USING ELASTICSEARCH WITH APACHE SPARK

ElasticSearch is a JSON database popular with log processing systems. For example, organizations often use ElasticSearch with logstash or filebeat to send web server logs, Windows events, Linux syslogs, and other data there. Then they use the Kibana web interface to query log events. All of this is important for cybersecurity, operations, etc.

Now, since Spark 2.1, Spark has included native ElasticSearch support, which they call Elasticsearch Hadoop. That means you can use Apache Pig and Hive to work with JSON documents ElasticSearch. ElasticSearch Spark is a connector that existed before 2.1 and is still supported. Here we show how to use ElasticSearch Spark.

These connectors means you can run analytics against ElasticSearch data. ElasticSearch by itself only supports Lucene Queries, meaning natural language queries. So you could write predictive and classification models to flag cybersecurity events or do other analysis, something that ES does not do by itself.

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

Installation ElasticSearch Spark

First you need ElasticSearch. Download a prebuilt version instead of using the Docker image. The downloaded version has authentication turned off, which saves us some steps.

```
wget
https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-5.5.2.tar.
gz
```

And then you need Apache Spark. Download that from here and unzip it.

Neither software requires configuration. And it is not necessary to start Spark to use it. Nor do you need to start Hadoop. You only need to start ElasticSearch. Run this command as a non-root user to do that:

```
/usr/share/elasticsearch/elasticsearch-5.5.2/bin/elasticsearch
```

Then download the ElasticSearch jar file:

```
wget
http://central.maven.org/maven2/org/elasticsearch/elasticsearch-spark-20_2.11
/5.5.2/elasticsearch-spark-20_2.11-5.5.2.jar
```

Now start the interactive Spark shell, supplying the jar file as a command line option:

```
spark-shell --jars /home/walker/Documents/jars/elasticsearch-
spark-20_2.11-5.5.2.jar
```

Now we can take the example program right from the ElasticSearch web site and create data in ElasticSearch. Copy this text into the spark-shell.

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.elasticsearch.spark._
val sc = new SparkContext(conf)
conf.set("es.index.auto.create", "true")
val numbers = Map("one" -> 1, "two" -> 2, "three" -> 3)
val airports = Map("arrival" -> "Otopeni", "SFO" -> "San Fran")
sc.makeRDD(Seq(numbers, airports)).saveToEs("spark/docs")
```

All this did was write an RDD to ElasticSearch. Now we can write a few simple lines of Scala to read that data from ES and make it an RDD again.

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.elasticsearch.spark._
import org.apache.spark.SparkConf
val RDD = sc.esRDD("spark/docs")
RDD.first()
```

That will return the lines below. Note that the RDD is a **Map**. It could also be a Scala Case Class or JavaBean. So those are the type of objects you can store there, which makes sense since the ES format is JSON.

```
RDD: org.apache.spark.rdd.RDD)] = ScalaEsRDD at RDD at AbstractEsRDD.scala:37
res4: (String, scala.collection.Map) = (AV5b8VamL-9DBQqpL3kV,Map(arrival ->
Otopeni, SFO -> San Fran))
```

Now we can query the ES index and see that the **spark** index is there:

```
curl 'localhost:9200/_cat/indices?v'
```

Responds:

```
health status index uuid                   pri rep docs.count docs.deleted
store.size pri.store.size
yellow open   spark n9aTT2HcTJ-RiRZ5Ct1Wiw   5   1          2            0
7.9kb          7.9kb
```