: Populators namespace reference**

### The <populator /> element

The <populator /> element allows to populate the a data store via the Spring Data repository infrastructure.[[1](https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/#_footnotedef_1)]

| *Table 4. Attributes* | |
| --- | --- |
| **Name** | **Description** |
| locations | Where to find the files to read the objects from the repository shall be populated with. |

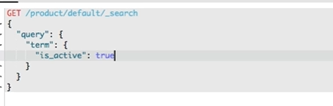
## **Appendix C: Repository query keywords**

### Supported query keywords

The following table lists the keywords generally supported by the Spring Data repository query derivation mechanism. However, consult the store-specific documentation for the exact list of supported keywords, because some keywords listed here might not be supported in a particular store.

| *Table 5. Query keywords* | |
| --- | --- |
| **Logical keyword** | **Keyword expressions** |
| AND | And |
| OR | Or |
| AFTER | After, IsAfter |
| BEFORE | Before, IsBefore |
| CONTAINING | Containing, IsContaining, Contains |
| BETWEEN | Between, IsBetween |
| ENDING\_WITH | EndingWith, IsEndingWith, EndsWith |
| EXISTS | Exists |
| FALSE | False, IsFalse |
| GREATER\_THAN | GreaterThan, IsGreaterThan |
| GREATER\_THAN\_EQUALS | GreaterThanEqual, IsGreaterThanEqual |
| IN | In, IsIn |
| IS | Is, Equals, (or no keyword) |
| IS\_EMPTY | IsEmpty, Empty |
| IS\_NOT\_EMPTY | IsNotEmpty, NotEmpty |
| IS\_NOT\_NULL | NotNull, IsNotNull |
| IS\_NULL | Null, IsNull |
| LESS\_THAN | LessThan, IsLessThan |
| LESS\_THAN\_EQUAL | LessThanEqual, IsLessThanEqual |
| LIKE | Like, IsLike |
| NEAR | Near, IsNear |
| NOT | Not, IsNot |
| NOT\_IN | NotIn, IsNotIn |
| NOT\_LIKE | NotLike, IsNotLike |
| REGEX | Regex, MatchesRegex, Matches |
| STARTING\_WITH | StartingWith, IsStartingWith, StartsWith |
| TRUE | True, IsTrue |
| WITHIN | Within, IsWithin |

Term level will do exact match



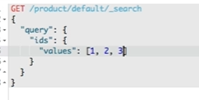


Both above are same

For matching multiple fields of tags field with ke



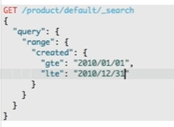
Search based multiple ids



Search based on range in\_stock field with range



Used for date range also with default format



Used for date range also with specified format

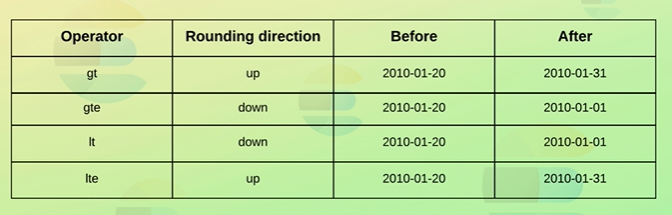


Matching by subtracting 1 year





Rounding my month











Checking non null fields

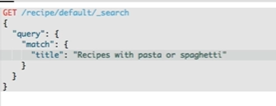




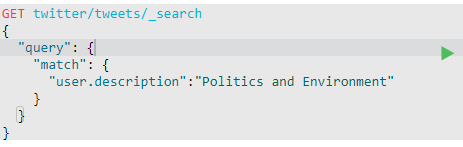




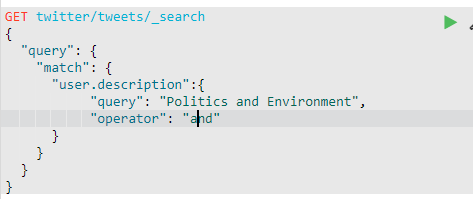
Match queries



Default is or operator for words

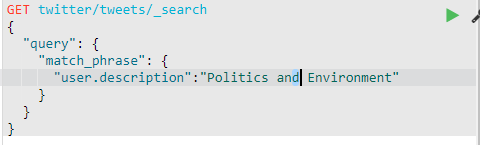






Match phrase maintains the order of the words

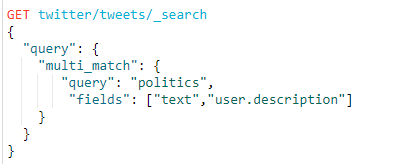




All words should be present in the same order

For searching in multiple fields





A query can run in query context or filter context

In query context relevance course is calculated documents are ordered by how well they match

Filter context only check for a document matches a query not how well it matches.



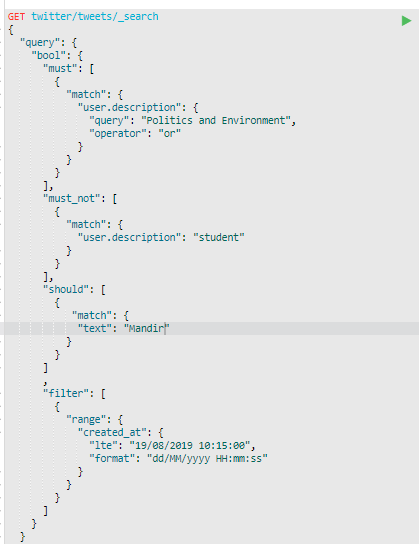








Should clause is not necessary. But if it is present the relevance score increases. It moves up in the result.



**Should** clause is optional till the time **must** and **filter** are present. In case one Should clause it is mandatory for document to get selected.