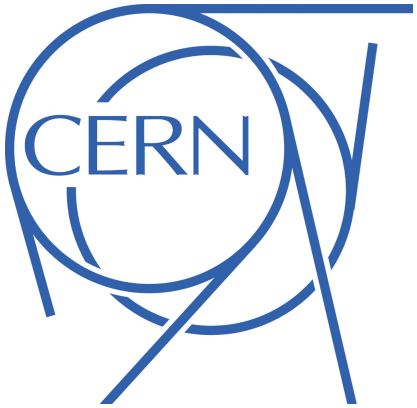


CERN DDM Lab Summer Student Report

Extending Rucio with Elasticsearch

Chuzhong PAN
Supervisor: Vincent Garonne, Cedric Serfon

August 24, 2013

Abstract

In this report, we talk about Elasticsearch as a candidate for the Query Engine backend for the next generation of Distributed Data Management system for Atlas experiment. This report focuses on the technical detail of Elasticsearch and provided code example on how to use it.

Contents

1 Main Goal: Rucio Project

The **Rucio project** is the new version of ATLAS Distributed Data Management (DDM) system services for allowing the ATLAS collaboration to manage the large volumes of data, both taken by the detector as well as generated or derived, in the ATLAS distributed computing system. Rucio uses to manage accounts, files, datasets and distibuted storage systems. One important aspect of this system is the capability of discovery data based on some metadata attributes like data type or physic run. With the predecessor system, DQ2, this represents a load of 1 million requests per day coming from physicists.

Figure 1: Rucio Project

My project is to explore open source and cloud computing solutions to offer fast and highly scalable search functionality into Rucio. To do that, we have 4 candidate engines, namely: **Elasticsearch, Lucene, Amazon CloudSearch,** and **Google Cloud Storage**.

In this report, we mainly focuse on Elasticsearch as the most promising backend engine for Rucio.

2 Introduction

Elasticsearch is a distributed, RESTful, free and open source search server based on Apache Lucene. It is developed by Shay Banon and is released under the terms of the Apache License. ElasticSearch is developed in Java[1]. It is powerful and easy to use. So we are trying to compare its performance in metadata query with the traditional database system, namely Oracle. We can sum up its features as:

Advantages

1. Distributed. No separate project required. Replicas are near real-time too, which is called "Push replication"
2. Fully supports the near real-time search of Apache Lucene.
3. Handling multitenancy is not a special configuration, whereas a more advanced setup is necessary with Solr.
4. Introduces the concept of the Gateway, which makes full backups easier.

Disadvantages

1. Only one main developer. This has to be mitigated since there is now the company ElasticSearch which has a large development team and provides commercial support.

3 Installation

3.1 Download Elasticsearch

Elasticsearch contains two main parts: the server and the client. There is only one version of the server, which is the java version and can be downloaded from:`http://www.elasticsearch.org/download/`. But there are a lot of versions of client (see the list of versions here: `http://www.elasticsearch.org/guide/clients/`). For our use, the suggestion is to use the python client, known as pyes, which can be downloaded from:`http://pypi.python.org/pypi/pyes/`

3.2 Install Server

Make sure you have Java Runtime Environment 6 or above. If you don't have it, download and install JDK from here: `http://www.oracle.com/technetwork/java/javase/downloads/index.html`.

Unzip your Elasticsearch Server package, run @bin/elasticsearch -f@ on unix, or @bin/elasticsearch.bat@ on windows. If it shows something like:

```
[2013-07-30 14:50:17,179][INFO ][node][Karima Shapandar] {0.90.2}[2783]: initializing
[2013-07-30 14:50:17,184][INFO ][plugins][Karima Shapandar] loaded [], sites []
[2013-07-30 14:50:18,894][INFO ][node ] [Karima Shapandar] {0.90.2}[2783]: initialized
[2013-07-30 14:50:18,895][INFO ][node ] [Karima Shapandar] {0.90.2}[2783]: starting
[2013-07-30 14:50:18,971][INFO ][transport ] [Karima Shapandar] bound\_address {inet
    [/0:0:0:0:0:0:0:0:9300]}, publish\_address {inet[/128.141.56.183:9300]}
[2013-07-30 14:50:21,999][INFO ][cluster.service ] [Karima Shapandar] new\_master [
    Karima Shapandar][h2SJMQ8\_T7G8kRgjYCqUXQ][inet[/128.141.56.183:9300]], reason: zen
    -disco-join (elected\_as\_master)
[2013-07-30 14:50:22,018][INFO ][discovery ] [Karima Shapandar] elasticsearch/h2SJMQ8\
    _T7G8kRgjYCqUXQ
[2013-07-30 14:50:22,028][INFO ][http ] [Karima Shapandar] bound\_address {inet
    [/0:0:0:0:0:0:0:0:9200]}, publish\_address {inet[/128.141.56.183:9200]}
[2013-07-30 14:50:22,028][INFO ][node ] [Karima Shapandar] {0.90.2}[2783]: started
[2013-07-30 14:50:22,061][INFO ][gateway ] [Karima Shapandar] recovered [0] indices
    into cluster\_state
```

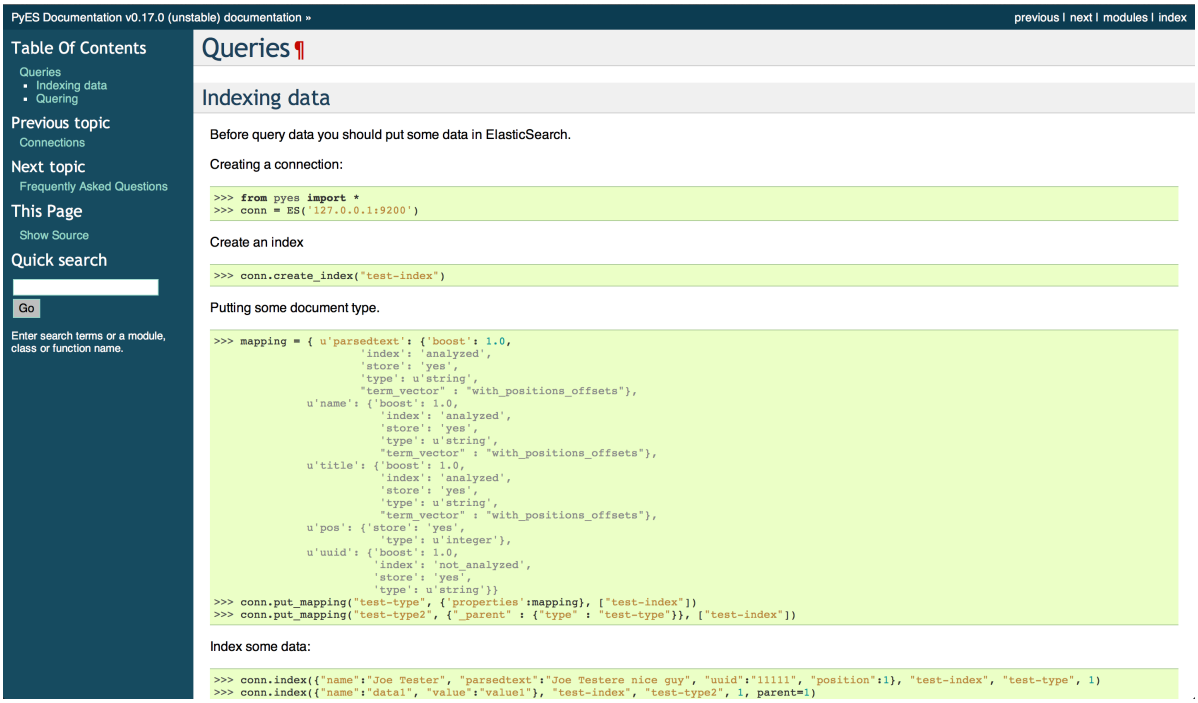
Then you have successfully launched the elasticsearch server.

3.3 Client

First, Make sure you have python 2.7 or above installed on your system. If not, download and install it from: `http://www.python.org/download/`.

Second, untar the pyes package and navigate into it. Run "python setup.py build" and "python setup.py install" as root.

Finally, open a new python terminal and type "*from pyes import* $*$". If there is no errors poping up, you have successfully installed the python elasticsearch client. The online Documentation can be found at: `http://pythonhosted.org/pyes/` which looks like:



Elasticsearch Python Api Documentation.

4 Using Elasticsearch

4.1 Inserting Data

Make sure elasticsearch server is up and running and your firewall allows incoming connection on port 9200. Open a new python terminal, type in the code shown below to create a connection.

```
from pyes import *
conn = ES('127.0.0.1:9200')
```

Then, create an index:

```
conn.create\_index("metadata")
```

An index is just like a directory. For example, after creating the index, we can use curl to get the current status. You should be able to open this link: `http://localhost:9200/metadata/_status`. It'll show a lot of useful information about the search engine.

After creating the index, we can now insert data into elasticsearch. Firstly, we need to "tell" elasticsearch what our data looks like. This process is called mapping.

```
mapping = { u'SCOPE': {'boost': 1.0,
                       'index': 'analyzed',
                       'store': 'yes',
                       'type': u'string',
                       "term\_vector" : "with\_positions\_offsets"},
            u'STREAM\_NAME': {'boost': 1.0,
                              'index': 'analyzed',
                              'store': 'yes',
                              'type': u'string',
                              "term\_vector" : "with\_positions\_offsets"},
            u'DATATYPE': {'boost': 1.0,
                          'index': 'analyzed',
                          'store': 'yes',
                          'type': u'string',
                          "term\_vector" : "with\_positions\_offsets"},
            u'PROD\_STEP': {'boost': 1.0,
                            'store': 'yes',
                            'type': u'string',
                            'index': 'analyzed',
                            "term\_vector" : "with\_positions\_offsets"},
            u'NAME': {'boost': 1.0,
                      'index': 'analyzed',
                      'store': 'yes',
                      'type': u'string',
                      "term\_vector" : "with\_positions\_offsets"},
            u'PROJECT': {'boost':1.0,
                         'index': 'analyzed',
                         'type': u'string',
                         'store': 'yes',
                         "term\_vector" : "with\_positions\_offsets"},
            u'VERSION': {'boost':1.0,
                         'index': 'analyzed',
                         'type': u'string',
                         'store': 'yes',
                         "term\_vector" : "with\_positions\_offsets"},
            u'CAMPAIGN': {'boost':1.0,
                          'index': 'analyzed',
                          'type': u'string',
                          'store': 'yes',
                          "term\_vector" : "with\_positions\_offsets"},
            u'RUN\_NUMBER': {'boost':1.0,
                             'index': 'analyzed',
                             'type': u'string',
                             'store': 'yes',
                             "term\_vector" : "with\_positions\_offsets"}}

conn.put\_mapping("maintype", {'properties':mapping}, ["metadata"])
```

Now, we can insert our data. The metadata we are dealing with looks like 4.1 without the line:

SCOPE	NAME	PROJECT	DATATYPE	RUN_NUMBER	STREAM_NAME	PROD_STEP	VERSION	CAMPAIGN
data12_8TeV	data12_8TeV.0089401.physics_IDCosmic.merge.HIST.5589	data12_8TeV	HIST	89401	physics_IDCosmic	merge	5589	(null)
data12_8TeV	data12_8TeV.0089591.physics_ZeroBias.merge.HIST.6399	data12_8TeV	HIST	89591	physics_ZeroBias	merge	6399	(null)
data12_8TeV	data12_8TeV.0304522.physics_JetTauEtmiss.merge.DESDM_RPVLL.719	data12_8TeV	DESDM_RPVLL	304522	physics_JetTauEtmiss	merge	719	(null)
data12_8TeV	data12_8TeV.0539601.physics_Background.merge.TAG.1251	data12_8TeV		539601	physics_Background	merge	1251	(null)
data12_8TeV	data12_8TeV.0627671.physics_JetCalibDelayed.recon.log.723	data12_8TeV	log	627671	physics_JetCalibDelayed	recon	723	(null)
data12_8TeV	data12_8TeV.0778304.debugrec_hltacc.merge.NTUP_FASTMON.5612	data12_8TeV	NTUP_FASTMON	778304	debugrec_hltacc	merge	5612	(null)
data12_8TeV	data12_8TeV.0685714.physics_MinBias.merge.HIST.2908	data12_8TeV	HIST	685714	physics_MinBias	merge	2908	(null)
data12_8TeV	data12_8TeV.0067187.calibration_PixelBeam.merge.TAG.2501	data12_8TeV	TAG	67187	calibration_PixelBeam	merge	2501	(null)
data12_8TeV	data12_8TeV.0845344.physics_JetTauEtmiss.merge.DESD_CALJET.684	data12_8TeV	DESD_CALJET	845344	physics_JetTauEtmiss	merge	684	(null)
data12_8TeV	data12_8TeV.0471321.calibration_PixelBeam.merge.AOD.5420	data12_8TeV	AOD	471321	calibration_PixelBeam	merge	5420	(null)
data12_8TeV	data12_8TeV.0791846.physics_HadDelayed.recon.DESD_CALJET.3234	data12_8TeV	DESD_CALJET	791846	physics_HadDelayed	recon	3234	(null)
data12_8TeV	data12_8TeV.0744338.physics_CosmicCalo.merge.TAG.5645	data12_8TeV	TAG	744338	physics_CosmicCalo	merge	5645	(null)
data12_8TeV	data12_8TeV.0828343.physics_Muons.merge.log.2436	data12_8TeV	log	828343	physics_Muons	merge	2436	(null)
data12_8TeV	data12_8TeV.0662874.physics_Muons.recon.DRAW_ZMUMU.9957	data12_8TeV	DRAW_ZMUMU	662874	physics_Muons	recon	9957	(null)
data12_8TeV	data12_8TeV.0999142.physics_Egamma.merge.DESD_PHOJET.3267	data12_8TeV	DESD_PHOJET	999142	physics_Egamma	merge	3267	(null)
data12_8TeV	data12_8TeV.0624711.physics_CosmicCalo.recon.HIST.8331	data12_8TeV	HIST	624711	physics_CosmicCalo	recon	8331	(null)
data12_8TeV	data12_8TeV.0687134.physics_CosmicCalo.recon.ESD.315	data12_8TeV	ESD	687134	physics_CosmicCalo	recon	315	(null)
data12_8TeV	data12_8TeV.0131885.debug_all.daq.RAW.5158	data12_8TeV	RAW	131885	debug_all	daq	5158	(null)

The syntax is regular and thus easy to process. In this case, the metadata is stored in a file named "dumpMetadata". We can insert them like :

```
with open("dumpMetadata.txt") as f : content=f.readlines();

for item in range(1,len(content)-1):
    conn.index({"NAME":content[item].split()[1],"SCOPE":content[item].
        split()[0],"STREAM_NAME":content[item].split()[5],"DATATYPE":
        content[item].split()[3],"PROD_STEP":content[item].split()[6],"
        PROJECT":content[item].split()[2],"VERSION":content[item].split()
        [7],"CAMPAIGN":content[item].split()[7],"RUN_NUMBER":content[item].
        split()[4]}, "metadata", "maintype");

conn.default_indices=["metadata"]
conn.refresh()
```

We have finished inserting all the metadata by now. The following step is the most exciting part, data query.

4.2 Querying Data

When querying data, we use a Search object to carry our search conditons. There are more than 10 kinds of query, we focus mainly on 2 kinds:TermQuery and WildcardQuery.

4.2.1 TermQuery

TermQuery is the most straightforward kind of query. It is a case-insensitive, full-text match query. An example of TermQuery is as follows.

```
q = TermQuery("RUN_NUMBER", "89401")
len(conn.search(query = q))
```

Be careful, the first parameter of TermQuery is the "mapping name" and must be **exactly the same** with those defined in your mapping. The second parameter is the

full-text of the item you search for in **lowercase string**. Do not use any upper case in the second parameter, or it'll refuse to work.

4.2.2 WildcardQuery

WildcardQuery is the most flexible kind of query. It partially allows for searching with regular expression. You can match any string with asterisk "*" or any single character with questionmark "?". But no more advanced regular expression is supported. We should also use lower case string in the second parameter and the exact mapping name in the first parameter. Here is a example.

```
q = WildcardQuery("RUN_NUMBER", "894*")
len(conn.search(query = q))
```

5 A Brief Introduction to Another Engine: Lucene

Lucene is a free/open source information retrieval software library, originally created in Java. Actually, Elasiticsearch is developed based on Lucene. We can sum up its features as follows.

Advantages

1. Scalable, High-Performance Indexing.
2. Designed for indexing files in the file system.
3. Many powerful query types: phrase, wildcard, proximity, range queries etc.
4. Configurable storage engine (codecs)

Disadvantages

1. Not easy to install and test.
2. Not stable on non-linux platforms.

6 Summary

The elasticsearch api version of DIDCore and DIDApi have been inplemented and tested. The future work is to inplement the DIDClient version and integrate it into rucio.

I had a wonderful research experience. It's definitely my **Best Summer Ever!**

References

[1] http://en.wikipedia.org/wiki/ElasticSearch

[2] http://pythonhosted.org/pyes/index.html

[3] http://rucio.cern.ch