Oracle NoSQL Database

Full Text Search

12c Release 1

(Library Version 12.1.4.3)

Legal Notice

17-Feb-2017

Table of Contents

Introduction

Full Text Search (or just text search) provides the capability to identify natural-language documents that satisfy a query, and optionally to sort them by relevance to the query. The most common type of search is to find all documents containing given query terms and return them in order of their similarity to the query. Notions of query and similarity are very flexible and depend on the specific application. The simplest search considers query as a set of words and similarity as the frequency of query words in the document.

Oracle NoSQL Database integrates with a third-party open-source search engine, Elasticsearch (ES) to enable text-searching capability in Oracle NoSQL Database, in-concert with the Tables interface. For more information on:

- Elasticsearch, see: https://www.elastic.co/products/elasticsearch
- Tables interface, see: Introducing Oracle NoSQL Database Tables and Indexes in the *Oracle NoSQL Database Getting Started with the Table API*.

Full-text search is an important aspect of any big data and Oracle NoSQL Database system. Users expect that when they input text into a box and click "search", they will get the relevant search results they are looking for in a fraction of a second. This feature provides:

- High performance full-text search of Tables stored in Oracle NoSQL Database
- Search which allows users to explore a collection of information by applying multiple filters

Note

So that the maintenance of indexes does not affect the performance of an Oracle NoSQL Database store, text indexes will not be maintained locally by Oracle NoSQL Database components, but will instead be maintained by a remote service (Elasticsearch) hosted on other nodes.

This feature provides a means of marking fields in an Oracle NoSQL Database Tables schema as being text searchable. Text indexing allows creating indexes on Oracle NoSQL Database tables that cause the indexed fields to automatically enter into an Elasticsearch cluster. Once the data is in Elasticsearch, you may use any native Elasticsearch API to search and retrieve it. The documents retrieved from Elasticsearch contain references back to the original Oracle NoSQL Database records, facilitating their retrieval.

Prerequisite

To use this feature, you must have an Elasticsearch 2.0 cluster running as well as the Oracle NoSQL Database store. In a production environment, for performance reasons, both Oracle NoSQL Database nodes and Elasticsearch nodes are intended to be used in distributed environments with different hosts.

- You can download Elasticsearch here:

  https://www.elastic.co/downloads/past-releases/elasticsearch-2-0-0

- You can find the installation instructions here:

  https://www.elastic.co/guide/en/elasticsearch/reference/2.0/_installation.html

When your Elasticsearch cluster is running, it will consist of one or more nodes. Some or all of the nodes will have services listening on two ports.

- The HTTP port, which is used for REST requests. It is 9200 by default.
- The Elasticsearch transport port, which is used for communication between Elasticsearch nodes. It is 9300 by default.

Note

You must know the host name and transport port of at least one node in the cluster, and the name of the cluster itself, which by default is "elasticsearch". See the command `show parameters` in Integrating Elasticsearch with Oracle NoSQL Database (page 4). This information will be provided to the Oracle NoSQL Database store so that it can connect to the Elasticsearch cluster.

Integrating Elasticsearch with Oracle NoSQL Database

Before you can create a text index, you must register the Elasticsearch cluster with the Oracle NoSQL Database store, using the `register-es` plan command. In this command you provide the Elasticsearch cluster name, and the host name and transport port of any node in the cluster as follows:

```
plan register-es -clustername <name> -host <host>
-port <transport port> [-force]
```

For example:

```
kv-> plan register-es -clustername elasticsearch
-host 127.0.0.1 -port 9300
Started plan 5. Use show plan -id 5 to check status.
To wait for completion, use plan wait -id 5
```

Note

You will see an error message if the Elasticsearch cluster already contains "stale indexes" corresponding to the Oracle NoSQL Database. A stale index is one that was not created by the current instance of the store, but by a previous instance of the store, or by a concurrent instance of a store with the same name as the current store, which is not allowed.

The optional `-force` argument causes the Oracle NoSQL Database store to initialize an Elasticsearch cluster regardless of whether it already contains a stale index corresponding to the Oracle NoSQL Database store. See for example:

```
kv-> plan register-es -clustername elasticsearch
-host localhost -port 9300 -force
Started plan 38. Use show plan -id 38 to check status.
To wait for completion, use plan wait -id 38
```

Oracle NoSQL Database store Admin communicates with the Elasticsearch node to verify its existence, and it will acquire a complete list of connection information for all the nodes in the Elasticsearch cluster. This information will be stored and distributed to all the nodes of the Oracle NoSQL Database store. This command can be repeated if the Elasticsearch cluster's population of nodes changes significantly, to update Oracle NoSQL Database's list of Elasticsearch node connections.

If you want to verify that Elasticsearch is registered with your Oracle NoSQL Database store, run the following command:

```
show parameters -service <storage node id>
```

This command produces a list of properties which includes the cluster instance and the name of the cluster. See the `searchClusterMembers=127.0.0.1:9300` and `searchClusterName=elasticsearch` in the output below:

```
kv-> show parameters -service sn1
capacity=1
haHostname=localhost
haPortRange=5005,5007
hostname=localhost
memoryMB=0
mgmtClass=oracle.kv.impl.mgmt.NoOpAgent
mgmtPollPort=0
mgmtTrapPort=0
numCPUs=8
registryPort=5000
rnHeapMaxMB=0
rnHeapPercent=85
rootDirPath=./kvroot
searchClusterMembers=127.0.0.1:9300
searchClusterName=elasticsearch
serviceLogFileCount=20
serviceLogFileLimit=2000000
storageNodeId=1
systemPercent=10
```

Deregistering Elasticsearch from Oracle NoSQL Database Store

To deregister an Elasticsearch cluster from the Oracle NoSQL Database store, use the following command:

```
kv-> plan deregister-es
Executed plan 16, waiting for completion...
Plan 16 ended successfully
```

This is allowed only if all full text indexes are first removed using the following command:

```
DROP INDEX [IF EXISTS] index_name ON table_name
```

For more information, see Drop Index (page 25). Otherwise, you get the following error message:

```
kv-> plan deregister-es
Cannot deregister ES because these text indexes exist:
```

```
mytestIndex
JokeIndex
```

To verify if the deregistration of the Elasticsearch was successful or not, run the following command:

```
show parameters -service <storage node id>
```

Creating Full Text Index

You can create text indexes on the Oracle NoSQL Database table by using this DDL command:

```
CREATE FULLTEXT INDEX [if not exists] <index-name> ON <table-name>
    (<field-name> [ <mapping-spec> ], ...)
    [ES_SHARDS = <n>] [ES_REPLICAS = <n>
    [OVERRIDE] [COMMENT <comment>]
```

where:

- IF NOT EXISTS is optional, and it causes the `CREATE FULLTEXT INDEX` statement to be ignored if an index by that name currently exists. If this phrase is not specified, and an index using the specified name does currently exist, then the `CREATE FULLTEXT INDEX` statement will fail with an error.
- `index-name` is the name of the index you want to create.
- `table-name` is the name of the table that you want to index.
- `field-name` is the name of a field that you want to index.
- `mapping-spec` is a small JSON document that influences Elasticsearch's handling of the field. For more information, see the section Mapping Full Text Index Field to Elasticsearch Index Field (page 8).
- `ES_SHARDS` and `ES_REPLICAS` are optional properties that are transmitted to Elasticsearch when the corresponding text index is created. It is explained further below.
- `OVERRIDE` is optional and is used to force a creation that otherwise would be prohibited.

For more information, see Example - Creating Full Text Index (page 19).

After you create the text index, you can verify the same by using the following statement:

```
show indexes -table <tableName>
```

After you create the index, you can run the `show table` command that lists the full text index that you have created. This command will give the table structure including the indexes that have been created for that table:

```
show table -name <tableName>
```

For example:

```
kv-> show table -name mytestTable
{
"type" : "table",
```

```
"name" : "mytestTable",
"owner" : null,
"comment" : null,
"shardKey" : [ "id" ],
"primaryKey" : [ "id" ],
"fields" : [ {
"name" : "id",
"type" : "INTEGER",
"nullable" : true,
"default" : null
}, {
"name" : "category",
"type" : "STRING",
"nullable" : true,
"default" : null
}, {
"name" : "txt",
"type" : "STRING",
"nullable" : true,
"default" : null
} ],
"indexes" : [ {
"name" : "mytestIndex",
"comment" : null,
"fields" : [ "category", "txt" ]
} ]
}
```

Note

You cannot evolve an index. If you want to change the index definition, for example, add more columns to the index, you have to delete the index and create a new one.

To configure the number of shards and replicas for an Elasticsearch index, (for more information on these parameters in Elasticsearch, see: https://www.elastic.co/guide/en/elasticsearch/reference/current/_basic_concepts.html) the keywords `ES_SHARDS` and `ES_REPLICAS` are included for these values in the `CREATE FULLTEXT INDEX` statement.

Note

The assignments of `ES_SHARDS` and `ES_REPLICAS` are optional.

Since Elasticsearch does not allow to modify the number of shards after the index is created, it is recommended that the users pass value for this number of shards if the default value is not suitable.

The values assigned to `ES_SHARDS` and `ES_REPLICAS` are given as the values for `number_of_shards` and `number_of_replicas` when you create an index in Elasticsearch. For more information about the Elasticsearch index properties, see: https://www.elastic.co/guide/en/elasticsearch/guide/current/_index_settings.html

While creating index, `CREATE FULLTEXT INDEX` statement uses the `OVERRIDE` flag, which allows to delete any index existing in Elasticsearch by the same name as would be created by the command.

For example:

```
CREATE FULLTEXT INDEX mytestIndex
on  mytestTable (category, txt) OVERRIDE
```

Mapping Full Text Index Field to Elasticsearch Index Field

The `CREATE FULLTEXT INDEX` command is similar to the command that creates regular secondary indexes. One difference is the addition of the optional <mapping-spec> clause that follows the field name. If present, <mapping-spec> is a small JSON document that influences Elasticsearch's treatment of the field.

The other difference is optional settings of shards and replicas for the Elasticsearch index.

Note

The <mapping-spec> is an optional clause.

When a user creates a text index, an Elasticsearch index will be created and named as:

```
ondb.<store-name>.<table>.<textIndex>
```

Note

Index Name is in lowercase.

For more information, see the section Indexes Created in Elasticsearch (page 18).

The index contains a single mapping which is generated from the CREATE FULLTEXT INDEX command. See, https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html#mapping-type

One aspect of the mapping is a list of fields that compose the document type, along with their types. In the absence of a <mapping-spec>, Oracle NoSQL Database will supply a default type for each field that corresponds to the type of the column in the Oracle NoSQL Database table. For example, if a table column A has the type string, then the mapping supplied to Elasticsearch will declare a field named A of type string. If you want Elasticsearch to treat column A as an integer despite its being a string in Oracle NoSQL Database, you must provide an explicit type by including a <mapping-spec> clause:

```
{ "type" : "integer" }
```

The <mapping-spec>, in addition to specifying the type of the field, can also contain any of a large set of parameters for determining how Elasticsearch handles the field. For example, if you want to store the field, but not index it (that is, not make it available for search), you would include the tuple "index" : "no". For information on a list of such parameters, see: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-params.html

You may supply a <mapping-spec> that does not include the `type` key, and Oracle NoSQL Database will supply the default type.

The following scalar column type will be mapped to ES, STRING, INTEGER, LONG, BOOLEAN, FLOAT, DOUBLE. For example a Long ("LONG") in Oracle NoSQL Database will be mapped to Long ("long") in ES.

Note

Indexed fields can include non-scalar types, which are specified in the same way and with the same limitations as those for secondary indexes. For more information, see Indexing Non-Scalar Data Types in *Oracle Getting Started with Oracle NoSQL Database Tables*.

You may want to use mapping spec to put different analyzers on a field. For example, let us assume that you want two indexed fields: "category" and "txt". "Category" uses standard analyzer and for the field "txt" stemming is required, and you want to use snowball analyzer. This example assumes you have Elasticsearch version 2.x.

Note that the snowball analyzer provides stemming for some languages including English. Stemming tries to index the stem of the word instead of the given word. For more information, see https://en.wikipedia.org/wiki/Stemming and https://www.elastic.co/guide/en/elasticsearch/reference/2.4/analysis-snowball-analyzer.html. For example, the word "fitted" would get indexed as "fit".

You must then do the following:

1.
```
execute 'CREATE FULLTEXT INDEX JokeIndex ON Joke (category, txt)'
```

2.
```
plan register-es -clustername elasticsearch -host localhost
-port 9300  &lt;CHANGE VALUES ACCORDINGLY&gt;
```

3.
```
execute 'CREATE FULLTEXT INDEX JokeIndex ON Joke
(category{"type":"string","analyzer":"standard"},
txt{"analyzer":"snowball"})'
```

4.
```
put table -name Joke -json
'{ "id" : 2, "category" : "self-referential", "txt" :
"Is it solipsistic in here, or is it just me?" }'
```

5.
```
curl -XGET 'http://localhost:7200/ondb.kvlightstore.joke.jokeindex/
_search? q=txt:solipsist&pretty'
```

Note

In search call, the word "solipsistic" is actually queried by using the stemmed word "solipsist" and the effect of snowball analyzer is seen as it fetches document with the word "solipsistic".

NoSQL TIMESTAMP, a scalar data type is also supported for full text index. The maximum precision of NoSQL TIMESTAMP is 9 digits (nanosecond precision) and ES "date" type only supports 3 (millisecond precision). The NoSQL TIMESTAMP type is therefore mapped to the following 2 kinds of ES data types based on the specified precision:

1. If precision of TIMESTAMP type is in [0..3], then it maps to a ES "date" field:

```
{
    "type" : "date",
    "format" : "strict_date_optional_time||epoch_millis"
  }
```

Here, the "date" type represents the date and time with fractional second.

2. If precision of TIMESTAMP type is in [4..9], then it maps to a ES object that contains 2 fields:

   - "date" - Here the "date" type represents the date and time with no fractional second.
   - "nanos" - Is an integer type field. Here the "nanos" is the number of nano seconds less than 1 second.

```
{
    "properties" : {
      "date" : {
      "type" : "date",
    "format" : "strict_date_optional_time||epoch_millis"
    },
    "nanos" : {
       "type" : "integer"
    }
  }
```

Hence, for the query on "date" field for TIMESTAMP with precision 0 ~ 3, the field name is "<timestamp-field>", and query on "date" field for TIMESTAMP with precision 4 ~ 9, the field name in ES query string is "<timestamp-field>.date". See the following example:

1. Register elasticsearch cluster with NoSQL store.

```
plan register-es -clustername oracle_kv -host localhost
-port 9300 -wait
```

2. Table `textts0` contains TIMESTAMP(0) field and with text index on it.

```
execute "create table IF NOT EXISTS textts0
(id integer, ts0 TIMESTAMP(0), primary key(id))"
  execute "CREATE FULLTEXT INDEX IF NOT EXISTS idxts0 ON textts0 (ts0)"
  put table -name textts0 -json '{"id":1,"ts0":"1970-01-01"}'
  put table -name textts0 -json '{"id":2,"ts0":"2016-10-18T23:59:59"}'
  ...
```

3. Query on ts0 TIMESTMAP(0) field in range "1970-01-01" ~ "2016-10-19", the field name `ts0`:

```
curl -XGET 'localhost:9200/ondb.kvstore.textts0.idxts0/_search?pretty'
-d '{
       "query": {
           "range": {
               "ts0": {
                   "gte": "1970-01-01",
```

```
                    "lt": "2016-10-19"
                  }
              }
          }
    }'
```

4. Table textts6 contains TIMESTAMP(6) field and with text index on it.

```
execute "create table IF NOT EXISTS textts6
(id integer, ts6 TIMESTAMP(6), primary key(id))"
  execute "CREATE FULLTEXT INDEX IF NOT EXISTS idxts6 ON textts6 (ts6)"
  put table -name textts6 -json '{"id":1,"ts6":"1970-01-01"}'
  put table -name textts6 -json
  '{"id":2,"ts6":"2016-10-18T23:59:59.999999"}'
    ...
                }
              }
          }
    }'
```

5. Query on ts6 TIMESTMAP(6) in range "1970-01-01" ~ "2016-10-19", the field name is ts6.date.

```
curl -XGET 'localhost:9200/ondb.kvstore.textts6.idxts6/_search?pretty'
-d '{
       "query": {
           "range": {
               "ts6.date": {
                   "gte": "1970-01-01",
                   "lt": "2016-10-19T00:00:00"
                 }
              }
          }
    }'
```

Here is an example with full text index on TIEMSTAMP(9), it is represented with "Date" and "Nanos" in ES, we can do query on one of them or both of them:

1. Register elasticsearch cluster with NoSQL store.

```
plan register-es -clustername oracle_kv -host localhost
-port 9300 -wait
```

2. Create table, full text index on ts9.

```
execute "CREATE TABLE IF NOT EXISTS textts9(id integer,
ts9 TIMESTAMP(9), primary key(id))"
execute 'CREATE FULLTEXT INDEX IF NOT EXISTS idxts9 ON textts9 (ts9)'
```

3. Load 6 rows to table:

```
put table -name textts9 -json '{"id":1,"ts9":
"2016-01-01T01:00:00.300000001"}'
```

```
put table -name textts9 -json '{"id":2,"ts9":
"2016-01-02T01:00:00.100000001"}'
put table -name textts9 -json '{"id":3,"ts9":
"2016-01-02T01:00:00.200000001"}'
put table -name textts9 -json '{"id":4,"ts9":
"2016-01-02T01:00:00.300000001"}'
put table -name textts9 -json '{"id":5,"ts9":
"2016-01-03T02:00:00.123456789"}'
```

4. Query on ts9 with ts9.date is "2016-01-02T01:00:00".

```
curl -XGET 'localhost:9200/ondb.kvstore.textts9.idxts9/_search?pretty'
-d '{"query": {
    "term" : { "ts9.date":"2016-01-02T01:00:00"}
}}'
>
{
  "took" : 3,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 3,
    "max_score" : 1.9162908,
    "hits" : [ {
      "_index" : "ondb.kvstore.textts9.idxts9",
      "_type" : "text_index_mapping",
      "_id" : "/v/¨0003",
      "_score" : 1.9162908,
      "_source" : {
        "_pkey" : {
          "_table" : "textts9",
          "id" : "3"
        },
        "ts9" : {
          "date" : "2016-01-02T01:00:00",
          "nanos" : "200000001"
        }
      }
    }, {
      "_index" : "ondb.kvstore.textts9.idxts9",
      "_type" : "text_index_mapping",
      "_id" : "/v/¨0002",
      "_score" : 1.0,
      "_source" : {
        "_pkey" : {
          "_table" : "textts9",
```

```
          "id" : "2"
        },
        "ts9" : {
          "date" : "2016-01-02T01:00:00",
          "nanos" : "100000001"
        }
      }
    }, {
      "_index" : "ondb.kvstore.textts9.idxts9",
      "_type" : "text_index_mapping",
      "_id" : "/v/¨0004",
      "_score" : 0.30685282,
      "_source" : {
        "_pkey" : {
          "_table" : "textts9",
          "id" : "4"
        },
        "ts9" : {
          "date" : "2016-01-02T01:00:00",
          "nanos" : "300000001"
        }
      }
    } ]
  }
}
```

5. Query on ts9 with ts9.nanos is 200000001.

```
curl -XGET 'localhost:9200/ondb.kvstore.textts9.idxts9/_search?pretty'
-d '{"query": {
>     "term" : { "ts9.nanos":200000001}
> }}'
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 1.5108256,
    "hits" : [ {
      "_index" : "ondb.kvstore.textts9.idxts9",
      "_type" : "text_index_mapping",
      "_id" : "/v/¨0003",
      "_score" : 1.5108256,
      "_source" : {
        "_pkey" : {
```

```
          "_table" : "textts9",
          "id" : "3"
        },
        "ts9" : {
          "date" : "2016-01-02T01:00:00",
          "nanos" : "200000001"
        }
      }
    } ]
  }
}
```

6. Query on ts9 with ts9.date is "2016-01-02T01:00:00" and ts9.nanos is 200000001.

```
curl -XGET 'localhost:9200/ondb.kvstore.textts9.idxts9/_search?pretty'
-d '{"query": {
    "bool" : {
      "must" : {
        "term" : { "ts9.date":"2016-01-02T01:00:00"}
      },
      "must" : {
        "term" : { "ts9.nanos":200000001}
      }
    }
   }
}'
>
{
  "took" : 4,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 2.4402385,
    "hits" : [ {
      "_index" : "ondb.kvstore.textts9.idxts9",
      "_type" : "text_index_mapping",
      "_id" : "/v/¨0003",
      "_score" : 2.4402385,
      "_source" : {
        "_pkey" : {
          "_table" : "textts9",
          "id" : "3"
        },
        "ts9" : {
          "date" : "2016-01-02T01:00:00",
```

```
          "nanos" : "200000001"
        }
      }
    } ]
  }
}
```

7. Query on ts9 with ts9.date is "2016-01-02T01:00:00" and ts9.nanos is in range of 100000000 ~ 300000000.

```
curl -XGET 'localhost:9200/ondb.kvstore.textts9.idxts9/_search?pretty'
-d '{"query": {
    "bool" : {
      "must" : {
          "term" : { "ts9.date":"2016-01-02T01:00:00"}
       },
      "must" : {
        "range" : { "ts9.nanos":{"gte":100000000,"lte":300000000}}
      }
    }
   }
}'
>
{
  "took" : 12,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 2,
    "max_score" : 2.1615205,
    "hits" : [ {
      "_index" : "ondb.kvstore.textts9.idxts9",
      "_type" : "text_index_mapping",
      "_id" : "/v/¨0003",
      "_score" : 2.1615205,
      "_source" : {
        "_pkey" : {
          "_table" : "textts9",
          "id" : "3"
        },
        "ts9" : {
          "date" : "2016-01-02T01:00:00",
          "nanos" : "200000001"
        }
      }
    }, {
```

```
      "_index" : "ondb.kvstore.textts9.idxts9",
      "_type" : "text_index_mapping",
      "_id" : "/v/¨0002",
      "_score" : 1.4142135,
      "_source" : {
        "_pkey" : {
          "_table" : "textts9",
          "id" : "2"
        },
        "ts9" : {
          "date" : "2016-01-02T01:00:00",
          "nanos" : "100000001"
        }
      }
    } ]
  }
}
```

8. Sort on ts9.date, ts9.nanos.

```
curl -XGET 'localhost:9200/ondb.kvstore.textts9.idxts9/_search?pretty'
-d '{"sort":[{"ts9.date":"asc"}, {"ts9.nanos":"asc"}]}'
>
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 6,
    "max_score" : null,
    "hits" : [ {
      "_index" : "ondb.kvstore.textts9.idxts9",
      "_type" : "text_index_mapping",
      "_id" : "/v/¨0001",
      "_score" : null,
      "_source" : {
        "_pkey" : {
          "_table" : "textts9",
          "id" : "1"
        },
        "ts9" : {
          "date" : "2016-01-01T01:00:00",
          "nanos" : "300000001"
        }
      },
      "sort" : [ 1451610000000, 300000001 ]
```

```
    }, {
      "_index" : "ondb.kvstore.textts9.idxts9",
      "_type" : "text_index_mapping",
      "_id" : "/v/¨0002",
      "_score" : null,
      "_source" : {
        "_pkey" : {
          "_table" : "textts9",
          "id" : "2"
        },
        "ts9" : {
          "date" : "2016-01-02T01:00:00",
          "nanos" : "100000001"
        }
      },
      "sort" : [ 1451696400000, 100000001 ]
    }, {
      "_index" : "ondb.kvstore.textts9.idxts9",
      "_type" : "text_index_mapping",
      "_id" : "/v/¨0003",
      "_score" : null,
      "_source" : {
        "_pkey" : {
          "_table" : "textts9",
          "id" : "3"
        },
        "ts9" : {
          "date" : "2016-01-02T01:00:00",
          "nanos" : "200000001"
        }
      },
      "sort" : [ 1451696400000, 200000001 ]
    }, {
      "_index" : "ondb.kvstore.textts9.idxts9",
      "_type" : "text_index_mapping",
      "_id" : "/v/¨0004",
      "_score" : null,
      "_source" : {
        "_pkey" : {
          "_table" : "textts9",
          "id" : "4"
        },
        "ts9" : {
          "date" : "2016-01-02T01:00:00",
          "nanos" : "300000001"
        }
      },
      "sort" : [ 1451696400000, 300000001 ]
    }, {
```

```
      "_index" : "ondb.kvstore.textts9.idxts9",
      "_type" : "text_index_mapping",
      "_id" : "/v/¨0005",
      "_score" : null,
      "_source" : {
        "_pkey" : {
          "_table" : "textts9",
          "id" : "5"
        },
        "ts9" : {
          "date" : "2016-01-03T02:00:00",
          "nanos" : "123456789"
        }
      },
      "sort" : [ 1451786400000, 123456789 ]
    }, {
      "_index" : "ondb.kvstore.textts9.idxts9",
      "_type" : "text_index_mapping",
      "_id" : "/v/¨0006",
      "_score" : null,
      "_source" : {
        "_pkey" : {
          "_table" : "textts9",
          "id" : "6"
        },
        "ts9" : {
          "date" : "2016-01-03T06:00:00",
          "nanos" : "600000001"
        }
      },
      "sort" : [ 1451800800000, 600000001 ]
    } ]
  }
}
```

See also:

- TIMESTAMP(<precision>) in the *Getting Started with Oracle NoSQL Database Tables* guide.
- Timestamp in the *Getting Started with SQL For Oracle NoSQL Database* guide.

Indexes Created in Elasticsearch

For each text index in Oracle NoSQL Database store, an Elasticsearch index is created with a unique name:

```
ondb.<store-name>.<table>.<textIndex>
```

For example, for a text index `mytestIndex` in table `mytestTable` in store `mystore`, the corresponding Elasticsearch index would be:

```
ondb.mystore.mytesttable.mytestindex
```

Here the name of the index is ondb.<storename>.<tablename>.<indexname>, where <tablename> itself might contain multiple dotted component names, if it is the name of a child table.

Note

You will notice that an extra index is created in Elasticsearch with a name like ondb.<store-name>._checkpoint. You must not remove or modify this index. It contains internal information to help with recovery during restarts of Oracle NoSQL Database components.

Example - Creating Full Text Index

1. Create a table as follows:

```
kv-> execute 'CREATE TABLE mytestTable
(id INTEGER, category STRING, txt STRING, PRIMARY KEY (id))'
Statement completed successfully
```

2. Use the following command to make a text index on that table that indexes the category and txt columns:

```
kv-> execute 'CREATE FULLTEXT INDEX mytestIndex
ON mytestTable (category, txt)'
Statement completed successfully
```

3. Insert data into the "mytestTable" Table:

```
kv-> put table -name mytestTable -json
'{ "id" : 1, "category" : "pun", "txt" : "Spring is natures way of
saying, Let us party" }'
Operation successful, row inserted.
kv-> put table -name mytestTable -json
'{ "id" : 2, "category" : "self-referential", "txt" : "I am thankful
for the mess to clean after a party because it means I have been
surrounded by friends" }'
Operation successful, row inserted.
kv-> put table -name mytestTable -json
'{ "id" : 3, "category" : "stupid", "txt" : "Doing nothing is hard,
you never know when you are done" }'
Operation successful, row inserted.
kv-> put table -name mytestTable -json
'{ "id" : 4, "category" : "thoughtful", "txt" : "Do not worry if plan
A fails, there are 25 more letters in the alphabet" }'
Operation successful, row inserted.
kv-> get table -name mytestTable
{"id":4,"category":"thoughtful","txt":"Do not worry if plan A fails,
there are 25 more letters in the alphabet"}
{"id":1,"category":"pun","txt":"Spring is natures way of saying,
Let us party"}
```

```
{"id":2,"category":"self-referential","txt":"I am thankful for the
mess to clean after a party because it means I have been surrounded
by friends"}
{"id":3,"category":"stupid","txt":"Doing nothing is hard, you never
know when you are done"}
4 rows returned
```

Note

As you enter these records, Oracle NoSQL Database produces documents that is sent to Elasticsearch for indexing. You can find the document by searching the Elasticsearch cluster. For more information, see: https://www.elastic.co/guide/en/elasticsearch/guide/current/index.html

4. Search Elasticsearch cluster to find the document that was created in the steps before. Elasticsearch allows REST calls as queries, so we can do REST calls using the cURL command. The cURL command sends a request to the Elasticsearch node's http port.

Note

cURL is a common utility program that can issue and display the results of http requests. Currently, it is supported on Microsoft Windows, Linux, and Mac OS X. For more information, see: https://en.wikipedia.org/wiki/CURL and Search using cURL Command (page 20). However, cURL is an alternative method/option for querying Elasticsearch. The other options can be using:

- elasticsearch-head, a web front end for browsing and interacting with an Elasticsearch cluster, helps to query. elasticsearch-head is part of ES standard installation and can be enabled by following the steps mentioned here: https://mobz.github.io/elasticsearch-head/

  For more information, see Search using elasticsearch-head (page 22).

- Java API commands, see section Search Text Index using JAVA APIs (page 23).

Search using cURL Command

- Use the following cURL command to produce every document that is indexed with the mytestIndex mapping:

```
curl -s localhost:9200/ondb.mystore
.mytesttable.mytestindex/_search\?pretty
{
  "took" : 4,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
```

```
  "hits" : {
    "total" : 4,
    "max_score" : 1.0,
    "hits" : [ {
      "_index" : "ondb.mystore.mytesttable.mytestindex",
      "_type" : "text_index_mapping",
      "_id" : "/w/¨0003",
      "_score" : 1.0,
      "_source":{"_pkey":{"_table":"mytestTable","id":"3"},"category":
      "stupid","txt":       "Doing nothing is hard, you never know when
      you are done"}
    }, {
      "_index" : "ondb.mystore.mytesttable.mytestindex",
      "_type" : "text_index_mapping",
      "_id" : "/w/¨0002",
      "_score" : 1.0,
      "_source":{"_pkey":{"_table":"mytestTable","id":"2"},"category":
      "self-referential","txt": "I am thankful for the mess to clean
      after a party because it means I have been surrounded by friends"}
    }, {
      "_index" : "ondb.mystore.mytesttable.mytestindex",
      "_type" : "text_index_mapping",
      "_id" : "/w/¨0004",
      "_score" : 1.0,
      "_source":{"_pkey":{"_table":"mytestTable","id":"4"},"category":
      "thoughtful","txt": "Do not worry if plan A fails, there are 25
      more letters in the alphabet"}
    }, {
      "_index" : "ondb.mystore.mytesttable.mytestindex",
      "_type" : "text_index_mapping",
      "_id" : "/w/¨0001",
      "_score" : 1.0,
      "_source":{"_pkey":{"_table":"mytestTable","id":"1"},"category":
      "pun","txt": "Spring is natures way of saying, Let us party"}
    } ]
  }
}
```

Note

- The name that we gave to the text index in the `CREATE FULLTEXT` statement was `mytestIndex`, so we are restricting the search to the Elasticsearch index associated with that Oracle NoSQL Database text index.
- The "cURL" command above asked to search for every record in mytestIndex (there is no search term, so every record matches the search). The argument "pretty" means to pretty-print the output.

- The result contains an array of "hits" with a single member. The interesting property is "_source" which contains "_pkey" which has the table and primary key for the original kvstore record; and the two indexed fields "category" and "txt".

- Each item has a "_score" field which Elasticsearch uses to indicate the level of relevance for search hits. For more information, see: https://www.elastic.co/guide/en/elasticsearch/guide/current/relevance-intro.html

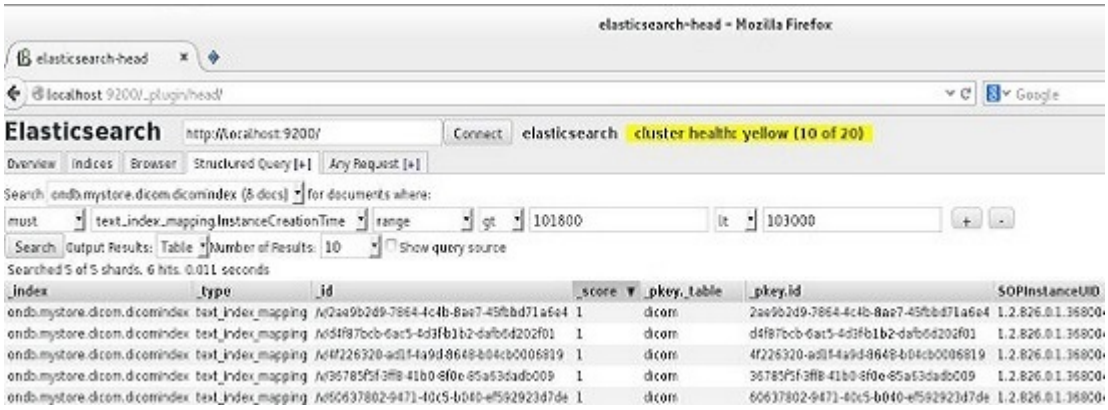
- You can narrow the search by putting a search term into request, using "q=" like the example below:

```
curl -s localhost:9200/ondb.mystore.mytesttable
.mytestindex/_search\?q=25\&pretty
{
  "took" : 11,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 0.25,
    "hits" : [ {
      "_index" : "ondb.mystore.mytesttable.mytestindex",
      "_type" : "text_index_mapping",
      "_id" : "/w/¨0004",
      "_score" : 0.25,
      "_source":{"_pkey":{"_table":"mytestTable","id":"4"},"category":
      "thoughtful","txt": "Do not worry if plan A fails, there are 25
      more letters in the alphabet"}
    } ]
  }
}
```

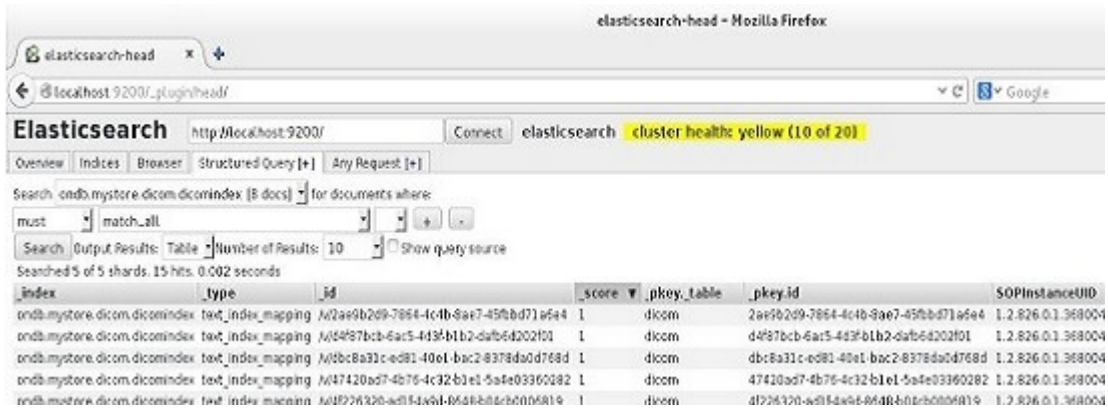
Search using elasticsearch-head

The following are the 2 query types:

- An aggregated query looks as follows:



- A query with "match_all" looks as follows:



For more information, see https://mobz.github.io/elasticsearch-head/

Search Text Index using JAVA APIs

For information on creating Oracle NoSQL Database tables, see Introducing Oracle NoSQL Database Tables and Indexes in the *Getting Started with Oracle NoSQL Database Tables* guide. To create a text index, you must use the method KVStore.executeSync to issue the DDL command CREATE FULLTEXT INDEX, as mentioned in the section Example - Creating Full Text Index (page 19).

Searching the index, on the other hand, must be done using Elasticsearch APIs. Here is a very simple example of a program that searches a document type that corresponds to an Oracle NoSQL Database text index. This command, given the arguments "localhost 9300 kvstore MyIndex 25" produces exactly the same output as the curl command in the section Search using cURL Command (page 20).

Note

To build and run this program, you will need all jar files supplied with the Elasticsearch distribution in your class path. One way to achieve this would be to use the java command's class path wildcard feature, for example:

```
java -cp ".:/home/…/elasticsearch-2.0.0/lib/*" \
    DoSearch localhost 9300 kvstore mytestIndex 25
```

See the following example program:

```
import java.net.InetAddress;

import org.elasticsearch.action.search.SearchRequestBuilder;
import org.elasticsearch.action.search.SearchResponse;
import org.elasticsearch.client.transport.TransportClient;
import org.elasticsearch.common.transport.InetSocketTransportAddress;
import org.elasticsearch.index.query.QueryBuilders;

public class DoSearch {

    public static void main(String args[]) throws Exception {

        if (args.length < 4 || args.length > 5) {
            System.err.println
                ("Usage: DoSearch <esTransportHost> <esTransportPort> " +
                 "<kvStoreName> <tableName> <indexName> [search-term]");
        }
        String esTransportHost = args[0];
        int esTransportPort = Integer.parseInt(args[1]);
        String kvStoreName = args[2];
        String tableName = args[3];
        String indexName = args[4];
        String searchTerm = args.length > 5 ? args[5] : null;

        TransportClient client =
            TransportClient.builder().build();

        client.addTransportAddress
            (new InetSocketTransportAddress
             (InetAddress.getByName(esTransportHost), esTransportPort));

        final String esIndexName = "ondb." + kvStoreName.toLowerCase()
        + "." +
            tableName + "." + indexName;

        SearchRequestBuilder sb = client.prepareSearch(esIndexName);
        if (searchTerm != null) {
            sb.setQuery(QueryBuilders.simpleQueryStringQuery(searchTerm));
        }
```

```
        SearchResponse response = sb.execute().actionGet();
        System.out.println(response);
    }
}
```

Drop Index

Text indexes share the same namespace as regular secondary indexes within a table. You can use the same statement to remove either type of index. `DROP INDEX` on a text index stops the population of the index from Oracle NoSQL Database shards, and removes the mapping and all related documents from Elasticsearch.

If a table to which a text index mapping refers is dropped, the text index will automatically be dropped as part of the process.

You can drop text indexes by using this DDL command:

```
DROP INDEX [IF EXISTS] index_name ON table_name [OVERRIDE]
DROP TABLE [IF EXISTS] table_name
```

For example:

```
kv-> execute 'drop table mytestTable'
Statement completed successfully
```

While deleting index, you can use the `OVERRIDE` flag. The `DROP INDEX` statement uses the `OVERRIDE` flag to enable overriding of the default constraints:

```
DROP INDEX [IF EXISTS] index_name ON table_name [OVERRIDE]
```

For example:

```
DROP INDEX mytestIndex on mytestTable OVERRIDE
```

For more information on the constraints, see Troubleshooting (page 25).

Troubleshooting

The most common problems that might arise when using Oracle NoSQL Database with Elasticsearch are those related to data transfer failure and data not getting indexed. For information on troubleshooting in Elasticsearch, see: https://www.elastic.co/guide/en/elasticsearch/hadoop/current/troubleshooting.html

The following sections describe some of the causes of these issues and provides steps you can follow to resolve these problems. Here are some things you can check to verify whether the data is successfully transferred and indexed:

- You can verify Oracle NoSQL Database's information that it uses to connect to the Elasticsearch cluster by issuing the runadmin command `show parameters -service sn1` where `sn1` is the id of any storage node in the store. The parameters of interest are `searchClusterName` which is the Elasticsearch cluster name; and `searchClusterMembers` which is a list of `host:port` representing the nodes in the Elasticsearch cluster.

- Be sure that when you are registering the Elasticsearch cluster, you give an ES node's transport port and not its http port.
- You can do a quick check of connectivity from the Oracle NoSQL Database master administrative node by re-registering the Elasticsearch cluster using the command `plan register-es`. This plan is safe to run multiple times. If it runs without errors, then the Elasticsearch cluster is at least available to the administrative node.
- Both Oracle NoSQL Database and Elasticsearch nodes should be configured to listen on appropriate network interfaces. If an Oracle NoSQL Database store is using a public interface, but Elasticsearch is using the loopback interface, then they will not be able to communicate properly. The network interface is configured for an Elasticsearch node by the property `network.host` in the `elasticsearch.yml`, and for an Oracle NoSQL Database storage node by the `-host` option to the `makebootconfig` command.
- You can issue a command like `curl http://localhost:9200/_cat/indices` to get a list of the indexes in an Elasticsearch cluster, to verify that they correspond to the text indexes you created in Oracle NoSQL Database. The output of this command will also show an indication of the status of each index using the color names green, yellow, and red. If the status is red, then the index cannot be populated.
- If you see unexplained failures to index, see the RepNode logs for SEVERE log messages or exceptions related to Elasticsearch. For information about finding logs, see Software Monitoring in the *Oracle NoSQL Database Runbook*.
- With a heavily update-dominated work load, the Elasticsearch cluster can lag quite far behind Oracle NoSQL Database. It can take minutes for a new Oracle NoSQL Database record to be reflected in search results from Elasticsearch. This issue can be mitigated by increasing the number of Elasticsearch nodes, tuning Elasticsearch, and especially by storing Elasticsearch data on solid state disks.
- Compare Record counts

  You can compare the number of records in table for Oracle NoSQL Database with the number of documents in your Elasticsearch cluster (this assumes that the Oracle NoSQL Database has a static number of items). This is particularly useful, for instance your cluster is in a test environment where the number of records is set and you do not add more. To find the number of records in Oracle NoSQL Database, use the following statement in the Command Line Interface (CLI):

  ```
  kv-> aggregate table -name mytestTable -count
  Row count: 100
  ```

  To get the number of records for Elasticsearch, use the following command: http://localhost:9200/ondb.mystore.joke.jokeindex/_count

  If it is successful, you get the following response:

  ```
  {"count":100,"_shards":
      {
      "total":5,"successful":5,"failed":0
  ```

```
    }
}
```

The count returned by Elasticsearch is the same value as the number of records shown in the CLI. The matching values provide assurance that all records have been transferred and indexed by Elasticsearch.

- ElasticServer Version Mismatch

  You must use Elasticsearch 2.0 version. A different version, for example ES 1.7.3, populates the following error:

```
kv-> plan register-es -clustername elasticsearch -host
localhost -port 9300
Can't connect to an Elasticsearch cluster at localhost:9300
{elasticsearch}
```

  The Admin log shows this:

```
2016-04-05 20:20:19.512 UTC INFO
[admin1] [Washout] loaded [], sites []
2016-04-05 20:20:20.434 UTC INFO [admin1] [Washout]
failed to get local cluster state for
{#transport#-1}{127.0.0.1}{localhost/127.0.0.1:9300},
disconnecting...
RemoteTransportException[[Failed to deserialize response
of type [org.elasticsearch.action.admin.cluster.state.
ClusterStateResponse]]];
nested: TransportSerializationException
[Failed to deserialize response of
type [org.elasticsearch.action.admin.cluster.state.
ClusterStateResponse]]; nested:
IndexOutOfBoundsException[Readable byte limit exceeded: 107];
```

- Check Elasticsearch Mappings

  You can influence the mapping by including a mapping-spec in the original CREATE command, but you cannot provide your own mapping. Be aware that this default mapping from Elasticsearch includes assumptions about data types and data structures in your documents. On the basis of these assumptions, Elasticsearch may omit your document from the index. For example, string mapped to object will not be indexed in the current version. Also, a record that contains only empty strings or nulls in the indexed fields will be omitted. For more information about expected data structures see: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html

- The population of a new text index begins immediately when it is created. If the existing database is large, this operation can take quite a long time. Furthermore, when a text index is created, if other text indexes that were created earlier already exist, they too will be populated over again from scratch. This is because all text indexes share the single gateway in server to stream data to the Elasticsearch cluster, and due to the newly added index, the gateway has to start from the very beginning, resulting re-populating all indexes. For

these reasons it is best to create all the text indexes that you need at the same time, and preferably before the database has been populated.

- When you execute the `CREATE FULLTEXT INDEX` statement, the Elasticsearch determines whether an index already exists with the same name. If such an index exists, the CREATE statement will fail, unless the OVERRIDE flag is given; in which case the existing index will be deleted before the new index is created by the same name. For more information on using OVERRIDE flag, see Creating Full Text Index (page 6).
- When the `DROP INDEX` statement is executed to remove a text index, the health status of the Elasticsearch cluster must be GREEN, unless the `OVERRIDE` flag is given, in which case the deletion will proceed. The metadata describing the index will be removed from Oracle NoSQL Database. However, there is a possibility that the corresponding Elasticsearch index will not be removed. For more information on using OVERRIDE flag, see Drop Index (page 25).

  This constraint helps to avoid an issue where an Elasticsearch index deletion can be undone when a node that was offline during the deletion returns to the cluster.

Security Configuration

This section details that Full Text Search and a secure Oracle NoSQL Database store are disjoint, that is, if Oracle NoSQL Database is configured as a secure store, Full Text Search should be disabled. On the other hand, if Full Text Search is enabled (that is, an external Elasticsearch cluster is registered) in a nonsecure store, users cannot reconfigure the nonsecure store to a secure store, unless Full Text Search is disabled before reconfiguration. Consider the following scenarios:

- For a new instance, if the user enables Full Text Search, then security cannot be enabled. For all new instances of Oracle NoSQL Database created from scratch, Full Text Search cannot be defined until the user registers an external Elasticsearch cluster successfully.
- For a new instance, if the user enables security, Full Text Search cannot be used. If a user configures a new Oracle NoSQL Database instance with security enabled, user will not be able to register any external Elasticsearch cluster. In this case, an IllegalCommandException will be raised with error message `Please unsecure the store to register ElasticSearch`.
- For an existing store that has security disabled, Full Text Search can be added. If the Oracle NoSQL Database instance is configured as nonsecure store, Full Text Search can be enabled and used as usual.
- For an existing instance that has security enabled, Full Text Search cannot be added. The user will not be able to register any external Elasticsearch cluster if the Oracle NoSQL Database is configured as a secure store.
- For an existing store that does not have security but has Full Text Search, the store cannot be made secure without removing Full Text Search. If the Oracle NoSQL Database instance is originally configured as a nonsecure store, and Full Text Search is enabled by registering an external Elasticsearch cluster, user will not be able to reconfigure the nonsecure store to a secure store. However, you can modify the configuration file by the following 2 ways:

1. Modifying the Configuration File using the Automated Tool

   You can use the automated tool `securityconfig` to configure a nonsecure store to a secure store. For more information on using the securityconfig tool to perform the security configuration of your store, see Security Configuration in the *Oracle NoSQL Database Security Guide*. If an external Elasticsearch is already registered, the configuration tool fails with an error messages that the user needs to deregister the Elasticsearch cluster first and then reconfigure:

   ```
   java -jar /Users/junyi/work/oracle/hg/lib/kvstore.jar securityconfig
   security-> config create -pwdmgr wallet -root /var/tmp/kvroot
   Enter a password for the Java KeyStore:
   Re-enter the KeyStore password for verification:
   Created files
   /var/tmp/kvroot/security/client.security
   /var/tmp/kvroot/security/client.trust
   /var/tmp/kvroot/security/security.xml
   /var/tmp/kvroot/security/store.keys
   /var/tmp/kvroot/security/store.trust
   /var/tmp/kvroot/security/store.wallet/cwallet.sso
   Created
   security-> config add-security -root /var/tmp/kvroot
   Error handling command config add-security -root /var/tmp/kvroot:
   The configuration cannot be enabled in a store with registered
   ES cluster <esClusterName>, please first deregister the
   ES cluster from the non-secure store, and reconfigure.
   ```

2. Modifying the Configuration File Manually

   If a user manually edits and changes the configuration file to convert to a secure store from nonsecure store with registered Elasticsearch cluster, without using the automated tool, the SNA would not be able to restart successfully. SNA raises an `IllegalStateException` error message `Secure store is not allowed if there is a registered ES cluster`. To fix it, the user must:

   - Withdraw the changes made to the configuration file
   - Restart Oracle NoSQL Database instance and get the original nonsecure store
   - Deregister the Elasticsearch cluster

     If you have any existing Text Index created, you need to drop those before deregistering the Elasticsearch cluster. For more information, see Deregistering Elasticsearch from Oracle NoSQL Database Store (page 5) section.

   - Restore the change of a secure store
   - Restart Oracle NoSQL Database instance and get a secure store without registering Elasticsearch cluster.

- For an existing store that does not have security enabled and also does not have Full Text Search, you can enable security. If the store is configured as nonsecure store and there is no Elasticsearch cluster registered, user will be able to reconfigure the store to a secure store normally.

  ## Note

  For a secure store, upgrading from version 4.0 to 4.1 is not permitted when Full Text Search is enabled. You should either disable security or Full Text Search before upgrading. Otherwise, after upgrading a storage node, restarting the node will fail with an error message asking users to restart Storage Node with previous library and disable either Full Text Search or security before upgrading.