SPARK ELASTICSEARCH HADOOP UPDATE AND UPSERT EXAMPLE AND EXPLANATION

Here we explain how to write Python to code to update an ElasticSearch document from an Apache Spark Dataframe and RDD.

There are few instructions on the internet. Those written by ElasticSearch are difficult to understand and offer no examples. So we make the simplest possible example here.

This code adds additional fields to an ElasticSearch (ES) JSON document. i.e. it updates the document. Spark has built-in native support for Scala and Java. But for Python you have to use the Elasticsearch-Hadoop connector, written by ElasticSearch. That makes this operation more complicated.

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

Code on Github

The code for this exercise is here:

Update ElasticSearch
Run code with spark-submit
Create Data

Prerequisites

- ES. Download the binary and do not use apt-get install as the version stored there is too old.

- Apache Spark.
- Hadoop-ElasticSearch jar file. When you download it from here, it will provide jars for various languages.

Add Data

First we need to add two data records to ES. The key is the notation: **school/doc/1**

Means to add this to the index **school**, type **doc**, with **id = 1**.

```
curl -XPUT --header 'Content-Type: application/json'
http://localhost:9200/school/doc/1 -d '{
   "school" : "Clemson"
}'

curl -XPUT --header 'Content-Type: application/json'
http://localhost:9200/schools/doc/2 -d '{
   "school" : "Harvard"
}'
```

Here is the code to run the python code below as a spark-submit job. You do not need this to step through the code one line at a time with pyspark.

```
#!/bin/bash

NUM_CORES=*
DRIVER_MEM=3g

JARS="/usr/share/spark/spark-2.3.2-bin-hadoop2.7/jars/elasticsearch-
hadoop-6.4.2/dist"

export SPARK_HOME="/usr/share/spark/spark-2.3.2-bin-hadoop2.7"
export PATH=$PATH:$SPARK_HOME/bin
export CODEPATH="/home/ubuntu/Documents/esearch"

cd $CODEPATH

$SPARK_HOME/bin/spark-submit --master local --driver-memory
$DRIVER_MEM --jars $JARS/elasticsearch-hadoop-6.4.2.jar
$CODEPATH/updateData.py
```

Code Explained

read.format opens a connection to ES. The important item to note is **"es.read.metadata", "true"**. We need the metadata as that provides the document _id that we will need to do the update operation

```
reader =
spark.read.format("org.elasticsearch.spark.sql").option("es.read.metadata",
"true").option("es.nodes.wan.only","true").option("es.port","9200").option("e
```

```
s.net.ssl","false").option("es.nodes", "http://localhost")
```

Next we read all the documents in the index **school**.

```
df = reader.load("school")
df.show()
```

Now we filter to find the one record that is equal to Harvard.

```
df.filter(df == "Harvard").show()
```

Now that the **df.filter()** operation works in situ. In other it does not return a new dataframe. Instead if operates on the current dataframe. You can see that below. Only Harvard is shown after the filter operation.

```
df.show()
+---------+-------+--------------------+
|  school|           _metadata|
+---------+-------+--------------------+
| Harvard| == "Harvard").show()
+---------+-------+--------------------+
|   school|           _metadata|
+---------+-------+--------------------+
| Harvard|
df2=df.withColumn("_id", lit(id))
```

Next we create the **esconf** object as a dictionary. In code examples on the internet you will see this as JSON. But the notation below is cleaner and easier to work with.

The import items to note are:

es.update.script.inline	**ctx._source.location** means to update or create a field called **location. ctx_source** is the ES object to do that.
es.update.script.params	**location:<Cambridge>** are the parameter values passed to the inline script **es.update.script.inline**. The **<>** means to write a literal. If we wanted to write a field value we would leave them off.
es.mapping.id	This tells ES to look in the dataframe for the **id** column and use that as the document ID **_id**. ES uses that to find the document we want to update in ES.
es.write.operation	**upsert** means to add the document if it does not exist, otherwise update it. **update** means to update it.

```
esconf={}
esconf = "_id"
```

```
esconf = "localhost"
esconf = "9200"
esconf = "ctx._source.location = params.location"
esconf = "location:"
esconf = "upsert"
```

This writes the data. The **esconf means to read the dictionary **escconf**. The **mode("append")** means to add the fields to the existing document.

```
df2.write.format("org.elasticsearch.spark.sql").options(**esconf).mode("appen
d").save("school/info")
```

Now we look up the document and notice that **location** field has been updated to **Cambridge**. Bunch of Ivy league snobs.

```
curl -X GET 'http://localhost:9chool/info/_search'

{"took":1,"timed_out":false,"_shards":{"total":5,"successful":5,"skipped":0,"
failed":0},"hits":{"total":1,"max_score":1.0,"hits":}}
```

Complete Code

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext
from pyspark.sql.functions import lit

conf = SparkConf().setAppName("updateSchools")
sc = SparkContext(conf=conf)
sc.setLogLevel("INFO")
spark = SQLContext(sc)

reader =
spark.read.format("org.elasticsearch.spark.sql").option("es.read.metadata",
"true").option("es.nodes.wan.only","true").option("es.port","9200").option("e
s.net.ssl","false").option("es.nodes", "http://localhost")

df = reader.load("school")
df.show()

df.filter(df == "Harvard").show()

r=df.rdd.collect()
id = r

df2=df.withColumn("_id", lit(id))

esconf={}
```

```
esconf = "_id"
esconf = "localhost"
esconf = "9200"
esconf = "ctx._source.location = params.location"
esconf = "location:"
esconf = "upsert"

df2.write.format("org.elasticsearch.spark.sql").options(**esconf).mode("appen
d").save("school/info")
```