**Transferring data from Oracle Golden Gate to Impala**

The aim of this integration is to capture transactions from Oracle databases (source) and to apply them into Impala database (target). The idea is to replicate the source database in almost real time or in a very short time.

**Architecture**

The architecture can be divided into two systems: source and target. The source system is where Oracle database, the Golden Gate instance and the Impala loader are installed. On the other side, we have the target system which involves a cluster of machines running Impala and HDFS.

The deployment of this architecture is shown below.

**Oracle database**

Source

Log

Trail

Adapter for Flat Files

Flat Files

**Oracle Golden Gate**

**Source machine**

**Target cluster**

Node 1

NameNode

Impala layer

HDFS

layer

Node 2

DataNode

Node N

DataNode

**…**

New

Flat files

SQL: creates and inserts

**Oracle Golden Gate solutions**

Oracle provides different solutions to connect Oracle Golden Gate with other products not related with Oracle. In our case, we want to connect Oracle Golden Gate with Apache Impala, in order to achieve that we can use different approaches such as develop our custom adapter or use an out-of-the-box adapter which generates text files. Both approaches consist of a process that reads the generated trail files and convert it to data that can be read by a custom application.

We have tried both solutions, regarding the first one, where you should **develop a custom adapter**, enable you to send data directly to the target system which sounds good but you must either send the data at the time is read from the trail or take care of different issues that can occur. If you send the data when every transaction is read, the network does not support the speed that we need to cover our needs (around 50000 inserts per second). We also do not want to develop a complex adapter that take care of different issues since we can use the next approach which take care of that.

Finally, we chose to use the adapter which write text files, this adapter is known as **Adapter for Flat Files** and it is instantiated in Oracle Golden Gate as an Extract process. The text files can be written in two formats, **DSV** (Delimiter Separated Values) which contains an operation per line and the values in each line are separated by a configured delimiter, or **LDV** (Length Delimited Values) which contains an operation per line and each line has a more complex format that indicates the length of each field to be read. Since DSV is easier to read, we use this format with the delimiter that Apache Impala uses by default (0x01), with this delimiter we can move the generated text files and use it to create Impala external tables which will be imported into the final table.

**How it works**

On the Oracle Golden Gate side, we use the adapter for Flat Files which generates DSV files with the transactional data. The generated files are transferred to the file system (HDFS) that Impala uses. An implemented loader checks periodically if there is new data to be transferred, and if there is new data, the loader follows several steps:

1. Copies new data to HDFS.
2. Creates temporal (external) table in Impala with the new data located in HDFS.
3. Inserts all data from temporal table into final Impala table.
4. Deletes temporal table.
5. Deletes new local data.

To understand better how the loader works, we show a flow diagram as following.

No

Yes

Copy local new data to HDFS

Create Impala temporal table

Insert data to final table

Delete temporal data

Start

New Flat

Files?

Wait N seconds

**Installation and configuration**

First of all, we have to configure Oracle Golden Gate in order to produce the Flat Files that the loader will read. On the other hand, we should deploy and configure the loader to import data into target Impala table.

|  |  |  |  |
| --- | --- | --- | --- |
| [**NAME**](https://twiki.cern.ch/twiki/bin/view/DB/Private/InstallationAndConfiguration?sortcol=0;table=1;up=0#sorted_table) | [**HOSTS**](https://twiki.cern.ch/twiki/bin/view/DB/Private/InstallationAndConfiguration?sortcol=1;table=1;up=0#sorted_table) | **INSTALLATION HOME** | [**TRAIL FILES**](https://twiki.cern.ch/twiki/bin/view/DB/Private/InstallationAndConfiguration?sortcol=3;table=1;up=0#sorted_table) |
| D3R | Itrac910 | /ORA/dbs01/oracle/product/OGG12 | /ORA/dbs02/trailsHadoop/TH |
| whatever | Itrac925 | /ORA/dbs91/GGhomes/ATLAS/prod/offline\_T1s/ogg121210\_rdbms11g | /ORA/dbs92/GGtrails/ATLR/OT |

**Oracle Golden Gate side**

1. First at all, we have created a table in D3R with the user “gguser”, which will be replicated. This table is called “data\_numeric” that contents three columns: VARIABLE\_ID, UTC\_STAMP and value. The primary key is a combinated one, set by (VARIABLE\_ID,UTC\_STAMP));
2. We do not add the supplemental logging to the table since we already have enabled it for the entire database D3R (from previous tests with Oracle GoldenGate)
3. As we already have Oracle GoldenGate installed, there is no need to install any new environment.
4. We create the processes corresponding to this configuration. We won’t have replicat, since the target is HDFS. At the same time, we won’t need the data pump. The reason is according to the previous architecture schema, only a second extract is needed to read the changes from trail files, connect to the loader and create the “Flat Files”.
5. Extract configuration

|  |
| --- |
| EXTRACT EXTFLUME  USERID ggadmin, PASSWORD ggtest  EXTTRAIL /ORA/dbs02/trailsHadoop/TH  GETUPDATEBEFORES  NOCOMPRESSUPDATES  TABLE gguser.data\_numeric; |

1. Extract configuration steps on Oracle GoldenGate

|  |
| --- |
| ADD EXTRACT EXTFLUME, INTEGRATED TRANLOG, BEGIN NOW  ADD EXTTRAIL /ORA/dbs02/trailsHadoop/TH, EXTRACT EXTFLUME, megabytes 50  REGISTER EXTRACT EXTFLUME DATABASE |

1. Second extract configuration

**NOTE:** The params SOURCEDEFS AND CUSEREXIT are in-depth explained in the next chapter “Installation and configuration of the adapter for Flat Files”

|  |
| --- |
| EXTRACT FFWRITER  SOURCEDEFS ./dirsql/datanumeric.sql  CUSEREXIT ./flatfilewriter.so CUSEREXIT PASSTHRU, INCLUDEUPDATEBEFORES, PARAMS "ffwriter.properties"  TABLE gguser.data\_numeric; |

1. Extract configuration steps on Oracle GoldenGate

|  |
| --- |
| ADD EXTRACT FFWRITER, EXTTRAILSOURCE /ORA/dbs02/trailsHadoop/TH  REGISTER EXTRACT FFWRITER DATABASE |

**Installation and configuration of the adapter for Flat Files**

1. This step is regarding to the parameter SOURCEDEFS in the second extract. Since the source and target table definitions are different, we first need to create what is called a data definition file which contains the definition of the source table. This is done using the DEFGEN utility and after the file is created, we transfer the same to the target server.
2. We first create a DEFGEN parameter file as shown below

|  |
| --- |
| defsfile ./dirsql/data\_numeric.sql  userid ggadmin password ggtest  table gguser.data\_numeric; |

OGG for Java includes a utility, DEFGEN, that is used to generate a data definitions file from the properties file settings and other parser specific data definition values. The DEFGEN output file is then specified in the Extract Data Pump or Replicat SOURCEDEFS parameter so the process can interpret the data contained in the OGG Trail created by the VAM.

1. We then run DEFGEN (OGG home) and specify the parameter file to be used which is the one we have just created

|  |
| --- |
| itrac910>-RAC>-d3r1:/ORA/dbs01/oracle/product/OGG12/dirsql$ ./defgen paramfile ./dirprm/defgen.prm |

1. This will create a file called data\_numeric.sql in the /OGG12/dirsql subdirectory with the following contents

|  |
| --- |
| \*+- Defgen version 4.0, Encoding UTF-8  \*  \* Definitions created/modified 2015-03-18 15:13  \*  \* Field descriptions for each column entry:  \*  \* 1 Name  \* 2 Data Type  \* 3 External Length  \* 4 Fetch Offset  \* 5 Scale  \* 6 Level  \* 7 Null  \* 8 Bump if Odd  \* 9 Internal Length  \* 10 Binary Length  \* 11 Table Length  \* 12 Most Significant DT  \* 13 Least Significant DT  \* 14 High Precision  \* 15 Low Precision  \* 16 Elementary Item  \* 17 Occurs  \* 18 Key Column  \* 19 Sub Data Type  \* 20 Native Data Type  \* 21 Character Set  \*  Database type: ORACLE  Character set ID: we8iso8859p1  National character set ID: UTF-16  Locale: neutral  Case sensitivity: 14 14 14 14 14 14 14 14 14 14 14 14 11 14 14 14  TimeZone: +02:00  \*  Definition for table GGUSER.DATA\_NUMERIC  Record length: 144  Syskey: 0  Columns: 3  VARIABLE\_ID 64 50 0 0 0 1 0 50 50 50 0 0 0 0 1 0 1 2 2 -1  UTC\_STAMP 192 29 56 0 0 1 0 29 29 29 0 6 0 0 1 0 1 0 187 -1  VALUE 64 50 88 0 0 1 0 50 50 50 0 0 0 0 1 0 0 2 2 -1  End of definition |

1. This step is regarding to the parameter CUSEREXIT in the second extract. This is to call a custom exit routine written in C programming code from a Windows DLL or UNIX shared object at a defined exit point within Oracle GoldenGate processing. Your user exit routine must be able to accept different events and information from the Extract and Replicat processes, process the information as desired, and then return a response and information to the caller (the Oracle GoldenGate process that called it).
2. We need to copy the Java Library from OGG JAVA Adapters (downloaded from MOS) to the directory of parameter files. This is the file indicated within this parameter (**./flatfilewriter.so**)
3. There is a customed file which we called **ffwriter.properties,** where we established all the parameters needed for the adapter. This file has to be located in the GGHOME.

|  |
| --- |
| goldengate.userexit.datetime.removecolon=true  goldengate.flatfilewriter.writers=dsvwriter  goldengate.log.logname=ffwriter  goldengate.log.level=INFO  goldengate.log.tostdout=false  goldengate.log.tofile=true  dsvwriter.mode=DSV  dsvwriter.rawchars=false  dsvwriter.includebefores=false  dsvwriter.includecolnames=false  dsvwriter.omitvalues=false  dsvwriter.diffsonly=false  dsvwriter.omitplaceholders=false  dsvwriter.files.onepertable=true  dsvwriter.files.data.ext=\_data.dsv  dsvwriter.files.data.tmpext=\_data.dsv.temp  dsvwriter.dsv.nullindicator.chars=  dsvwriter.dsv.fielddelim.code=01  dsvwriter.dsv.fielddelim.escaped.chars=  dsvwriter.files.data.rootdir=./out  dsvwriter.files.data.rollover.size=500  dsvwriter.files.data.rollover.time=2  dsvwriter.dsv.quotes.policy=none |

**Impala loader side**

**Deployment and configuration**

The loader is a custom client of HDFS and Apache Impala that reads the Flat Files generated by Oracle Golden Gate and inserts this data into Impala. We must follow some steps to get a functional loader. That commands must be executed in the same machine where Oracle Golden Gate is installed and with the same system user that Oracle Golgen Gate is executed (e.g. oracle).

1. We use Git for versioning and GitHub as remote repository. In order to facilitate the building of the project we include a POM file which is a descriptor for Maven, that file is used by this tool to build the binaries. We must **clone** and **build** the project from the console as following.

$> cd <ogg\_directory>

$> git clone https://github.com/dlanza1/ogg-impala.git

$> cd ogg-impala

$> mvn install

1. Once we build the project, several JAR files have been generated. One of these binaries contains the compiled code and all the dependencies. We must copy the JAR with dependencies to a new directory located where OGG is installed.

$> cd <ogg\_directory>

$> mkdir impala-loader

$> cp ogg-impala/target ogg-impala-\*-jar-with-dependencies.jar impala-loader

1. Since the Impala loader produces logging outputs, we should configure it with a properties file.

$> cd <ogg\_directory>/impala-loader

$> nano log4j.properties

An example of the content of this file could be:

|  |
| --- |
| # Root logger option  log4j.rootLogger=INFO, stdout    # Redirect log messages to console  log4j.appender.stdout=org.apache.log4j.ConsoleAppender  log4j.appender.stdout.Target=System.out  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout  log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n |

This example configures the logger to print the logging activity in the standard output, commonly the console.

1. The loader has several parameters that must be configured, so we must create a parameters file for the loader.

$> cd <ogg\_directory>/impala-loader

$> nano config.params

An example of the content of this file could be:

|  |
| --- |
| impala.host = itrac925.cern.ch  ogg.data.folder = ./out  ogg.control.file.name = GGUSER.DATA\_NUMERICcontrol  hdfs.staging.directory = ogg/staging |

1. Since the loader is a HDFS client, it must know where the NameNode (master process of HDFS) is running, so must create a configuration file with the address and port the that process.

$> cd <ogg\_directory>/impala-loader

$> nano core-site.xml

The file must contain the following content, we must only change the address and port.

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!--Autogenerated by Cloudera Manager-->  <configuration>  <property>  <name>fs.defaultFS</name>  <value>hdfs://itrac925.cern.ch:8020</value>  <description>NameNode host and port</description>  </property>  </configuration> |

After the above steps, the current state of impala-loader directory must be:

$> ls -l <ogg\_directory>/impala-loader

config.params

core-site.xml

log4j.properties

ogg-impala-\*-jar-with-dependencies.jar

Finally, we must choose if we want to run the loader in **background**, which implies when we close the ssh connection the loader will be killed, so it should be only used for testing. Or in **background**, which is recommended for production environment since the process will be always running untill it is explicitly killed.

* 1. Foreground.

$> cd <ogg\_directory,>

$> export CLASSPATH=./impala-loader/:./impala-loader/\*

$> java ch.cern.impala.ogg.datapump.ImpalaDataLoader ./impala-loader/config.params

Thanks to we configured the logger to print the logging messages in the console, we can observe an output similar to:

|  |
| --- |
| 2015-04-20 13:49:23 INFO ImpalaDataLoader:22 - inicializing loader (properties file = ./impala/config.params)  2015-04-20 13:49:23 INFO ImpalaDataLoader:26 - reading control data from ./out/GGUSER.DATA\_NUMERICcontrol  2015-04-20 13:49:23 WARN PropertiesE:71 - the number of seconds between batches has been set to the default value (10 seconds)  2015-04-20 13:49:23 INFO ImpalaDataLoader:47 - there is no data to process  2015-04-20 13:49:33 INFO ImpalaDataLoader:47 - there is no data to process |

* 1. Background.

$> cd <ogg\_directory>

$> export CLASSPATH=./impala-loader/:./impala-loader/\*

$> nohup java ch.cern.impala.ogg.datapump.ImpalaDataLoader ./impala-loader/config.params &

# To check the output of the command

$> cat nohup.out

# It will show the some output when we run in foreground (this file stored the standard output of the process)