
Elassandra Documentation

Release 5.5.0.4

Strapdata

Nov 19, 2017

Contents

1	Architecture	3
1.1	Concepts Mapping	4
1.2	Durability	4
1.3	Shards and Replica	5
1.4	Write path	8
1.5	Search path	9
2	Installation	11
2.1	Tarball	11
2.2	deb	12
2.3	rpm	13
2.4	Docker image	13
2.4.1	Start an elassandra server instance	13
2.4.2	Connect to Cassandra from an application in another Docker container	14
2.4.3	Environment Variables	14
2.4.4	Files locations	15
2.4.5	Exposed ports	15
2.4.6	Make a cluster	16
2.5	Build from source	16
3	Configuration	17
3.1	Directory Layout	17
3.2	Configuration	17
3.3	Logging configuration	18
3.4	Multi datacenter configuration	18
3.5	Elassandra Settings	19
3.6	Sizing and tuning	22
3.6.1	Write performances	22
3.6.2	Search performances	22
4	Mapping	25
4.1	Type mapping	25
4.2	Bidirectionnal mapping	27
4.3	Meta-Fields	27
4.4	Mapping change with zero downtime	28
4.5	Partitioned Index	29
4.6	Object and Nested mapping	31

4.7	Dynamic mapping of Cassandra Map	31
4.7.1	Dynamic Template with Dynamic Mapping	33
4.8	Parent-Child Relationship	34
4.9	Indexing Cassandra static columns	36
4.10	Elassandra as a JSON-REST Gateway	38
4.11	Check Cassandra consistency with elasticsearch	40
5	Operations	43
5.1	Indexing	43
5.2	GETing	44
5.3	Updates	46
5.4	Searching	46
5.4.1	Optimizing search requests	47
5.4.2	Caching features	48
5.5	Create, delete and rebuild index	49
5.6	Open, close, index	50
5.7	Flush, refresh index	51
5.8	Managing Elassandra nodes	51
5.9	Backup and restore	51
5.9.1	Restoring a snapshot	52
5.9.2	Point in time recovery	52
5.9.3	Restoring to a different cluster	52
5.10	How to change the elassandra cluster name	53
6	Enterprise	55
6.1	Installation	55
6.2	License management	56
6.2.1	License installation	56
6.2.2	Checking your license	57
6.2.3	Upgrading your license	57
6.3	JMX Managment & Monitoring	58
6.3.1	JMX Monitoring	58
6.3.2	Enable/Disable search on a node	60
6.4	SSL Network Encryption	60
6.4.1	Elasticsearch SSL configuration	62
6.4.2	JMX traffic Encryption	62
6.5	Authentication and Authorization	63
6.5.1	Cassandra internal authentication	63
6.5.2	Privileges	64
6.5.3	Permissions	65
6.5.4	Privilege caching	66
6.6	Integration	67
6.6.1	Secured Transport Client	67
6.6.2	Multi-user Kibana configuration	68
6.6.3	Elasticsearch Spark connector	69
6.7	Elasticsearch Auditing	69
6.7.1	Logback Audit	70
6.7.2	CQL Audit	71
6.8	Limitations	72
6.8.1	Content-Based Security Limitations	72
7	Integration	75
7.1	Integration with an existing cassandra cluster	75
7.1.1	Rolling upgrade to elassandra	75

7.1.2	Create a new elassandra datacenter	75
7.2	Installing an Elasticsearch plugins	76
7.3	Running Kibana with Elassandra	76
7.4	JDBC Driver sql4es + Elassandra	77
7.5	Running Spark with Elassandra	77
8	Testing	79
8.1	Testing environnement	79
8.2	Elassandra unit test	79
9	Breaking changes and limitations	81
9.1	Deleting an index does not delete cassandra data	81
9.2	Cannot index document with empty mapping	81
9.3	Nested or Object types cannot be empty	82
9.4	Document version is meaningless	82
9.5	Index and type names	83
9.6	Column names	83
9.7	Null values	83
9.8	Elasticsearch unsupported features	83
9.9	Cassandra limitations	83
10	Indices and tables	85

Elassandra tightly integrates [Elasticsearch](#) in [Cassandra](#).

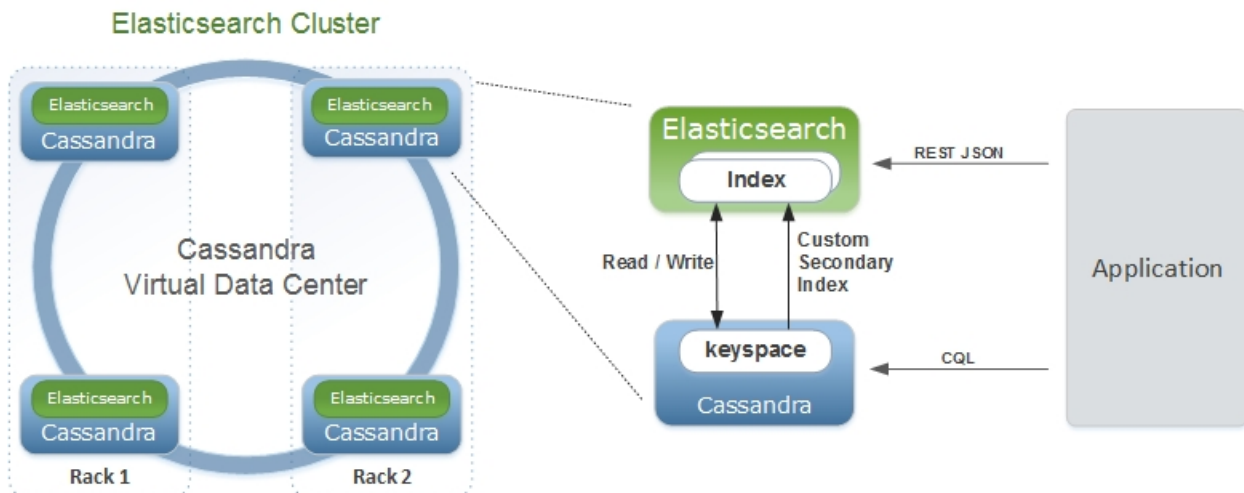
Contents:

CHAPTER 1

Architecture

Elassandra tightly integrates elasticsearch within cassandra as a secondary index, allowing near-realtime search with all existing elasticsearch APIs, plugins and tools like Kibana.

When you index a document, the JSON document is stored as a row in a cassandra table and synchronously indexed in elasticsearch.



1.1 Concepts Mapping

Elasticsearch	Cassandra	Description
Cluster	Virtual Datacenter	All nodes of a datacenter forms an Elasticsearch cluster
Shard	Node	Each cassandra node is an elasticsearch shard for each indexed keyspace
Index	Keyspace	An elasticsearch index is backed by a keyspace
Type	Table	Each elasticsearch document type is backed by a cassandra table
Document	Row	An elasticsearch document is backed by a cassandra row
Field	Cell	Each indexed field is backed by a cassandra cell (row x column)
Object or nested field	User Defined Type	Automatically create User Defined Type to store elasticsearch object

From an Elasticsearch perspective :

- An Elasticsearch cluster is a Cassandra virtual datacenter.
- Every Elassandra node is a master primary data node.
- Each node only index local data and acts as a primary local shard.
- Elasticsearch data is not more stored in lucene indices, but in cassandra tables.
 - An Elasticsearch index is mapped to a cassandra keyspace,
 - Elasticsearch document type is mapped to a cassandra table.
 - Elasticsearch document `_id` is a string representation of the cassandra primary key.
- Elasticsearch discovery now rely on the cassandra [gossip protocol](#). When a node join or leave the cluster, or when a schema change occurs, each nodes update nodes status and its local routing table.
- Elasticsearch [gateway](#) now store metadata in a cassandra table and in the cassandra schema. Metadata updates are played sequentially through a [cassandra lightweight transaction](#). Metadata UUID is the cassandra `hostId` of the last modifier node.
- Elasticsearch REST and java API remain unchanged.
- Logging is now based on [logback](#) as cassandra.

From a Cassandra perspective :

- Columns with an `ElasticSecondaryIndex` are indexed in Elasticsearch.
- By default, Elasticsearch document fields are multivalued, so every field is backed by a list. Single valued document field can be mapped to a basic types by setting `'cql_collection: singleton'` in our type mapping. See [Elasticsearch document mapping](#) for details.
- Nested documents are stored using cassandra [User Defined Type](#) or `map`.
- Elasticsearch provides a JSON-REST API to cassandra, see [Elasticsearch API](#).

1.2 Durability

All writes to a cassandra node are recorded both in a memory table and in a commit log. When a memtable flush occurs, it flushes the elasticsearch secondary index on disk. When restarting after a failure, cassandra replays commitlogs and re-indexes elasticsearch documents that were no flushed by elasticsearch. This the reason why [elasticsearch translog](#) is disabled in elassandra.

1.3 Shards and Replica

Unlike Elasticsearch, sharding depends on the number of nodes in the datacenter, and number of replica is defined by your keyspace [Replication Factor](#) . Elasticsearch *numberOfShards* is just an information about number of nodes.

- When adding a new elasticassandra node, the cassandra bootstrap process gets some token ranges from the existing ring and pull the corresponding data. Pulled data are automatically indexed and each node update its routing table to distribute search requests according to the ring topology.
- When updating the Replication Factor, you will need to run a `nodetool repair <keyspace>` on the new node to effectively copy and index the data.
- If a node become unavailable, the routing table is updated on all nodes in order to route search requests on available nodes. The actual default strategy routes search requests on primary token ranges' owner first, then to replica nodes if available. If some token ranges become unreachable, the cluster status is red, otherwise cluster status is yellow.

After starting a new Elassandra node, data and elasticsearch indices are distributed on 2 nodes (with no replication).

```
nodetool status twitter
Datacenter: DC1
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address      Load          Tokens   Owns (effective)  Host ID
--  --
UN  127.0.0.1     156,9 KB      2        70,3%             74ae1629-0149-4e65-b790-
    ↪cd25c7406675  RAC1
UN  127.0.0.2     129,01 KB     2        29,7%             e5df0651-8608-4590-92e1-
    ↪4e523e4582b9  RAC2
```

The routing table now distributes search request on 2 elasticassandra nodes covering 100% of the ring.

```
curl -XGET 'http://localhost:9200/_cluster/state/?pretty=true'
{
  "cluster_name" : "Test Cluster",
  "version" : 12,
  "master_node" : "74ae1629-0149-4e65-b790-cd25c7406675",
  "blocks" : { },
  "nodes" : {
    "74ae1629-0149-4e65-b790-cd25c7406675" : {
      "name" : "localhost",
      "status" : "ALIVE",
      "transport_address" : "inet[localhost/127.0.0.1:9300]",
      "attributes" : {
        "data" : "true",
        "rack" : "RAC1",
        "data_center" : "DC1",
        "master" : "true"
      }
    },
    "e5df0651-8608-4590-92e1-4e523e4582b9" : {
      "name" : "127.0.0.2",
      "status" : "ALIVE",
      "transport_address" : "inet[127.0.0.2/127.0.0.2:9300]",
      "attributes" : {
        "data" : "true",
        "rack" : "RAC2",
        "data_center" : "DC1",
```

```
        "master" : "true"
      }
    },
    "metadata" : {
      "version" : 1,
      "uuid" : "e5df0651-8608-4590-92e1-4e523e4582b9",
      "templates" : { },
      "indices" : {
        "twitter" : {
          "state" : "open",
          "settings" : {
            "index" : {
              "creation_date" : "1440659762584",
              "uuid" : "fyqNMDfnRgeRE9KgTqxFWw",
              "number_of_replicas" : "1",
              "number_of_shards" : "1",
              "version" : {
                "created" : "1050299"
              }
            }
          }
        },
        "mappings" : {
          "user" : {
            "properties" : {
              "name" : {
                "type" : "string"
              }
            }
          }
        },
        "tweet" : {
          "properties" : {
            "message" : {
              "type" : "string"
            },
            "postDate" : {
              "format" : "dateOptionalTime",
              "type" : "date"
            },
            "user" : {
              "type" : "string"
            },
            "_token" : {
              "type" : "long"
            }
          }
        }
      },
      "aliases" : [ ]
    },
    "routing_table" : {
      "indices" : {
        "twitter" : {
          "shards" : {
            "0" : [ {
              "state" : "STARTED",
```

```

        "primary" : true,
        "node" : "74ae1629-0149-4e65-b790-cd25c7406675",
        "token_ranges" : [ "(-8879901672822909480,4094576844402756550)" ],
        "shard" : 0,
        "index" : "twitter"
    } ],
    "1" : [ {
        "state" : "STARTED",
        "primary" : true,
        "node" : "e5df0651-8608-4590-92e1-4e523e4582b9",
        "token_ranges" : [ "(-9223372036854775808,-8879901672822909480)",
↪ "(4094576844402756550,9223372036854775807)" ],
        "shard" : 1,
        "index" : "twitter"
    } ]
    }
}
},
"routing_nodes" : {
    "unassigned" : [ ],
    "nodes" : {
        "e5df0651-8608-4590-92e1-4e523e4582b9" : [ {
            "state" : "STARTED",
            "primary" : true,
            "node" : "e5df0651-8608-4590-92e1-4e523e4582b9",
            "token_ranges" : [ "(-9223372036854775808,-8879901672822909480)",
↪ "(4094576844402756550,9223372036854775807)" ],
            "shard" : 1,
            "index" : "twitter"
        } ],
        "74ae1629-0149-4e65-b790-cd25c7406675" : [ {
            "state" : "STARTED",
            "primary" : true,
            "node" : "74ae1629-0149-4e65-b790-cd25c7406675",
            "token_ranges" : [ "(-8879901672822909480,4094576844402756550)" ],
            "shard" : 0,
            "index" : "twitter"
        } ]
    }
}
},
"allocations" : [ ]
}

```

Internally, each node broadcasts its local shard status in the gossip application state X1 ("twitter":STARTED) and its current metadata UUID/version in application state X2.

```

nodetool gossipinfo
127.0.0.2/127.0.0.2
generation:1440659838
heartbeat:396197
DC:DC1
NET_VERSION:8
SEVERITY:-1.3877787807814457E-17
X1:{"twitter":3}
X2:e5df0651-8608-4590-92e1-4e523e4582b9/1
RELEASE_VERSION:2.1.8
RACK:RAC2

```

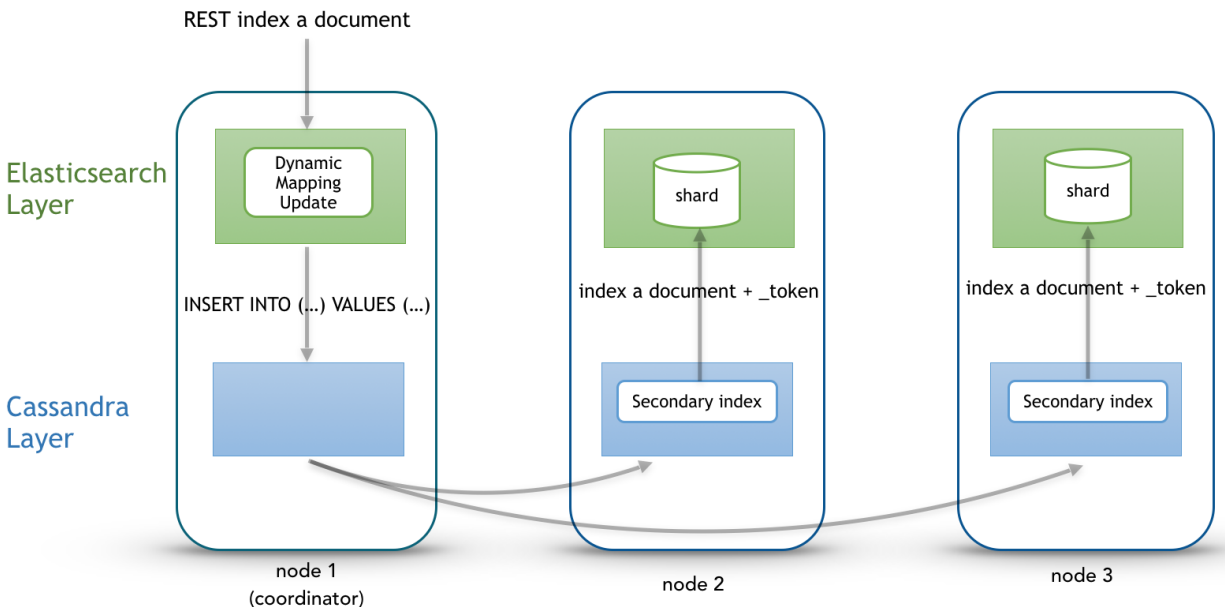
```

STATUS:NORMAL,-8879901672822909480
SCHEMA:ce6febf4-571d-30d2-afeb-b8db9d578fd1
INTERNAL_IP:127.0.0.2
RPC_ADDRESS:127.0.0.2
LOAD:131314.0
HOST_ID:e5df0651-8608-4590-92e1-4e523e4582b9
localhost/127.0.0.1
generation:1440659739
heartbeat:396550
DC:DC1
NET_VERSION:8
SEVERITY:2.220446049250313E-16
X1:{"twitter":3}
X2:e5df0651-8608-4590-92e1-4e523e4582b9/1
RELEASE_VERSION:2.1.8
RACK:RAC1
STATUS:NORMAL,-4318747828927358946
SCHEMA:ce6febf4-571d-30d2-afeb-b8db9d578fd1
RPC_ADDRESS:127.0.0.1
INTERNAL_IP:127.0.0.1
LOAD:154824.0
HOST_ID:74ae1629-0149-4e65-b790-cd25c7406675

```

1.4 Write path

Write operations (Elasticsearch index, update, delete and bulk operations) are converted to CQL write requests managed by the coordinator node. The elasticsearch document `_id` is converted to the underlying primary key, and the corresponding row is stored on many nodes according to the Cassandra replication factor. Then, on each node hosting this row, an Elasticsearch document is indexed through a Cassandra custom secondary index. Every document includes a `_token` fields used when searching.

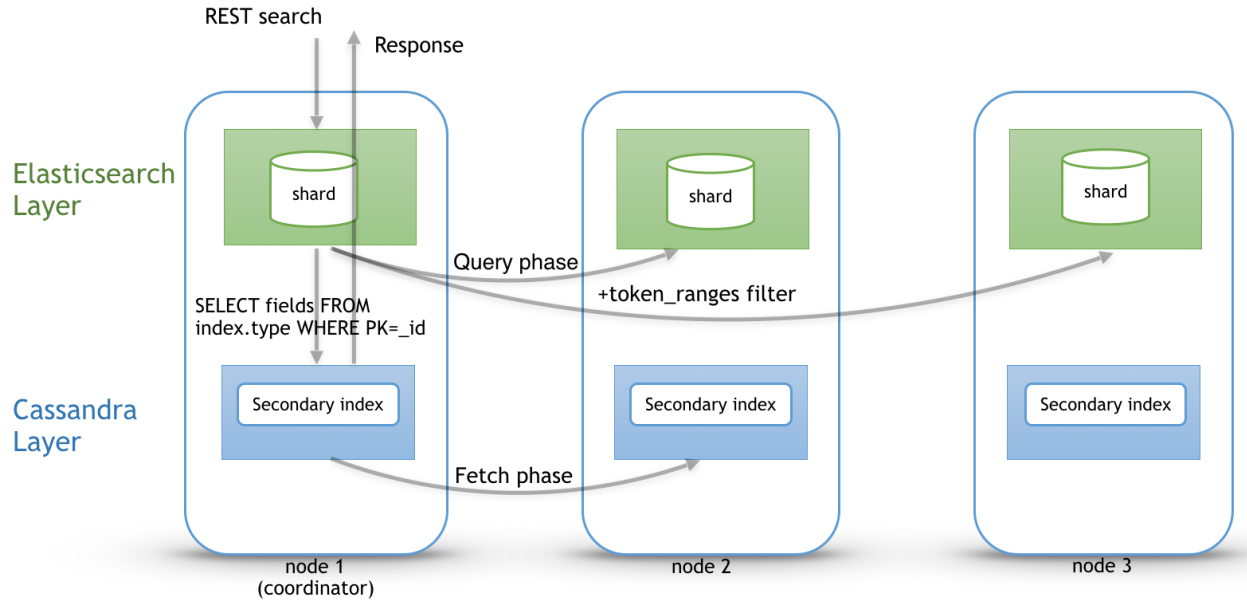


At index time, every nodes directly generates lucene fields without any JSON parsing overhead, and Lucene files does not contains any version number, because version-based concurrency management becomes meaningless in a

multi-master database like Cassandra.

1.5 Search path

Search request is done in two phases. In the query phase, the coordinator node add a `token_ranges` filter to the query and broadcasts a search request to all nodes. This `token_ranges` filter covers all the Cassandra ring and avoid duplicate results. Then, in the fetch phases, the coordinator fetches the required fields by issuing a CQL request in the underlying Cassandra table, and builds the final JSON response.



Elassandra provides a random search strategy requesting the minimum of nodes to cover the whole Cassandra ring. For example, if you have a datacenter with four nodes and a replication factor of two, it will request only two nodes with simplified `token_ranges` filters (adjacent token ranges are automatically merged).

Additionally, as these `token_ranges` filters only change when the datacenter topology change (for example when a node is down or when adding a new node), Elassandra introduces a `token_range` bitset cache for each lucene segment. With this cache, out of range documents are seen as deleted documents at the lucene segment layer for subsequent queries using the same `token_range` filter. This drastically improves search performances.

Finally, the CQL fetch overhead can be mitigated by using keys and rows Cassandra caching, eventually using the off-heap caching features of Cassandra.

CHAPTER 2

Installation

There are a number of ways to install Elassandra: from the *tarball*, with the *deb* or *rpm* package, with a *docker image*, or even from *source*.

Elassandra is based on Cassandra and Elasticsearch, thus it will be easier if you're already familiar with one on these technologies.

2.1 Tarball

Elassandra requires at least Java 8. Oracle JDK is the recommended version, but OpenJDK should work as well. You can check which version is installed on your computer:

```
$ java -version
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)
```

Once java is correctly installed, download the Elassandra tarball:

```
wget https://github.com/strapdata/elassandra/releases/download/v5.5.0.4/
elassandra-5.5.0.4.tar.gz
```

Then extract its content:

```
tar -xzf elassandra-5.5.0.4.tar.gz
```

Go to the extracted directory:

```
cd elassandra-5.5.0.4
```

Configure `conf/cassandra.yaml` if necessary, then run:

```
bin/cassandra -e
```

This has started cassandra with elasticsearch enabled (according to the `-e` option).

Get the node status:

```
bin/nodetool status
```

Now connect to the node with cqlsh:

```
bin/cqlsh
```

You're now able to type CQL commands. See the [CQL reference](#).

Check the elasticsearch API:

```
curl -X GET http://localhost:9200/
```

You should get something like:

```
{
  "name" : "127.0.0.1",
  "cluster_name" : "Test Cluster",
  "cluster_uuid" : "7cb65cea-09c1-4d6a-a17a-24efb9eb7d2b",
  "version" : {
    "number" : "5.5.0",
    "build_hash" : "b0b4cb025cb8aa74538124a30a00b137419983a3",
    "build_timestamp" : "2017-04-19T13:11:11Z",
    "build_snapshot" : true,
    "lucene_version" : "5.5.2"
  },
  "tagline" : "You Know, for Search"
}
```

You're done !

On a production environment, it's better to modify some system settings like disabling swap. This [guide](#) shows you how to. On linux, consider installing [jemalloc](#).

2.2 deb

Download the last deb package from [github relases page](#)

```
wget -O elassandra.deb https://github.com/strapdata/elassandra/releases/download/v5.5.0.4/elassandra-5.5.0.4.deb
```

Install it with dpkg tool:

```
sudo dpkg -i elassandra.deb
```

To start elassandra, just run:

```
sudo systemctl start cassandra
```

or:

```
sudo service cassandra start
```

Files locations:

- /usr/bin: startup script, cqlsh, nodetool, elasticsearch-plugin

- `/etc/cassandra` and `/etc/default/cassandra`: configurations
- `/var/lib/cassandra`: data
- `/var/log/cassandra`: logs
- `/usr/share/cassandra`: plugins, modules, libs, ...
- `/usr/lib/python2.7/dist-packages/cqlshlib/`: python library for cqlsh

2.3 rpm

Download the last rpm package from [github relases page](#)

```
wget -O elassandra.rpm https://github.com/strapdata/elassandra/releases/download/v5.5.0.4/elassandra-5.5.0.4.rpm
```

Install it with yum:

```
sudo yum install elassandra.rpm
```

To start elassandra, just run:

```
sudo systemctl start cassandra
```

or:

```
sudo service cassandra start
```

Files locations:

- `/usr/bin`: startup script, cqlsh, nodetool, elasticsearch-plugin
- `/etc/cassandra` and `/etc/sysconfig/cassandra`: configurations
- `/var/lib/cassandra`: data
- `/var/log/cassandra`: logs
- `/usr/share/cassandra`: plugins, modules, libs, ...
- `/usr/lib/python2.7/site-packages/cqlshlib/`: python library for cqlsh

2.4 Docker image

We provide an [image on docker hub](#):

```
docker pull strapdata/elassandra
```

This image is based on the [official Cassandra image](#) whose the [documentation](#) is valid as well for Elassandra.

The source code is on github at [strapdata/docker-elassandra](#).

2.4.1 Start an elassandra server instance

Starting an Elassandra instance is simple:

```
docker run --name some-elasticassandra -d strapdata/elasticassandra:tag
```

...where `some-elasticassandra` is the name you want to assign to your container and `tag` is the tag specifying the Elassandra version you want. Default is `latest`.

Run `nodetool` and `cqlsh`:

```
docker exec -it some-elasticassandra nodetool status
docker exec -it some-elasticassandra cqlsh
```

2.4.2 Connect to Cassandra from an application in another Docker container

This image exposes the standard Cassandra and ElasticSearch ports, so container linking makes the Elassandra instance available to other application containers. Start your application container like this in order to link it to the Elassandra container:

```
docker run --name some-app --link some-elasticassandra:elasticassandra -d app-that-uses-
↳elasticassandra
```

For instance, consuming the elasticsearch API from another container can be done like this:

```
docker run --link some-elasticassandra:elasticassandra -it strapdata/elasticassandra curl http://
↳elasticassandra:9200
```

... where `strapdata/elasticassandra` could be any image with `curl` installed.

2.4.3 Environment Variables

When you start the Elassandra image, you can adjust the configuration of the Elassandra instance by passing one or more environment variables on the `docker run` command line.

Variable Name	Description
CASSAN-DRA_LISTEN_ADDRESS	This variable is for controlling which IP address to listen for incoming connections on. The default is 0.0.0.0, which will set the listen_address option in cassandra.yaml to the IP address of the container as it starts. This default should work in most use cases.
CASSAN-DRA_BROADCAST_ADDRESS	This variable is for controlling which IP address to advertise to other nodes. The default value is the CASSANDRA_LISTEN_ADDRESS. It will set the broadcast_address and broadcast_rpc_address options in cassandra.yaml.
CASSAN-DRA_RPC_ADDRESS	This variable is for controlling which address to bind the thrift rpc server to. If you do not specify an address, the wildcard address (0.0.0.0) will be used. It will set the rpc_address option in cassandra.yaml.
CASSAN-DRA_START_RPC	This variable is for controlling if the thrift rpc server is started. It will set the start_rpc option in cassandra.yaml. As Elastic search used this port in Elassandra, it will be set ON by default.
CASSAN-DRA_SEEDS	This variable is the comma-separated list of IP addresses used by gossip for bootstrapping new nodes joining a cluster. It will set the seeds value of the seed_provider option in cassandra.yaml. The CASSANDRA_BROADCAST_ADDRESS will be added to the seeds passed in so that the server will talk to itself as well.
CASSAN-DRA_CLUSTER_NAME	This variable sets the name of the cluster and must be the same for all nodes in the cluster. It will set the cluster_name option of cassandra.yaml.
CASSAN-DRA_NUM_TOKENS	This variable sets number of tokens for this node. It will set the num_tokens option of cassandra.yaml.
CASSAN-DRA_DC	This variable sets the datacenter name of this node. It will set the dc option of cassandra-rackdc.properties.
CASSAN-DRA_RACK	This variable sets the rack name of this node. It will set the rack option of cassandra-rackdc.properties.
CASSAN-DRA_ENDPOINT_SNITCH	This variable sets the snitch implementation this node will use. It will set the endpoint_snitch option in cassandra.yaml.
CASSAN-DRA_DAEMON	The Cassandra entry-point class: <code>org.apache.cassandra.service.ElassandraDaemon</code> to start with ElasticSearch enabled (default), <code>org.apache.cassandra.service.ElassandraDaemon</code> otherwise.

2.4.4 Files locations

- `/opt/elassandra-|release|`: elassandra installation
- `/var/lib/cassandra`: data (sstables, lucene segment, commitlogs, ...)
- `var/log/cassandra`: logs files.

`/var/lib/cassandra` is automatically managed as a docker volume. But it's a good target to bind mount from the host filesystem.

2.4.5 Exposed ports

- 7000: intra-node communication
- 7001: TLS intra-node communication
- 7199: JMX
- 9042: CQL
- 9160: thrift service
- 9200: ElasticSearch HTTP

- 9300: ElasticSearch transport

2.4.6 Make a cluster

In case there is only one elassandra instance per docker host, the easiest way is to start the container with `--net=host`.

When using the host network is not an option, you could just map the necessary ports with `-p 9042:9042`, `-p 9200:9200` and so on... but you should be aware that docker default network will considerably slow down performances.

Also, elassandra cluster can be fully managed over a swarm cluster. But this will basically require some more customization. Feel free to open an issue on our github repository to discuss about this.

2.5 Build from source

Requirements:

- Oracle JDK 1.8 or OpenJDK 8
- gradle >= 3.5

Clone Elassandra repository and Cassandra sub-module:

```
git clone --recursive git@github.com:strapdata/elassandra.git
cd elassandra
```

Elassandra 5.5+ uses [Gradle](#) for its build system. Simply run:

```
gradle assemble
```

or to build only the tgz tarball for a release version :

```
gradle assemble distribution:tar:assemble -Dbuild.snapshot=false
```

It's gonna take a while, you might go for a cup of tea. If everything succeed, tarballs will be built in:

```
distribution/ (tar|zip|rpm|deb) /build/distributions/
```

Then follow the instructions for [tarball](#) installation.

3.1 Directory Layout

Elassandra merge the cassandra and elasticsearch directories as follow :

- `conf` : Cassandra configuration directory + elasticsearch.yml default configuration file.
- `bin` : Cassandra scripts + elasticsearch plugin script.
- `lib` : Cassandra and elasticsearch jar dependency
- `pylib` : Cqlsh python library.
- `tools` : Cassandra tools.
- `plugins` : Elasticsearch plugins installation directory.
- `modules` : Elasticsearch modules directory.
- `work` : Elasticsearch working directory.

Elasticsearch paths are set according to the following environnement variables and system properties :

- `path.home` : **CASSANDRA_HOME** environnement variable, `cassandra.home` system property, the current directory.
- `path.conf` : **CASSANDRA_CONF** environnement variable, `path.conf` or `path.home`.
- `path.data` : **cassandra.storagedir**/data/elasticsearch.data, `path.data` system property or `path.home`/data/elasticsearch.data

3.2 Configuration

Elasticsearch configuration rely on cassandra configuration file **conf/cassandra.yaml** for the following parameters.

Cassandra	Elasticsearch	Description
<code>cluster.name</code>	<code>cluster_name</code>	Elasticsearch <code>cluster_name</code> is mapped to the cassandra cluster name.
<code>rpc_address</code>	<code>network.host</code>	Elasticsearch <code>network.host</code> is set to the cassandra <code>rpc_address</code> .
<code>broadcast_rpc_address</code>	<code>network.publish_host</code>	Elasticsearch <code>network.publish_host</code> is set to the cassandra <code>broadcast_rpc_address</code> .
<code>listen_address</code>	<code>transport.host</code>	Elasticsearch <code>transport.host</code> is set to the cassandra <code>listen_address</code> .
<code>broadcast_address</code>	<code>transport.publish_host</code>	Elasticsearch <code>transport.publish_host</code> is set to the cassandra <code>broadcast_address</code> .

Node role (master, primary, data) is automatically set by elassandra, standard configuration should only set **cluster_name**, **rpc_address** in the `conf/cassandra.yaml`.

By default, Elasticsearch HTTP is bound to Cassandra RPC addresses, while Elasticsearch transport protocol is bound to Cassandra internal addresses. You can overload these default settings by defining Elasticsearch network settings in `conf/elasticsearch.yaml` (in order to bind Elasticsearch transport on a public interface if you want to use the Elasticsearch transport client from your application).

Caution: If you use the [GossipPropertyFile](#) Snitch to configure your cassandra datacenter and rack properties in **conf/cassandra-rackdc.properties**, keep in mind this snitch falls back to the `PropertyFileSnitch` when gossip is not enabled. So, when re-starting the first node, dead nodes can appear in the default DC and rack configured in **conf/cassandra-topology.properties**. This also breaks the replica placement strategy and the computation of the Elasticsearch routing tables. So it is strongly recommended to set the same default rack and datacenter in both the **conf/cassandra-topology.properties** and **conf/cassandra-rackdc.properties**.

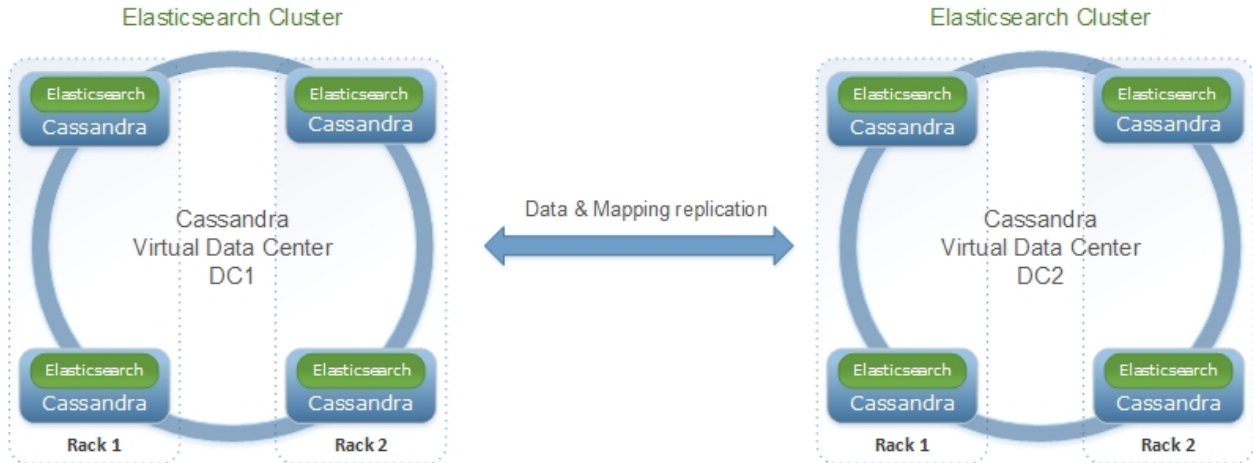
3.3 Logging configuration

The cassandra logs in `logs/system.log` includes elasticsearch logs according to the your `conf/logback.conf` settings. See [cassandra logging configuration](#).

Per keyspace (or per table) logging level can be configured using the logger name `org.elassandra.index.ExtendedElasticSecondaryIndex.<keyspace>.<table>`.

3.4 Multi datacenter configuration

By default, all elassandra datacenters share the same Elasticsearch cluster name and mapping. This mapping is stored in the `elastic_admin` keyspace.



If you want to manage distinct Elasticsearch clusters inside a cassandra cluster (when indexing different tables in different datacenter), you can set a `datacenter.group` in `conf/elasticsearch.yml` and thus, all elassandra datacenters sharing the same datacenter group name will share the same mapping. Those elasticsearch clusters will be named `<cluster_name>@<datacenter.group>` and mapping will be stored in a dedicated keyspace `table elastic_admin_<datacenter.group>.metadata`.

All `elastic_admin[_<datacenter.group>]` keyspaces are configured with **NetworkReplicationStrategy** (see [data replication](#)), where the replication factor is automatically set to the number of nodes in each datacenter. This ensures maximum availability for the elasticsearch metadata. When removing a node from an elassandra datacenter, you should manually decrease the `elastic_admin[_<datacenter.group>]` replication factor to the number of nodes.

When a mapping change occurs, Elassandra updates Elasticsearch metadata in `elastic_admin[_<datacenter.group>].metadata` within a **lightweight transaction** to avoid conflict with concurrent updates. This transaction requires QUORUM available nodes, that is more than half the nodes of one or more datacenters regarding your `datacenter.group` configuration. It also involves cross-datacenter network latency for each mapping update.

Tip: Cassandra cross-datacenter writes are not sent directly to each replica; instead, they are sent to a single replica with a parameter telling that replica to forward to the other replicas in that datacenter; those replicas will respond directly to the original coordinator. This reduces network traffic between datacenters when having many replicas.

3.5 Elassandra Settings

Most of the settings can be set at various levels :

- As a system property, default property is `es.<property_name>`
- At cluster level, default setting is `cluster.default_<property_name>`
- At index level, setting is `index.<property_name>`
- At table level, setting is configured as a `_meta:{ "<property_name>" : <value> }` for a document type.

For example, `drop_on_delete_index` can be :

- set as a system property `es.drop_on_delete_index` for all created indices.
- set at the cluster level with the `cluster.default_drop_on_delete_index` dynamic settings,
- set at the index level with the `index.drop_on_delete_index` dynamic index settings,
- set as the Elasticsearch document type level with `_meta : { "drop_on_delete_index":true }` in the document type mapping.

When a settings is dynamic, it's relevant only for index and cluster setting levels, system and document type setting levels are immutables.

Setting	Update	Levels	Default value	Description
secondary_index_ static	static class	index, cluster	ExtendedElasticSecondaryIndex	org.apache.cassandra.index.Index secondary index implementation class. This class must implements <i>org.apache.cassandra.index.Index</i> interface.
search_strategy_ dynamic	dynamic	index, cluster	PrimaryFirstSearchStrategy	Search strategy class. Available strategy are : <i>PrimaryFirstSearchStrategy</i> distributes search requests to all available nodes <i>RandomSearchStrategy</i> distributes search requests to a subset of available nodes covering the whole cassandra ring. This improves search performance when RF > 1.
partition_function_ static	static class	index, cluster	MessageFormatPartitionFunction	Partition function implementation class. Available implementations are : <i>MessageFormatPartitionFunction</i> based on the java MessageFormat.format() <i>StringPartitionFunction</i> based on the java String.format().
version_less_engine_ static	static	index, cluster, system	true	If true, use the optimized lucene <i>VersionLessEngine</i> (does not more manage any document version)
3.5. Elassandra Settings				21 otherwise, use the standard Elastic-search Engine.
			22	

3.6 Sizing and tuning

Basically, Elassandra requires much CPU than standalone Cassandra or Elasticsearch and Elassandra write throughput should be half the cassandra write throughput if you index all columns. If you only index a subset of columns, write performances would be better.

Design recommendations :

- Increase number of Elassandra node or use partitioned index to keep shards size below 50Gb.
- Avoid huge wide rows, write-lock on a wide row can dramatically affect write performance.
- Choose the right Cassandra compaction strategy to fit your workload (See this [blog](#) post by Justin Cameron)

System recommendations :

- Turn swapping off.
- Configure less than half the total memory of your server and up to 30.5Gb. Minimum recommended DRAM for production deployments is 32Gb. If you are not aggregating on text fields, you can probably use less memory to improve file system cache used by Doc Values (See this [excellent blog](#) post by Chris Earle).
- Set -Xms to the same value as -Xmx.
- Ensure JNA and jemalloc are correctly installed and enabled.

3.6.1 Write performances

- By default, Elasticsearch analyzes the input data of all fields in a special **_all** field. If you don't need it, disable it.
- By default, Elasticsearch all fields names in a special **_field_names** field. If you don't need it, disable it (elasticsearch-hadoop requires **_field_names** to be enabled).
- By default, Elasticsearch shards are refreshed every second, making new document visible for search within a second. If you don't need it, increase the refresh interval to more than a second, or even turn it off temporarily by setting the refresh interval to -1.
- Use the optimized version less Lucene engine (the default) to reduce index size.
- Disable `index_on_compaction` (Default is *false*) to avoid the Lucene segments merge overhead when compacting SSTables.
- Index partitioning may increase write throughput by writing to several Elasticsearch indexes in parallel, but choose an efficient partition function implementation. For example, *String.format()* is much more faster than *Message.format()*.

3.6.2 Search performances

- Use 16 to 64 vnodes per node to reduce the complexity of the token_ranges filter.
- Use the *RandomSearchStrategy* and increase the Cassandra Replication Factor to reduce the number of nodes requires for a search request.
- Enable the `token_ranges_bitset_cache`. This cache compute the token ranges filter once per Lucene segment. Check the token range bitset cache statistics to ensure this caching is efficient.
- Enable Cassandra row caching to reduce the overhead introduced by fetching the requested fields from the underlying Cassandra table.
- Enable Cassandra off-heap row caching in your Cassandra configuration.

- When this is possible, reduce the number of Lucene segments by forcing a merge.

Basically, an Elasticsearch index is mapped to a cassandra keyspace, and a document type to a cassandra table.

4.1 Type mapping

Here is the mapping from Elasticsearch field basic types to CQL3 types :

Elasticsearch Types	CQL Types	Comment
keyword	text	Not analyzed text
text	text	Analyzed text
date	timestamp	
date	date	Existing Cassandra <i>date</i> columns are mapped to an Elasticsearch date. (the number of day since epoch is converted to a timestamp)
integer, short, byte	int	
long	bigint	
long	time	Existing Cassandra <i>time</i> columns are mapped to an Elasticsearch long.
double	double	
float	float	
boolean	boolean	
binary	blob	
ip	inet	Internet address
keyword	uuid	Existing Cassandra <i>uuid</i> columns are mapped to an Elasticsearch keyword.
keyword or date	timeuuid	Existing Cassandra <i>timeuuid</i> columns are mapped to an Elasticsearch keyword by default, or can explicitly be mapped to an Elasticsearch date.
geo_point	UDT geo_point or text	Built-In User Defined Type (1)
geo_shape	text	Requires <i>_source</i> enabled (2)
object, nested	Custom User Defined Type	

1. Geo shapes require *_source* to be enabled to store the original JSON document (default is disabled).
2. Existing Cassandra text columns containing a geohash string can be mapped to an Elasticsearch *geo_point*.

These parameters control the cassandra mapping.

Parameter	Values	Description
cql_collection_type	list , set or singleton	Control how the field of type X is mapped to a column list<X>, set<X> or X. Default is list because Elasticsearch fields are multivalued.
cql_struct_type	udt or map	Control how an object or nested field is mapped to a User Defined Type or to a cassandra map<text,?>. Default is udt .
cql_mandatory	true or false	Elasticsearch index full document. For partial CQL updates, this control which fields should be read to index a full document from a row. Default is true meaning that updates involve reading all missing fields.
cql_primary_key_order	integer	Field position in the cassandra the primary key of the underlying cassandra table. Default is -1 meaning that the field is not part of the cassandra primary key.
cql_partition_key	true or false	When the cql_primary_key_order >= 0, specify if the field is part of the cassandra partition key. Default is false meaning that the field is not part of the cassandra partition key.
cql_udt_name	table_name or table_name.field_name	Specify the Cassandra User Defined Type name to use to store an object. By default, table_name is used. table_name.field_name automatically build (dots in <i>field_names</i> are replaced by underscores)

For more information about cassandra collection types and compound primary key, see [CQL Collections](#) and [Compound keys](#).

4.2 Bidirectionnal mapping

Elassandra supports the [Elasticsearch Indice API](#) and automatically creates the underlying cassandra keyspaces and tables. For each Elasticsearch document type, a cassandra table is created to reflect the Elasticsearch mapping. However, deleting an index does not remove the underlying keyspace, it just removes cassandra secondary indices associated to mapped columns.

Additionally, with the new put mapping parameter `discover`, Elassandra create or update the Elasticsearch mapping for an existing cassandra table. Columns matching the provided regular expression are mapped as Elasticsearch fields. The following command creates the elasticsearch mapping for all columns starting by 'a' of the cassandra table `my_keyspace.my_table` and set a specific analyzer for column `name`.

```
curl -XPUT "http://localhost:9200/my_keyspace/_mapping/my_table" -d '{
  "my_table" : {
    "discover" : "a.*",
    "properties" : {
      "name" : {
        "type" : "string",
        "index" : "analyzed"
      }
    }
  }
}
```

By default, all text columns are mapped with `"index": "not_analyzed"`.

Tip: When creating the first Elasticsearch index for a given cassandra table, elassandra creates a custom CQL secondary index asynchronously for each mapped field when all shards are started. Cassandra build index on all nodes for all existing data. Subsequent CQL inserts or updates are automatically indexed in Elasticsearch.

If you then add a second or more Elasticsearch indices to an existing indexed table, existing data are not automatically re-indexed because cassandra has already indexed existing data. Instead of re-insert your data in the cassandra table, you may use the following command to force a cassandra index rebuild. It will re-index your cassandra table to all associated elasticsearch indices :

```
nodetool rebuild_index --threads <N> <keyspace_name> <table_name> elastic_<table_name>
↪ _idx
```

- `column_name` is any indexed columns (or elasticsearch top-level document field).
- `rebuild_index` reindexes SSTables from disk, but not from MEMtables. In order to index the very last inserted document, run a **nodetool flush <kespace_name>** before rebuilding your elasticsearch indices.
- When deleting an elasticsearch index, elasticsearch index files are removed form the `data/elasticsearch.data` directory, but cassandra secondary indices remains in the CQL schema until the last associated elasticsearch index is removed. Cassandra is acting as a primary data storage, so keyspace and tables and data are never removed when deleting an elasticsearch index.

4.3 Meta-Fields

Elasticsearch [meta-fields](#) meaning is slightly different in Elassandra :

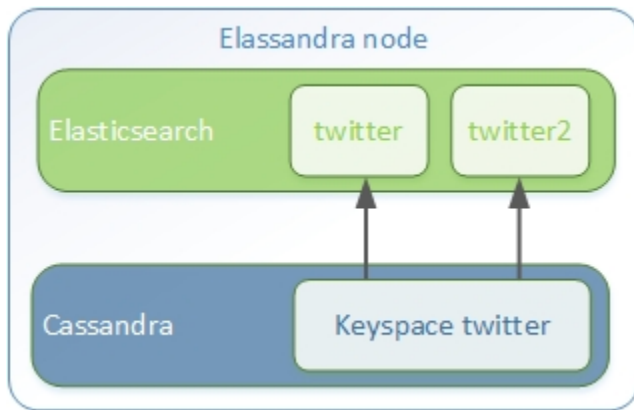
- `_index` is the index name mapped to the underlying cassandra keyspace name (dash [-] and dot[.] are automatically replaced by underscore [_]).

- `_type` is the document type name mapped to the underlying cassandra table name (dash [-] and dot[.] are automatically replaced by underscore [_]).
- `_id` is the document ID is a string representation of the primary key of the underlying cassandra table. Single field primary key is converted to a string, compound primary key is converted to a JSON array.
- `_source` is the indexed JSON document. By default, `_source` is disabled in Elassandra, meaning that `_source` is rebuild from the underlying cassandra columns. If `_source` is enabled (see [Mapping _source field](#)) Elassandra stores documents indexed by with the Elasticsearch API in a dedicated Cassandra text column named `_source`. This allows to retrieve the original JSON document for **‘GeoShape Query’**<https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-geo-shape-query.html> **‘_’**.
- `_routing` is valued with a string representation of the partition key of the underlying cassandra table. Single partition key is converted to a string, compound partition key is converted to a JSON array. Specifying `_routing` on get, index or delete operations is useless, since the partition key is included in `_id`. On search operations, Elassandra compute the cassandra token associated to `_routing` for the search type, and reduce the search only to a cassandra node hosting this token. (WARNING: Without any search types, Elassandra cannot compute the cassandra token and returns an error **all shards failed**).
- `_ttl` and `_timestamp` are mapped to the cassandra `TTL` and `WRITETIME`. The returned `_ttl` and `_timestamp` for a document will be the one of a regular cassandra columns if there is one in the underlying table. Moreover, when indexing a document through the Elasticsearch API, all cassandra cells carry the same `WRITETIME` and `TTL`, but this could be different when upserting some cells using CQL.
- `_parent` is string representation of the parent document primary key. If the parent document primary key is composite, this is string representation of columns defined by `cql_parent_pk` in the mapping. See [Parent-Child Relationship](#).
- `_token` is a meta-field introduced by Elassandra, valued with `token(<partition_key>)`.
- `_node` is an optional meta-field introduced by Elassandra, valued with the cassandra host id, allowing to check the datacenter consistency.

4.4 Mapping change with zero downtime

You can map several Elasticsearch indices with different mapping to the same cassandra keyspace. By default, an index is mapped to a keyspace with the same name, but you can specify a target `keyspace` in your index settings.

For example, you can create a new index **twitter2** mapped to the cassandra keyspace **twitter** and set a mapping for type **tweet** associated to the existing cassandra table **twitter.tweet**.



```
curl -XPUT "http://localhost:9200/twitter2/" -d '{
  "settings" : { "keyspace" : "twitter" } },
  "mappings" : {
    "tweet" : {
      "properties" : {
        "message" : { "type" : "string", "index" : "not_analyzed" },
        "post_date" : { "type" : "date", "format": "yyyy-MM-dd" },
        "user" : { "type" : "string", "index" : "not_analyzed" },
        "size" : { "type" : "long" }
      }
    }
  }
}
```

You can set a specific mapping for **twitter2** and re-index existing data on each cassandra node with the following command (indices are named **elastic_<tablename>_idx**).

```
nodetool rebuild_index [--threads <N>] twitter tweet elastic_tweet_idx
```

By default, **rebuild_index** use only one thread, but Elassandra supports multi-threaded index rebuild with the new parameter **-threads**. Index name is <elastic>_<table_name>_idx where *column_name* is any indexed column name. Once your **twitter2** index is ready, set an alias **twitter** for **twitter2** to switch from the old mapping to the new one, and delete the old **twitter** index.

```
curl -XPOST "http://localhost:9200/_aliases" -d '{ "actions" : [ { "add" : { "index" :
↪: "twitter2", "alias" : "twitter" } } ] }'
curl -XDELETE "http://localhost:9200/twitter"
```

4.5 Partitioned Index

Elasticsearch TTL support is deprecated since Elasticsearch 2.0 and the Elasticsearch TTLSERVICE is disabled in Elassandra. Rather than periodically looking for expired documents, Elassandra supports partitioned index allowing to manage per time-frame indices. Thus, old data can be removed by simply deleting old indices.

Partitioned index also allows to index more than 2^{31} documents on a node (2^{31} is the lucene max documents per index).

An index partition function acts as a selector when many indices are associated to a cassandra table. A partition function is defined by 3 or more fields separated by a space character :

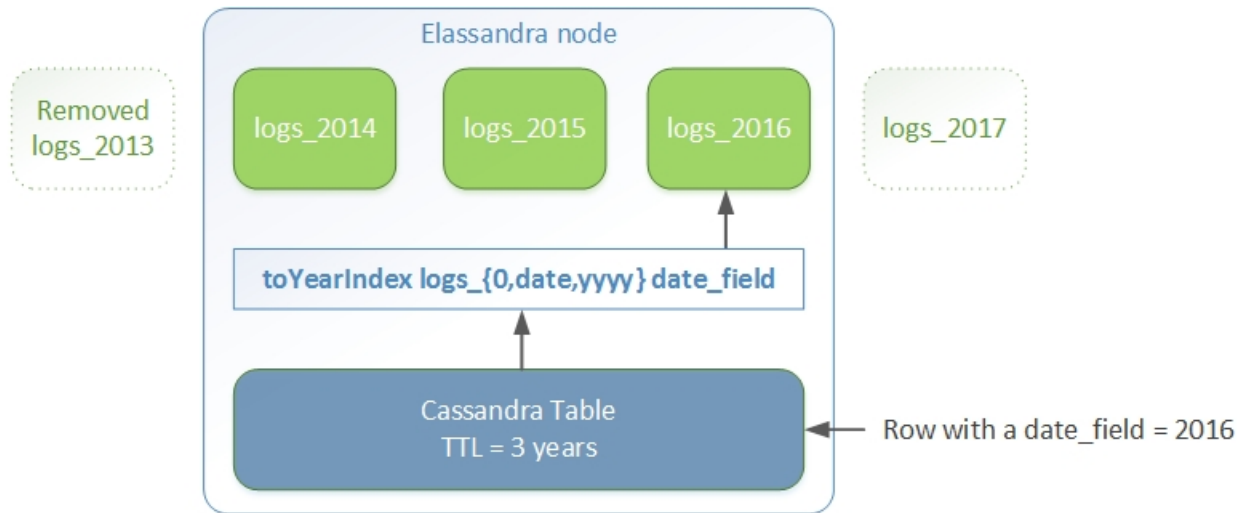
- Function name.
- Index name pattern.
- 1 to N document field names.

The target index name is the result your partition function,

A partition function must implements the java interface **org.lassandra.index.PartitionFunction**. Two implementation classes are provided :

- **StringFormatPartitionFunction** (the default) based on the JDK function `String.format(Locale locale, <parttern>,<arg1>,...)`.
- **MessageFormatPartitionFunction** based on the JDK function `MessageFormat.format(<parttern>,<arg1>,...)`.

Index partition function are stored in a map, so a given index function is executed exactly once for all mapped index. For example, the **toYearIndex** function generates the target index **logs_<year>** depending on the value of the **date_field** for each document (or row).



You can define each per-year index as follow, with the same `index.partition_function` for all **logs_<year>**. All those indices will be mapped to the keyspace **logs**, and all columns of the table **mylog** automatically mapped to the document type **mylog**.

```
curl -XPUT "http://localhost:9200/logs_2016" -d '{
  "settings": {
    "keyspace": "logs",
    "index.partition_function": "toYearIndex logs_{0,date,yyyy} date_field",
    "index.partition_function_class": "MessageFormatPartitionFunction"
  },
  "mappings": {
    "mylog" : { "discover" : ".*" }
  }
}'
```

Tip: Partition function is executed for each indexed document, so if write throughput is a concern, you should choose an efficient implementation class.

To remove an old index.

```
curl -XDELETE "http://localhost:9200/logs_2013"
```

Cassandra **TTL** can be used in conjunction with partitioned index to automatically removed rows during the normal cassandra compaction and repair processes when `index_on_compaction` is *true*, but this introduce a lucene merge

overhead because document are re-indexed when compacting. You can also use the [DateTieredCompactionStrategy](#) to the [TimeWindowTieredCompactionStrategy](#) to improve performance of time series-like workloads.

4.6 Object and Nested mapping

By default, Elasticsearch [Object](#) or [nested](#) types are mapped to dynamically created Cassandra [User Defined Types](#).

```
curl -XPUT 'http://localhost:9200/twitter/tweet/1' -d '{
  "user" : {
    "name" : {
      "first_name" : "Vincent",
      "last_name" : "Royer"
    },
    "uid" : "12345"
  },
  "message" : "This is a tweet!"
}'

curl -XGET 'http://localhost:9200/twitter/tweet/1/_source'
{"message":"This is a tweet!","user":{"uid":["12345"],"name":[{"first_name":["Vincent
↪"],"last_name":["Royer"]}]]}}
```

The resulting cassandra user defined types and table.

```
cqlsh>describe keyspace twitter;
CREATE TYPE twitter.tweet_user (
  name frozen<list<frozen<tweet_user_name>>>,
  uid frozen<list<text>>
);

CREATE TYPE twitter.tweet_user_name (
  last_name frozen<list<text>>,
  first_name frozen<list<text>>
);

CREATE TABLE twitter.tweet (
  "_id" text PRIMARY KEY,
  message list<text>,
  person list<frozen<tweet_person>>
)

cqlsh> select * from twitter.tweet;
_id | message | user
-----+-----+-----
↪-----
1 | ['This is a tweet!'] | [{name: [{last_name: ['Royer'], first_name: ['Vincent']}],
↪uid: ['12345']}]
```

4.7 Dynamic mapping of Cassandra Map

Nested document can be mapped to [User Defined Type](#) or to CQL [map](#) having a *text* key. In the following example, the cassandra map is automatically mapped with `cql_mandatory:true`, so a partial CQL update cause a read of the whole map to re-index a document in the elasticsearch index.

```
cqlsh>CREATE KEYSPACE IF NOT EXISTS twitter WITH replication={ 'class':
↪ 'NetworkTopologyStrategy', 'dc1':'1' };
cqlsh>CREATE TABLE twitter.user (
    name text,
    attrs map<text,text>,
    PRIMARY KEY (name)
);
cqlsh>INSERT INTO twitter.user (name,attrs) VALUES ('bob',{'email':'bob@gmail.com',
↪ 'firstname':'bob'});
```

Create the type mapping from the cassandra table and search for the *bob* entry.

```
curl -XPUT "http://localhost:9200/twitter/_mapping/user" -d '{ "user" : { "discover"↪
↪: ".*" } }'
{"acknowledged":true}

curl -XGET 'http://localhost:9200/twitter/_mapping/user?pretty=true'
{
  "twitter" : {
    "mappings" : {
      "user" : {
        "properties" : {
          "attrs" : {
            "type" : "nested",
            "cql_struct" : "map",
            "cql_collection" : "singleton",
            "properties" : {
              "email" : {
                "type" : "string"
              },
              "firstname" : {
                "type" : "string"
              }
            }
          },
          "name" : {
            "type" : "string",
            "cql_collection" : "singleton",
            "cql_partition_key" : true,
            "cql_primary_key_order" : 0
          }
        }
      }
    }
  }
}

curl -XGET "http://localhost:9200/twitter/user/bob?pretty=true"
{
  "_index" : "twitter",
  "_type" : "user",
  "_id" : "bob",
  "_version" : 0,
  "found" : true,
  "_source":{"name":"bob","attrs":{"email":"bob@gmail.com","firstname":"bob"}}
```

Now insert a new entry in the attrs map column and search for a nested field *attrs.city:paris*.

```

cqlsh>UPDATE twitter.user SET attrs = attrs + { 'city':'paris' } WHERE name = 'bob';

curl -XGET "http://localhost:9200/twitter/_search?pretty=true" -d '{
  "query":{
    "nested":{
      "path":"attrs",
      "query":{"match":{"attrs.city":"paris" } }
    }
  }
}'
{
  "took" : 3,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 2.3862944,
    "hits" : [ {
      "_index" : "twitter",
      "_type" : "user",
      "_id" : "bob",
      "_score" : 2.3862944,
      "_source":{"attrs":{"city":"paris","email":"bob@gmail.com","firstname":"bob"},
↪ "name":"bob"}
    } ]
  }
}

```

4.7.1 Dynamic Template with Dynamic Mapping

Dynamic templates can be used when creating a dynamic field from a Cassandra map.

```

"mappings" : {
  "event_test" : {
    "dynamic_templates": [
      { "strings_template": {
        "match": "strings.*",
        "mapping": {
          "type": "string",
          "index": "not_analyzed"
        }
      }
    ]
  },
  "properties" : {
    "id" : {
      "type" : "string",
      "index" : "not_analyzed",
      "cql_collection" : "singleton",
      "cql_partition_key" : true,
      "cql_primary_key_order" : 0
    },
    "strings" : {

```

```
    "type" : "object",
    "cql_struct" : "map",
    "cql_collection" : "singleton"
  }
}
```

Then, a new entry *key1* in the underlying cassandra map will have the following mapping:

```
"mappings" : {
  "event_test" : {
    "dynamic_templates" : [ {
      "strings_template" : {
        "mapping" : {
          "index" : "not_analyzed",
          "type" : "string",
          "doc_values" : true
        },
        "match" : "strings.*"
      }
    ] ,
    "properties" : {
      "strings" : {
        "cql_struct" : "map",
        "cql_collection" : "singleton",
        "type" : "nested",
        "properties" : {
          "key1" : {
            "index" : "not_analyzed",
            "type" : "string"
          }
        }
      },
      "id" : {
        "index" : "not_analyzed",
        "type" : "string",
        "cql_partition_key" : true,
        "cql_primary_key_order" : 0,
        "cql_collection" : "singleton"
      }
    }
  }
}
```

Note that because `doc_values` is true by default for a not analyzed field, it does not appear in the mapping.

4.8 Parent-Child Relationship

Elassandra supports [parent-child relationship](#) when parent and child document are located on the same cassandra node. This condition is met :

- when running a single node cluster,
- when the keyspace replication factor equals the number of nodes or
- when the parent and child documents share the same cassandra partition key, as shown in the following example.

Create an index company (a cassandra keyspace), a cassandra table, insert 2 rows and map this table as document type *employee*.

```
cqlsh <<EOF
CREATE KEYSPACE IF NOT EXISTS company WITH replication={ 'class':
↪'NetworkTopologyStrategy', 'dc1':'1' };
CREATE TABLE company.employee (
  "_parent" text,
  "_id" text,
  name text,
  dob timestamp,
  hobby text,
  primary key ((_parent), "_id")
);
INSERT INTO company.employee ("_parent", "_id", name, dob, hobby) VALUES ('london', '1',
↪'Alice Smith', '1970-10-24', 'hiking');
INSERT INTO company.employee ("_parent", "_id", name, dob, hobby) VALUES ('london', '2',
↪'Alice Smith', '1990-10-24', 'hiking');
EOF

curl -XPUT "http://$NODE:9200/company2" -d '{
  "mappings" : {
    "employee" : {
      "discover" : ".*",
      "_parent" : { "type": "branch", "cql_parent_pk": "branch" }
    }
  }
}'

curl -XPOST "http://127.0.0.1:9200/company/branch/_bulk" -d '
{ "index": { "_id": "london" }}
{ "district": "London Westminster", "city": "London", "country": "UK" }
{ "index": { "_id": "liverpool" }}
{ "district": "Liverpool Central", "city": "Liverpool", "country": "UK" }
{ "index": { "_id": "paris" }}
{ "district": "Champs Élysées", "city": "Paris", "country": "France" }
'
```

Search for documents having children document of type *employee* with *dob* date greater than 1980.

```
curl -XGET "http://$NODE:9200/company2/branch/_search?pretty=true" -d '{
  "query": {
    "has_child": {
      "type": "employee",
      "query": {
        "range": {
          "dob": {
            "gte": "1980-01-01"
          }
        }
      }
    }
  }
}'
```

Search for employee documents having a parent document where *country* match UK.

```
curl -XGET "http://$NODE:9200/company2/employee/_search?pretty=true" -d '{
  "query": {
    "has_parent": {
```

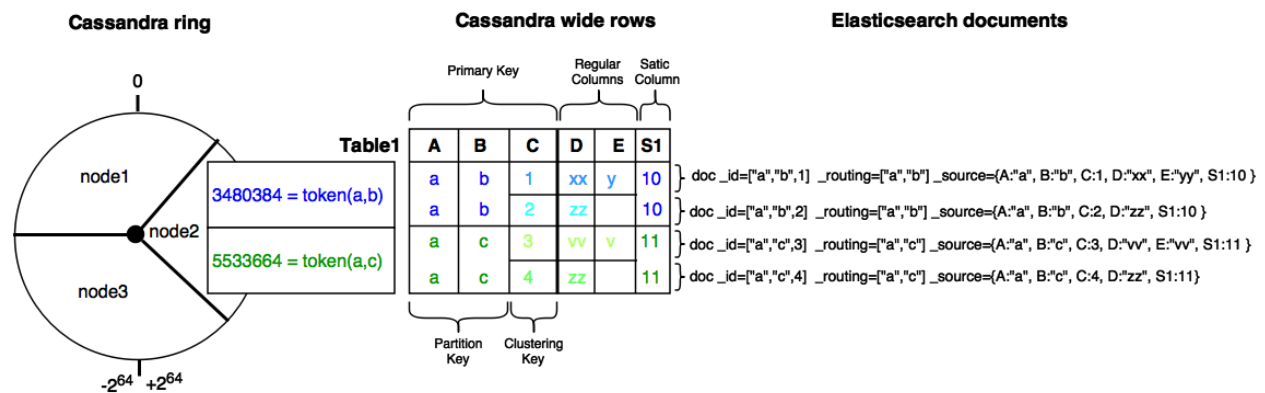
```

    "parent_type": "branch",
    "query": {
      "match": { "country": "UK"
    }
  }
}
}
}

```

4.9 Indexing Cassandra static columns

When a Cassandra table have one or more clustering columns, a **static columns** is shared by all the rows with the same partition key.



Each time a static columns is modified, a document containing the partition key and only static columns is indexed in Elasticsearch. By default, static columns are not indexed with every **wide rows** because any update on a static column would require reindexation of all wide rows. However, you can request for fields backed by a static columns on any get/search request.

The following example demonstrates how to use static columns to store meta information of a timeserie.

```

curl -XPUT "http://localhost:9200/test" -d '{
  "mappings" : {
    "timeseries" : {
      "properties" : {
        "t" : {
          "type" : "date",
          "format" : "strict_date_optional_time||epoch_millis",
          "cql_primary_key_order" : 1,
          "cql_collection" : "singleton"
        },
        "meta" : {
          "type" : "nested",
          "cql_struct" : "map",
          "cql_static_column" : true,
          "cql_collection" : "singleton",
          "include_in_parent" : true,
          "index_static_document": true,
          "index_static_columns": true,
          "properties" : {
            "region" : {

```

```

        "type" : "string"
    }
}
},
"v" : {
    "type" : "double",
    "cql_collection" : "singleton"
},
"m" : {
    "type" : "string",
    "cql_partition_key" : true,
    "cql_primary_key_order" : 0,
    "cql_collection" : "singleton"
}
}
}
}
}'

cqlsh <<EOF
INSERT INTO test.timeseries (m, t, v) VALUES ('server1-cpu', '2016-04-10 13:30', 10);
INSERT INTO test.timeseries (m, t, v) VALUES ('server1-cpu', '2016-04-10 13:31', 20);
INSERT INTO test.timeseries (m, t, v) VALUES ('server1-cpu', '2016-04-10 13:32', 15);
INSERT INTO test.timeseries (m, meta) VALUES ('server1-cpu', { 'region':'west' });
SELECT * FROM test.timeseries;
EOF

```

m	t	meta	v
server1-cpu	2016-04-10 11:30:00.000000z	{'region': 'west'}	10
server1-cpu	2016-04-10 11:31:00.000000z	{'region': 'west'}	20
server1-cpu	2016-04-10 11:32:00.000000z	{'region': 'west'}	15

Search for wide rows only where v=10 and fetch the meta.region field.

```
curl -XGET "http://localhost:9200/test/timeseries/_search?pretty=true&q=v:10&fields=m,
↪t,v,meta.region,_source"

"hits" : [ {
  "_index" : "test",
  "_type" : "timeseries",
  "_id" : "[\"server1-cpu\",1460287800000]",
  "_score" : 1.9162908,
  "_routing" : "server1-cpu",
  "_source" : {
    "t" : "2016-04-10T11:30:00.000Z",
    "v" : 10.0,
    "meta" : { "region" : "west" },
    "m" : "server1-cpu"
  },
  "fields" : {
    "meta.region" : [ "west" ],
    "t" : [ "2016-04-10T11:30:00.000Z" ],
    "m" : [ "server1-cpu" ],
    "v" : [ 10.0 ]
  }
} ]
```

Search for rows where `meta.region=west`, returns only a static document (i.e. document containing the partition key and static columns) because `index_static_document` is `true`.

```
curl -XGET "http://localhost:9200/test/timeseries/_search?pretty=true&q=meta.
↪region:west&fields=m,t,v,meta.region"
{"hits": {
  "total": 1,
  "max_score": 1.5108256,
  "hits": [ {
    "_index": "test",
    "_type": "timeseries",
    "_id": "server1-cpu",
    "_score": 1.5108256,
    "_routing": "server1-cpu",
    "fields": {
      "m": [ "server1-cpu" ],
      "meta.region": [ "west" ]
    }
  } ]
}
```

If needed, you can change the default behavior for a specific cassandra table (or elasticsearch document type), by using the following custom metadata :

- `index_static_document` controls whether or not static document (i.e. document containing the partition key and static columns) are indexed (default is `false`).
- `index_static_only` if `true`, it only indexes static documents with partition key as `_id` and static columns as fields.
- `index_static_columns` controls whether or not static columns are included in indexed documents (default is `false`).

Be careful, if `“index_static_document“=*false*` and `“index_static_only“=*true*`, it does not index any document. In our example with the following mapping, static columns are indexed in every documents, allowing to search on.

```
curl -XPUT http://localhost:9200/test/_mapping/timeseries -d '{
  "timeseries": {
    "discover": ".*",
    "_meta": {
      "index_static_document":true,
      "index_static_columns":true
    }
  }
}'
```

4.10 Elassandra as a JSON-REST Gateway

When dynamic mapping is disabled and a mapping type has no indexed field, elassandra nodes can act as a JSON-REST gateway for cassandra to get, set or delete a cassandra row without any indexing overhead. In this case, the mapping may be use to cast types or format date fields, as shown below.

```
CREATE TABLE twitter.tweet (
  "_id" text PRIMARY KEY,
  message list<text>,
  post_date list<timestamp>,
  size list<bigint>,
  user list<text>
```

```
)

curl -XPUT "http://$NODE:9200/twitter/" -d'{
  "settings":{ "index.mapper.dynamic":false },
  "mappings":{
    "tweet":{
      "properties":{
        "size":      { "type":"long", "index":"no" },
        "post_date":{ "type":"date", "index":"no", "format" : "strict_date_
↪optional_time||epoch_millis" }
      }
    }
  }
}'
```

As the result, you can index, get or delete a cassandra row, including any column of your cassandra table.

```
curl -XPUT "http://localhost:9200/twitter/tweet/1?consistency=one" -d '{
  "user" : "vince",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "look at Elassandra !!",
  "size": 50
}'
{"_index":"twitter","_type":"tweet","_id":"1","_version":1,"_shards":{"total":1,
↪"successful":1,"failed":0},"created":true}

$ curl -XGET "http://localhost:9200/twitter/tweet/1?pretty=true&fields=message,user,
↪size,post_date"
{
  "_index" : "twitter",
  "_type" : "tweet",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "fields" : {
    "size" : [ 50 ],
    "post_date" : [ "2009-11-15T14:12:12.000Z" ],
    "message" : [ "look at Elassandra !" ],
    "user" : [ "vince" ]
  }
}

$ curl -XDELETE "http://localhost:9200/twitter/tweet/1?pretty=true"
{
  "found" : true,
  "_index" : "twitter",
  "_type" : "tweet",
  "_id" : "1",
  "_version" : 0,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  }
}
```

4.11 Check Cassandra consistency with elasticsearch

When the `index.include_node = true` (default is false), the `_node` metafield containing the Cassandra host id is included in every indexed document. This allows to distinguish multiple copies of a document when the datacenter replication factor is greater than one. Then a token range aggregation allows to count the number of documents for each token range and for each Cassandra node.

In the following example, we have 1000 accounts documents in a keyspace with RF=2 in a two nodes datacenter, and each token ranges have the same number of document for the two nodes.

```
curl -XGET "http://$NODE:9200/accounts/_search?pretty=true&size=0" -d'{
  "aggs" : {
    "tokens" : {
      "token_range" : {
        "field" : "_token"
      },
      "aggs": {
        "nodes" : {
          "terms" : { "field" : "_node" }
        }
      }
    }
  }
}'
{
  "took" : 23,
  "timed_out" : false,
  "_shards" : {
    "total" : 2,
    "successful" : 2,
    "failed" : 0
  },
  "hits" : {
    "total" : 2000,
    "max_score" : 0.0,
    "hits" : [ ]
  },
  "aggregations" : {
    "tokens" : {
      "buckets" : [ {
        "key" : "(-9223372036854775807,-4215073831085397715]",
        "from" : -9223372036854775807,
        "from_as_string" : "-9223372036854775807",
        "to" : -4215073831085397715,
        "to_as_string" : "-4215073831085397715",
        "doc_count" : 562,
        "nodes" : {
          "doc_count_error_upper_bound" : 0,
          "sum_other_doc_count" : 0,
          "buckets" : [ {
            "key" : "528b78d3-fae9-49ae-969a-96668566f1c3",
            "doc_count" : 281
          }, {
            "key" : "7f0b782e-5b75-409b-85e9-f5f96a75a7dc",
            "doc_count" : 281
          } ]
        }
      } ]
    }
  },
}
```

```

    "key" : "(-4215073831085397714,7919694572960951318]",
    "from" : -4215073831085397714,
    "from_as_string" : "-4215073831085397714",
    "to" : 7919694572960951318,
    "to_as_string" : "7919694572960951318",
    "doc_count" : 1268,
    "nodes" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [ {
        "key" : "528b78d3-fae9-49ae-969a-96668566f1c3",
        "doc_count" : 634
      }, {
        "key" : "7f0b782e-5b75-409b-85e9-f5f96a75a7dc",
        "doc_count" : 634
      } ]
    }
  }, {
    "key" : "(7919694572960951319,9223372036854775807]",
    "from" : 7919694572960951319,
    "from_as_string" : "7919694572960951319",
    "to" : 9223372036854775807,
    "to_as_string" : "9223372036854775807",
    "doc_count" : 170,
    "nodes" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [ {
        "key" : "528b78d3-fae9-49ae-969a-96668566f1c3",
        "doc_count" : 85
      }, {
        "key" : "7f0b782e-5b75-409b-85e9-f5f96a75a7dc",
        "doc_count" : 85
      } ]
    }
  }
]
}
}
}

```

Of course, according to your use case, you should add a filter to your query to ignore write operations occurring during the check.

5.1 Indexing

Let's try and index some *twitter* like information (demo from [Elasticsearch](#)). First, let's create a twitter user, and add some tweets (the *twitter* index will be created automatically, see automatic index and mapping creation in Elasticsearch documentation):

```
curl -XPUT 'http://localhost:9200/twitter/user/kimchy' -d '{ "name" : "Shay Banon" }'
curl -XPUT 'http://localhost:9200/twitter/tweet/1' -d '
{
  "user": "kimchy",
  "postDate": "2009-11-15T13:12:00",
  "message": "Trying out Elasticsearch, so far so good?"
}'
curl -XPUT 'http://localhost:9200/twitter/tweet/2' -d '
{
  "user": "kimchy",
  "postDate": "2009-11-15T14:12:12",
  "message": "Another tweet, will it be indexed?"
}'
```

You now have two rows in the Cassandra **twitter.tweet** table.

```
cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.1.8 | CQL spec 3.2.0 | Native protocol v3]
Use HELP for help.
cqlsh> select * from twitter.tweet;
 _id | message                                     | postDate                                     |
-----+-----+-----+-----+-----+
  2 | ['Another tweet, will it be indexed?'] | ['2009-11-15 15:12:12+0100'] | ['kimchy']
  1 | ['Trying out Elasticsearch, so far so good?'] | ['2009-11-15 14:12:00+0100'] | ['kimchy']
```

```
(2 rows)
```

Apache Cassandra is a column store that only support upsert operation. This means that deleting a cell or a row involves the creation of a tombstone (insert a null) kept until the compaction later removes both the obsolete data and the tombstone (See this blog about [Cassandra tombstone](#)).

By default, when using the Elasticsearch API to replace a document by a new one, Elassandra insert a row corresponding to the new document including null for unset fields. Without these null (cell tombstones), old fields not present in the new document would be kept at the Cassandra level as zombie cells.

Moreover, indexing with `op_type=create` (See [‘Elasticsearch indexing’](#) `<https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-index\_.html#operation-type>`) require a Cassandra PAXOS transaction to check if the document exists in the underlying datacenter. This comes with useless performance cost if you use automatic generated document ID (See [Automatic ID generation](#)), as this ID will be the Cassandra primary key.

Depending on `op_type` and document ID, CQL requests are issued as follow when indexing with the Elasticsearch API :

<code>op_type</code>	Generated ID	Provided ID	Comment
<code>create</code>	<code>INSERT INTO ... VALUES(...)</code>	<code>INSERT INTO ... VALUES(...) IF NOT EXISTS (I)</code>	Index a new document.
<code>index</code>	<code>INSERT INTO ... VALUES(...)</code>	<code>DELETE FROM ... WHERE ... INSERT INTO ... VALUES(...)</code>	Replace a document that may already exists

(1) The *IF NOT EXISTS* comes with the cost of the PAXOS transaction. If you don’t need to check the uniqueness of the provided ID, add parameter `check_unique_id=false`.

5.2 GETting

Now, let’s see if the information was added by GETting it:

```
curl -XGET 'http://localhost:9200/twitter/user/kimchy?pretty=true'
curl -XGET 'http://localhost:9200/twitter/tweet/1?pretty=true'
curl -XGET 'http://localhost:9200/twitter/tweet/2?pretty=true'
```

Elasticsearch state now show reflect the new twitter index. Because we are currently running on one node, the **token_ranges** routing attribute match 100% of the ring from Long.MIN_VALUE to Long.MAX_VALUE.

```
curl -XGET 'http://localhost:9200/_cluster/state/?pretty=true'
{
  "cluster_name" : "Test Cluster",
  "version" : 5,
  "master_node" : "74ae1629-0149-4e65-b790-cd25c7406675",
  "blocks" : { },
  "nodes" : {
    "74ae1629-0149-4e65-b790-cd25c7406675" : {
      "name" : "localhost",
      "status" : "ALIVE",
      "transport_address" : "inet[localhost/127.0.0.1:9300]",
      "attributes" : {
        "data" : "true",
        "rack" : "RAC1",
        "data_center" : "DC1",
```

```

        "master" : "true"
      }
    },
    "metadata" : {
      "version" : 3,
      "uuid" : "74ae1629-0149-4e65-b790-cd25c7406675",
      "templates" : { },
      "indices" : {
        "twitter" : {
          "state" : "open",
          "settings" : {
            "index" : {
              "creation_date" : "1440659762584",
              "uuid" : "fyqNMDfnRgeRE9KgTqxFWw",
              "number_of_replicas" : "1",
              "number_of_shards" : "1",
              "version" : {
                "created" : "1050299"
              }
            }
          }
        },
        "mappings" : {
          "user" : {
            "properties" : {
              "name" : {
                "type" : "string"
              }
            }
          }
        },
        "tweet" : {
          "properties" : {
            "message" : {
              "type" : "string"
            },
            "postDate" : {
              "format" : "dateOptionalTime",
              "type" : "date"
            },
            "user" : {
              "type" : "string"
            }
          }
        }
      },
      "aliases" : [ ]
    }
  },
  "routing_table" : {
    "indices" : {
      "twitter" : {
        "shards" : {
          "0" : [ {
            "state" : "STARTED",
            "primary" : true,
            "node" : "74ae1629-0149-4e65-b790-cd25c7406675",
            "token_ranges" : [ "(-9223372036854775808,9223372036854775807]" ],

```

```
        "shard" : 0,
        "index" : "twitter"
      } ]
    }
  }
},
"routing_nodes" : {
  "unassigned" : [ ],
  "nodes" : {
    "74ae1629-0149-4e65-b790-cd25c7406675" : [ {
      "state" : "STARTED",
      "primary" : true,
      "node" : "74ae1629-0149-4e65-b790-cd25c7406675",
      "token_ranges" : [ "(-9223372036854775808,9223372036854775807]" ],
      "shard" : 0,
      "index" : "twitter"
    } ]
  }
},
"allocations" : [ ]
}
```

5.3 Updates

In Cassandra, an update is an upsert operation (if the row does not exists, it's an insert). As Elasticsearch, Elassandra issue a GET operation before any update. Then, to keep the same semantic as Elasticsearch, update operations are converted to upsert with the ALL consistency level. Thus, later get operations are consistent. (You should consider [CQL UPDATE](#) operation to avoid this performance cost)

Scripted updates, upsert (scripted_upsert and doc_as_upsert) are also supported.

5.4 Searching

Let's find all the tweets that *kimchy* posted:

```
curl -XGET 'http://localhost:9200/twitter/tweet/_search?q=user:kimchy&pretty=true'
```

We can also use the JSON query language Elasticsearch provides instead of a query string:

```
curl -XGET 'http://localhost:9200/twitter/tweet/_search?pretty=true' -d '{
  "query" : {
    "match" : { "user": "kimchy" }
  }
}'
```

To avoid duplicates results when the Cassandra replication factor is greater than one, Elassandra adds a `token_ranges` filter to every queries distributed to all nodes. Because every document contains a `_token` fields computed at index-time, this ensure that a node only retrieves documents for the requested token ranges. The `token_ranges` parameter is a conjunction of Lucene [NumericRangeQuery](#) build from the Elasticsearch routing tables to cover the entire Cassandra ring. .. code:

```
curl -XGET 'http://localhost:9200/twitter/tweet/_search?pretty=true&token_ranges=(0,
↪9223372036854775807)' -d '
{
  "query" : {
    "match" : { "user": "kimchy" }
  }
}'
```

Of course, if the token range filter cover all ranges (Long.MIN_VALUE to Long.MAX_VALUE), Elassandra automatically remove the useless filter.

Finally, you can restrict a query to the coordinator node with *preference=_only_local* parameter, for all token_ranges as shown below :

```
curl -XGET 'http://localhost:9200/twitter/tweet/_search?pretty=true&preference=_only_
↪local&token_ranges=' -d '
{
  "query" : {
    "match" : { "user": "kimchy" }
  }
}'
```

5.4.1 Optimizing search requests

The search strategy

Elassandra supports various search strategies to distribute a search request over the Elasticsearch cluster. A search strategy is configured at index-level with the `index.search_strategy_class` dynamic parameter.

Strategy	Description
<code>org.lassandra.cluster.routing.PrimaryFirstSearchStrategy</code> (Default)	Search on all alive nodes in the datacenter. All alive nodes responds for their primary token ranges, and for replica token ranges when there is some unavailable nodes. This strategy is always used to build the routing table in the cluster state.
<code>org.lassandra.cluster.routing.RandomSearchStrategy</code>	For each query, randomly distribute a search request to a minimum of nodes to reduce the network traffic. For example, if your underlying keypace replication factor is N, a search only involves 1/N of the nodes.

You can create an index with the `RandomSearchStrategy` as shown below (or change it dynamically).

```
curl -XPUT "http://localhost:9200/twitter/" -d '{
  "settings" : {
    "index.search_strategy_class": "RandomSearchStrategy"
  }
}'
```

Tip: When changing a keypace replication factor, you can force an Elasticsearch routing table update by closing and re-opening all associated elasticsearch indices. To troubleshoot search request routing, set the logging level to **DEBUG** for class `org.lassandra.cluster.routing` in the `conf/logback.xml` file.

5.4.2 Caching features

Compared to Elasticsearch, Elassandra adds to each query a token ranges filter and by fetching fields through a CQL request at the Cassandra layer.

Token Ranges Query Cache

Token ranges filter depends on the node or vnodes configuration, are quite stable and shared for all keyspaces having the same replication factor. These filters only change when the datacenter topology changes, for example when a node is temporary down or when a node is added to the datacenter. So, Elassandra use a cache to keep these queries, a conjunction of Lucene [NumericRangeQuery](#) often reused for every search requests.

As a classic caching strategy, the `token_ranges_query_expire` controls the expiration time of useless token ranges filter queries into memory. The default is 5 minutes.

Token Ranges Bitset Cache

When enabled, the token ranges bitset cache keeps into memory the results of the token range filter for each Lucene segment. This in-memory bitset, acting as the liveDocs Lucene thumbstones mechanism, is then reused for subsequent Lucene search queries. For each Lucene segment, this document bitset is updated when the Lucene thumbstones count increase (it's a bitwise AND between the actual Lucene thumbstones and the token range filter result), or removed if the corresponding token ranges query is removed because unused from the token range query cache.

You can enable the token range bitset cache at index level by setting `index.token_ranges_bitset_cache` to *true* (Default is *false*), or configure the its default value for newly created indices at cluster or system levels.

You can also bypass this cache by adding `token_ranges_bitset_cache=false` in your search request :

```
curl -XPUT "http://localhost:9200/twitter/_search?token_ranges_bitset_cache=false&
↪q=*:*"
```

Finally, you can check the in-memory size of the token ranges bitset cache with the Elasticsearch stats API, and clear it when clearing the Elasticsearch `query_cache` :

```
curl -XGET "http://localhost:9200/_stats?pretty=true"
...
"segments" : {
  "count" : 3,
  "memory_in_bytes" : 26711,
  "terms_memory_in_bytes" : 23563,
  "stored_fields_memory_in_bytes" : 1032,
  "term_vectors_memory_in_bytes" : 0,
  "norms_memory_in_bytes" : 384,
  "doc_values_memory_in_bytes" : 1732,
  "index_writer_memory_in_bytes" : 0,
  "index_writer_max_memory_in_bytes" : 421108121,
  "version_map_memory_in_bytes" : 0,
  "fixed_bit_set_memory_in_bytes" : 0,
  "token_ranges_bit_set_memory_in_bytes" : 240
},
...
```

Cassandra Key and Row Cache

To improve CQL fetch requests response time, Cassandra provides key and row caching features configured for each Cassandra table as follow :

```
ALTER TABLE ... WITH caching = {'keys': 'ALL', 'rows_per_partition': '1'};
```

To enable Cassandra row caching, set the `row_cache_size_in_mb` parameter in your `conf/cassandra.yaml`, and set `row_cache_class_name: org.apache.cassandra.cache.OHCPProvider` to use off-heap memory.

Tip: Elasticsearch also provides a Lucene query cache, used for segments having more than 10k documents, and for some frequent queries (queries done more than 5 or 20 times depending of the nature of the query). The shard request cache, can also be enable if the token range bitset cache is disabled.

5.5 Create, delete and rebuild index

In order to create an Elasticsearch index from an existing Cassandra table, you can specify the underlying keyspace. In the following example, all columns but `message` is automatically mapped with the default mapping, and the `message` is explicitly mapped with a custom mapping.

```
curl -XGET 'http://localhost:9200/twitter_index' -d '{
  "settings": { "keyspace": "twitter" }
  "mappings": {
    "tweet" : {
      "discover": "(?!message).*",
      "properties" : {
        "message" : { "type": "string", "index": "analyzed", "cql_collection":
↪ "singleton" }
      }
    }
  }
}
```

Tip: By default, as the standard elasticsearch, index creation only returns a response to the client when all primary shards have been started, or the request times out (default is 30 seconds). To emulate the elasticsearch routing table, shards hosted by dead nodes are primary or not according to the underlying Cassandra replication factor. So, when there are some dead nodes, if the number of dead nodes is lower than the number of replicas in your create index request, index creation succeed immediately with `shards_acknowledged=true` and index status is yellow, otherwise, index creation timeouts, `shards_acknowledged=false` and the index status is red, meaning that search requests will be inconsistent. Finally, the elasticsearch parameter `wait_for_active_shards` is useless in elassandra, because Cassandra ensure write consistency.

Deleting an Elasticsearch index does not remove any Cassandra data, it keeps the underlying Cassandra tables but remove Elasticsearch index files.

```
curl -XDELETE 'http://localhost:9200/twitter_index'
```

To re-index your existing data, for example after a mapping change to index a new column, run a **nodetool rebuild_index** as follow :

```
nodetool rebuild_index [--threads <N>] <keyspace> <table> elastic_<table>_idx
```

Tip: By default, rebuild index runs on a single thread. In order to improve re-indexing performance, Elassandra comes with a multi-threaded rebuild_index implementation. The **–threads** parameter allows to specify the number of threads dedicated to re-index a Cassandra table. Number of indexing threads should be tuned carefully to avoid CPU exhaustion. Moreover, indexing throughput is limited by locking at the lucene level, but this limit can be exceeded by using a partitioned index involving many independent shards.

Re-index existing data rely on the Cassandra compaction manager. You can trigger a [Cassandra compaction](#) when :

- Creating the first Elasticsearch index on a Cassandra table with existing data automatically involves an index rebuild executed by the compaction manager,
- Running a `nodetool rebuild_index` command,
- Running a `nodetool repair` on a keyspace having indexed tables (a repair actually creates new SSTables triggering index build).

If the compaction manager is busy, secondary index rebuild is added as a pending task and executed later on. You can check current running compactions with a **nodetool compactionstats** and check pending compaction tasks with a **nodetool tpstats**.

```
nodetool -h 52.43.156.196 compactionstats
pending tasks: 1
```

	id	compaction type	keyspace	table
→completed	total	unit	progress	
052c70f0-8690-11e6-aa56-674c194215f6	Secondary index build	lastfm	playlist	
→66347424	330228366	bytes	20,09%	

```
Active compaction remaining time : 0h00m00s
```

To stop a compaction task (including a rebuild index task), you can either use a **nodetool stop –compaction-id <uuid>** or use the JMX management operation **stopCompactionById** (on MBean `org.apache.cassandra.db.CompactionManager`).

5.6 Open, close, index

Open and close operations allow to close and open an Elasticsearch index. Even if the Cassandra secondary index remains in the CQL schema while the index is closed, it has no overhead, it's just a dummy function call. Obviously, when several Elasticsearch indices are associated to the same Cassandra table, data are indexed in opened indices, but not in closed ones.

```
curl -XPOST 'localhost:9200/my_index/_close'
curl -XPOST 'localhost:9200/my_index/_open'
```

Warning: Elasticsearch `translog` is disabled in Elassandra, so you might loose some indexed documents when closing an index if `index.flush_on_close` is `false`.

5.7 Flush, refresh index

A refresh makes all index updates performed since the last refresh available for search. By default, refresh is scheduled every second. By design, setting `refresh=true` on a index operation has no effect with Elassandra, because write operations are converted to CQL queries and documents are indexed later by a custom secondary index. So, the per-index refresh interval should be set carefully according to your needs.

```
curl -XPOST 'localhost:9200/my_index/_refresh'
```

A flush basically write a lucene index on disk. Because document `_source` is stored in Cassandra table in elassandra, it make sense to execute a `nodetool flush <keyspace> <table>` to flush both Cassandra Memtables to SSTables and lucene files for all associated Elasticsearch indices. Moreover, remember that a `nodetool snapshot` also involve a flush before creating a snapshot.

```
curl -XPOST 'localhost:9200/my_index/_flush'
```

Tip: Elasticsearch automatically triggers a flush when an index shard is inactive for more than `indices.memory.shard_inactive_time` (default is 5 minutes) or when `Translog` size is greater than `index.translog.flush_threshold_size` (Default is 512Mb). Elassandra implements a dummy Translog to track the size of indexed data and triggers a flush on the same size threshold. Elassandra also triggers an Elasticsearch flush when flushing [Cassandra SSTables](#).

5.8 Managing Elassandra nodes

You can add, remove or replace an Elassandra node by using the same procedure as for Cassandra (see [Adding nodes to an existing cluster](#)). Even if it's technically possible, you should never bootstrap more than one node at a time,

During the bootstrap process, pulled data from existing nodes are automatically indexed by Elasticsearch on the new node, involving a kind of an automatic Elasticsearch resharding. You can monitor and resume the Cassandra bootstrap process with the `nodetool bootstrap` command.

After bootstrap successfully ends, you should cleanup nodes to throw out any data that is no longer owned by that node, with a `nodetool cleanup`. Because cleanup involves by a Delete-by-query in Elasticsearch indices, it is recommended to smoothly schedule cleanups one at a time in you datacenter.

5.9 Backup and restore

By design, Elassandra synchronously update Elasticsearch indices on Cassandra write path and flushing a Cassandra table invlove a flush of all associated elasticsearch indices. Therefore, elassandra can backup data by taking a snapshot of Cassandra SSTables and Elasticsearch Lucene files on the same time on each node, as follow :

1. `nodetool snapshot --tag <snapshot_name> <keyspace_name>`
2. For all indices associated to `<keyspace_name>`

```
cp -al $CASSANDRA_DATA/elasticsearch.data/<cluster_name>/nodes/0/indices/
<index_name>/0/index/(*|segment*) $CASSANDRA_DATA/elasticsearch.data/
snapshots/<index_name>/<snapshot_name>/
```

Of course, rebuilding Elasticsearch indices after a Cassandra restore is another option.

5.9.1 Restoring a snapshot

Restoring Cassandra SSTable and Elasticsearch Lucene files allow to recover a keyspace and its associated Elasticsearch indices without stopping any node. (but it is not intended to duplicate data to another virtual datacenter or cluster)

To perform a hot restore of Cassandra keyspace and its Elasticsearch indices :

1. Close all Elasticsearch indices associated to the keyspace
2. Truncate all Cassandra tables of the keyspace (because of delete operation later than the snapshot)
3. Restore the Cassandra table with your snapshot on each node
4. Restore Elasticsearch snapshot on each nodes (if ES index is open during nodetool refresh, this cause Elasticsearch index rebuild by the compaction manager, usually 2 threads).
5. Load restored SSTables with a `nodetool refresh`
6. Open all indices associated to the keyspace

5.9.2 Point in time recovery

Point-in-time recovery is intended to recover the data at any time. This require a restore of the last available Cassandra and Elasticsearch snapshot before your recovery point and then apply the commitlogs from this restore point to the recovery point. In this case, replaying commitlogs on startup also re-index data in Elasticsearch indices, ensuring consistency at the recovery point.

Of course, when stopping a production cluster is not possible, you should restore on a temporary cluster, make a full snapshot, and restore it on your production cluster as describe by the hot restore procedure.

To perform a point-in-time-recovery of a Cassandra keyspace and its Elasticsearch indices, for all nodes in the same time :

1. Stop all the datacenter nodes.
2. Restore the last Cassandra snapshot before the restore point and commitlogs from that point to the restore point
3. Restore the last Elasticsearch snapshot before the restore point.
4. Restart your nodes

5.9.3 Restoring to a different cluster

It is possible to restore a Cassandra keyspace and its associated Elasticsearch indices to another cluster.

1. On the target cluster, create the same Cassandra schema without any custom secondary indices
2. From the source cluster, extract the mapping of your associated indices and apply it to your destination cluster. Your keyspace and indices should be open and empty at this step.

If you are restoring into a new cluster having the same number of nodes, configure it with the same token ranges (see https://docs.datastax.com/en/Cassandra/2.1/cassandra/operations/ops_snapshot_restore_new_cluster.html). In this case, you can restore from Cassandra and Elasticsearch snapshots as describe in step 1, 3 and 4 of the snapshot restore procedure.

Otherwise, when the number of node and the token ranges from the source and destination cluster does not match, use the `sstableloader` to restore your Cassandra snapshots (see https://docs.datastax.com/en/cassandra/2.0/cassandra/tools/toolsBulkloader_t.html). This approach is much time-and-io-consuming because all rows are read from the sstables and injected into the Cassandra cluster, causing an full Elasticsearch index rebuild.

5.10 How to change the elassandra cluster name

Because the cluster name is a part of the Elasticsearch directory structure, managing snapshots with shell scripts could be a nightmare when cluster name contains space characters. Therefore, it is recommended to avoid space characters in your elassandra cluster name.

On all nodes:

1. In a cqlsh, **UPDATE system.local SET cluster_name = '<new_cluster_name>' where key='local';**
2. Update the cluster_name parameter with the same value in your conf/cassandra.yaml
3. Run a `nodetool flush system` (this flush your system keyspace on disk)

Then:

4. On one node only, change the primary key of your cluster metadata in the elastic_admin.metadata table, using cqlsh :
 - **COPY elastic_admin.metadata (cluster_name, metadata, owner, version) TO 'metadata.csv';**
 - Update the cluster name in the file metadata.csv (first field in the JSON document).
 - **COPY elastic_admin.metadata (cluster_name, metadata, owner, version) FROM 'metadata.csv';**
 - **DELETE FROM elastic_admin.metadata WHERE cluster_name='<old_cluster_name>;**
5. Stop all nodes in the cluster
6. On all nodes, in your Cassandra data directory, move `elasticsearch.data/<old_cluster_name>` to `elasticsearch.data/<new_cluster_name>`
7. Restart all nodes
8. Check the cluster name in the Elasticsearch cluster state and that you can update the mapping.

Elassandra Enterprise plugin provides advanced features to monitor, manage and secure Elassandra. See [strapdata](#) for more details.

6.1 Installation

You must install the Enterprise plugin on ALL nodes of Elasticsearch enabled datacenters in your Cassandra cluster, and the Enterprise plugin version must match the version of Elassandra you are running. Installation require a full restart, usally a rolling restart of nodes.

1. Install the Enterprise Plugin :

```
bin/plugin install file:///path/to/enterprise-<version>.zip
```

2. Create a symbolic link from the **conf/triggers** directory to **plugins/enterprise/enterprise-<version>.jar**

```
(cd conf/triggers/ && ln -s ../../plugins/enterprise/enterprise-<version>.jar)
```

3. Restart your Elassandra node.

4. Check for installed plugins :

```
$curl -XGET "http://localhost:9200/_nodes?pretty"
...
"plugins" : [
  {
    "name" : "enterprise",
    "version" : "5.5.0",
    "description" : "Elassandra enterprise plugin.",
    "classname" : "com.strapdata.elasticsearch.plugin.EnterprisePlugin",
    "has_native_controller" : false
  },
  ...
]
```

To uninstall the Enterprise plugin :

1. Remove the symbolic link from the **conf/triggers** directory.

```
rm conf/triggers/enterprise-<version>.jar
```

2. Uninstalling the Enterprise plugin :

```
bin/plugin remove enterprise
```

3. Restart your Elassandra node.

6.2 License management

Enterprise plugin require a valid license. You can request for a free 30 days trail license or purchase a souscription including technical support for Elassandra. If your license expires, the enterprise plugin operates in a restricted mode until a valid license is installed.

Feature	Description	Restricted mode
JMX	JMX monitoring of Elasticsearch indices	Node restart required to see new index metrics, JMX attributes become read-only
SSL	SSL encryption of Elasticsearch connections	
AAA	User Authentication, Authorization and Audit	Node restart required to reload users' privileges, no more audit trails.
CBS	Content-Based Security rules	Node restart required to reload users' privileges.

Caution: If the number of nodes of licensed datacenters becomes greater than your license maximum number of nodes, the license becomes invalid on all your nodes.

6.2.1 License installation

Licenses are stored in a Cassandra table `elastic_admin.licenses`. You can also put a **conf/license.json** file, this file is automatically loaded at boot time if `elastic_admin.licenses` is empty.

```
cassandra@cqlsh> select * from elastic_admin.licenses;

id | clustername | company | datacenters | email |
--+-----+-----+-----+-----+
expire | features |
generated | issuer | maxnodes | production | signature |
start | type |
-----+-----+-----+-----+-----+
bbbef903-bbea-401d-838d-faf696e53547 | TestCluster | thecorp | ['DC1'] |
contact@thecorp | 2018-10-01 22:00:00.000000+0000 | ['JMX', 'SSL', 'AAA', 'CBS'] |
2017-10-02 13:23:09.227000+0000 | Strapdata | 3 | False |
0x302c02141404c757c3d0e387a8f6194669d5b0a677fbb82102145b88c2785ffabc26b3aa9df72ba03b65f4a829fe |
2017-10-01 22:00:00.000000+0000 | TRIAL
```

6.2.2 Checking your license

You can use the REST license API to check the currently active license. If your current configuration require SSL encryption and user authentication, you must provide a valid login, password and root CA certificate.

```
$ curl --user <username>:<password> --cacert conf/cacert.pem -XGET "https://
↪localhost:9200/_license?pretty"
{
  "id" : "bbbef903-bbea-401d-838d-faf696e53547",
  "issuer" : "Strapdata",
  "company" : "thecorp",
  "email" : "contact@thecorp",
  "generated" : "2017-10-02T13:23:09.227Z",
  "start" : "2017-10-01T22:00:00.000Z",
  "expire" : "2018-10-01T22:00:00.000Z",
  "production" : false,
  "max_nodes" : 3,
  "cluster_name" : "TestCluster",
  "datacenters" : [
    "DC1"
  ],
  "type" : "TRIAL",
  "features" : [
    "JMX",
    "SSL",
    "AAA",
    "CBS"
  ],
  "status" : "valid",
  "signature" :
↪"0x302c02141404c757c3d0e387a8f6194669d5b0a677fbb82102145b88c2785ffabc26b3aa9df72ba03b65f4a829fe
↪"
}
```

6.2.3 Upgrading your license

You can update your licence by inserting additional license row in the Cassandra `elastic_admin.licenses` table.

```
cassandra@cqlsh> INSERT INTO elastic_admin.licenses JSON '{"id":"bb0a181c-dbc6-4255-
↪8d69-67b6eld276ce","issuer":"Strapdata","company":"thecorp","email":"contact@thecorp
↪","type":"TRIAL","features":["JMX","SSL","AAA"],"production":false,"generated":
↪"2017-09-26 09:10:15.604Z","start":"2017-09-25 22:00:00.000Z","expire":"2018-09-25
↪22:00:00.000Z","clustername":"TestCluster","datacenters":["DC1"],"maxnodes":1,
↪"signature":
↪"0x302d02140b49e8c00b3606c66fe22378acb1ab781410460d02150092b666041dd97887b7d624fd6a12b6bd434a955ed
↪"}';
```

Then reload the license with a POST REST request as shown below, each nodes returns its active license If you have several licenses in `elastic_admin.licenses`, the **most recently generated valid** license is used.

```
$ curl --user <username>:<password> --cacert /opt/elassandra-5.5.0/conf/cacert.pe -
↳XPOST "https://localhost:9200/_license?pretty"
{
  "_nodes" : {
    "total" : 2,
    "successful" : 2,
    "failed" : 0
  },
  "cluster_name" : "TestCluster",
  "nodes" : {
    "d607917d-8c68-4cc5-8dc2-2aa21f5ea986" : {
      "name" : "127.0.0.2",
      "license_id" : "bbbef903-bbea-401d-838d-faf696e53547"
    },
    "a1c5307c-5f5a-4676-a6f0-50f221dd655b" : {
      "name" : "127.0.0.1",
      "license_id" : "bbbef903-bbea-401d-838d-faf696e53547"
    }
  }
}
```

Tip: If you have several Elasticsearch clusters in your Cassandra cluster, reload the license on each datacenter where Elasticsearch is enabled.

6.3 JMX Managment & Monitoring

The **JMX** technology provides a standard solution for managing and monitoring java applications. With the JMX feature, you can manage and monitor both Cassandra and Elasticsearch.

6.3.1 JMX Monitoring

The JMX feature expose Elasticsearch metrcis over JMX, allowing to monitor Elasticsearch cluster, index shards, threadpool and networks activities. You can browse these metrics with various JMX clients like [VisualVM](#) or [jmxterm](#).

JMXTerm example :

```
java -jar jmxterm-1.0.0-uber.jar -l localhost:7199
$>domain org.elasticsearch.index
#domain is set to org.elasticsearch.index
$>bean org.elasticsearch.index:name=sales_2017,scope=search,type=IndexShard
#bean is set to org.elasticsearch.index:name=sales_2017,scope=search,type=IndexShard
$>get *
#mbean = org.elasticsearch.index:name=sales_2017,scope=search,type=IndexShard:
QueryTotal = 21;
FetchTotal = 0;
ScrollTotal = 0;
QueryTimeInMillis = 56038;
QueryCurrent = 0;
FetchTimeInMillis = 0;
FetchCurrent = 0;
ScrollTimeInMillis = 0;
ScrollCurrent = 0;
```

```
SuggestCount = 0;
SuggestTimeInMillis = 0;
SuggestCurrent = 0;
$>
```

These metrics can be pulled, or pushed to various tools ([graphite](#), [ganglia](#) or [influxdb](#)) using the popular [Metrics Library](#) embedded in Apache Cassandra.

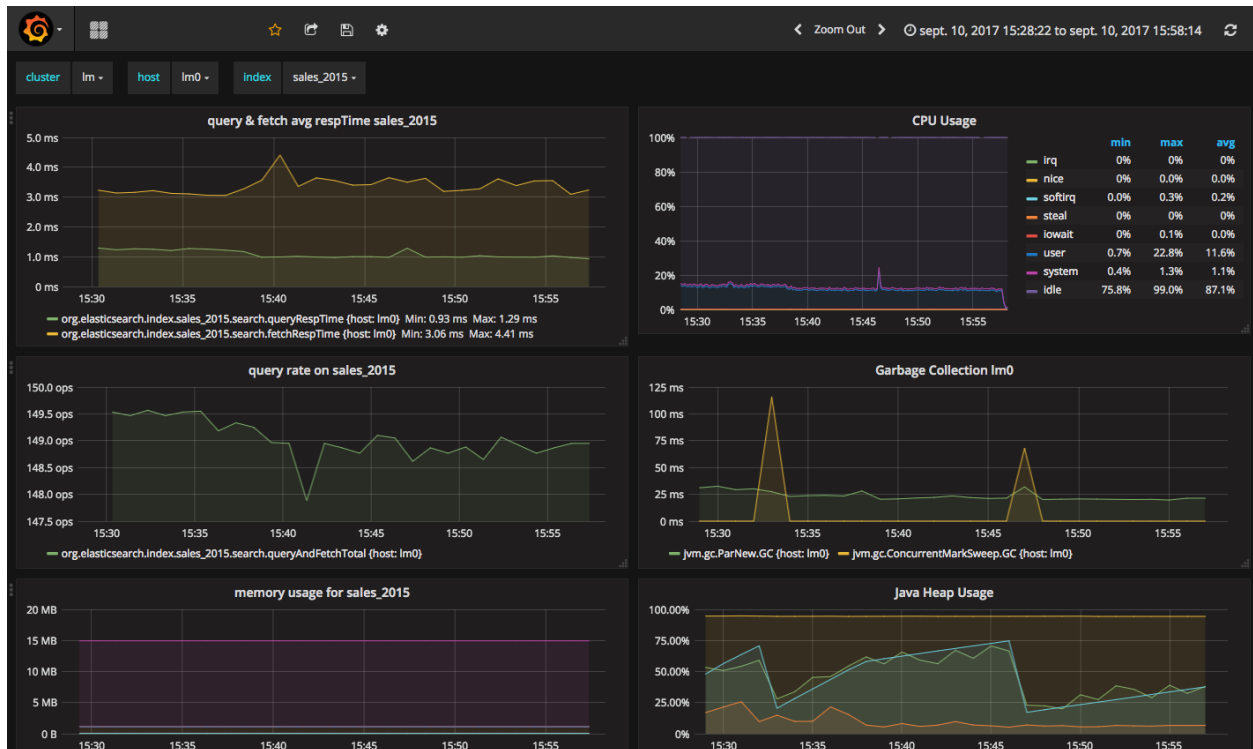
Here is a sample configuration located in **conf/influxdb-reporting.yaml** sending JMX metrics to an influxdb database named *elassandra*.

```
influxdb:
-
  dbName: 'elassandra'
  protocol: 'http'
  tags:
    environment: 'test'
    cluster: 'test_cluster'
    host: 'vm1'
  hosts:
    - host: 'vm1'
      port: 8086
  timeunit: 'SECONDS'
  period: 60
  prefix: ''
  groupGauges: true
```

To enable this configuration :

- add the following jars files to your elassandra **lib** directory :
 - [reporter-config-base-3.0.4.jar](#)
 - [reporter-config3-3.0.4.jar](#)
 - [metrics-influxdb-1.1.10-SNAPSHOT.jar](#)
 - [dropwizard-metrics-influxdb-1.1.10-SNAPSHOT.jar](#)
- add ***JVM_OPTS="\$JVM_OPTS -Dcassandra.metricsReporterConfigFile=influxdb-reporting.yaml"*** in your **conf/cassandra-env.sh**

Then configure Grafana to build your Elassandra dashboard.



6.3.2 Enable/Disable search on a node

The JMX feature allows to exclude/include a node from distributed search while still receiving CQL write, repairing or rebuilding its elasticsearch indices, by setting the following attributes on the JMX Bean `org.elasticsearch.node:type=node`

JMX At-tribute	De-fault value	Description
SearchEnabled	true	Set whether or not the node is involved in distributed search queries from other nodes. When SearchEnabled is false on a node, all its shards are seen UNASSIGNED from other nodes.
AutoEnableSearch	true	If true, the node automatically set SearchEnabled to true when it becomes available, participating to distributed search queries. In order to restart a node in a maintenance mode for search requests, you can set AutoEnableSearch to false with the system property <code>es.auto_enable_search</code> .

To set SearchEnabled on command line, just use **jmxterm** as in the following exemple.

```
echo "set -b org.elasticsearch.node:type=node SearchEnabled false" | java -jar \
  ↪ jmxterm-1.0.0-uber.jar -l localhost:7199
```

6.4 SSL Network Encryption

The SSL Feature provides traffic encryption for both HTTP and Elasticsearch transport connections.

Note: Elasticsearch transport protocol is the native binary protocol used for Elasticsearch node-to-node communication. You can also use the transport protocol from a client application written in java, as describe in the [elasticsearch documentation](#).

SSL configuration is defined in your **conf/cassandra.yaml** for both Cassandra and Elasticsearch :

- Server options defines node-to-node encryption for both Cassandra and Elasticsearch. Obviously, Elasticsearch transport connections are encrypted when *internode_encryption* is set to **all** or **rack** (there is no elasticsearch cross-datacenter traffic).
- Client options defines client-to-node encryption to request both Cassandra and Elasticsearch. If *optional* is **true**, Elasticsearch still accepts clear connections for HTTP and transport request.

To ensure support for all encryption algorithms, it is highly recommended to install the [JCE Unlimited Strength Jurisdiction policy files](#) on all nodes.

Here an SSL configuration in your **conf/cassandra.yaml** file :

```
# Enable or disable inter-node encryption
# Default settings are TLS v1, RSA 1024-bit keys (it is imperative that
# users generate their own keys) TLS_RSA_WITH_AES_128_CBC_SHA as the cipher
# suite for authentication, key exchange and encryption of the actual data transfers.
# Use the DHE/ECDHE ciphers if running in FIPS 140 compliant mode.
# NOTE: No custom encryption options are enabled at the moment
# The available internode options are : all, none, dc, rack
#
# If set to dc cassandra will encrypt the traffic between the DCs
# If set to rack cassandra will encrypt the traffic between the racks
#
# The passwords used in these options must match the passwords used when generating
# the keystore and truststore.  For instructions on generating these files, see:
# http://download.oracle.com/javase/6/docs/technotes/guides/security/jsse/
# ↪JSSERefGuide.html#CreateKeystore
#
server_encryption_options:
  internode_encryption: all
  keystore: conf/.keystore.jks
  keystore_password: changeit
  truststore: conf/.truststore.jks
  truststore_password: changeit
  # More advanced defaults below:
  protocol: TLSv1.2
  # algorithm: SunX509
  # store_type: JKS
  # cipher_suites: [TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_
# ↪DHE_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_
# ↪AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA]
  # require_client_auth: true

# enable or disable client/server encryption.
client_encryption_options:
  enabled: true
  # If enabled and optional is set to true encrypted and unencrypted connections_
# ↪are handled.
  optional: true
  keystore: conf/.keystore.jks
  keystore_password: changeit
  require_client_auth: true
```

```
# Set trustore and truststore_password if require_client_auth is true
truststore: conf/.truststore.jks
truststore_password: changeit
# More advanced defaults below:
protocol: TLSv1.2
# algorithm: SunX509
# store_type: JKS
# cipher_suites: [TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_
↪DHE_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_
↪AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA]
```

Caution: If paths to keystores are relative, you could face an issue when starting elassandra from another directory than the install directory. You should use absolute keystore paths to avoid such issues.

6.4.1 Elasticsearch SSL configuration

SSL for Elasticsearch is activated according to the following settings in your **conf/elasticsearch.yml** :

Setting	De- fault	Description
<code>https.enabled</code>	false	Enable HTTPS on client-to-node Elasticsearch connections
<code>ssl.transport. enabled</code>	false	Enable SSL on Elastisearch transport connections (node-to-node connections)

Once HTTPS is enabled, accessing your elasticsearch cluster requires the HTTPS protocol and a trusted certificate to validate the server side certificate :

```
curl -XGET --cacert conf/cacert.pem "https://localhost:9200/my_index/_search"
```

You can also check your SSL configuration with a GET `/_sslinfo` request.

```
curl -XGET --cacert conf/cacert.pem "https://localhost:9200/_sslinfo"
{
  "https_protocol" : "TLSv1.2",
  "https_cipher" : "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384"
}
```

If client encryption is enabled in your **conf/cassandra.yaml**, and `require_client_auth=true`, a client certificate is required to connect.

6.4.2 JMX traffic Encryption

Enable SSL for JMX by setting the following parameters.

```
JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.ssl=true"
JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.ssl.need.client.auth=true"
JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.registry.ssl=true"
#JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.ssl.enabled.protocols=<enabled-
↪protocols>"
#JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.ssl.enabled.cipher.suites=
↪<enabled-cipher-suites>"
```

```
JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.keyStore=<install_dir>/conf/server-keystore.jks"
JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.keyStorePassword=changeit"
JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.trustStore=<install_dir>/cassandra/conf/server-
↪truststore.jks"
JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.trustStorePassword=changeit"
```

Once SSL is enabled on JMX, `nodetool` utility requires the `-ssl` option.

6.5 Authentication and Authorization

Elasticsearch authentication and authorization is based on Cassandra internal [Authentication and Role-Based Access Control](#), allowing to get an homogeneous security policy.

6.5.1 Cassandra internal authentication

To enable Cassandra authentication, set the following settings in your `conf/cassandra.yaml`, and restart your node :

```
authenticator: PasswordAuthenticator
authorizer: CassandraAuthorizer
```

Once authentication is enabled, create a new Cassandra superuser to avoid issue with the default “cassandra” superuser (Authentication with the cassandra superuser require QUORUM nodes to be available in your cluster), and change the default cassandra password.

```
cqlsh> CREATE ROLE admin WITH PASSWORD='*****' AND LOGIN=true AND SUPERUSER=true;
cqlsh> ALTER ROLE cassandra WITH PASSWORD='*****';
```

Then configure the replication factor for the `system_auth` keyspace according to your cluster configuration (see [Configure Native Authentication](#)). Finally, adjust roles and credential cache settings and disable JMX configuration of authentication and authorization `cacheElasticsearch Authentication, Authorization and Content-Based Security`

Elasticsearch authentication settings are defined in `conf/elasticsearch.yml`. To be effective, these settings must be the same on all nodes of a Cassandra datacenter.

Setting	Default	Description
aaa.enabled	false	Enable Elasticsearch authentication and authorization.
aaa.rest.prompt	true	By default, a rejected HTTP request returns with a 403 code, meaning access is forbidden. When prompt is configured, rejected anonymous HTTP requests return a code 401 this prompt in the authorization header.
aaa.rest.realm	\${cluster_name} authentication required	Prompted realm when HTTP authentication is required.
aaa.user_header		When user is already authenticated by an HTTP proxy, you can define the HTTP header name used to carry the cassandra user's name used to execute an elasticsearch request. To avoid security breach, you should properly restrict unauthenticated access to elassandra when using such mechanism.
aaa.anonymous_user		Defines the cassandra user's name used to execute unauthenticated request. If undefined, unauthenticated requests are rejected.
aaa.shared_secret	Base64 encoded cluster name	Shared secret used to tag authorized requests on the coordinator node. This should be a confidential per datacenter secret.
cbs.enabled	false	Enable or disable Content-Based Security.

Tip: Elasticsearch **user authentication requires HTTPS**. (User authentication without HTTPS is not supported).

In order to grant an Elasticsearch request, Elassandra checks two levels of access rights :

1. First, Elassandra looks up for a **privilege** matching your elasticsearch request in the Cassandra table `elastic_admin.privileges`.
2. If no privilege matches and request is related to indices, Elassandra looks up for a Cassandra **permission** associated to the user's roles.

6.5.2 Privileges

Privileges are defined in the Cassandra table `elastic_admin.privileges`.

```
CREATE TABLE elastic_admin.privileges (
  role text,
  actions text,
  indices text,
  fields set<text>,
  query text,
  PRIMARY KEY (role, actions, indices)
);
```

- **role:** The user's role.
- **actions:** Regular expression defining the authorized actions.
- **indices:** Regular expression defining the authorized target indices. If null, all indices backed by keyspaces associated to the role.

- **fields:** List of visible fields of documents when Content-Base Security is enabled. Supports wilcards, for exemple `foo*` will match all fields starting by `foo`. If your request matches multiple privileges, returned document may contains all associated fields.
- **query:** Filter query when Content-Base Security is enabled. If your request matches multiple privileges, returned document are filtered with all queries.

Important:

- Cassandra superuser have full access to Elasticsearch.
- All cluster-level access should be granted using privileges.
- Content-Based Security should be used with read-only accounts.

6.5.3 Permissions

Cassandra permission associated to a role are **granted** or **revoked** as shown below :

```
cassandra@cqlsh> GRANT SELECT ON KEYSPACE sales TO sales;
cassandra@cqlsh> LIST ALL PERMISSIONS;
```

role	username	resource	permission
cassandra	cassandra	<role sales>	ALTER
cassandra	cassandra	<role sales>	DROP
cassandra	cassandra	<role sales>	AUTHORIZE
sales	sales	<keyspace sales>	MODIFY

```
(4 rows)

cassandra@cqlsh> REVOKE SELECT ON KEYSPACE sales FROM sales;
```

Cassandra permissions associated to a role are mapped to Elasticsearch Document and Indices APIs as follow.

Cassandra privilege	Cassandra Permissions	Elasticsearch Action	Elasticsearch API
CREATE	CREATE KEYSPACE and CREATE TABLE in any keyspace.	indices:admin/create	Create Index
ALTER	ALTER KEYSPACE and ALTER TABLE in any keyspace.	indices:admin/mapping indices:admin/alias indices:admin/template indices:admin/settings/update	Put Mapping Index Alias Index Templates Update Indices Settings
DROP	DROP KEYSPACE and DROP TABLE in any keyspace.	indices:admin/delete	Delete Index
EXECUTE	Execute operations on any elasticsearch indices associated to the granted keyspaces.	indices:admin/refresh indices:admin/flush indices:admin/optimize indices:admin/open indices:admin/close indices:admin/cache/clear indices:admin/analyze	Refresh Flush Force Merge Open Index Close Index Clear Cache Analyze
DESCRIBE	Retrieve stats about elasticsearch indices associated to the granted mbeans.	indices:monitor/stats indices:monitor/segments	Indices Stats Indices Segments
SELECT	SELECT on any table.	indices:data/read/* indices:admin/get indices:admin/exists indices:admin/types/exists indices:admin/mapping indices:admin/mappings/fields/get	All document reading API Get Index Indices Exists Type Exists Get Mapping Get Field Mapping
MODIFY	INSERT, UPDATE, DELETE on any table.	indices:data/write/*	All document writing API

6.5.4 Privilege caching

For performance reasons, the elasticsearch privilege table is cached into memory, according the the following settings in `conf/elasticsearch.yml` :

Setting	Default	Description
<code>aaa.privilege_cache_expire</code>	1h	Privilege cache entry TTL
<code>aaa.privilege_cache_size</code>	1024	Privilege cache max entries.

When changing a privilege in `elastic_admin.privileges`, you should clear the cache with the follwing REST request to put the change into effect on available nodes :

```
curl -XPOST --user admin:admin --cacert conf/cacert.pem "https://localhost:9200/_aaa_
→clear_privilege_cache?pretty"
{
  "_nodes" : {
    "total" : 2,
    "successful" : 2,
    "failed" : 0
  },
  "cluster_name" : "TestCluster",
  "nodes" : {
    "d607917d-8c68-4cc5-8dc2-2aa21f5ea986" : {
```

```

    "name" : "127.0.0.2"
  },
  "a1c5307c-5f5a-4676-a6f0-50f221dd655b" : {
    "name" : "127.0.0.1"
  }
}
}

```

If you just want to invalidate the privilege cache for some roles, you can specify roles :

```
POST _aaa_clear_privilege_cache?pretty&roles=sales,kibana"
```

Tip: If you are running multiple Elasticsearch cluster in your Cassandra cluster, you should clear privilege cache on each datacenter where Elasticsearch is enabled.

6.6 Integration

6.6.1 Secured Transport Client

The elasticsearch transport protocol used for inter-node communication can be used directly from your java application. It is very efficient as it does not have to deal with JSON serialization. Strapdata provides a SSL transport client to work with a secured Elassandra cluster :

1. If your Elassandra cluster requires user authentication, check that your user have access to the cluster topology with the *Nodes Info API* (action **cluster:monitor/nodes/info**).
2. Add the **ssl-transport-client.jar** and its dependencies in your CLASSPATH.
3. Add the desired configuration to your client settings, as shown in the following exemple.
4. Add an **Authorization** header to your client containing your based-64 encoded login and password.

```

...
import com.strapdata.elasticsearch.plugins.ssl.PreBuiltSslTransportClient;

TransportClient client = new PreBuiltSslTransportClient(Settings.builder()
    .put("cluster.name", "myClusterName")
    .put("ssl.transport.enabled", true)
    .put("ssl.truststore.path", "/path/to/truststore.jks")
    .put("ssl.truststore.password", "*****")
    ...
    .build())
    .build()
    .addTransportAddress(new InetSocketAddress(InetAddress.getByName("localhost"), 9300))

client.filterWithHeader(Collections.singletonMap("Authorization",
    PreBuiltSslTransportClient.encodeBasicHeader("bob", "password")))
    .prepareSearch().get();

```

Available security settings for the secured transport client for Elassandra :

Setting	Default	Description
ssl.transport.enabled	false	Enable SSL on transport connections.
ssl.algorithm	SunX509	Algorithm used to manage keys and certificates.
ssl.storetype	JKS	Cryptographic stores file format.
ssl.trust_all_cert	false	Trust all certificates
ssl.truststore.path	conf/.truststore	Path to your truststore.
ssl.truststore.password	cassandra	Truststore password.
ssl.protocol	TLSv1.2	Secure protocol.
ssl.ciphers	JCE default	SSL Cipher suite
ssl.require_client_auth	false	Enable SSL client authentication.
ssl.keystore.path	conf/.truststore	Path to your keystore when using SSL client authentication.
ssl.keystore.password	cassandra	Truststore password when using SSL client authentication.
ssl.require_endpoint_verification	false	Enable server hostname verification.

6.6.2 Multi-user Kibana configuration

Kibana needs a dedicated kibana account to manage kibana configuration, with the CREATE, ALTER, MODIFY, SELECT cassandra permissions.

```
cassandra@cqlsh> CREATE ROLE kibana WITH PASSWORD = '*****' AND LOGIN = true;
cassandra@cqlsh> CREATE KEYSPACE "_kibana" WITH replication = {'class':
↪ 'NetworkTopologyStrategy', 'DC1':'1'};
cassandra@cqlsh> GRANT CREATE ON KEYSPACE "_kibana" TO kibana;
cassandra@cqlsh> GRANT ALTER ON KEYSPACE "_kibana" TO kibana;
cassandra@cqlsh> GRANT SELECT ON KEYSPACE "_kibana" TO kibana;
cassandra@cqlsh> GRANT MODIFY ON KEYSPACE "_kibana" TO kibana;
cassandra@cqlsh> LIST ALL PERMISSIONS OF kibana;
```

role	username	resource	permission
kibana	kibana	<keyspace _kibana>	CREATE
kibana	kibana	<keyspace _kibana>	ALTER
kibana	kibana	<keyspace _kibana>	SELECT
kibana	kibana	<keyspace _kibana>	MODIFY

Add cluster monitoring access rights to the *kibana* user.

```
cassandra@cqlsh> INSERT INTO elastic_admin.privileges (role,actions,indices) VALUES (
↪ 'kibana','cluster:monitor/.*','.*');
cassandra@cqlsh> SELECT * FROM elastic_admin.privileges ;
```

role	actions	indices	fields	query
kibana	cluster:monitor/.*	.*	null	null

Finally, user accounts must have :

- the SELECT permission on visualized indices, especially on your default kibana index.
- the SELECT permission on the kibana keyspace to read kibana configuration.
- the MODIFY permission on the kibana keyspace to store kibana configuration if authorized to create/update kibana objects.

Tip: Once a user is authenticated by kibana, kibana keeps this information. In order to logout from your browser,

clear cookies and data associated to your kibana server.

6.6.3 Elasticsearch Spark connector

The `elasticsearch-hadoop` connector can access a secured Elassandra cluster by providing the same SSL/TLS and Username/Password authentication parameters as the original `elasticsearch-hadoop` connector. Here is an example with the `spark-submit`.

```
ES_OPTS="$ES_OPTS --conf spark.es.nodes=127.0.0.1"
ES_OPTS="$ES_OPTS --conf spark.es.nodes.wan.only=true"
ES_OPTS="$ES_OPTS --conf spark.es.net.ssl=true"
ES_OPTS="$ES_OPTS --conf spark.es.net.ssl.truststore.location=file:///path/to/
↳truststore.jks"
ES_OPTS="$ES_OPTS --conf spark.es.net.ssl.truststore.pass=*****"
ES_OPTS="$ES_OPTS --conf spark.es.net.http.auth.user=john"
ES_OPTS="$ES_OPTS --conf spark.es.net.http.auth.pass=*****"

bin/spark-shell --driver-class-path path/to/elasticsearch-hadoop-5.5.0.jar $ES_OPTS
```

In order to work, the elasticsearch spark connector requires privileges to monitor your cluster and request for available shards for search. You can associate these privileges to a dedicated cassandra role `spark`, and grant this role to the account used in your spark application. The `spark` role has no cassandra permission, but user `john` inherits its privileges from the `elastic_admin.privileges` table.

```
cassandra@cqlsh> CREATE ROLE spark;
cassandra@cqlsh> INSERT INTO elastic_admin.privileges (role,actions,indices) VALUES (
↳'spark','cluster:monitor/.*','.*');
cassandra@cqlsh> INSERT INTO elastic_admin.privileges (role,actions,indices) VALUES (
↳'spark','indices:admin/shards/search_shards','.*');
cassandra@cqlsh> SELECT * FROM elastic_admin.privileges WHERE role='spark';
```

role	actions	indices	fields	query
spark	cluster:monitor/.*	.*	null	null
spark	indices:admin/shards/search_shards	.*	null	null

```
(2 rows)
cassandra@cqlsh> GRANT spark TO john;
cassandra@cqlsh> LIST ROLES OF john;
```

role	super	login	options
spark	False	False	{}
john	False	True	{}

```
(2 rows)
```

6.7 Elasticsearch Auditing

Elasticsearch auditing tracks security events with the following fields :

Field	Description
status	GRANTED(200), UNAUTHORIZED(401), FORBIDDEN(403), BLOCKED(409)
type	PRIVILEGE, PERMISSION, UNAUTHORIZED, UNSUPPORTED, TAMPERED
login	User login
role	Cassandra role
source	Source IP of the elasticsearch request
action	Elasticsearch action
indices	Requested indices

Audits events are recorded in a Cassandra table or in a log file configured as an appender in your **conf/logback.xml** file.

Setting	De- fault	Description
aaa.audit.enabled	false	Enable or disable Elasticsearch auditing.
aaa.audit.appender	none	Audits events are recorded in a Cassandra table (cql) or in a logback appender (log).
aaa.audit. include_login		Comma separated list of logins to audit
aaa.audit. exclude_login		Comma separated list of logins not audited

6.7.1 Logback Audit

When using the **log** appender for audit, you should configure a dedicated logback appender in your **conf/logback.xml** file :

```
<appender name="AUDIT" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${cassandra.logdir}/audit.log</file>
  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
    <fileNamePattern>${cassandra.logdir}/audit.log.%i.zip</fileNamePattern>
    <minIndex>1</minIndex>
    <maxIndex>20</maxIndex>
  </rollingPolicy>
  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
    <maxFileSize>500MB</maxFileSize>
  </triggeringPolicy>
  <encoder>
    <pattern>%date{ISO8601} %msg%n</pattern>
  </encoder>
</appender>
```

And add a logger named **LogbackAuditor** with additivity set to **false** :

```
<logger name="LogbackAuditor" level="DEBUG" additivity="false" >
  <appender-ref ref="AUDIT" />
</logger>
```

Here an exemple of audit logs in the **logs/audit.log** file :

```
2017-10-20 14:11:49,854 200,PERMISSION,sales,roles/sales,/10.0.1.5,indices:data/read/
↪search,[sales_*]
2017-10-20 14:11:51,607 200,PERMISSION,sales,roles/sales,/10.0.1.5,indices:data/read/
↪search,[.kibana]
```

```

2017-10-20 14:11:52,377 200,PRIVILEGE,kibana,roles/kibana,/10.0.1.5,cluster:monitor/
↪main,null
2017-10-20 14:11:52,501 200,PRIVILEGE,kibana,roles/kibana,/10.0.1.5,cluster:monitor/
↪nodes/info,null
2017-10-20 14:11:52,627 200,PRIVILEGE,kibana,roles/kibana,/10.0.1.5,cluster:monitor/
↪nodes/info,null
2017-10-20 14:11:52,679 200,PERMISSION,sales,roles/sales,/10.0.1.5,indices:data/read/
↪mget[shard],[.kibana]
2017-10-20 14:11:52,751 200,PERMISSION,kibana,roles/kibana,/10.0.1.5,indices:data/
↪read/mget[shard],[.kibana]
2017-10-20 14:11:52,868 200,PRIVILEGE,kibana,roles/kibana,/10.0.1.5,cluster:monitor/
↪health,[.kibana]
2017-10-20 14:11:52,990 200,PERMISSION,kibana,roles/kibana,/10.0.1.5,indices:data/
↪read/search,[.kibana]

```

6.7.2 CQL Audit

When using the `cql` appender for audit, audit events are recorded in the cassandra table `elastic_audit.events`.

```

cassandra@cqlsh> select * from elastic_audit.events ;

node      | event
↪indices  | level      | login  | role      | action      | source      | status
-----+-----+-----+-----+-----+-----+-----
↪-----+-----+-----+-----+-----+-----+-----
10.0.0.4 | cf74fed0-b5a2-11e7-9508-157b11ac2561 |      cluster:monitor/main |
↪      null | PRIVILEGE | kibana | roles/kibana | 10.0.1.5 | 200
10.0.0.4 | d2026070-b5a2-11e7-9508-157b11ac2561 |      cluster:monitor/state |
↪      null | PRIVILEGE | kibana | roles/kibana | 10.0.1.5 | 200
10.0.0.4 | da709470-b5a2-11e7-9508-157b11ac2561 |      indices:data/read/search | [
↪'sales_*'] | PERMISSION | sales | roles/sales | 10.0.1.5 | 200
10.0.0.4 | d8025390-b5a2-11e7-9508-157b11ac2561 |      cluster:monitor/health | ['.
↪kibana'] | PRIVILEGE | kibana | roles/kibana | 10.0.1.5 | 200
10.0.0.4 | cf9de390-b5a2-11e7-9508-157b11ac2561 |      cluster:monitor/nodes/info |
↪      null | PRIVILEGE | kibana | roles/kibana | 10.0.1.5 | 200

```

If you want to have multiple copies of audit events in your cluster, you can alter the following default settings :

Setting	Default	Description
<code>aaa.audit.cql.rf</code>	1	Cassandra <i>Replication Factor</i> used when creating the <code>elastic_audit</code> keyspace.
<code>aaa.audit.cql.cl</code>	LOCAL_ONE	Write <i>Consistency Level</i> for audit events.

You can index with elasticsearch the `elastic_audit.events` table with the following mapping, where the `eventtimeuuid` column is explicitly mapped to a date :

```

curl -XPUT --user admin:admin --cacert conf/cacert.pem "https://localhost:9200/
↪elastic_audit/" -d'
{
  "mappings":{
    "events":{
      "discover":"^((?!event).*)",
      "properties":{
        "event":{

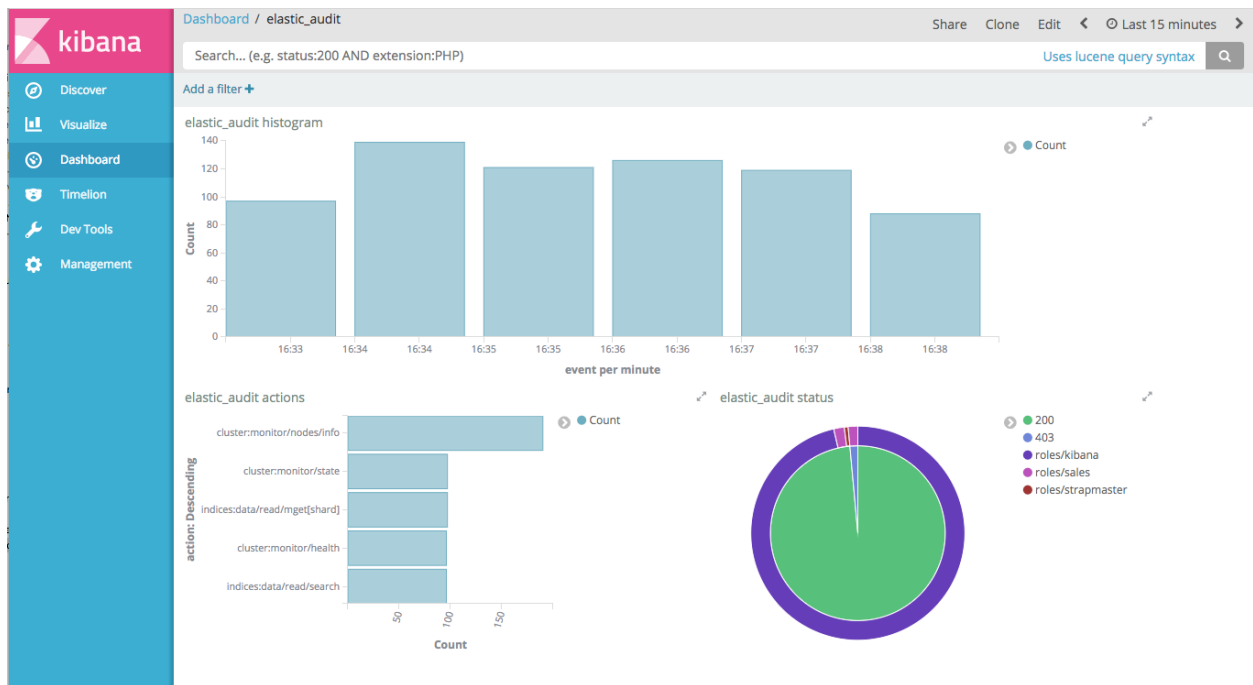
```

```

        "type": "date",
        "cql_collection": "singleton"
      }
    }
  }
}

```

Then you can build you audit trail kibana report.



Tip: Keep in mind that CQL audit trail involves a network overhead because each node send some events to all other nodes. For better performances, you should use the Logback audit and collect events with Beat+Logstash into a dedicated elassandra cluster.

6.8 Limitations

6.8.1 Content-Based Security Limitations

- The request cache is disabled for search requests.
- The following queries are not supported for document-level filtering :
 - **Has Parent, Has Child** queries.
 - **Terms** queries with lookups.
 - **Geo Shape** queries without inline shape definition.
 - **Percolate** queries.

If you try to insert an unsupported query in `elastic_admin.privileges.query`, you will get a syntax error as show bellow :

```
cassandra@cqlsh> insert into elastic_admin."privileges" (role,actions,indices,query)
↪VALUES ('blogger','indices:data/read/*','blog','{"query":{ "has_parent":{"parent_
↪type":"blog","query":{"term":{"tag":"something"}}}}});
SyntaxException: Unsupported query for content-based filtering
```


7.1 Integration with an existing cassandra cluster

Elassandra include a modified version of cassandra, available at [strapdata-cassandra repro](#), so **all nodes of a cluster should run elassandra binaries**. However, you can start a node with or without the elasticsearch support. Obviously, all nodes of a datacenter should run cassandra only or cassandra with elasticsearch.

7.1.1 Rolling upgrade to elassandra

Before starting any elassandra node with elasticsearch enable, do a rolling replace of the cassandra binaries by the elassandra ones. For each node :

- Install elassandra.
- Replace the elassandra configuration files by the one from your existing cluster (cassandra.yml and snitch configuration file)
- Stop your cassandra node.
- Restart cassandra `elassandra bin/cassandra` or cassandra with elasticsearch enable `elassandra bin/cassandra -e`

7.1.2 Create a new elassandra datacenter

The overall procedure is similar the cassandra one describe on [Adding a datacenter to a cluster](#).

For each nodes in your new datacenter :

- Install elassandra.
- Set `auto_bootstrap: false` in your **conf/cassandra.yaml**.
- Start cassandra-only nodes in your new datacenter and check that all nodes join the cluster.

```
bin/cassandra
```

- Restart all nodes in your new datacenter with elasticsearch enable. You should see started shards but empty indices.

```
bin/cassandra -e
```

- Set the replication factor of indexed keyspaces to one or more in your new datacenter.
- Pull data from your existaing datacenter.

```
nodetool rebuild <source-datacenter-name>
```

After rebuild on all your new nodes, you should see the same number of document for each indices in your new and existing datacenters.

- Set `auto_bootstrap: true` (default value) in your **conf/cassandra.yaml**
- Create new elasticsearch index or map some existing cassandra tables.

Tip: If you need to replay this procedure for a node :

- stop your node
 - `nodetool removenode <id-of-node-to-remove>`
 - clear data, commitlogs and saved_cache directories.
-

7.2 Installing an Elasticsearch plugins

Elasticsearch plugin installation remains unchanged, see [elasticsearch plugin installation](#).

- `bin/plugin install <url>`

7.3 Running Kibana with Elassandra

Kibana can run with Elassandra, providing a visualization tool for cassandra and elasticsearch data.

- If you want to load sample data from the [Kibana Getting started](#), apply the following changes to `logstash.jsonl` with a `sed` command.

```
s/logstash-2015.05.18/logstash_20150518/g
s/logstash-2015.05.19/logstash_20150519/g
s/logstash-2015.05.20/logstash_20150520/g

s/article:modified_time/articleModified_time/g
s/article:published_time/articlePublished_time/g
s/article:section/articleSection/g
s/article:tag/articleTag/g

s/og:type/ogType/g
s/og:title/ogTitle/g
s/og:description/ogDescription/g
s/og:site_name/ogSite_name/g
```

```
s/og:url/ogUrl/g
s/og:image:width/ogImageWidth/g
s/og:image:height/ogImageHeight/g
s/og:image/ogImage/g

s/twitter:title/twitterTitle/g
s/twitter:description/twitterDescription/g
s/twitter:card/twitterCard/g
s/twitter:image/twitterImage/g
s/twitter:site/twitterSite/g
```

7.4 JDBC Driver sql4es + Elassandra

The [Elasticsearch JDBC driver](#), can be used with elassandra. Here is a code example :

```
Class.forName("nl.anchorment.sql4es.jdbc.ESDriver");
Connection con = DriverManager.getConnection("jdbc:sql4es://localhost:9300/twitter?
↳cluster.name=Test%20Cluster");
Statement st = con.createStatement();
ResultSet rs = st.executeQuery("SELECT user,avg(size),count(*) FROM tweet GROUP BY_
↳user");
ResultSetMetaData rsmd = rs.getMetaData();
int nrCols = rsmd.getColumnCount();
while(rs.next()){
    for(int i=1; i<=nrCols; i++){
        System.out.println(rs.getObject(i));
    }
}
rs.close();
con.close();
```

7.5 Running Spark with Elassandra

A modified version of the [elasticsearch-hadoop](#) connector is available for elassandra at <https://github.com/strapdata/elasticsearch-hadoop>. This connector works with spark as describe in the elasticsearch documentation available at <https://www.elastic.co/guide/en/elasticsearch/hadoop/current/index.html>.

For example, in order to submit a spark job in client mode.

```
bin/spark-submit --driver-class-path <yourpath>/elasticsearch-spark_2.10-2.2.0.jar --
↳master spark://<sparkmaster>:7077 --deploy-mode client <application.jar>
```


Elasticsearch comes with a testing framework based on [JUNIT](#) and [RandomizedRunner](#) provided by the randomized-testing project. Most of these tests work with Elassandra to ensure compatibility between Elasticsearch and Elassandra.

8.1 Testing environnement

By default, JUnit creates one instance of each test class and executes each *@Test* method in parallel in many threads. Because Cassandra use many static variables, concurrent testing is not possible, so each test is executed sequentially (using a semaphore to serialize tests) on a single node Elassandra cluster listening on localhost, see [ESSingleNodeTestCase](#)). Test configuration is located in **core/src/test/resources/conf**, data and logs are generated in **core/build/testrun/test/J0**.

Between each test, all indices (and underlying keyspaces and tables) are removed to have idempotent testings and avoid conflicts on index names. System settings `es.synchronous_refresh` and `es.drop_on_delete_index` are set to `true` in the parent *pom.xml*.

Finally, the testing framework randomizes the locale settings representing a specific geographical, political, or cultural region, but Apache Cassandra does not support such setting because string manipulation are implemented with the default locale settings (see CASSANDRA-12334). For example, `String.format("SELECT %s FROM ...",...)` is computed as `String.format(Local.getDefault(),"SELECT %s FROM ...",...)`, involving errors for some Locale setting. As a workaround, a javassist byte-code manipulation in the Ant build step adds a *Locale.ROOT* argument to weak method calls in all Cassandra classes.

8.2 Elassandra unit test

Elassandra unit test allows to use both the Elasticsearch API and CQL requests as shown in the following sample.

```
public class BasicTests extends ESSingleNodeTestCase {  
  
    @Test  
    public void testTest() throws Exception {
```

```
createIndex("cmdb");
ensureGreen("cmdb");

process(ConsistencyLevel.ONE,"CREATE TABLE cmdb.server ( name text, ip inet,
↪netmask int, prod boolean, primary key (name))");
assertAcked(client().admin().indices().preparePutMapping("cmdb")
    .setType("server")
    .setSource("{ \"server\" : { \"discover\" : \".*\", \"properties\": { \
↪\"name\":{ \"type\": \"keyword\" }}}}")
    .get());

process(ConsistencyLevel.ONE,"insert into cmdb.server (name,ip,netmask,prod)
↪VALUES ('localhost','127.0.0.1',8,true)");
process(ConsistencyLevel.ONE,"insert into cmdb.server (name,ip,netmask,prod)
↪VALUES ('my-server','123.45.67.78',24,true)");

assertThat(client().prepareGet().setIndex("cmdb").setType("server").setId("my-
↪server").get().isExists(), equalTo(true));
assertThat(client().prepareGet().setIndex("cmdb").setType("server").setId(
↪"localhost").get().isExists(), equalTo(true));

assertEquals(client().prepareIndex("cmdb", "server", "bigserver234")
    .setSource("{ \"ip\": \"22.22.22.22\", \"netmask\":32, \"prod\" : true, \
↪\"description\": \"my big server\" }")
    .get().getResult(), DocWriteResponse.Result.CREATED);

assertThat(client().prepareSearch().setIndices("cmdb").setTypes("server")
↪setQuery(QueryBuilders.queryStringQuery("*:~")).get().getHits().getTotalHits(),
↪equalTo(3L));
}
```

To run this specific test :

```
$gradle :core:test -Dtests.seed=96A0B026F3E89763 -Dtests.class=org.elassandra.
↪BasicTests -Dtests.security.manager=false -Dtests.locale=it-IT -Dtests.
↪timezone=Asia/Tomsk
```

To run all core unit tests :

```
$gradle core:test
```

Breaking changes and limitations

9.1 Deleting an index does not delete cassandra data

By default, Cassandra is considered as a primary data storage for Elasticsearch, so deleting an Elasticsearch index does not delete Cassandra content, keyspace and tables remain unchanged. If you want to use Elasticsearch as Elasticsearch, you can configure your cluster or only some indices with the `drop_on_delete_index` like this.

```
$curl -XPUT "$NODE:9200/twitter/" -d'{
  "settings":{ "index":{ "drop_on_delete_index":true } }
}'
```

Or to set `drop_on_delete_index` at cluster level :

```
$curl -XPUT "$NODE:9200/_cluster/settings" -d'{
  "persistent":{ "cluster.drop_on_delete_index":true }
}'
```

9.2 Cannot index document with empty mapping

Elasticsearch cannot index any document for a type having no mapped properties and no underlying clustering key because Elasticsearch cannot create a secondary index on the partition key and there is no other indexed columns. Example :

```
$curl -XPUT "$NODE:9200/foo/bar/1?pretty" -d'{}'
{
  "_index" : "foo",
  "_type" : "bar",
  "_id" : "1",
  "_version" : 1,
  "_shards" : {
    "total" : 1,
    "successful" : 1,

```

```
    "failed" : 0
  },
  "created" : true
}
```

The underlying cassandra table *foo.bar* has only a primary key column with no secondary index. So, search operations won't return any result.

```
cqlsh> desc KEYSPACE foo ;

CREATE KEYSPACE foo WITH replication = {'class': 'NetworkTopologyStrategy', 'DC1': '1
↪'} AND durable_writes = true;

CREATE TABLE foo.bar (
  "_id" text PRIMARY KEY
) WITH bloom_filter_fp_chance = 0.01
  AND caching = '{"keys":"ALL", "rows_per_partition":"NONE"}'
  AND comment = 'Auto-created by Elassandra'
  AND compaction = {'class': 'org.apache.cassandra.db.compaction.
↪SizeTieredCompactionStrategy'}
  AND compression = {'sstable_compression': 'org.apache.cassandra.io.compress.
↪LZ4Compressor'}
  AND dclocal_read_repair_chance = 0.1
  AND default_time_to_live = 0
  AND gc_grace_seconds = 864000
  AND max_index_interval = 2048
  AND memtable_flush_period_in_ms = 0
  AND min_index_interval = 128
  AND read_repair_chance = 0.0
  AND speculative_retry = '99.0PERCENTILE';

cqlsh> SELECT * FROM foo.bar ;

 _id
-----
  1

(1 rows)
```

To get the same behavior as Elasticsearch, just add a dummy field in your mapping.

9.3 Nested or Object types cannot be empty

Because Elasticsearch nested and object types are backed by a Cassandra User Defined Type, it requires at least one sub-field in the mapping.

9.4 Document version is meaningless

Elasticsearch's versioning system helps to cope with conflicts, but in a multi-master database like Apache Cassandra, versioning cannot ensure global consistency of compare-and-set operations.

In Elassandra, Elasticsearch version management is disabled by default, document version is not more indexed in lucene files and **document version is always 1**. This simplification improves write throughput and reduce the memory footprint by eliminating the in-memory version cache implemented in the Elasticsearch internal lucene engine.

If you want to keep the Elasticsearch internal lucene file format including a version number for each document, you should create your index with `index.version_less_engine` set to *false* like this :

```
$curl -XPUT "$NODE:9200/twitter/" -d'{
  "settings":{ "index.version_less_engine":false } }
}'
```

Finally, if you need to avoid conflicts on write operations, you should use Cassandra [lightweight transactions](#) (or PAXOS transaction). Such lightweight transactions is also used when updating the Elassandra mapping or when indexing a document with `op_type=create`, but of course, it comes with a network cost.

9.5 Index and type names

Because cassandra does not support special caraters in keyspace and table names, Elassandra automatically replaces dot (.) and dash (-) characters by underscore (_) in index and type names to create underlying Cassandra keyspaces and tables. When such a modification occurs, Elassandra keeps this change in memory to correctly convert keyspace/table to index/type.

Moreover, Cassandra table names are limited to 48 caraters, so Elasticsearch type names are also limited to 48 characters.

9.6 Column names

For Elasticsearch, field mapping is unique in an index. So, two columns having the same name, indexed in an index, should have the same CQL type and share the same Elasticsearch mapping.

9.7 Null values

To be able to search for null values, Elasticsearch can replace null by a default value (see <https://www.elastic.co/guide/en/elasticsearch/reference/2.4/null-value.html>). In Elasticsearch, an empty array is not a null value, whereas in Cassandra, an empty array is stored as null and replaced by the default null value at index time.

9.8 Elasticsearch unsupported features

- Tribe node allows to query multiple Elasticsearch clusters. This feature is not currently supported by Elassandra.
- Elasticsearch snapshot and restore operations are disabled (See Elassandra backup and restore in operations).
- Elasticsearch Ingest node is not supported (Use the cassandra driver to safely ingest your data).
- Elasticsearch percolator, reindex and shrink API are not supported.
- `copy_to` is not supported.
- `range` field is not supported.

9.9 Cassandra limitations

- Elassandra only supports the murmur3 partitioner.

- The thrift protocol is supported only for read operations.
- Elassandra synchronously indexes rows into Elasticsearch. This may increase the write duration, particularly when indexing complex documents like [GeoShape](#), so Cassandra `write_request_timeout_in_ms` is set to 5 seconds (Cassandra default is 2000ms, see [Cassandra config](#))
- In order to avoid concurrent mapping or persistent cluster settings updates, Elassandra plays a PAXOS transaction that requires QUORUM available nodes for the keyspace `elastic_admin` to succeed. So it is recommended to have at least 3 nodes in 3 distinct racks (A 2 nodes datacenter won't accept any mapping update when a node is unavailable).
- CQL3 **TRUNCATE** on a Cassandra table deletes all associated Elasticsearch documents by playing a `delete_by_query` where `_type = <table_name>`. Of course, such a `delete_by_query` comes with a performance cost and won't notify `IndexingOperationListeners` for `preDelete` and `postDelete` events if used in an Elasticsearch plugin.

CHAPTER 10

Indices and tables

- `genindex`
- `modindex`
- `search`