ORCHESTRATING ORACLE GOLDENGATE MICROSERVICES USING PL/SQL

Contents

1 INTRODUCTION

Oracle GoldenGate (OGG) Microservices Architecture is a new architecture that provides REST-enabled services as part of the OGG environment. This document assumes that the reader is familiar with the concepts and features of the OGG Microservices Architecture. The REST-enabled services enable remote configuration, administration, and monitoring through HTML5 web pages, command line, and APIs

> ***The concepts, scripts and information presented in this article are for educational purposes only. They are not supported by Oracle Development or Support, and come with no guarantee or warrant for functionality in any environment other than the test system used to prepare this article. Before applying any changes presented in this article it should be thoroughly tested to assess functionality and performance implications***

PL/SQL can also be used to invoke the REST API's to manage OGG Microservices. UTL_HTTP is a PL/SQL package that enables making HTTP(S) calls using PL/SQL. This demo is built on UTL_HTTP to make REST calls to OGG deployments.

This document outlines how to setup one-way replication and bidirectional replication, between a source and a target database. The example provided utilizes an orchestration Database to make the REST calls via PL/SQL to setup replication, however, the use of a separate orchestration database is optional.

The example will walk through how to create all of the components for replication, such as EXTRACT, DISTRIBUTION PATH and a REPLICAT and start them from the remote orchestration database.

An outline for managing or querying the created components, such as stopping EXTRACT or modifying EXTRACT parameter file, will also be demonstrated.

An overview of the setup used for this demo

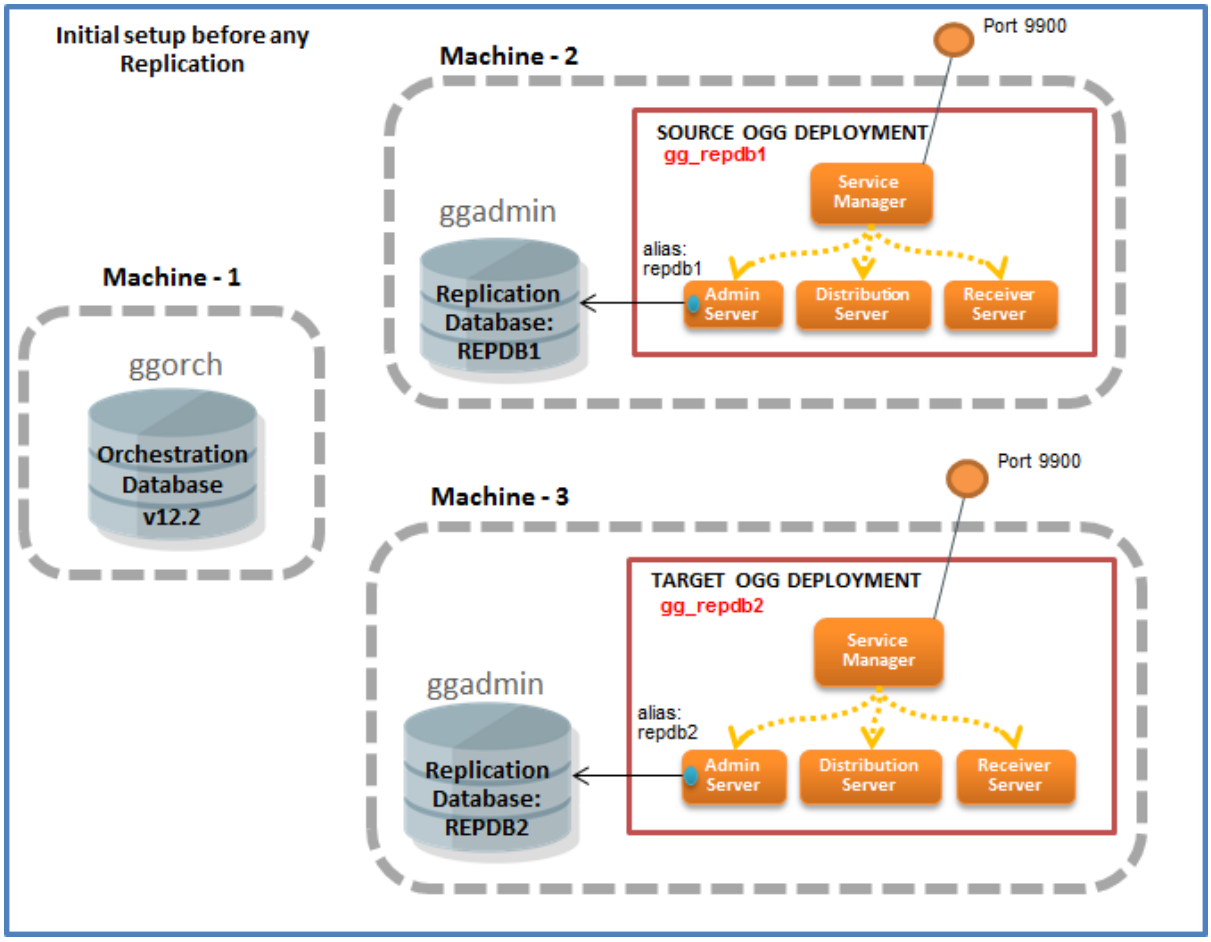

Figure-1

The legend for **Figure-1** is given below:

ggorch	OGG orchestration user in orchestration database
ggadmin	OGG replication admin user in replication database
repdb1, repdb2	Names of replication databases
Port ***9900***	Service Manager port of OGG deployment
gg_repdb1, gg_repdb2	Deployment names in the OGG deployment
alias:***repdb1***, alias:***repdb2***	Alias names of the credentials in the OGG deployment.

2 PREREQUISITES

The demo orchestration script assumes the following setup as shown in **Figure-1**:

- Machine-1 has a database with version 12.2 or higher. The demo script is to be installed in this orchestration database.
- Machine-2 has a replication database setup and Oracle GoldenGate Microservices deployed
- Machine-3 has a replication database setup and Oracle GoldenGate Microservices deployed.

Note: In order to try the demo, the entire setup of the databases and the Oracle GoldenGate deployments can be hosted in the same machine.

3 BACKGROUND

3.1 OGG NEW ARCHITECTURE

The Oracle GoldenGate Microservices Architecture (MA) is based on Representational State Transfer Application Programming Interface (REST API). It allows configuring, monitoring, and managing Oracle GoldenGate services using a web-based interface, command line, or REST APIs.

In contrast to OGG classic architecture, OGG Microservices architecture has different processes such as Service Manager, adminsrvr, distsrvr and recvsrvr (see Components of Oracle GoldenGate Microservices Architecture).

3.2 OGG INSTALLATION AND DEPLOYMENT

The Getting Started with Oracle GoldenGate Microservices Architecture document details the steps for setting up replication using the new architecture. The deployments can be secure (https based) or non-secure (http based). The security model for the OGG REST API's requires both Authentication and Authorization. More details of the security model can be found in the documentation for Setting Up Secure or Non-Secure Deployments and Securing The Oracle GoldenGate Environment.

Secure deployments use a certificate authority (CA) certificate, issued by Verisign, to enable clients to connect securely to the server.
Please note:
a) Certificates issued by CA's other than Verisign can also be used
b) Self Signed certificates can also be used.
You can use ORAPKI to generate self-signed certificates for testing.

The sample code provided along with this document is only for Secure OGG deployments (i.e. HTTPS) that are configured for certificate based Authentication and Authorization.

3.3 OGG REST API

OGG REST API is used to manage OGG deployments and replication services. You can create, modify, remove or query the OGG components. The prerequisites for using the REST API are mentioned in the QuickStart section of the OGG documentation.

Each Oracle GoldenGate Service (ServiceManager, adminsrvr, distsrvr, recvsrvr) runs on a separate port. The OGG REST resources can be accessed using the following URL structure:

http://*localhost*:*port*/or https://*localhost*:*port*/ where *localhost* and *port* are where OGG service is running.

OGG can also be configured with reverse proxy at port 443 for secure (HTTPS) setup. The ports of individual services need not be remembered when using reverse proxy.

More about OGG REST API can be found in the documentation About the REST APIs and Reverse Proxy Support documentation gives more information about reverse proxy setup.

4 SAMPLE ORCHESTRATION SCRIPT

The sample code is structured as below

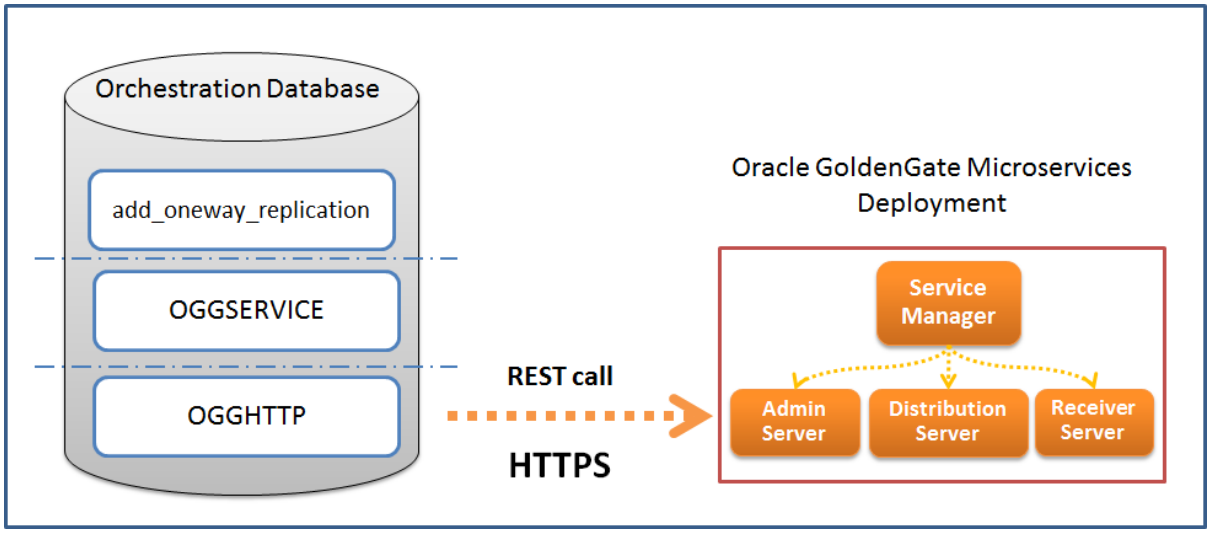

Figure-2

4.1 ADD_ONEWAY_REPLICATION – USAGE

ADD_ONEWAY_REPLICATION	A sample standalone PL/SQL function that sets up a simple one-way replication between a source database and a target database for the specified tables. This uses the underneath provided functions from OGGSERVICE.
OGGSERVICE	Package that illustrates how different actions can be performed on a given OGG deployment using REST calls. The actions are - Create component - Modify component - Remove component - Query component This uses OGGHTTP packages. Each action performs the following : 1. Construct the URL for the action 2. Construct the payload for the action 3. Make the HTTP call using the URL and the payload 4. Handle the response.
OGGHTTP	Package that illustrates a way to make secure HTTP calls using UTL_HTTP.

ADD_ONEWAY_REPLICAION is a sample standalone PL/SQL function that sets up a simple one-way replication between a source database and a target database for the specified tables. This uses the underneath provided functions from OGGSERVICE.

The replication setup instruction assumes that there are three machines (refer to **Figure-1** above)

Orchestration DB site

- Orchestration Database is created where the sample orchestration script is installed as database user GGORCH.

- The client certificate to connect to the OGG deployments is available inside a wallet placed under the directory $ORACLE_HOME/admin/ggorch_wallet. The user name specified in the client certificate is authorized to make REST API calls to both the OGG deployments in REPDB1 site and REPDB2 site mentioned below.

REPDB1 Site

- Replication database ***repdb1*** setup for acting as replication source
- Secure OGG deployment is setup with ServiceManager listening on port ***9900***
- dblogin credential alias, ***repdb1,*** is created with domain as *OracleGoldenGate*, in the admin server credential store of OGG deployment
- OGG Deployment ***gg_repdb1*** is created

REPDB2 Site

- Replication database ***repdb2*** is setup for acting as replication target
- Secure OGG deployment is setup with ServiceManager listening on port ***9900***
- dblogin credential alias, ***repdb2,*** is created with domain as *OracleGoldenGate*, in the admin server credential store of OGG deployment
- OGG Deployment ***gg_repdb2*** is created

The following steps are to be performed in the orchestration database to configure the GGORCH user.

```
SQL> connect sys as sysdba
SQL> grant dba to ggorch identified by <password>;
SQL> @oggorch_setup ggorch <absolute_path_to_client_certificate>
SQL> connect ggorch
SQL> @orchestration_db_setup.sql
```

*Note: <**absolute_path_to_client_certificate**> can be skipped if the directory $ORACLE_HOME/admin/ggorch_wallet directory is created in the orchestration database and the client wallet copied into it. This is the default directory that will be used by the script to look for the client wallet.*

4.1.1 SETUP ONE-WAY REPLICATION

add_oneway_replication FUNCTION sets up a one-way replication between the SCOTT schema and HR schema of the databases ***repdb1*** and ***repdb2.***

The key inputs to this function are

- Source OGG deployment
- Source database OGG alias
- Target OGG deployment
- Target database OGG alias
- List of tables to be replicated

It can optionally take inputs for instantiating the target tables (see add_oneway_replication)
This function uses REST calls to retrieve information about the adminsrvr, distsrvr and recvsrvr running in the OGG deployments. It then creates the following replication components from the remote Orchestration database using REST calls

- Extract in the source OGG deployment.
- Distribution path between source and target OGG deployment
- Replicat in the target OGG deployment.

The parameter files of Extract and Replicat is configured to replicate the provided list of tables.

OGG Deployment in REPDB1 Site

- Creates and start an EXTRACT
- Creates and start a DISTRIBUTION PATH to OGG deployment in REPDB2 site

OGG Deployment in REPDB2 Site

- Creates and start a REPLICAT

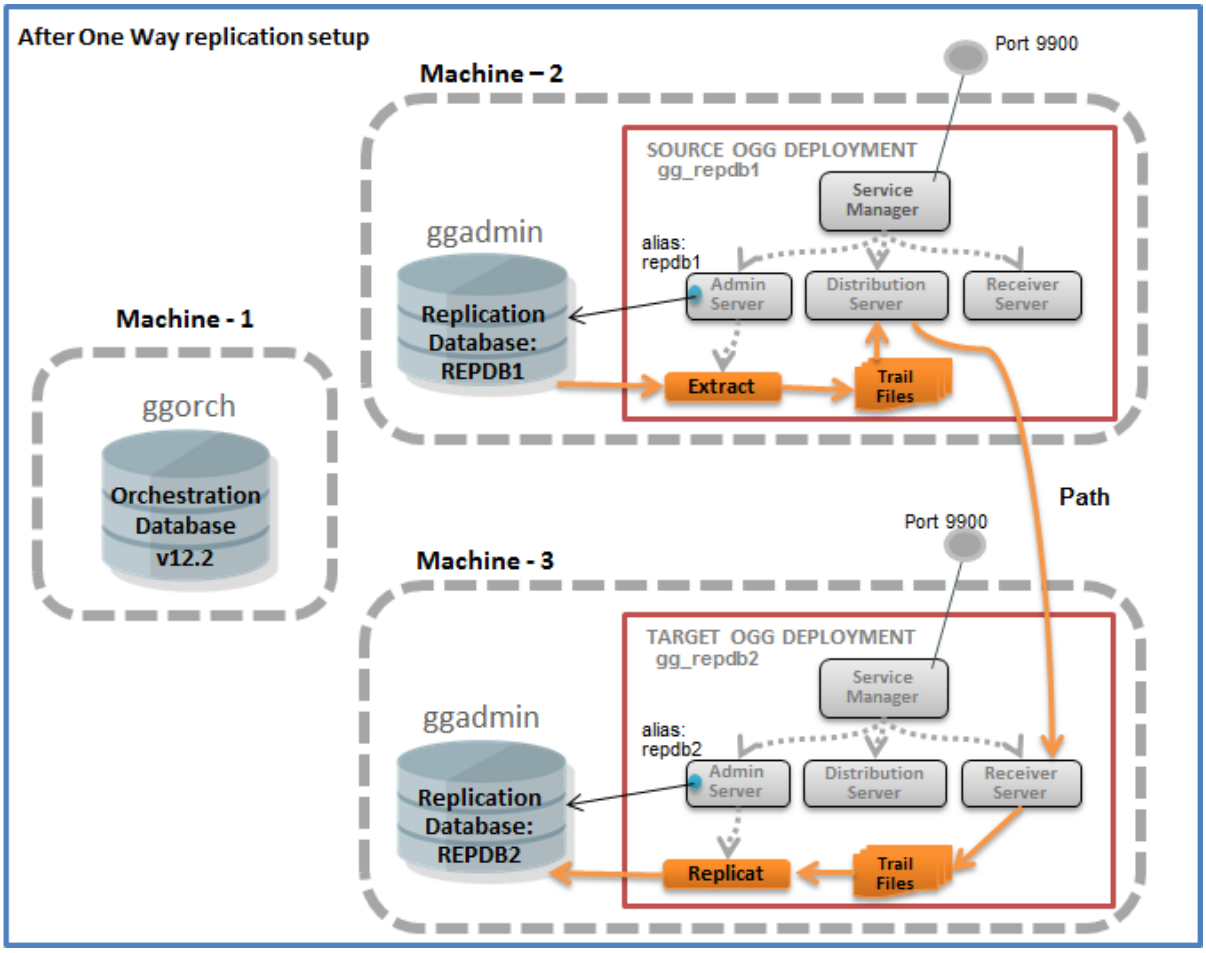

Figure-3

CREATE ONE WAY REPLICATION FROM REPDB1 DATABASE TO REPDB2 DATABASE.

The following code needs to be executed as GGORCH in the orchestration database (see add_oneway_replication). This creates a one-way replication between databases referenced by the aliases ***repdb1*** and ***repdb2*** with former as the source and latter as the target. The resulting setup replicates 'scott' schema and 'hr' schema from source to target.

```
set serverout on
DECLARE
  response_info clob;
  ret number;
BEGIN
  ret := add_oneway_replication(
             ogg_source                       => ogg_connection('&machine2', 9900, 'gg_repdb1'),
             source_dblogin_credential_alias  => 'repdb1',
             ogg_target                       => ogg_connection('&machine3', 9900, 'gg_repdb2'),
             target_dblogin_credential_alias  => 'repdb2',
             tables                           => '"scott.*","hr.*"',
             response_info                    => response_info);
END;
/
```

The output below shows that Extract ***EFIML***, Distribution path ***PFIML*** and replicat ***RFIML*** are created. The Extract and Distribution path are shown to be 'running' in the SOURCE OGG DEPLOYMENT and the replicat is 'starting' in the TARGET OGG DEPLOYMENT.

OUTPUT

```
---------------- REPLICATION:REPDB1 =>REPDB2 -------------------
---------------- EXTRACT -------------------
EXTRACT (Integrated) added.
Extract EFIML successfully registered with database at SCN 4623804.
EXTTRAIL added.
EXTRACT EFIML starting
EXTRACT EFIML started


---------------- DISTRIBUTION PATH -------------------
The path 'PFIML' has been added.


---------------- REPLICAT -------------------
REPLICAT (Integrated) added.
REPLICAT RFIML starting
REPLICAT RFIML started


---------- SOURCE OGG DEPLOYMENT  ---------

Group    Status      Program
EFIML : ["running"] - EXTRACT
PFIML : ["running"] - PATH


---------- TARGET OGG DEPLOYMENT  ---------

Group    Status      Program
RFIML : ["starting"] - REPLICAT
```

*Note: The names are like **E<suffix>** for EXTRACT, **R<suffix>** for REPLICAT and **P<suffix>** for DISTRIBUTION PATH, where **<suffix>** is a random generated 4 character sequence.*

4.1.2 SETUP BI-DIRECTIONAL REPLICATION

The above setup can be made bi-directional replication between the SCOTT schema and HR schema of the databases ***repdb1*** and ***repdb2*** by making another call to ***add_oneway_replication*** by exchanging the source and target details. An additional call to create replication in the reverse direction does the following:

OGG Deployment in REPDB2 Site

- Creates and start an EXTRACT
- Creates and start a DISTRIBUTION PATH to OGG deployment in REPDB1 site

OGG Deployment in REPDB1 Site

- Creates and start a REPLICAT

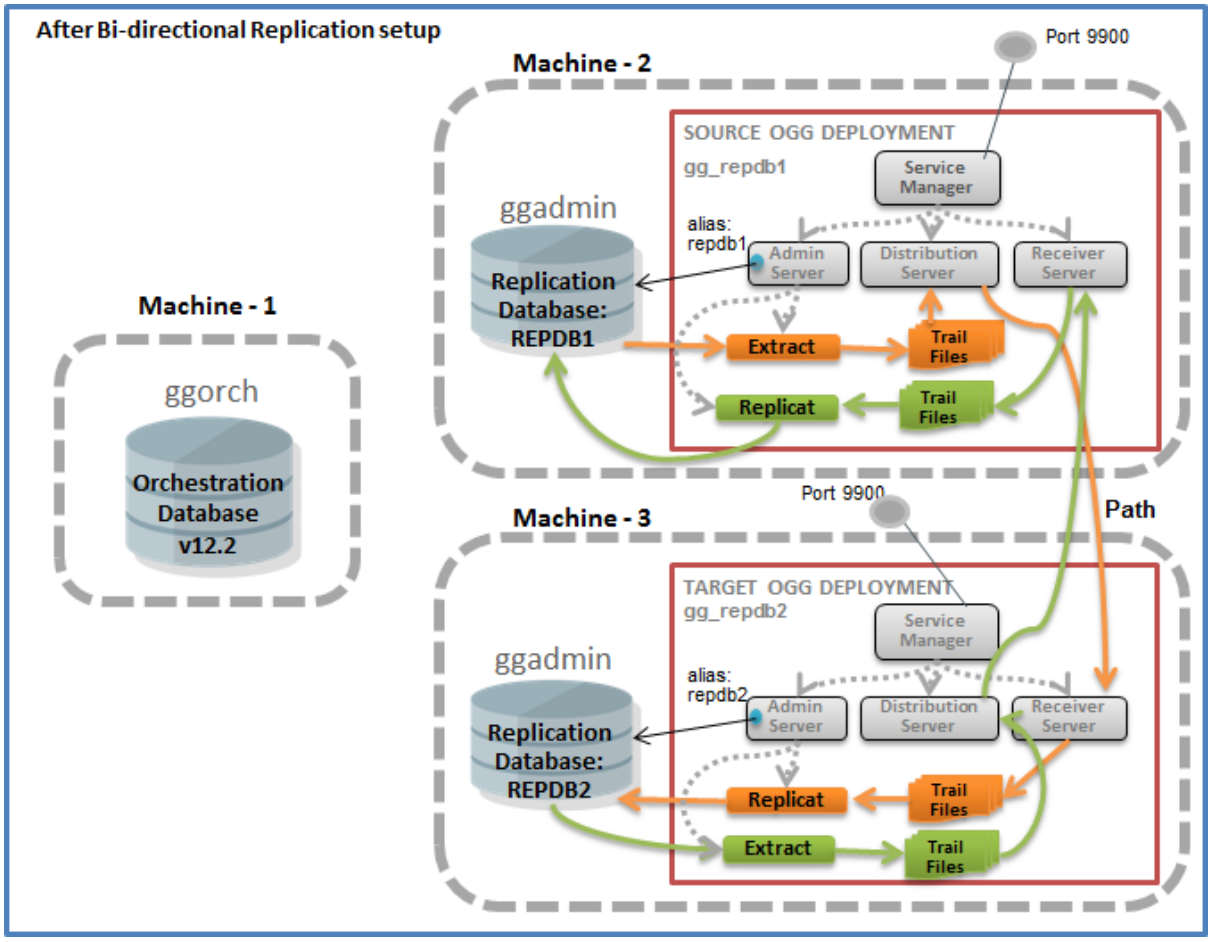

Figure-4

BI-DIRECTIONAL REPLICATION FROM REPDB2 DATABASE TO REPDB1 DATABASE.

The following code need to be executed as GGORCH in the orchestration database. This creates a one-way replication between databases referenced by the aliases ***repdb2*** and ***repdb1*** with former as the source and latter as the target. The resulting setup replicates 'scott' schema and 'hr' schema from source to target. This step, combined with the earlier step, creates a bi-directional setup between ***repdb1*** and ***repdb2*** for 'scott' and 'hr' schemas.

```
set serverout on
DECLARE
  response_info clob;
  ret number;
BEGIN
  ret := add_oneway_replication(
             ogg_source                       => ogg_connection('&machine3',9900,'gg_repdb1'),
             source_dblogin_credential_alias  => 'repdb2',
             ogg_target                       => ogg_connection('&machine2',9900,'gg_repdb2'),
             target_dblogin_credential_alias  => 'repdb1',
             tables                           => '"scott.*","hr.*"',
             response_info                    => response_info);
END;
/
```

The output below shows that Extract ***EFCUG***, Distribution path ***PFCUG*** and replicat ***RFCUG*** are created. The Extract and Distribution path are shown to be 'running' in the SOURCE OGG DEPLOYMENT and the replicat is 'starting' in the TARGET OGG DEPLOYMENT. The replication components ***EFIML, PFIML, RFIML*** created in the earlier steps may also be seen in the output.

OUTPUT

```
---------------- REPLICATION:REPDB2 =>REP DB1 -------------------
---------------- EXTRACT -------------------
EXTRACT (Integrated) added.
Extract EFCUG successfully registered with database at SCN 4624869.
EXTTRAIL added.
EXTRACT EFCUG starting
EXTRACT EFCUG started


---------------- DISTRIBUTION PATH -------------------
The path 'PFCUG' has been added.


---------------- REPLICAT -------------------
REPLICAT (Integrated) added.
REPLICAT RFCUG starting
REPLICAT RFCUG started

---------- SOURCE OGG DEPLOYMENT  ---------

Group    Status      Program
EFCUG : ["running"] - EXTRACT
RFIML : ["running"] - REPLICAT
PFCUG : ["running"] - PATH

---------- TARGET OGG DEPLOYMENT  ---------

Group    Status      Program
EFIML : ["running"] - EXTRACT
RFCUG : ["starting"] - REPLICAT
PFIML : ["running"] - PATH
```

4.2 MANAGING REPLICATION COMPONENTS

Once the replication components are created using ***add_oneway_replication*** they can be updated, started, stopped or removed.

The examples below show a few actions that can be performed on Extract. Similar actions can be performed on other replication components as appropriate.

4.2.1 GET EXTRACT DETAILS

The below code will get the extract details of the extract ***&extract_name*** under OGG deployment ***gg_repdb2*** on machine ***&machine3*** listening on port ***9900***

```
set serverout on
DECLARE
 response_info  CLOB;
 ret            NUMBER;
BEGIN
 ret := OGGSERVICE.get_extract_details(
                ogg_instance  => ogg_connection('&machine3',9900,'gg_repdb2'),
                extract_name  => '&extract_name',
                response_info => response_info);

 pretty_json(response_info);
END;
/
```

The OUTPUT shown below are the details fetched for the extract ***EFCUG***. It may be noted that this shows the "status" as "running" and the "source" attribute says it is an "integrated" extract. The entire parameter file of the extract can also be seen in the "config" attribute. The json schema of this output can be found in the docs.

OUTPUT

```
[
  {
    "messages" :
    [
      [
      ]
```

```
    ],
    "response" :
    [
      {
        "$schema" : "ogg:extract",
        "credentials" :
        {
          "alias" : "repdb2",
          "domain" : "OracleGoldenGate"
        },
        "begin" : "now",
        "targets" :
        [
          {
            "name" : "jq",
            "sizeMB" : 500,
            "sequenceLength" : 9,
            "sequenceLengthFlip" : false,
            "sequence" : 0,
            "offset" : 31556912,
           "remote" : false
          }
        ],
        "config" :
        [
          "extract EFCUG;",
          "useridalias repdb2 domain OracleGoldenGate",
          "exttrail jq",
          "table scott.*;",
          "table hr.*;"
        ],
        "description" : "EFCUG",
        "source" :
        {
          "tranlogs" : "integrated"
        },
        "registration" :
        {
          "csn" : 4624869,
          "share" : true
        },
        "status" : "running"
      }
    ]
  }
]
```


4.2.2 UPDATE EXTRACT PARAMETER FILE

The lines of EXTRACT parameter file is presented as a JSON array (see “config” in the OUTPUT of section 4.2.1) in the REST API payload.

This example updates the parameter file of an EXTRACT by adding an additional line to the existing parameter file. This is done by first retrieving the existing parameters of the EXTRACT and adding the new parameter to the existing list and updating the parameter file with the revised list of lines.

The below code updates the parameter file of extract ***&extract_name*** under the OGG deployment ***gg_repdb2*** on machine ***&machine3*** listening on port ***9900***

```
set serverout on
DECLARE
 response_info  CLOB;
 ret            NUMBER;
 ogg_deployment OGG_CONNECTION;
 config_jarray  JON_ARRAY_T;
 config_str     VARCHAR2(4000);
BEGIN

 ogg_deployment := ogg_connection('&machine3',9900,'gg_repdb2');

 /* Get the existing parameters for the extract */
 ret := OGGSERVICE.get_extract_details(
           ogg_instance  => ogg_deployment,
           extract_name  => '&extract_name',
           response_info => response_info);
```



```
config_jarray := JSON_ARRAY_T.PARSE(JSON_QUERY(response_info,
                                    '$.response.config'));

/* Add the new parameter to the existing set of parameters */
config_jarray.append('DDLOPTIONS ADDTRANDATA RETRYOP RETRYDELAY 10 MAXRETRIES 10');

/* Remove the brackets from the JSON array */
config_str := config_jarray.to_string;
config_str := replace(replace(config_str,'[',''),']','');

/* Update the revised parameters of the extract */
ret := OGGSERVICE.update_extract(
                ogg_instance   => ogg_deployment,
                extract_name   => '&extract_name',
                extract_params => config_str,
                response_info  => response_info);

/* If the update is successful then retrieve the extract details to check if the
   update went through
 */
IF ret != OGGHTTP.resp_success THEN
  pretty_json(response_info);
ELSE
  ret := OGGSERVICE.get_extract_details(
          ogg_instance  => ogg_deployment,
          extract_name  => '&extract_name',
          response_info => response_info);

  Pretty_json(response_info);
END IF;
END;
/
```

The OUTPUT shown below displays the effect of parameter file update for the extract ***EFCUG***. It may be noted that this shows the line "*DDLOPTIONS ADDTRANDATA RETRYOP RETRYDELAY 10 MAXRETRIES 10* " added to the parameter file. (Refer to OUTPUT of section 4.2.1 for the parameter file before this update)

OUTPUT


```
[
  {
    "messages" :
    [
      [
      ]
    ],
    "response" :
    [
      {
        "$schema" : "ogg:extract",
        "credentials" :
        {
          "alias" : "repdb2",
          "domain" : "OracleGoldenGate"
        },
        "begin" : "now",
        "targets" :
        [
          {
            "name" : "jq",
            "sizeMB" : 500,
            "sequenceLength" : 9,
            "sequenceLengthFlip" : false,
            "sequence" : 0,
            "offset" : 31556912,
           "remote" : false
          }
        ],
        "config" :
        [
          "extract EFCUG;",
          "useridalias repdb2 domain OracleGoldenGate",
          "exttrail jq",
          "table scott.*;",
          "table hr.*;",
          "DDLOPTIONS ADDTRANDATA RETRYOP RETRYDELAY 10 MAXRETRIES 10"
        ],
```



```
        "description" : "EFCUG",
        "source" :
        {
          "tranlogs" : "integrated"
        },
        "registration" :
        {
          "csn" : 4624869,
          "share" : true
        },
        "status" : "running"
      }
    ]
  }
]
```


4.2.3 STOP EXTRACT

The below code stops the extract ***&extract_name*** under the OGG deployment ***gg_repdb2*** on ***&machine3*** listening on port ***9900***.

```
set serverout on
DECLARE
  response_info clob;
  ret number;
BEGIN
  ret := OGGSERVICE.STOP_EXTRACT(
              ogg_instance  => ogg_connection('&machine3',9900,'gg_repdb2'),
              extract_name  => '&extract_name',
              response_info => response_info);

  pretty_json(response_info);

END;
/
```

The OUTPUT shown below is for the extract ***EFCUG***. It says that the extract is stopped.

OUTPUT


```
[
  {
    "messages" :
    [
      [
        {
          "$schema" : "ogg:message",
          "title" : "EXTRACT EFCUG stopped",
          "code" : "OGG-00975",
          "severity" : "INFO",
          "issued" : "2018-08-17T12:04:22Z",
          "type" : "http://docs.oracle.com/goldengate/c1910/gg-winux/GMESG/oggus.htm#OGG-00975"
        },
        {
          "$schema" : "ogg:message",
          "title" : "EXTRACT EFCUG stopped",
          "code" : "OGG-15426",
          "severity" : "INFO",
          "issued" : "2018-08-17T12:04:22Z",
          "type" : "http://docs.oracle.com/goldengate/c1910/gg-winux/GMESG/oggus.htm#OGG-15426"
        }
      ]
    ],
    "response" : " "
  }
]
```


4.2.4 DELETE EXTRACT

The below code deletes the extract ***&extract_name*** under the OGG deployment ***gg_repdb2*** on ***&machine3*** listening on port ***9900***.


```
set serverout on
DECLARE
  response_info clob;
  ret number;
BEGIN
  ret := OGGSERVICE.DELETE_EXTRACT(
             ogg_instance  => ogg_connection('&machine3',9900,'gg_repdb2'),
             extract_name  => '&extract_name',
             response_info => response_info);
  pretty_json(response_info);
END;
/
```

The OUTPUT below is for extract ***EFCUG.*** It may be noted that before deleting the extract it was unregistered from the database, as this is an integrated extract. Any rows corresponding to the extract in the heartbeat table is also deleted.

OUTPUT


```
[
  {
    "messages" :
    [
      [
        {
          "$schema" : "ogg:message",
          "title" : "Sending STOP request to EXTRACT EFCUG ",
          "code" : "OGG-08100",
          "severity" : "INFO",
          "issued" : "2018-08-17T12:25:03Z",
          "type" : "http://docs.oracle.com/goldengate/c1910/gg-winux/GMESG/oggus.htm#OGG-08100"
        },
        {
          "$schema" : "ogg:message",
          "title" : "EXTRACT EFCUG is down (gracefully)",
          "code" : "OGG-00979",
          "severity" : "INFO",
          "issued" : "2018-08-17T12:25:03Z",
          "type" : "http://docs.oracle.com/goldengate/c1910/gg-winux/GMESG/oggus.htm#OGG-00979"
        },
        {
          "$schema" : "ogg:message",
          "title" : "Successfully unregistered EXTRACT EFCUG from database.",
          "code" : "OGG-01750",
          "severity" : "INFO",
          "issued" : "2018-08-17T12:25:14Z",
          "type" : "http://docs.oracle.com/goldengate/c1910/gg-winux/GMESG/oggus.htm#OGG-01750"
        },
        {
          "$schema" : "ogg:message",
          "title" : "No Heartbeat entries with [EFCUG], none deleted.",
          "code" : "OGG-14052",
          "severity" : "INFO",
          "issued" : "2018-08-17T12:25:14Z",
          "type" : "http://docs.oracle.com/goldengate/c1910/gg-winux/GMESG/oggus.htm#OGG-14052"
        },
        {
          "$schema" : "ogg:message",
          "title" : "Deleted EXTRACT EFCUG.",
          "code" : "OGG-08100",
          "severity" : "INFO",
          "issued" : "2018-08-17T12:25:14Z",
          "type" : "http://docs.oracle.com/goldengate/c1910/gg-winux/GMESG/oggus.htm#OGG-08100"
        }
      ]
    ],
    "response" : " "
  }
]
```


5 TROUBLESHOOTING

Following are some of the errors that might occur, when using the sample orchestration script and their possible

reasons.

Error	Possible Reason
"file:$ORACLE_HOME/admin/ggorch_wallet is not configured for access to user: <USER>"] }	Access to wallet directory to the orchestration user is not granted. Sample code for granting access to wallet directory can be found in *oggorch_setup.sql*
"title" : "The authorization information for <service> is missing, invalid or not properly formed.",	Possibly the user name present in the certificate in $ORACLE_HOME/ggorch_wallet is not authorized to access the RESTAPI calls in the OGG deployment.
ORA-29024: Certificate validation Failure	The certificate placed in $ORACLE_HOME/ggorch_wallet is possibly not signed by a Certificate authority that is trusted by the OGG deployment.
ORA-24247: network access denied by access control list (ACL)	The DB admin user, where orchestration packages are installed, does not have privilege to make HTTPS calls. A sample code for granting HTTPS access to a user is given in *oggorch_setup.sql*
ORA-28759: failure to open file	Possibly no wallet is present in $ORACLE_HOME/admin/ggorch_wallet directory

6 CONCLUSION

It may be noticed that the provided package does not exhaustively provide functions to perform all possible actions on the OGG Microservices deployment. It does not also provide different security modes of authentication. However, it may be easily extended to provide complete functional REST API client to maintain an OGG deployment, only using PL/SQL.

7 APPENDIX – I

If the OGG deployments are configured with reverse proxy at port 443 then all the examples given above needs the below modification

```
ogg_connection('&machine2',9900,'gg_repdb1') should be modified to
ogg_connection('&machine2','gg_repdb1'),
```

e.g.

```
set serverout on
DECLARE
  response_info clob;
  ret number;
BEGIN
  ret := OGGSERVICE.DELETE_EXTRACT(
              ogg_instance  => ogg_connection('&machine2', 'gg_repdb1'),
              extract_name  => '&extract_name',
              response_info => response_info);
  pretty_json(response_info);
END;
/
```

If the reverse proxy is configured at a non-standard port then use the following modification

```
ogg_connection('&machine2',9900,'gg_repdb1') should be modified to
ogg_connection('&machine2',&reverse_proxy_port,'gg_repdb1', 1)
```

e.g.

```
set serverout on
DECLARE
  response_info clob;
  ret number;
BEGIN
```

```
  ret := OGGSERVICE.DELETE_EXTRACT(
             ogg_instance  => ogg_connection('&machine2', &reverse_proxy_port,'gg_repdb1', 1),
             extract_name  => '&extract_name',
             response_info => response_info);
  pretty_json(response_info);
END;
/
```

8 APPENDIX – II

The top level function that creates a one way replication between a source and target database is below:

```
FUNCTION ADD_ONEWAY_REPLICATION  (
   ogg_source       IN  OGG_CONNECTION,
   db_source_alias  IN  VARCHAR2,
   ogg_target       IN  OGG_CONNECTION,
   db_target_alias  IN  VARCHAR2,
   tables           IN  VARCHAR2,
   response_info   OUT  CLOB)
```

Parameter	Description
ogg_source	Connection details of source ogg deployment
db_source_alias	Alias to the source database created in the ogg_source_deployment
ogg_target	Connection details of target ogg deployment
db_target_alias	Alias to the target database created in the ogg_target_deployment
tables	Comma separated list of tables and schemas to be replicated. SCOTT schema can be specified as SCOTT.* EMP table in SCOTT schema can be specified as SCOTT.EMP. e.g 'SCOTT.*, USER1.TAB, USER2.*' The above string sets up replication between SCOTT schema from source to target USER2 schema from source to target USER1.TAB table in source to USER1.TAB table in target
response_info	response message from OGG server.

There are a number of configuration variables inside the FUNCTION that can be used to customize it for different requirements.

```
Wherever names are required (extract, replicat, path, trail) they are generated, if NULL. The following
variables can be initialized with desired value so that the initialized value is used instead of
generating them.
extract_name            VARCHAR2(5)     := '';
extract_mode            VARCHAR2(20)    := 'integrated';
path_name               VARCHAR2(5)     := '';
replicat_name           VARCHAR2(5)     := '';
replicat_mode           VARCHAR2(20)    := 'integrated';
source_trail            VARCHAR2(2)     := '';
source_trail_path       VARCHAR2(2)     := '';
target_trail            VARCHAR2(2)     := '';
target_trail_path       VARCHAR2(2)     := '';
extract_params_template VARCHAR2(4000)  :=
                    '"extract <extract_name>;",'                        ||
                    '"useridalias <db_alias> domain OracleGoldenGate",' ||
                    '"exttrail <extract_trail>",'                       ||
                    '<tables_stmt>' ;
replicat_params_template VARCHAR2(4000) :=
                    '"replicat <replicat_name>;",'                      ||
                    '"useridalias <db_alias> domain OracleGoldenGate",' ||
                    '<map_stmt>';
```


9 APPENDIX – III

The replication objects of the target database can be instantiated from the replication objects of the source database using the package OGGINSTANTIATE that is provided in the file ***ogginstantiate.sql***

The user, executing the package OGGINSTANTIATE, needs to have the following privileges

- grant execute on dbms_datapump to <user>
- grant create table to <user>

The OGGINSTANTIATE package needs to be executed from the target replication database before setting up the replication. Instantiation is done by importing objects from the source database using a database link. The package OGGINSTANTIATE is built using the code provided in the DBMS_DATAPUMP API examples.

The signature of the only procedure provided by OGGINSTANTIATE is given below

```
/*
   PROCEDURE instantiate
     Instantiates the target schema/table using the schema/table in the source
   INPUT
     source_obj    - Source object to be used for instantiation
                     can be of the form '*.*' for Full database instantiation.
                     'SCOTT.*' for schema instantiation or 'SCOTT.EMP' for table instantiation
     target_obj    - Target object to be used for instantiation
                     can be of the form '*.*' for Full database instantiation,
                     'SCOTT.*' for schema instantiation or 'SCOTT.EMP' for table instantiation
     source_dblink - Database link to the source database

*/
--------------------------------------------------------------------------------

  PROCEDURE instantiate(source_obj     IN VARCHAR2,
                        target_obj     IN VARCHAR2,
                        source_dblink  IN VARCHAR2);
--------------------------------------------------------------------------------
```

EXAMPLE – 1

Instantiate the target database, from the source database, by doing a full database import.
src_dblink is the database link to the source database

```
sqlplus> set serverout on
sqlplus> execute  OGGINSTANTIATE.INSTANTIATE('*.*', '*.*', src_dblink)
```

EXAMPLE – 2

Instantiate only scott schema of the target database from the scott schema of the source database.
src_dblink is the database link to the source database

```
sqlplus> set serverout on
sqlplus> execute  OGGINSTANTIATE.INSTANTIATE('scott.*', 'scott.*', src_dblink)
```

Note: If target_obj is specified as 'hr.' instead of 'scott.*' then the 'scott' schema of source database will be imported into 'hr' schema of the target.*

EXAMPLE – 3

Instantiate only scott.emp table of the target database with the scott.emp table of the source database
src_dblink is the database link to the source database

```
sqlplus> set serverout on
sqlplus> execute  OGGINSTANTIATE.INSTANTIATE('scott.emp', 'scott.emp', src_dblink)
```

Note: If target_obj is specified as 'hr.emp' instead of 'scott.emp' then the 'scott.emp' table of source database will be imported into 'hr.emp' table of the target.

In all the above cases, if the object that is being imported from the source database exists on the target database then the object will not be imported.

10 REFERENCES

[1] Oracle GoldenGate Microservices Architecture

[2] Self-signed Certificate creation

[3] Setting Up Secure or Non-Secure Deployments

[4] Components of Oracle GoldenGate Microservices Architecture

[5] OGG Microservices REST API Reference guide

[6] Securing Oracle GoldenGate Environment

[7] UTL_HTTP

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

Orchestration Utilities for Oracle GoldenGate Microservices using PL/SQL

Oct 2018
Author: Parthasarathy Raghunathan