MariaDB ColumnStore PySpark API Usage Documentation

Release 1.2.3-3d1ab30

MariaDB Corporation

Mar 07, 2019

CONTENTS

CHAPTER

ONE

LICENSING

1.1 Documentation Content

The Spark mcsapi documentation is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

1.2 MariaDB ColumnStore PySpark API

The MariaDB ColumnStore PySpark API (pyspark_mcsapi) is licensed under the GNU Lesser General Public License, version 2.1.

CHAPTER

TWO

VERSION HISTORY

This is a version history of PySpark API interface changes. It does not include internal fixes and changes.

Version	Changes
1.2.3	• Documentation added

CHAPTER

THREE

USING MCSAPI WITH PYSPARK

3.1 Usage Introduction

mcsapi for PySpark is built upon the Python bulk insert API (pymcsapi) and provides export functions to ingest PySpark DataFrames into MariaDB ColumnStore.

3.2 Installation and Configuration

This document describes the installation and configuration of MariaDB ColumnStore 1.2, Apache Spark 2.4.0, and mcsapi for PySpark in a dockerized lab environment. Production system installations need to follow the same steps. Installation and configuration commands and paths might change depending on your operating systems, software versions, and network setup.

3.2.1 Lab environment setup

The lab environment consists of:

- A multi node MariaDB ColumnStore 1.2 installation with 1 user module (UM) and 2 performance modules (PMs)
- A multi node Apache Spark 2.4 installation with 1 Spark driver and 2 Spark workers

It is defined through following docker-compose.yml configuration.

To start the lab environment download it and go to the folder containing the docker-compose.yml file. Then execute:

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

This will spin up the environment with six container.

3.2.2 Installation of mcsapi for PySpark

To utilize mcsapi for PySpark's functions you have to install it on the Spark master. Therefore, you first have to set up the regarding software repository via:

```
docker exec -it SPARK_MASTER bash #to get a shell in the docker container instance
apt-get update
apt-get install -y apt-transport-https dirmngr wget
echo "deb https://downloads.mariadb.com/MariaDB/mariadb-columnstore-api/latest/repo/
↪debian9 stretch main" > /etc/apt/sources.list.d/mariadb-columnstore-api.list
```

Then add the repository key and refresh the repositories via:

```
wget -qO - https://downloads.mariadb.com/MariaDB/mariadb-columnstore/MariaDB-
↪ColumnStore.gpg.key | apt-key add -
apt-get update
```

And finally install mcsapi for PySpark and its dependencies:

```
#apt-get install -y mariadb-columnstore-api-pyspark  # PySpark for Python 2.7
apt-get install -y mariadb-columnstore-api-pyspark3  # PySpark for Python 3
```

It is further advised to install the MySQL Python package on the Spark driver to be able to execute DDL.

```
#apt-get install -y python-pip             # For Python 2.7
#pip2 install mysql-connector==2.1.6       # For Python 2.7
apt-get install -y python3-pip             # For Python 3
pip3 install mysql-connector==2.1.6        # For Python 3
```

For other operating systems, please follow the dedicated installation document in our Knowledge Base.

3.2.3 Spark configuration

To configure Spark to use mcsapi for PySpark one more action needs to be executed.

mcsapi for PySpark needs information about the ColumnStore cluster to write data into. This information needs to be provided in form of a Columnstore.xml configuration file. This needs to be copied from ColumnStore's um1 node to the Spark master.

```
docker cp COLUMNSTORE_UM_1:/usr/local/mariadb/columnstore/etc/Columnstore.xml .
docker exec -it SPARK_MASTER mkdir -p /usr/local/mariadb/columnstore/etc
docker cp Columnstore.xml SPARK_MASTER:/usr/local/mariadb/columnstore/etc
```

More information about creating appropriate Columnstore.xml configuration files and Spark configuration changes can be found in our Knowledge Base.

3.2.4 Firewall setup

In production environments with installed firewalls you have to ensure that the Spark master and worker nodes can reach TCP port 3306 on the ColumnStore user modules, and TCP ports 8616, 8630, and 8800 on the ColumnStore performance modules. The lab environment is already fully configured, therefore there is nothing to do in this case.

3.2.5 Finishing note

Note that the configured Spark container aren't persistent. Once the container are stopped you have to install and configure mcsapi for PySpark again. You could use `docker commit` to save your changes. Feel free to check out our *Interactive test environments* if you want to tinker around further with mcsapi for PySpark.

3.3 A simple DataFrame export application

In this example we will export a synthetic DataFrame out of Spark into a non existing table in MariaDB ColumnStore. The full code for this can be found in the ExportDataFrame.py file in the mcsapi codebase.

First, all needed libraries are imported and the SparkContext is created.

Listing 1: ExportDataFrame.py

```
import columnStoreExporter
from pyspark.sql import SparkSession, Row
import mysql.connector as mariadb

#Create the spark session
spark = SparkSession.builder.appName("DataFrame export into MariaDB ColumnStore").
↪getOrCreate()
```

Second, an example DataFrame that is going to be exported into ColumnStore is created. The DataFrame consists of two columns. One containing numbers from 0-127 and the other containing its ASCII character representation.

Listing 2: ExportDataFrame.py

```
#Generate the DataFrame to export
df = spark.createDataFrame(spark.sparkContext.parallelize(range(0, 128)).map(lambda␣
↪i: Row(number=i, ASCII_representation= chr(i))))
```

Third, the JDBC connection necessary to execute DML statements to create the target table is set up.

Listing 3: ExportDataFrame.py

```
#JDBC connection parameter
user = "root"
host = "um1"
password = ""
```

Fourth, the target table "pyspark_export" is created in the database "test". Note that `ColumnStoreExporter.generateTableStatement` infers a suitable DML statement based on the DataFrame's structure.

Listing 4: ExportDataFrame.py

```
#Create the target table
createTableStatement = columnStoreExporter.generateTableStatement(df, "test",
↪"pyspark_export")
try:
    conn = mariadb.connect(user=user, database='', host=host, password=password)
    cursor = conn.cursor()
    cursor.execute("CREATE DATABASE IF NOT EXISTS test")
    cursor.execute(createTableStatement)

except mariadb.Error as err:
    print("Error during table creation: ", err)
finally:
    if cursor: cursor.close()
    if conn: conn.close()
```

Finally, the DataFrame gets imported into MariaDB ColumnStore by bypassing the SQL layer and injecting the data directly through MariaDB's Bulk Write SDK.

Listing 5: ExportDataFrame.py

```
47 #Export the DataFrame into ColumnStore
48 columnStoreExporter.export("test","pyspark_export",df)
49 spark.stop()
```

3.4 Application execution

To submit last section's sample application to your Spark setup you simply have to copy it to the Spark master and execute it though *spark-submit*.

```
docker cp ExportDataFrame.py SPARK_MASTER:/root
docker exec -it SPARK_MASTER spark-submit --master spark://master:7077 /root/
↪ExportDataFrame.py
```

3.5 Interactive test environments

Feel free to check out our interactive test environments for pymcsapi, javamcsapi, mcsapi for Spark, and mcsapi for PySpark.

- Zeppelin environment: A multi node ColumnStore environment featuring Apache Spark and Zeppelin UI
- Jupyter environment: A multi node ColumnStore environment featuring Apache Spark and Jupyter UI

CHAPTER

FOUR

API REFERENCE

4.1 columnStoreExporter Module

4.1.1 Functions

generateTableStatement

columnStoreExporter.**generateTableStatement**(*dataFrame*, *database=None*, *table="spark_export"*, *determineTypeLengths=False*)

Generates a CREATE TABLE statement based on the schema of the submitted DataFrame.

Returns A DML CREATE TABLE statement based on the schema of the submitted DataFrame.

Parameters

- **dataFrame** – The DataFrame from whom the structure for the generated table statement will be inferred.
- **database** – The database name used in the generated table statement.
- **table** – The table name used in the generated table statement.
- **determineTypeLength** – If set to True the content DataFrame will be analysed to determine the best SQL datatype for each column. Otherwise reasonable default types will be used.

Note: The submitted database and table names will automatically be parsed into the ColumnStore naming convention, if not already compatible.

---

export

columnStoreExporter.**export**(*database*, *table*, *df*, *configuration=None*)

Exports the given DataFrame into an existing ColumnStore table.

Parameters

- **database** – The target database the DataFrame is exported into.
- **table** – The target table the DataFrame is exported into.
- **df** – The DataFrame to export.

- **configuration** – Path to the Columnstore.xml configuration to use for the export. If None is given, the default Columnstore.xml will be used.

Note: To guarantee that the DataFrame import into ColumnStore is a single transaction, that is rollbacked in case of error, the DataFrame is first collected at the Spark master and from there written to the ColumnStore system. Therefore, it needs to fit into the memory of the Spark master.

Note: The schema of the DataFrame to export and the ColumnStore table to import have to match. Otherwise, the import will fail.

---

parseTableColumnNameToCSConvention

columnStoreExporter.**parseTableColumnNameToCSConvention**(*input*)

Parses the input String according to the ColumnStore naming convention and returns it.

Returns The parsed input String in ColumnStore naming convention.

Parameters **input** – The String that is going to be parsed.

PYTHON MODULE INDEX

C

INDEX

C