ORACLE GOLDENGATE BIG DATA ADAPTER FOR HIVE

Version 1.0

Oracle Corporation

Table of Contents

1. INTRODUCTION

The Oracle GoldenGate Big Data Adapter for Hive is designed to stream change capture data into the Hadoop Distributed File System (HDFS) and query the records using Hive. This example is developed to provide a ready-made functionality, while at time maintaining simplicity so that the example is easily understood. Oracle GoldenGate customers can use, modify, and extend this example to fulfill their specific needs.

1.1. Functionality

The Hive Adapter takes the unfiltered operations from the source trail and writes them into multiple files in HDFS based on the file size configured.

1. Records for each table will be written into its own directory structure. With this approach tables can be created in Hive for each source table by pointing the LOCATION of the Hive table to the corresponding directory path in HDFS.

   Directory structure and format of records in the file are as noted below.

Directory structure of files
`HDFS_ROOT_DIRECTORY/SCHEMA_NAME/TABLE_NAME`/

`HDFS_ROOT_DIRECTORY`:- The root directory structure in HDFS as configured in the properties.
`SCHEMA_NAME`:- The schema in which the source table exists.
`TABLE_NAME`:- The name of the table for which the change data is being captured.

File Naming
`<TABLE_NAME>_<SYSTEM_TIMESTAMP>`

`TABLE_NAME`:- The name of the table for which the change data is being captured.
`SYSTEM_TIMESTAMP`:- The timestamp of the system when the file is being created. Timestamp is being used in the file name to maintain unique file names.

Change data records
```
<OPERATION_TYPE><COLUMN_DELIMITER><COLUMN_1_VALUE><COLUMN_DELIMITER>...<COLUM
N_N_VALUE><COLUMN_DELIMITER><OPERATION_TIMESTAMP><LINE_DELIMITER>
```

`OPERATION_TYPE`:- Indicated the type of operation i.e. Insert or Update or Delete.
`COLUMN_DELIMITER`:- Column delimiter as specified in the properties.
`COLUMN_1_VALUE`:- Value of the 1st column.
`...` :- `COLUMN_DELIMITER` and `COLUMN_VALUE` will be repeated for all column values between the first and last column.
`COLUMN_N_VALUE`:- Value of the last column.
`OPERATION_TIMESTAMP`:- Timestamp of the operation.
`LINE_DELIMITER`:- Line delimiter as specified in properties.

2. All records written to the HDFS file will be in Delimiter Separated Values (DSV) format. Column and row delimiters can be configured in the properties.
3. Files for each table will be rolled based on size configured in properties.
4. On every restart of the adapter pump process, the current open files, if any, will be closed, resulting in the opening of a new file in HDFS.

1.2. Supported Operations

- Inserts
- Updates including change to primary key(s)
- Deletes

1.3. Unsupported Operations

- Truncate Table

2. BUILDING HIVE ADAPTER

2.1 Prerequisite

1. The Oracle GoldenGate Java Adapter 12 is built and runs with Java 7. It is required that the Hive Adapter also be built with the Java SE Development Kit 7.

2. The recommended method of building the Hive Adapter is using Apache Maven. Maven is an open source build tool available for download at http://maven.apache.org. One of the biggest benefits provided by Maven is that it will resolve and download the build and runtime dependencies of the Hive Adapter from the Maven central repository. Internet connectivity is required to access the Maven central repository. The Hive Adapter was built and tested with version 3.0.5 of Maven.

3. The precompiled Hive Adapter was built and tested using version 2.5.1 of the HDFS client and 0.13.1 version of Hive client.

NOTE: - **Step 2** is required only if maven is used to build the Hive adapter.

2.2 Building with Maven

1. Navigate to GoldenGate (Java Adapters) root directory
   `GG_ROOT_DIRECTORY /AdapterExamples/big-data/hive`

2. Type the following command to build
   ```
   mvn clean package
   ```

3. Maven will download all the required dependencies and build the Hive Adapter. The created jar and the downloaded dependent jars will be placed in the following directory
   `GG_ROOT_DIRECTORY/AdapterExamples/big-data/hive/target/hive-lib`

4. One of the dependencies that Maven does not download is the Oracle GoldenGate Java adapter (`ggjava.jar`). This dependency is located in the following directory
   `GG_ROOT_DIRECTORY/ggjava/ggjava.jar`

   This dependency is instead resolved using a relative path inside the `pom.xml` file. If the Hive Adapter project is copied to a different directory, the relative path to the `ggjava.jar` file will need to be fixed by editing the `pom.xml` file in:
   `GG_ROOT_DIRECTORY/AdapterExamples/big-data/hive`

Note: - The pom.xml file in the GG_ROOT_DIRECTORY/AdapterExamples/big-data/hive directory instructs Maven what version of the HDFS client to download and to build against. It is important that the HDFS client version match the version of HDFS to which it is connecting. Using mismatched versions of the HDFS client and HDFS server can result in serious problems. The version of the HDFS client in the pom.xml file is 2.5.1. If using a version of HDFS other than 2.5.1 it is highly recommended that the user modify the pom.xml file so that HDFS Adapter is built against the correct version of the HDFS client, and the correct jar files are downloaded for use at runtime.

WARNING: - Many companies employ proxy servers to help protect their network from the hazards of the Internet. Performing a Maven build while connected to the company network or VPN may require that dependency downloads from the Maven central repository go through a proxy server. This may require special configuration of the Maven settings.xml file which is generally located in the USER HOME/.m2 directory. Consult your network administrator for proxy settings. Consult the Maven documentation on how to correctly configure the settings.xml file for your required proxy settings.

2.3. Building with Build Script

A build script, `build.sh`, is provided in the `*GG_ROOT_DIRECTORY*/AdapterExamples/big-data/hdfs` directory. This script is only an alternative to the Maven build mentioned above.

Note: - The build script requires *HDFS_HOME* environment variable to be set.

3. Configuration

3.1. Hadoop Prerequisite

In order to successfully run the Hive Adapter, Hadoop single instance or Hadoop cluster must be installed, running, and network accessible to the machine running the Hive Adapter.

3.2. Runtime and Classpath

1. Hadoop Connectivity

   The Oracle GoldenGate Big Data Adapter for Hive obtains connectivity information to HDFS via the Hadoop `core-site.xml` file. This file is generally located in the *`HADOOP_HOME`*`/etc/hadoop` directory. The directory containing the properly configured `core-site.xml` file must be in the `CLASSPATH`.

2. Hive Dependencies Classpath

   The GoldenGate Hive Adapter requires the HDFS client, which in turn requires a number of dependent jars. The easiest way to fulfill the runtime dependencies of the Hive Adapter is by configuring the `gg.classpath` property in the properties file using the wildcard asterisk character (*). . The GoldenGate Java properties file is very specific as to how the wildcard character is used.

   The following works:

   ```
   gg.classpath= GG_ROOT_DIRECTORY/AdapterExamples/big-
   data/hive/target/hive-lib/*
   ```

   The following does NOT work:

   ```
   gg.classpath= GG_ROOT_DIRECTORY/AdapterExamples/big-
   data/hive/target/hive-lib/*.jar
   ```

3. GG Adapter Boot Options

   The following can be configured using the boot options property.

   - Memory allocation for GG Java Adapter JVM (-Xmx and -Xms).
   - HDFS `core-site.xml`
   - GG Adapter dependencies (`ggjava.jar`)

   javawriter.bootoptions=`-Xmx512m -Xms32m -Djava.class.path=ggjava/ggjava.jar` *`-Dlog4j.configuration=log4j.properties`*

`PATH_TO_HADOOP_CONFIG_DIR`: Directory containing Hadoop ***core-site.xml.***
`CLASSPATH_DELIMITER`: The Classpath delimiter for Windows is semicolon (;) and Linux is colon (:).

3.3. Accessing the Precompiled Hive Adapter

The Oracle GoldenGate Big Data Adapter for Hive installation includes the pre-built jar file. The Hive Adapter binary is available in the following location

`GG_ROOT_DIRCTORY/AdapterExamples/big-data/hive/bin/ogg-hive-adapter-1.0.jar`

If the If the above jar is being used instead of building the Hive Adapter from source, then ogg-hive-adapter-1.0.jar should be added to the `gg.classpath` variable in the properties file. The `gg.classpath` variable must also include the HDFS dependency jars. This is generally resolved by adding `HDFS_ROOT_DIRECTORY/share/hadoop/common/lib/*:HDFS_ROOT_DIRECTORY/share/hadoop/common/*` to the `gg.classpath` variable.

3.4. Hive Adapter Properties

The below properties are specific to the Hive Adapter. The *`name`* attribute represents the handler name configured as part of `gg.handlerlist`.

1. **`gg.handler.`*`name`*`.type`**

   The value of this property should not be changed and always should be `com.goldengate.delivery.handler.hdfs.HiveHandler`.
   This property is mandatory.

2. **`gg.handler.`*`name`*`.fieldDelimiter`**

   The delimiter to be used for field/column value separation. Values like semicolon (;) or comma (,) or any other character can be set. Nonprintable characters like `\u0001`, `\u0002` are also supported. Use the Unicode format in the configuration file when using nonprintable characters as delimiters. The default value is `\u0001`. This property is optional.

3. **`gg.handler.`*`name`*`.lineDelimiter`**

   The delimiter to be used for line separation.This property is optional. The default value is \n .

`gg.handler.`*`name`*`.rootdirectoryPath`

The root directory in HDFS under which the directories and files will be created. This property is mandatory.

4. **`gg.handler.`*`name`*`.deleteOpKey`**

   The actual key that should be used to represent the `DELETE` operation type in DSV. This property is optional. The default value is `D`.

5. **`gg.handler.`*`name`*`.updateOpKey`**

   The actual key that should be used to represent the `UPDATE` operation type in DSV
   This property is optional. Default value is `U`.

6. **`gg.handler.`*`name`*`.insertOpKey`**

   The actual key that should be used to represent the `INSERT` operation type in DSV. This property is optional. Default value is `I`.

7. **`gg.handler.`*`name`*`.pKUpdateOpKey`**

   The actual key that should be used to represent the `PRIMARY KEY UPDATE` operation type in DSV. This property is optional. Default value is `P`.

8. **`gg.handler.`*`name`*`.includeOpType`**

   A value of `true` indicates the operation type is to be included in DSV. A value of `false` indicates the operation type is not to be included in DSV. This property is optional. The default value is `true`.

9. **`gg.handler.`*`name`*`.includeOpTimestamp`**

   A value of `true` indicates the operation timestamp is to be included in DSV. A value of `false` indicates the operation timestamp is not to be included in DSV. This property is optional. The default value is `true`.

10. **`gg.handler.`*`name`*`.maxFileSize`**

    The maximum size of the file in HDFS. Once the maximum size is reached, the current file will be closed and a new file will be opened. The value should be entered in bytes. This property is optional. The default value is 10485760 (i.e.10 MB).

4. Performance Considerations

By default, data is flushed to the HDFS server at the end of each transaction. This behavior is the same if operating in `op` or `tx` mode, so performance differences are not likely to be significant.

If transactions are small (contain few operations), performance may be significantly increased by employing the new transaction grouping functionality. The transaction grouping functionality allows operations which cross multiple transaction boundaries to be grouped together as a single transaction. Because performance is dependent upon many variables, the use of the transaction grouping functionality cannot guarantee increased performance. It is simply a mechanism to help customers tune the Oracle GoldenGate Java Adapter to their specific needs.

4.1 Grouping Configuration

1. **`gg.handler.`*`name`*`.mode`**

   To enable grouping the value of this property should be set to `tx`.

2. **`gg.handler.`*`name`*`.maxGroupSize`**

   Controls the maximum number of operations that can be held by an operation group – irrespective of whether the operation group holds operations from a single transaction or multiple transactions.
   The operation group will send a transaction commit and end the group as soon as this number of operations is reache. This property leads to splitting of transactions across multiple operation groups

3. **`gg.handler.`*`name`*`.minGroupSize`**

   This is the minimum number of operations that must exist in a group before the group can end. This property helps to avoid groups that are too small by grouping multiple small transactions into one operation group so that it can be more efficiently processed.

NOTE: maxGroupSize should always be greater than or equal to minGroupSize; i.e. maxGroupSize >= minGroupSize.

Example*:-* Consider a scenario where the source trail has 100 transactions with 10 operations each and the handler is configured in `tx` mode.

Without grouping functionality, transaction commit occurs for every 10 records ultimately flushing the records to HDFS for every 10 records.

With grouping enabled by setting max and min group size to 1000, transaction commit occurs for 1000 records ultimately flushing 1000 records to HDFS at once.

5. Guide to configure and run Hive Adapter

The following steps assume `GG_ROOT_DIRECTORY` to be the GoldenGate installation directory.

5.1. Hadoop Setup

Make sure Hadoop is started using the script available in `$HADOOP_HOME/sbin`.

```
1. ./start-dfs.sh
2. ./start-yarn.sh
```

5.2. Building Hive Adapter using Maven

1. Navigate to `$GG_ROOT_DIRECTORY/AdapterExamples/big-data/hive`
2. Execute command `mvn clean package`

5.3. Configuring Hive Adapter

1. Copy `$GG_ROOT_DIRECTORY/AdapterExamples/big-data/hive/dirprm/hive.prm to $GG_ROOT_DIRECTORY/dirprm`
2. Copy `$GG_ROOT_DIRECTORY/AdapterExamples/big-data/hive/dirprm/hive.props` to `$GG_ROOT_DIRECTORY/dirprm`
3. Edit {*HDFS_INSTALL_DIRECTORY*" in $GG_ROOT_DIRECTORY/dirprm/hive.props to point to a valid Hadoop installation path
4. At the GGSCI command prompt, execute the following command:
   GGSCI> `ADD EXTRACT HIVE, EXTTRAILSOURCE ./AdapterExamples/java-delivery/tc`

5.4. Starting Hive Adapter

1. Make sure that the `LD_LIBRARY_PATH` environment variable set. For example:
   ```
   export LD_LIBRARY_PATH=$JAVA_HOME/jre/lib/amd64/server/
   ```

2. At the GGSCI command prompt, execute the following commands:
   ```
   GGSCI  1> START MGR
   GGSCI  2> START HIVE
   GGSCI  3> INFO ALL
   ```
   Both the Manager and Hive processes should be in `RUNNING` status.

5.5. Creating tables in Hive

1. Navigate to `$HIVE_HOME/bin` and execute the following command:

   ```
   ./hive
   ```

2. Execute the following create table scripts:

   ```
   hive> CREATE EXTERNAL TABLE TCUSTMER(
   OPERATION_TYPE                STRING,
   CUST_CODE                     STRING,
   NAME                          STRING,
   CITY                          STRING,
   STATE                         STRING,
   OPERATION_TIMESTAMP           STRING
   )
   ROW FORMAT DELIMITED FIELDS TERMINATED BY '\u0001' LINES TERMINATED BY
   '\n'
   LOCATION '/gg/replication/hive/gg/tcustmer';

   hive> CREATE EXTERNAL TABLE TCUSTORD(
   OPERATION_TYPE                STRING,
   CUST_CODE                     STRING,
   ORDER_DATE                    STRING,
   PRODUCT_CODE                  STRING,
   ORDER_ID                      INT,
   PRODUCT_PRICE                 DOUBLE,
   PRODUCT_AMOUNT                INT,
   TRANSACTION_ID                INT,
   OPERATION_TIMESTAMP           STRING
   )
   ROW FORMAT DELIMITED FIELDS TERMINATED BY '\u0001' LINES TERMINATED BY
   '\n'
   LOCATION '/gg/replication/hive/gg/tcustord';
   ```

5.6. Querying records from HDFS using Hive

1. Navigate to `$HIVE_HOME/bin` and execute the following command:

   ```
   ./hive
   ```

2. Verify data replication into HDFS using the following queries:

   ```
   hive> select * from TCUSTMER;
   hive> select * from TCUSTORD
   ```

6. Limitation

1. Tables in Hive will not be created by the Hive Adapter. Tables should be manually created in Hive using the Hive create table syntax.

2. HDFS DataStream will be flushed using `hflush` on every transaction commit event, but records may not be written onto the physical disc until the DataStream is closed (i.e. once the max file size is reached or if the adapter pump process is stopped). Due to this, the Hive client cannot retrieve the latest records from HDFS until the data is written to disc.

3. The new unified update record is not yet supported by the Hive Adapter and will cause the Extract process to `ABEND`.

4. Hive adapter does not support the truncate table operation. A truncate table operation will cause the Extract process to `ABEND`.