1. **DOCUMENT**

Creating Hive Tables From JSON Using Pyspark and Spark DataFrames

1. **WHAT IT’S FOR**

To convert dynamic JSON formatted-data into partitioned, parquet format tables stored in Hive

1. **LIMITATIONS/PROBLEMS ENCOUNTERED**

Nested JSON problem not yet addressed when saved as a Hive table – the script can only go first level deep or what is referred to as the column families

1. **ASSUMPTIONS**

Pyspark is installed and configured

I am using ipython notebook to build and test the individual elements inside the script

1. **PROCESS**

There are a couple of approaches to dealing with JSON in Hadoop. Both approaches use Spark as the data format conversion, from JSON to parquet.

The first and more straightforward approach is to convert the JSON data into parquet format and query the HDFS directory using Apache Drill, no need to build schemas. It is important to note though that Apache Drill can only query files in HDFS which have extensions (e.g. parquet, csv, json, etc).

* 1. Spark
     1. Be sure to add the following configurations when starting the pyspark ipython notebook, spark-shell or running the spark script

--conf spark.sql.parquet.mergeSchema=true \*

--conf spark.shuffle.service.enabled=true

--conf spark.dynamicAllocation.enabled=true

--conf spark.executor.memory=60g \*\*

--conf spark.dynamicAllocation.maxExecutors=15 \*\*

\* mergeSchema gives the Spark DataFrame the capability to create a general

schema from multiple JSON/parquet files

\*\* these properties can be tweaked to your requirements

* + 1. If the json file(s) is/are to be loaded as is and there is no need for data processing, use sqlContext.read.json(“<path to your directory or json file>”) to read the file in as a Spark DataFrame.

You can view the json schema by calling the printSchema() method.

Example:

Df = sqlContext.read.json(“”)

Df.printSchema()

You can also access the top level keys through the DataFrame columns attribute.

Example:

Df.columns

If the json file(s) need processing, use

sc.textFile(“<path to your directory or json file>”) to read the file in as a Spark RDD and you can do map/filter lambda functions before saving the RDD as a Spark DataFrame.

To save a RDD as a DataFrame, use sqlContext.jsonRDD(<RDDfile>).

Save the DataFrame in HDFS in parquet format.

Df.write.parquet(“<path in HDFS>”)

* 1. Apache Drill
     1. Drill queries

Drill uses SQL-like syntax to query any file or directory in a filesystem like HDFS. To query, simple do a regular SELECT on the directory.

Example:

SELECT \* FROM hdfs.`<path in HDFS>` LIMIT 10;

An advantage of this approach is the simplicity offered when saving and retrieving nested JSON data. To query nested JSON, create an alias for the file or directory and reference the nested JSON keys using dot notation.

Example:

SELECT tbl.level1key.level2key.level3key FROM hdfs.`<path in HDFS>` tbl LIMIT 10;

The second approach is to create 2 tables in Hive: staging and final. The staging table is an external table with columns identical to the Spark DataFrame columns and with the table location set to the parquet directory created in Spark. The final table is similar to the staging table but with added partition set and snappy compression.

* 1. Hive
     1. Be sure to add the following configurations when starting the hive shell or running the hive script

set mapreduce.map.memory.mb=10000; \*

set mapreduce.reduce.memory.mb=3120; \*

set PARQUET\_COMPRESSION\_CODEC=snappy;

set hive.cli.print.header=true;

set hive.exec.dynamic.partition.mode=nonstrict;

\*check if you have enough resources

* + 1. \*\*Create external table for generated parquet directory from Spark with column names from df.columns attribute.

CREATE EXTERNAL TABLE pq\_staging

(

column1 STRING,

column2 STRING

)

LOCATION ‘<path in HDFS’>;

Create internal partitioned snappy parquet table for generated parquet directory from Spark with column names from df.columns attribute.

CREATE TABLE pq\_full

(

column1 STRING,

column2 STRING

)

PARTITIONED BY (col1 datatype, col2 datatype, col3 datatype)

STORED AS PARQUET

TBLPROPERTIES ('parquet.compression'='SNAPPY');

\*\*Note: Cannot use Impala’s infer schema to create internal table because CREATE TABLE… LIKE needs a file and can’t process an entire directory

Load staging data into partition table

INSERT INTO pq\_full

PARTITION BY (col1, col2, col3)

SELECT column1, column2,

Col1,

Col2,

Col3

FROM pq\_staging;