HOW TO WRITE APACHE SPARK DATA TO ELASTICSEARCH USING PYTHON

Here we explain how to write Apache Spark data to ElasticSearch (ES) using Python. We will write Apache log data into ES.

This topic is made complicated, because of all the bad, convoluted examples on the internet. But here, we make it easy.

One complicating factor is that Spark provides native support for writing to ElasticSearch in Scala and Java but not Python. For you need to download ES-Hadoop, which is written by ElasticSearch, available here.

You then bring that into scope and make it available to pyspark like this:

```
pyspark --jars elasticsearch-hadoop-6.4.1.jar
```

Set PySpark to use Python 3 like this:

```
export PYSPARK_PYTHON=/usr/bin/python3
```

The key to understanding writing to ElasticSearch is that, while ES is a JSON database, it does have one requirement. The data has to be in this format:

```
{ "id: { the rest of your json}}
```

Below we show how to make that transformation.

At the bottom is the complete code and it is online here. Here we explain it in sections:

(This article is part of our ElasticSearch Guide. Use the right-hand menu to navigate.)

Parsing Apache Log Files

We read an Apache log into a Spark RDD. We then write a **parse()** function to read each string into into regular expression groups, pick the fields we want, and pass it back as a dictionary:.

```
rdd = sc.textFile("/home/ubuntu/walker/apache_logs")

regex='^(\S+) (\S+) (\S+) \+\s\d{4})\] "(\S+)\s?(\S+)?\s?(\S+)?" (\d{3}|-)
(\d+|-)\s?"?(*)"?\s?"?(*)?"?$'

p=re.compile(regex)

def parse(str):
    s=p.match(str)
    d = {}
    d=s.group(1)
    d=s.group(4)
    d=s.group(5)
    d=s.group(6)
    return d
```

In other words, when we first read the text file logs into an RDD it looks like this:

```
 "GET /presentations/logstash-monitorama-2013/images/kibana-search.png
HTTP/1.1" 200 203023
"http://semicomplete.com/presentations/logstash-monitorama-2013/"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_1) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/32.0.1700.77 Safari/537.36"']
```

Then we use the rdd **map()** function to pass each line into the **parse()** function to yield this.

```
rdd2 = rdd.map(parse)

rdd2.take(1)
```

Now, that looks like JSON, but it's not JSON yet. We will use **json.dumps**, which per the technical description in the Python documentation will "serialize obj as a JSON formatted stream."

We also add a ID. In the ES configuration below we tell ES what field will be the unique document identifier: **"es.mapping.id": "doc_id"**.

We calculate a SHA digest over the whole JSON document first to create that ID as a unique number.

The results are returned like this. You can see that the ID is a very long SHA number in front follow by the JSON.

```
rdd3.take(1)
```

Now we specify the ElasticSearch configuration. The important items to note are:

"es.resource" : 'walker/apache'	"walker" is the index and "apache" is the type. The whole thing together is often called "the index."
"es.mapping.id": "doc_id"	Here we tell ES which document to use as the document ID, which is the same as saying the **_id** field.

The rest of the fields are self explanatory.

Then we use the **saveAsNewAPIHadoopFile()** method to save the RDD to ES. There is nothing study there as the syntax is always the same for ES, so there is no need to understand all the pieces of that.

```
es_write_conf = {
        "es.nodes" : "localhost",
        "es.port" : "9200",
        "es.resource" : 'walker/apache',
        "es.input.json": "yes",
        "es.mapping.id": "doc_id"
    }
rdd3.saveAsNewAPIHadoopFile(
        path='-',
     outputFormatClass="org.elasticsearch.hadoop.mr.EsOutputFormat",
keyClass="org.apache.hadoop.io.NullWritable",
        valueClass="org.elasticsearch.hadoop.mr.LinkedMapWritable",
        conf=es_write_conf)

rdd3 = rdd2.map(addID)

def addId(data):
    j=json.dumps(data).encode('ascii', 'ignore')
    data = hashlib.sha224(j).hexdigest()
    return (data, json.dumps(data))
```

Now we can query ES from the command line and look at one document:

```
curl http://localhost:9200s/walker/apache/_search?pretty=true&?q=*


{
        "_index" : "walker",
        "_type" : "apache",
        "_id" : "227e977849bfd5f8d1fca69b04f7a766560745c6cb3712c106d590c2",
        "_score" : 1.0,
```

```
        "_source" : {
          "date" : "17/May/2015:10:05:32 +0000",
          "ip" : "91.177.205.119",
          "operation" : "GET",
          "doc_id" :
"227e977849bfd5f8d1fca69b04f7a766560745c6cb3712c106d590c2",
          "uri" : "/favicon.ico"
        }
```

Here is the complete code:

```
import json
import hashlib
import re

def addId(data):
    j=json.dumps(data).encode('ascii', 'ignore')
    data = hashlib.sha224(j).hexdigest()
    return (data, json.dumps(data))

def parse(str):
    s=p.match(str)
    d = {}
    d=s.group(1)
    d=s.group(4)
    d=s.group(5)
    d=s.group(6)
    return d

regex='^(\S+) (\S+) (\S+) \+\s\d{4})\] "(\S+)\s?(\S+)?\s?(\S+)?" (\d{3}|-)
(\d+|-)\s?"?(*)"?\s?"?(*)?"?$'

p=re.compile(regex)

rdd = sc.textFile("/home/ubuntu/walker/apache_logs")

rdd2 = rdd.map(parse)

rdd3 = rdd2.map(addID)

es_write_conf = {
        "es.nodes" : "localhost",
        "es.port" : "9200",
        "es.resource" : 'walker/apache',
        "es.input.json": "yes",
        "es.mapping.id": "doc_id"
    }
rdd3.saveAsNewAPIHadoopFile(
```

```
        path='-',
     outputFormatClass="org.elasticsearch.hadoop.mr.EsOutputFormat",
keyClass="org.apache.hadoop.io.NullWritable",
        valueClass="org.elasticsearch.hadoop.mr.LinkedMapWritable",
        conf=es_write_conf)
```