Adam Rempter Follow
Feb 21, 2020 · 3 min read · Listen

Running Spark 3 with standalone Hive Metastore 3.0

Intro

Recently I have spent some time testing **Spark 3** Preview2 running "outside" Hadoop. I was checking mainly how to run spark jobs on Kubernetes like schedulers (as an alternative to Yarn) with S3 storage.
I used both plain K8S cluster and Openshift. There are already some nice articles about it…

The Second thing I checked was if I could use **Hive Metastore as a standalone** service. And here I was able to find some code snippets on how to achieve it, but I couldn't find the full how-to article so far…

So, I decided to summarize steps I did, to hopefully help others to speed up Spark testing with standalone Hive Metastore.

In this article I will:

- show how to run Hive Metastore as a docker service (with MariaDB in a separate container). I will not use embedded Derby for Hive Metastore
- I will also use Minio as S3 Storage to test storing data in external table

Environment setup:

First, we need to start above mentioned containers. I have created a helper **repository**, so the article can be shorter 😄
All details of how it has been setup can be checked in **this Dockerfile**.

1. Clone hive-metastore-docker repository

```
$ git clone https://github.com/arempter/hive-metastore-docker.git
```

2. Run docker-compose to build Hive Metastore docker image locally

```
$ docker-compose build
```

3. In the last step start all containers (MariaDB, Metastore, and Minio)

```
$ docker-compose up -d
```

***NOTE:** you may need to start MariaDB container first, depending on the environment, otherwise **schemaTool** may fail to connect*

The most important part really is enabling spark support for Hive and pointing spark to our local metastore:

```
val spark = SparkSession
 .builder()
 .appName("SparkHiveTest")
 .config("hive.metastore.uris", "thrift://localhost:9083")
 .config("spark.sql.warehouse.dir", warehouseLocation)
 .enableHiveSupport()
 .getOrCreate()
```

Then we create a table:

```
sql("CREATE EXTERNAL TABLE IF NOT EXISTS spark_tests.s3_table_1 (key INT, value STRING) STORED AS PARQUET
LOCATION 's3a://spark/s3_table_1'")
```

Or we can store the table in S3 bucket:

```
df
 .write.mode(SaveMode.Overwrite)
 .option("path","s3a://spark/s3_table_1")
 .saveAsTable("spark_tests.s3_table_1")
```

Spark submit

Let's finally run our spark application and test if metastore works!


```
$ bin/spark-submit — master local — class tests.SparkHiveTest $REPO_DIR/spark_hive_test/target/scala-
2.12/spark_hive_test_2.12-0.1.jar
```

NOTE**: Detailed info on setting up spark binaries can be found in **this repository

Spark job output

You should see the following logs in completed spark job:

```
20/02/21 10:44:18 INFO HiveClientImpl: Warehouse location for Hive client (version 2.3.6) is /tmp/spark-
warehouse
20/02/21 10:44:18 INFO metastore: Trying to connect to metastore with URI thrift://localhost:9083
20/02/21 10:44:18 INFO metastore: Opened a connection to metastore, current connections: 1
20/02/21 10:44:18 INFO metastore: Connected to metastore.
…
20/02/21 10:44:30 INFO HiveExternalCatalog: Persisting file based data source table
`spark_tests`.`s3_table_1` into Hive metastore in Hive compatible format.
```

Also let's check if the table has been stored in S3:

```
$ aws s3 --endpoint=http://localhost:9000 ls s3://spark/s3_table_1/
2020-02-21 11:02:21          0 _SUCCESS
2020-02-21 11:02:18        859 part-00000-e9365ee5-d5a2-4566-9479-97662f87805e-c000.snappy.parquet
```

After successfully running spark job, you can now see table definition in a hive:

```
hive> show create table spark_tests.s3_table_1;
OK
CREATE EXTERNAL TABLE `spark_tests.s3_table_1`(
  `_1` int,
  `_2` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
WITH SERDEPROPERTIES (
  'path'='s3a://spark/s3_table_1')
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3a://spark/s3_table_1'
TBLPROPERTIES (
  'spark.sql.create.version'='3.0.0-preview2',
  'spark.sql.sources.provider'='parquet',
  'spark.sql.sources.schema.numParts'='1',
  'spark.sql.sources.schema.part.0'='{"type":"struct","fields":
[{"name":"_1","type":"integer","nullable":true,"metadata":{}},
{"name":"_2","type":"string","nullable":true,"metadata":{}}]}',
  'transient_lastDdlTime'='1582278270')
Time taken: 0.038 seconds, Fetched: 19 row(s)
```

And as the last step you can use hive to query data:

```
hive> select * from spark_tests.s3_table_1 limit 5;
OK
1 elem_1
2 elem_2
3 elem_3
4 elem_4
5 elem_5
Time taken: 0.083 seconds, Fetched: 5 row(s)
```

Useful links:

- https://cwiki.apache.org/confluence/display/Hive/AdminManual+Metastore+3.0+Administration
- https://spark.apache.org/docs/3.0.0-preview/sql-data-sources-hive-tables.html