Greenplum-Spark Connector Examples Documentation

kong-yew,chan

Feb 06, 2019

Contents

CHAPTER 1

Overview

1.1 Pivotal Greenplum

The Pivotal Greenplum Database is an advanced, fully featured, open source data warehouse. It provides powerful and rapid analytics on petabyte scale data volumes. Uniquely geared toward big data analytics, Greenplum Database is powered by the world's most advanced cost-based query optimizer delivering high analytical query performance on large data volumes.

1.2 Pivotal Greenplum-Spark Connector

The Pivotal Greenplum Spark Connector provides high speed, parallel data transfer between Greenplum Database and Apache Spark clusters to support:

- Interactive data analysis
- In-memory analytics processing
- Batch ETL

1.3 Apache Spark

Apache Spark is a fast and general cluster computing system for Big Data. It provides high-level APIs in Scala, Java, Python, and R, and an optimized engine that supports general computation graphs for data analysis. It also supports a rich set of higher-level tools including Spark SQL for SQL and DataFrames, MLlib for machine learning, GraphX for graph processing, and Spark Streaming for stream processing.

References: - [Introduction](https://gitpitch.com/kongyew/greenplum-spark-connector) - [Greenplum-Spark connector docs](http://greenplum-spark.docs.pivotal.io/latest/index.html)

CHAPTER 2

Prerequisites

2.1 What you need to know

The tutorial assumes some basic familiarity with commandline prompt in a terminal.

You'll need to know basic knowledge about Pivotal Greenplum. and Apache Spark.

Git repository : 'https://github.com/kongyew/greenplum-spark-connector'.

Greenplum-Spark connector documentation: <http://greenplum-spark.docs.pivotal.io>

2.2 Software

The tutorial assumes that you're using a Unix-like system and docker.

If your system already has Docker and Docker-compose running that you've installed, you probably already have what you need and know what you need to know.

2.2.1 Docker-Compose

You'll need a reasonably up-to-date version of Docker-compose installed on your machine. 1.14.0 or newer should be fine.

2.2.2 Greenplum-Spark connector

Please download the Greenplum-Spark connector jar from Pivotal Network.

CHAPTER 3

Setup Greenplum and Spark

This page describes how to setup Greenpum and Spark dockers

3.1 Pre-requisites:

- docker compose
- Greenplum Spark connector
- Postgres JDBC driver - if you want to write data from Spark into Greenplum.

3.2 Using docker-compose

To create a standalone Greenplum cluster with the following command in the root directory. It builds a docker image with Pivotal Greenplum binaries and download some existing images such as Spark master and worker. Initially, it may take some time to download the docker image.

```
$ ./runDocker.sh -t usecase1 -c up
```

Creating network "usecase1_default" with the default driver Creating sparkmaster ... done Creating gpdbsne ... done Creating sparkworker ... done ... ...

The SparkUI will be running at *http://localhost:8081* with one worker listed.

3.3 Setup Greenplum with sample tables

Click on the section "Create database and table"

3.4 Connect to Spark master instance

1. Connect to the Spark master docker image

```
$ docker exec -it sparkmaster /bin/bash
```

3.5 Connect to Greenplum instance

1. Connect to the GPDB docker image

```
$ docker exec -it gpdbsne bin/bash
  root@master:/usr/spark-2.1.0#
```

CHAPTER 4

How to setup Greenplum database and table

This readme describes how to setup Greenplum database and table(s).

4.1 Pre-requisites:

- docker-compose.

4.2 Run the docker instances:

You can run spark and GPDB instances by using existing scripts.

```
$./runDocker.sh -t usecase1 -c up
```

4.3 Verify the docker instance is running:

Make sure the docker instances are running by running *docker ps*

```
$ docker ps
```

4.4 How to run the setupDB:

This setupDB.sh script automatically creates default database and table(s). The script is located under *<src>/data/scripts/setupDB.sh*.

1. Connect to the GPDB docker image The Greenplum DB cluster will be running with this instance name: *gpdbsne* with two segments. To access this docker instance, exec into a container:

```
$ docker exec -it gpdbsne bin/bash
```

2. Execute the command below to access the scripts folder under "/code/data"

```
[root@d632f535db87]# cd /code/data
```

3. Run *scripts/setupDB.sh*, in order to create a database and table.

```
[root@d632f535db87 data]# scripts/setupDB.sh
psql:./sample_table.sql:1: NOTICE:  table "basictable" does not exist, skipping
DROP TABLE
psql:./sample_table.sql:5: NOTICE:  CREATE TABLE will create implicit sequence
↪"basictable_id_seq" for serial column "basictable.id"
CREATE TABLE
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 9
INSERT 0 18
INSERT 0 36
INSERT 0 72
INSERT 0 144
INSERT 0 288
INSERT 0 576
INSERT 0 1152
INSERT 0 2304
INSERT 0 4608
```

4. Run the following psql command to verify database (basic_db) and table (basictable) are created correctly.

```
[root@d632f535db87 data]# psql -h localhost -U gpadmin -d basic_db -c "\dt"
           List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+---------
 public | basictable | table | gpadmin
(1 row)
```

```
[root@d632f535db87 data]# psql -h localhost -U gpadmin -d basic_db -c "select␣
↪count(*) from basictable"
 count
-------
  9216
(1 row)
```

CHAPTER 5

Reading data from Greenplum into Spark

In this example, we will describe how to configure Greenplum-Spark connector when you run Spark-shell. It assumes the database and table are already created.

1. Make sure you download greenplum-spark_2.11-x.x.jar from Pivotal Network.
2. Connect to the Spark master instance.

```
$ docker exec -it sparkmaster /bin/bash
```

3. Run the command to start a spark shell that loads Greenplum-Spark connector. This section assumes you have downloaded greenplum-spark_2.11.jar under the github repo with subfolder *scripts*. The root directory is mounted by the docker images under /code directory. You can also use scripts such as *scripts/download_postgresql.sh* and *scripts/download_greenplum-spark-connector.sh* to download required binaries.

Also, we included Postgresql, in order to write data from Spark into Greenplum. Greenplum-Spark connector will support write features in future release and support parallel data transfer that performs significantly better than JDBC driver.

```
    root@master:/usr/spark-2.1.0#GSC_JAR=$(ls /code/scripts/greenplum-spark_2.11-*.
↪jar)
    root@master:/usr/spark-2.1.0#POSTGRES_JAR=$(ls /code/scripts/postgresql-*.jar)
    root@master:/usr/spark-2.1.0#spark-shell --jars "${GSC_JAR},${POSTGRES_JAR}" --
↪driver-class-path ${POSTGRES_JAR}
    ...
    Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_112)
    Type in expressions to have them evaluated.
    Type :help for more information.
    scala>
```

4. Verify Greenplum-Spark driver is successfully loaded by Spark Shell. You can follow the example below to verify the Greenplum-Spark driver. The scala repl confirms the driver is accessible by returning *res0* result.

```
scala> Class.forName("io.pivotal.greenplum.spark.GreenplumRelationProvider")
res0: Class[_] = class io.pivotal.greenplum.spark.GreenplumRelationProvider
```

Verify JDBC driver is successfully loaded by Spark Shell. You can follow the example below to verify the JDBC driver. The scala repl confirms the driver is accessible by returning *res1* result.

```
scala> Class.forName("org.postgresql.Driver")
res1: Class[_] = class org.postgresql.Driver
```

5. By default, you can run the command below to retrieve data from Greenplum with a single data partition in Spark cluster. In order to paste the command, you need to type *:paste* in the scala environment and paste the code below, followed by *Ctrl-D*.

```
scala> :paste
// Entering paste mode (ctrl-D to finish)
// that gives an one-partition Dataset
val dataFrame = spark.read.format("io.pivotal.greenplum.spark.
↪GreenplumRelationProvider")
.option("dbtable", "basictable")
.option("url", "jdbc:postgresql://gpdbsne/basic_db")
.option("user", "gpadmin")
.option("password", "pivotal")
.option("driver", "org.postgresql.Driver")
.option("partitionColumn", "id")
.load()
// Exiting paste mode, now interpreting.
```

2. You can verify the Spark DataFrame by running these commands *dataFrame.printSchema* and *dataFrame.show()*.

```
scala> dataFrame.printSchema
root
 |-- id: integer (nullable = false)
 |-- value: string (nullable = true)
scala> dataFrame.show()
+---+--------+
| id|   value|
+---+--------+
|  1|   Alice|
|  3| Charlie|
|  5|     Jim|
|  7|    Jack|
|  9|     Zim|
| 15|     Jim|
| 11|     Bob|
| 13|     Eve|
| 17|Victoria|
| 25|Victoria|
| 27|   Alice|
| 29| Charlie|
| 31|     Zim|
| 19|   Alice|
| 21| Charlie|
| 23|     Jim|
| 33|     Jim|
| 35|     Eve|
| 43|Victoria|
| 45|   Alice|
+---+--------+
only showing top 20 rows
scala> dataFrame.filter(dataFrame("id") > 40).show()
```

(continues on next page)

(continued from previous page)

```
+---+--------+
| id|   value|
+---+--------+
| 41|     Jim|
| 43|    Jack|
| 45|     Zim|
| 47|   Alice|
| 49| Charlie|
| 51|     Jim|
| 53|    Jack|
| 55|     Bob|
| 57|     Eve|
| 59|    John|
| 61|Victoria|
| 63|     Zim|
| 65|     Bob|
| 67|     Eve|
| 69|    John|
| 71|Victoria|
| 73|     Bob|
| 75|   Alice|
| 77| Charlie|
| 79|     Jim|
+---+--------+
only showing top 20 rows
```

3. You create a temporary table to cache the results from Greenplum and using option to speed your in-memory processing in Spark cluster. Global temporary view. is tied to a system preserved database global_temp, and we must use the qualified name to refer it, e.g. SELECT * FROM global_temp.view1. Meanwhile, Temporary views in Spark SQL are session-scoped and will disappear if the session that creates it terminates.

```
scala>
// Register the DataFrame as a global temporary view
dataFrame.createGlobalTempView("tempdataFrame")
// Global temporary view is tied to a system preserved database `global_temp`
spark.sql("SELECT * FROM global_temp.tempdataFrame").show()
```

5.1 Conclusions

Greenplum-Spark connector uses Greenplum gpfdist protocol to parallelize data transfer between Greenplum and Spark clusters. Therefore, this connector provides better read throughput, compared to typical JDBC driver.

CHAPTER 6

Writing data from Spark into Greenplum

In this section, you can write data from Spark DataFrame into Greenplum table by using Greenplum-Spark connector.

1. Run the script under scripts/download_postgresql.sh to download postgresql jar to the directory 'scripts'.

```
    $ scripts/download_postgreql.sh
    ...
    HTTP request sent, awaiting response... 200 OK
    Length: 713037 (696K) [application/java-archive]
    Saving to: 'postgresql-42.1.4.jar'
    postgresql-42.1.4.jar          100
↪%[=================================================>] 696.33K   850KB/s    in 0.8s
    2017-09-24 20:59:25 (850 KB/s) - 'postgresql-42.1.4.jar' saved [713037/713037]
```

2. Make sure your spark shell is loaded the Postgresql jar.

3. Determine the number of records in the "basictable" table by using psql command.

```
$ docker exec -it docker_gpdb_1 /bin/bash
[root@d632f535db87 data]# psql -h localhost -U gpadmin -d basic_db -c "select␣
↪count(*) from basictable"
```

4. Configure JDBC URL and connection Properties and use DataFrame write operation to write data from Spark into Greenplum. You can use different write mode

```
scala> :paste
// Entering paste mode (ctrl-D to finish)
val jdbcUrl = s"jdbc:postgresql://docker_gpdb_1/basic_db?user=gpadmin&password=pivotal
↪"
val connectionProperties = new java.util.Properties()
dataFrame.write.mode("Append") .jdbc( url = jdbcUrl, table = "basictable",␣
↪connectionProperties = connectionProperties)
// Exiting paste mode, now interpreting.
```

5. Verify the write operation is successful by exec into GPDB container and run psql command-line. The total number records in the Greenplum table must be 2x of the original data.

```
$ docker exec -it docker_gpdb_1 /bin/bash
[root@d632f535db87 data]# psql -h localhost -U gpadmin -d basic_db -c "select␣
↪count(*) from basictable"
```

6. Next, you can write DataFrame data into an new Greenplum table via *append* mode.

```
scala>dataFrame.write.mode("Append") .jdbc( url = jdbcUrl, table = "NEWTable",␣
↪connectionProperties = connectionProperties)
```

7. Run psql commands to verify the new table with new records.

```
[root@d632f535db87 scripts]# psql -h localhost -U gpadmin -d basic_db -c "\dt"
List of relations
Schema |            Name              | Type  |  Owner
--------+------------------------------+-------+---------
public | basictable                   | table | gpadmin
public | newtable                     | table | gpadmin
public | spark_7ac1947b17a17725_0_41 | table | gpadmin
public | spark_7ac1947b17a17725_0_42 | table | gpadmin
(4 rows)
```

CHAPTER 7

Writing data from Spark into Greenplum via JDBC

In this section, you can write data from Spark DataFrame into Greenplum table by using JDBC driver.

1. Run the script under scripts/download_postgresql.sh to download postgresql jar to the directory 'scripts'.

```
    $ scripts/download_postgreql.sh
    ...
    HTTP request sent, awaiting response... 200 OK
    Length: 713037 (696K) [application/java-archive]
    Saving to: 'postgresql-42.1.4.jar'
    postgresql-42.1.4.jar         100
↪%[=================================================>] 696.33K   850KB/s    in 0.8s
    2017-09-24 20:59:25 (850 KB/s) - 'postgresql-42.1.4.jar' saved [713037/713037]
```

2. Make sure your spark shell is loaded the Postgresql jar.

3. Determine the number of records in the "basictable" table by using psql command.

```
$ docker exec -it docker_gpdb_1 /bin/bash
[root@d632f535db87 data]# psql -h localhost -U gpadmin -d basic_db -c "select␣
↪count(*) from basictable"
```

4. Configure JDBC URL and connection Properties and use DataFrame write operation to write data from Spark into Greenplum. You can use different write mode

```
scala> :paste
// Entering paste mode (ctrl-D to finish)
val jdbcUrl = s"jdbc:postgresql://docker_gpdb_1/basic_db?user=gpadmin&password=pivotal
↪"
val connectionProperties = new java.util.Properties()
dataFrame.write.mode("Append") .jdbc( url = jdbcUrl, table = "basictable",␣
↪connectionProperties = connectionProperties)
// Exiting paste mode, now interpreting.
```

5. Verify the write operation is successful by exec into GPDB container and run psql command-line. The total number records in the Greenplum table must be 2x of the original data.

```
$ docker exec -it docker_gpdb_1 /bin/bash
[root@d632f535db87 data]# psql -h localhost -U gpadmin -d basic_db -c "select␣
↪count(*) from basictable"
```

6. Next, you can write DataFrame data into an new Greenplum table via *append* mode.

```
scala>dataFrame.write.mode("Append") .jdbc( url = jdbcUrl, table = "NEWTable",␣
↪connectionProperties = connectionProperties)
```

7. Run psql commands to verify the new table with new records.

```
[root@d632f535db87 scripts]# psql -h localhost -U gpadmin -d basic_db -c "\dt"
List of relations
Schema |            Name              | Type  |  Owner
--------+------------------------------+-------+---------
public | basictable                   | table | gpadmin
public | newtable                     | table | gpadmin
public | spark_7ac1947b17a17725_0_41 | table | gpadmin
public | spark_7ac1947b17a17725_0_42 | table | gpadmin
(4 rows)
```

CHAPTER 8

Using PySpark

In this example, we will describe how to run PySpark-shell.

8.1 How to read data from Greenplum into Spark

1. Connect to the Spark master docker image

```
$ docker exec -it sparkmaster /bin/bash
```

2. Execute the command below to run pySpark

```
root@master:/usr/spark-2.1.0#GSC_JAR=$(ls /code/greenplum-spark_2.11-*.jar)
root@master:/usr/spark-2.1.0#pyspark --jars "${GSC_JAR}"
Python 3.4.2 (default, Oct  8 2014, 10:45:20)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use␣
↪setLogLevel(newLevel).
17/09/23 18:51:37 WARN util.NativeCodeLoader: Unable to load native-hadoop library␣
↪for your platform... using builtin-java classes where applicable
Welcome to
...
Using Python version 3.4.2 (default, Oct  8 2014 10:45:20)
SparkSession available as 'spark'.
```

3. Verfiy the Greenplum-Spark connector is loaded by pySpark

   Use the command *sc.getConf().getAll()* to verify spark.repl.local.jars is referring to Greenplum-Spark connector jar.

4. To load a DataFrame from a Greenplum table in PySpark

```
>>>source_df = sqlContext.read.format('io.pivotal.greenplum.spark.
↪GreenplumRelationProvider').options(
          url='jdbc:postgresql://docker_gpdb_1/basic_db',
          dbtable='basictable',
          user='gpadmin',
          password='pivotal',
          driver='org.postgresql.Driver',
          partitionColumn='id').load()
```

5. Verify source dataframe by running these commands

```
>>> source_df.count()
18432
>>> source_df.printSchema()
root
 |-- id: integer (nullable = false)
 |-- value: string (nullable = true)
>>> source_df.show()
+---+--------+
| id|   value|
+---+--------+
|  1|   Alice|
|  3| Charlie|
|  5|     Jim|
|  7|    Jack|
|  9|     Zim|
| 13|    John|
| 11|   Alice|
| 15| Charlie|
| 17|    Jack|
| 19|   Alice|
| 21|     Jim|
| 23|     Zim|
| 25|   Alice|
| 27|    Jack|
| 29|     Eve|
| 31|Victoria|
| 33|     Eve|
| 35|     Jim|
| 37|     Bob|
| 39|     Eve|
+---+--------+
only showing top 20 rows
```

8.2 How to write data from Spark DataFrame into Greenplum

In this section, you can write data from Spark DataFrame into Greenplum table.

1. Determine the number of records in the “basictable” table by using psql command.

```
$ docker exec -it gpdbsne /bin/bash
[root@d632f535db87 data]# psql -h localhost -U gpadmin -d basic_db -c "select␣
↪count(*) from basictable"
```

2. Configure JDBC URL and connection Properties and use DataFrame write operation to write data from Spark into Greenplum.

```
source_df.write.format('jdbc').options(
    url='jdbc:postgresql://gpdbsne/basic_db',
    dbtable='basictable',
    user='gpadmin',
    password='pivotal',
    driver='org.postgresql.Driver').mode('append').save()
```

3. Verify the write operation is successful by exec into GPDB container and run psql command-line. The total number records in the Greenplum table must be 2x of the original data.

```
$ docker exec -it gpdbsne /bin/bash
[root@d632f535db87 data]# psql -h localhost -U gpadmin -d basic_db -c "select␣
↪count(*) from basictable"
 count
-------
`36864`
(1 row)
```

CHAPTER 9

About Greenplum-Spark examples

This documentation describes examples with Pivotal Greenplum and Spark by using GPDB-Spark connector and Postgres JDBC driver.

9.1 About the Author

Kong-Yew,Chan has more than 15 years of experience in product management & development for enterprise and consumer applications. Currently, he is working as Product Manager @Pivotal to develop data integrations for Greenplum and GemFire platforms. He completed BSc Computer Engineering degree at the prestigious Nanyang Technological University and MBA at Babson.

9.2 Question:

If you have a problem with any aspect of this documentations, you can contact me at kochan@pivotal.io, and I will do my best to address the problem.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search