ORACLE

Migration to the Oracle Cloud with an Oracle GoldenGate Hub Configuration

April 6, 2020

DISCLAIMER

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle software license and service agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

Due to the nature of the product architecture, it may not be possible to safely include all features described in this document without risking significant destabilization of the code.

TABLE OF CONTENTS

PURPOSE STATEMENT & INTENDED AUDIENCE

The intended audience for this document are individuals investigating or charged with the migration of Oracle databases to the Oracle Cloud. The document's organization and content assumes familiarity with Oracle database concepts but is suitable for both individuals well versed in Oracle GoldenGate as well as those that are new to Oracle GoldenGate and replication concepts. The primary intent of this document is to provide education and guidance in regards to how best to utilize Oracle GoldenGate to migrate an Oracle database to the Oracle Cloud with minimal downtime. This includes conceptual education, best practices, and step-by-step examples to help guide the reader.

INTRODUCTION

This Oracle GoldenGate Maximum Availability Architecture (MAA) logical migration solution provides step-by-step instructions to set up and migrate from source databases with these characteristics.

- Oracle Database 11gR2 or later releases
- Any Operating System platform (AIX, HP UX, Linux, Solaris, Windows)
- Single tenant or multitenant database
- Non-encrypted or encrypted database

You can migrate to a cloud certified encrypted database target residing in

- Exadata Cloud Service (ExaCS) in Oracle Cloud Infrastructure (OCI)
- Autonomous Database Transaction Processing Dedicated (ATP-D)

This solution uses Oracle GoldenGate on the Oracle Cloud Marketplace to create an Oracle GoldenGate Hub configuration that delivers the following advantages:

- Minimal to zero downtime
- Cross-database version support
- Cross-platform or cross-endianness support (e.g. IBM AIX to Linux)
- Single tenant or multitenant agnostic
- Reduced Oracle GoldenGate resource impact on the source and target database systems
- Can be used for multiple Oracle Database migrations

This migration solution integrates all of the benefits and flexibility of Oracle GoldenGate. However, there are some data type and operational limitations that need to be considered before using the steps documented in this paper. For more information, see Evaluate Source Database Support for Oracle GoldenGate.

The diagram below illustrates the high-level migration flow as presented in this white paper.

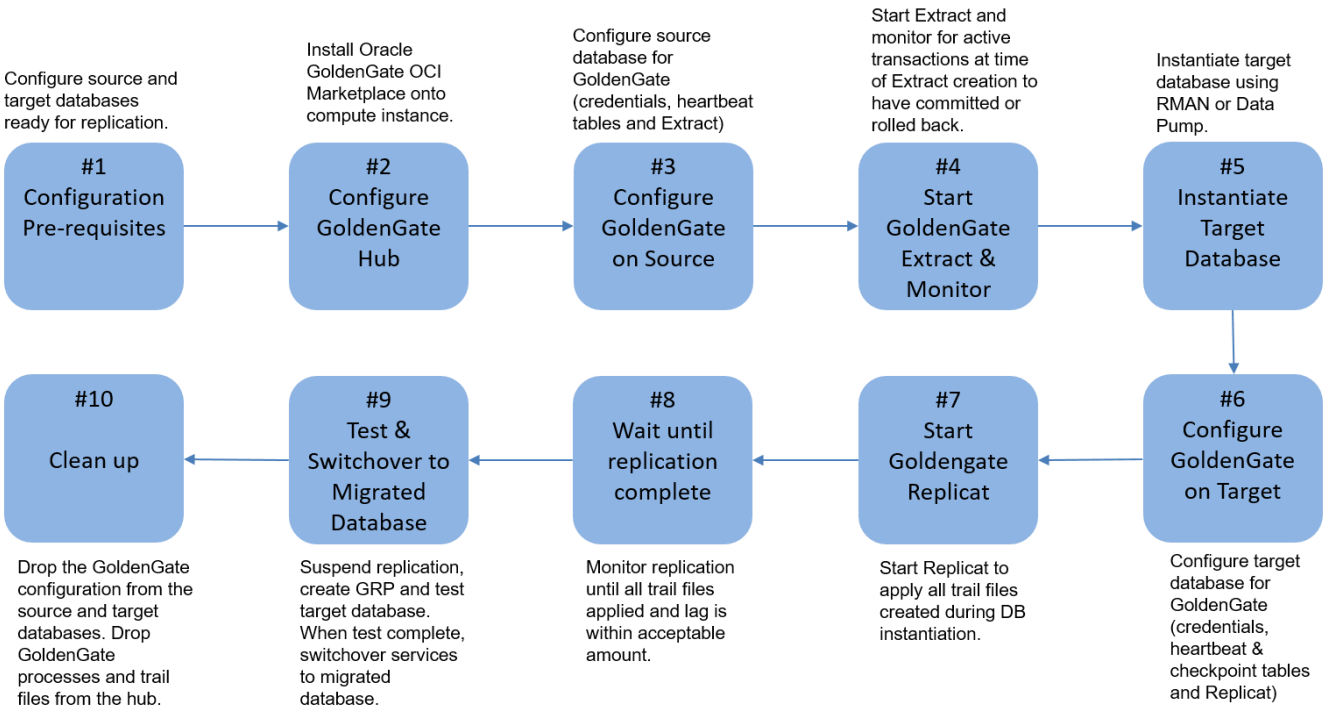

Figure 1: High-level migration flow

CONFIGURATION OVERVIEW

Prior to beginning any migration project, it is essential to understand the overall architecture of the migration solution being utilized in order to ensure best practices are followed and the solution is deployed in an ideal configuration. This section briefly introduces Oracle GoldenGate as well as the Oracle GoldenGate Microservices Architecture and how it is configured within the Oracle GoldenGate Hub solution.

Oracle GoldenGate

Oracle GoldenGate provides real-time, log-based change data capture and delivery between homogenous and heterogeneous systems. This technology enables you to construct a cost-effective and low-impact real-time data integration and continuous availability solution.

Oracle GoldenGate replicates data from committed transactions with transaction integrity and minimal overhead on your existing infrastructure. The architecture supports multiple data replication topologies such as one-to-many, many-to-many, cascading, and bidirectional. Its wide variety of use cases includes real-time business intelligence; query offloading; zero-downtime upgrades and migrations; and active-active databases for data distribution, data synchronization, and high availability.

Oracle GoldenGate Microservices Architecture was introduced in Oracle GoldenGate release 12.3, as a new administration architecture that provides REST-enabled services as part of the Oracle GoldenGate environment. The REST-enabled services provide remote configuration, administration, and monitoring through HTML5 web pages, command-line interfaces, and APIs. Figure 2 shows the Oracle GoldenGate Microservices Architecture.

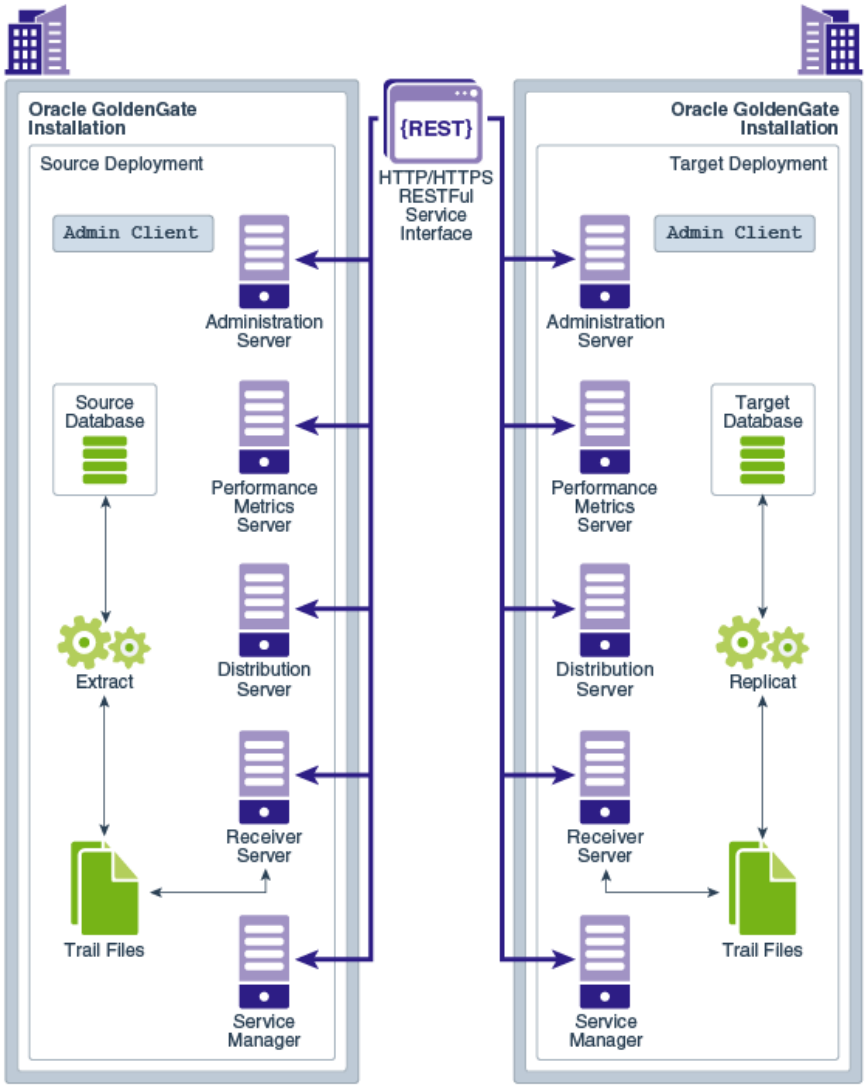

Figure 2: Oracle GoldenGate Microservices Architecture

For more information about Oracle GoldenGate Microservices Architecture, see the Oracle GoldenGate documentation.

https://docs.oracle.com/en/middleware/goldengate/core/19.1/using/getting-started-oracle-goldengate.html#GUID-61088509-F951-4737-AE06-29DAEAD01C0C

Oracle GoldenGate Hub

The Oracle GoldenGate Hub is an architectural concept that places the Oracle GoldenGate software on a different host than the databases being operated against. The GoldenGate Hub must be located in the same Oracle Cloud Infrastructure (OCI) region or preferably the same OCI availability domain as the target cloud database. The GoldenGate Hub must be in close network proximity to the target database with the expectation that network latency should never exceed 2-3ms.

The GoldenGate software and processes run on a separate server, shown in figure 3. One advantage of this architecture is that it isolates most of the GoldenGate resource usage from the source and target database servers. Another advantage is that the GoldenGate configuration can be administered and monitored from a single server, without the need of accessing a separate GoldenGate installation on each database server.

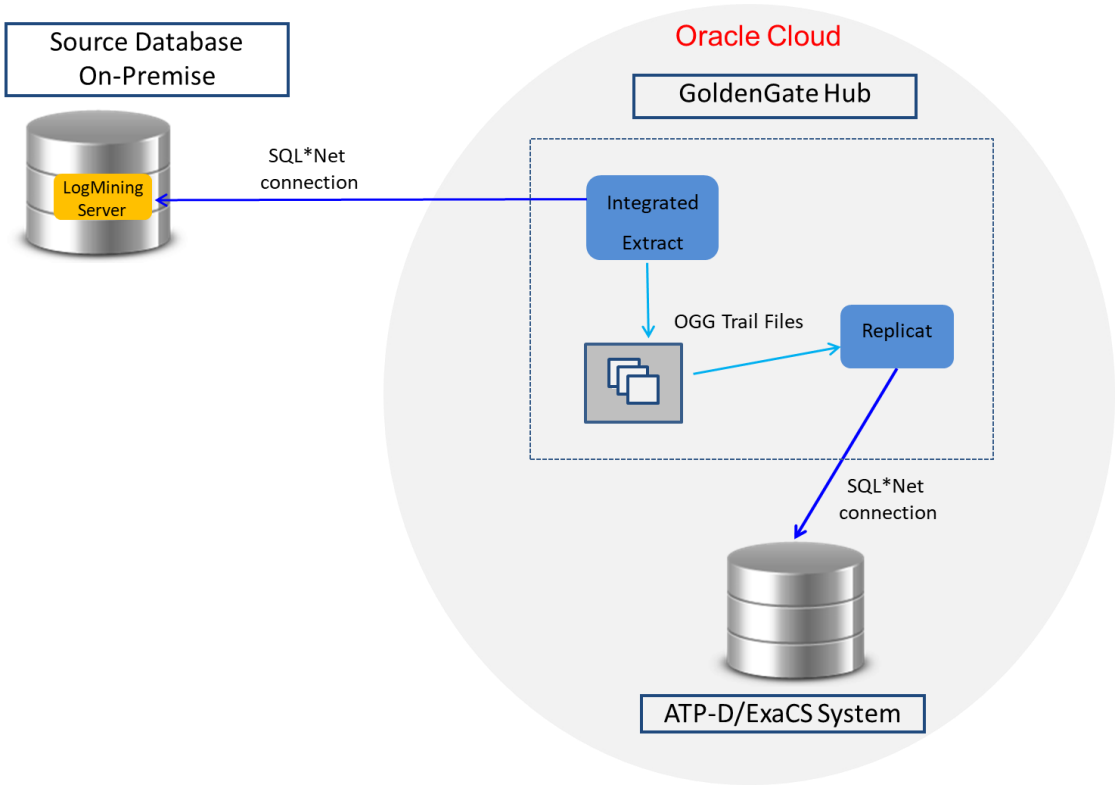

Figure 3: Oracle GoldenGate Hub architecture in the Oracle Cloud

The GoldenGate Hub software and deployment configuration is installed and configured using Oracle GoldenGate Microservices from Oracle Cloud Marketplace, which contains the latest Oracle GoldenGate Microservices release along with Oracle Database client software for all supported versions of Oracle Database. Up to two GoldenGate deployments are created as part of the automated GoldenGate installation, based on which database versions are selected for both the source and the target Oracle databases. Installation of the GoldenGate Hub is detailed in a later step (Oracle GoldenGate Hub Configuration). Oracle Cloud Marketplace and its GoldenGate Hub automation is only available in Oracle Cloud Infrastructure (OCI). For more information on Oracle GoldenGate Microservices on Oracle Cloud Marketplace, refer to the Using Oracle GoldenGate on Oracle Cloud Marketplace documentation at

https://docs.oracle.com/en/middleware/goldengate/core/19.1/oggmp/oracle-goldengate-microservices-oracle.html#GUID-06FC1224-DBC0-4145-A485-4C273F3199C2

Nginx Reverse Proxy Server

The Oracle GoldenGate reverse proxy feature allows a single point of contact for all of the GoldenGate microservices associated with a GoldenGate deployment. Without reverse proxy, the GoldenGate deployment microservices are contacted using a URL consisting of a hostname or IP address and separate port numbers, one for each of the services. For example, to contact the Service Manager you may use `http://gghub.example.com:9100`, the Administration Server is `http://gghub.example.com:9101`, the administration server of deployment #2 may be `https://gghub.example.com:9111`, and so on.

With reverse proxy, port numbers are not required to connect to the microservices, as they are replaced with the deployment name. Using the previous example, to connect to the Service Manager you use the URL `https://gghub.example.com`, the Admin Server of deployment #1 (named Source) would use `https://gghub.example.com/Source/adminsrvr`, and the administration server of deployment #2 (named Target) would be `https://gghub.example.com/Target/adminsrvr`.

The Oracle GoldenGate reverse proxy is recommended and used by default with GoldenGate from the Oracle Cloud Marketplace, to ensure easy access to microservices and provide enhanced security and manageability.

Target Database Instantiation

Before replicating data between the source and target databases with Oracle GoldenGate, both databases must contain the objects that will be replicated in a consistent state, such that they won't cause data conflicts when the Replicat process applies changes. The method of target database instantiation is driven by the source and target Oracle Database version, the endianness of the operating system of the database server, and any major database structure differences, such as migrating to an encrypted or multitenant database from one that is not.

The following instantiation methods are discussed in this paper.

1. **Data Pump export/import -** This instantiation method is required when the source and destination database versions are different, the platform endian format is different, or there is a database structure change, such as migrating from single tenant to multitenant architecture, or to an encrypted database.

   This is the only supported method for migrating an on-premises database to ATP-D. For migration to ExaCS use mv2oci (My Oracle Support note 2514026.1), and for migration to ATP-D use mv2adb (My Oracle Support note 2463574.1). Both of these tools use Data Pump to migrate data to the target database.

2. **Physical database instantiation using RMAN -** The source database can be instantiated by RMAN when the source and destination database versions and platform (endian format) are the same. This instantiation method is the fastest way to instantiate the target database.

   This method will not work for migration to ATP-D. For information about the recommended instantiation approach, based on database version, see "Creating a Physical Standby Database in an 11.2, 12.1, 12.2 or later environment" (Doc ID 2275154.1).

In both of the above cases, the source database is online throughout the instantiation.

These two instantiation methods are described in detail in the Target Database Instantiation section.

During the migration period it is recommended that you avoid certain database operations to provide the most optimal environment for fast database replication. The following should be avoiding during the database migration.

1. **Data Definition Language (DDL) -** While DDL is replicated, the Replicat serializes data to ensure that there are no locking issues between DML and DDL on the same objects.
2. **Large batch DML -** Running large batch operations, like a single transaction that affects multi-millions of rows, can slow down replication rates.

Naming Conventions Used Throughout This Paper

Throughout this paper, examples are provided for REST API endpoints to manage the Oracle GoldenGate configuration using two deployments (SOURCE and TARGET). The REST API is used so that the commands can easily be integrated into an automated database migration script. The script can be run locally or remotely from any server with access to the GoldenGate Hub with curl and python installed.

Alternatively, you can use Admin Client commands to manage the GoldenGate Hub. Admin Client is a standalone command-line interface used to create and manage GoldenGate replication. For Admin Client commands, see *Command Line Interface Reference for Oracle GoldenGate*.

https://docs.oracle.com/en/middleware/goldengate/core/19.1/gclir/index.html

REST API Arguments

The following is an example of key arguments given in the GoldenGate REST APIs used throughout this paper.

```
$ curl -s -K access.cfg https://<GG Hub>/<Deployment
Name>/adminsrvr/services/v2/credentials/goldengate -XGET | python -m json.tool
```

» **access.cfg -** To prevent the GoldenGate administrator account name and password from being exposed on the command line, it is recommended that you include the user name and password in a configuration file, which is read by curl. For example

```
user = "oggadmin:password"
```

» **GG Hub -** The hostname or IP address of the GoldenGate Hub server. For example, `gghub-server`.

» **Deployment Name -** The name of the Oracle GoldenGate deployment. For example, `SOURCE` or `TARGET`.

For example,

```
$ curl -s -K access.cfg https://gghub-server/SOURCE/adminsrvr/services/v2/credentials/goldengate -
XGET | python -m json.tool
```

REST Calls

The REST calls can be made from either the Oracle GoldenGate Hub, or any machine, on-premises or in OCI that can access the GoldenGate hub through the HTTPS protocol.

Extract and Replicat Names

The example Extract name used in this paper is **EXT1** and the Replicat is called **REP1**.

CONFIGURATION PREREQUISITES

Before you configure Oracle GoldenGate Hub, complete the prerequisites described in the following sections.

Configure the Oracle Cloud Network

For GoldenGate remote Extract to run on the hub, extracting data from the source on-premises database, additional network configuration is required to allow access between the systems.

There are three options to connect your cloud network GoldenGate Hub to the on-premises database.

IPSec VPN

Internet Protocol Security (IPSec or IP Security) is a protocol suite that encrypts the IP traffic before the packets are transferred from the source to the destination. For an overview of IPSec in OCI, see

https://docs.cloud.oracle.com/iaas/Content/Network/Tasks/overviewIPsec.htm

FastConnect

OCI FastConnect lets you create a dedicated, private connection between your data center and OCI. FastConnect provides higher-bandwidth options and a more reliable and consistent networking experience, compared to internet-based connections. For more details about FastConnect, see

https://docs.cloud.oracle.com/iaas/Content/Network/Concepts/fastconnectoverview.htm

Internet Gateway

Connectivity between OCI and on-premises systems can be achieved through the public internet as well as using an internet gateway. An internet gateway is an optional virtual router that connects the edge of the Oracle Virtual Cloud Network (VCN) with the internet.

To use the gateway, the hosts on both ends of the connection must have public IP addresses for routing. If VCN for the GoldenGate Hub doesn't have Internet Gateway, you have to add one. For information about creating an internet gateway, see

https://docs.cloud.oracle.com/iaas/Content/Network/Tasks/managingIGs.htm

Be sure to edit the ingress and egress rule in your security list to connect to the on-premises database from the GoldenGate Hub. For additional information, see

https://docs.cloud.oracle.com/iaas/Content/Network/Concepts/securitylists.htm

The remainder of this white paper assumes public internet connectivity.

Evaluate Source Database Support for Oracle GoldenGate

Oracle GoldenGate has a number of requirements of the source database, detailed below.

Database Patch Requirements

It is a best practice to apply the latest bundle patch and PSU (Patch Set Update) on both the source and target databases. The full recommended patch list can be found in My Oracle Support note 2193391.1, which covers Oracle Database release 11.2.0.4 and later releases.

It is recommended that you also apply the patch for bug 28849751 to the source database if network round trip latency between the source database and GoldenGate Hub is greater than 8ms, and if the latest database bundle patch or PS/CPU (Critical Patch Update) does not include it. Refer to My Oracle Support note 28849751.8 for patchset version information.

Data Type Support

On the source database, use the dictionary view `DBA_GOLDENGATE_SUPPORT_MODE` to determine if there are any objects that are not fully supported for extraction by Oracle GoldenGate due to data type replication limitations.

```
SQL> SELECT owner, object_name FROM DBA_GOLDENGATE_SUPPORT_MODE
WHERE support_mode NOT IN ('FULL','ID_KEY');
```

Any tables that are listed in the above query must be excluded from capture using the GoldenGate Extract parameter *`TABLEEXCLUDE owner.object_name`*. These objects must be manually copied to the target database at the end of the database migration process.

For a list of unsupported data types for Integrated Extract, see the Oracle GoldenGate documentation

https://docs.oracle.com/en/middleware/goldengate/core/19.1/oracle-db/1-understanding-whats-supported.html#GUID-110CD372-2F7E-4262-B8D2-DC0A80422806

For more details about the GoldenGate `TABLEEXCLUDE` parameter, see

https://docs.oracle.com/en/middleware/goldengate/core/19.1/reference/index.html

Row Uniqueness

Oracle GoldenGate requires a unique row identifier on the source and target tables to locate the correct target rows for replicated updates and deletes.

This is normally taken care of with primary key or unique key indexes. If there are tables identified that do not have any such keys, GoldenGate must construct a pseudo key that contains all of the allowable columns in the table, excluding virtual columns, UDTs, function-based columns, extended (32K) `VARCHAR2/NVARCHAR2` columns, and any columns that are explicitly excluded from the Oracle GoldenGate configuration by an Oracle GoldenGate user.

If the source database is version 12g Release 2 or later, use the data dictionary view `DBA_GOLDENGATE_NOT_UNIQUE` to identify all of the tables that do not have a primary key or non-null unique columns.

You can define a substitute key if the table has columns that always contain unique values. You define this substitute key by including a KEYCOLS clause within the Extract TABLE parameter and the Replicat MAP parameter. The specified key will override any existing primary or unique key that Oracle GoldenGate finds.

For more details about ensuring row uniqueness in source tables, see the Oracle GoldenGate documentation.

https://docs.oracle.com/en/middleware/goldengate/core/19.1/oracle-db/additional-oracle-goldengate-configuration-considerations.html#GUID-644099C5-8950-496C-8592-446FB1566AFD

Configure the Source Database

The source database prerequisites are described in the MAA white paper “Oracle GoldenGate Performance Best Practices” at https://www.oracle.com/technetwork/database/availability/maa-gg-performance-1969630.pdf under "Configuring the Source Database” section.

The key prerequisites are:

- Enable ARCHIVELOG mode for the database.
- Enable database force logging to ensure that all changes are found in the redo by the Oracle GoldenGate Extract process.
- Enable database minimal supplemental logging. Additional schema level supplemental logging for replicated objects also required.
- Configure the streams pool with the initialization parameter STREAMS_POOL_SIZE.
- Enable GoldenGate replication by enabling the initialization parameter ENABLE_GOLDENGATE_REPLICATION.
- Install the UTL_SPADV/UTL_RPADV package for Integrated Extract performance analysis.

Additionally, you must configure Oracle Net connectivity and create an Oracle GoldenGate database administrator account, if none exists, by following the instructions below.

1. Configure Oracle Net Connectivity

Oracle Net connectivity should be optimized to provide the best performance for the remote GoldenGate Extract running on the hub. This can be done by creating a separate Oracle Net Listener on the source database host, which also serves not to interfere with the current production listener service.

It is recommended that you set the source database listener to use a higher Session Data Unit (SDU) size due to performance improvements with a remote GoldenGate Extract. Please note that even though the listener is setting the maximum possible SDU size, the smallest size will be used if requested by the client. For example, if the listener sets SDU to 2MB but the client requests the default 8KB SDU, the connection to the database will use an 8KB SDU size. The maximum value for the SDU size for Oracle Database 11g Release 2 is 64KB (65536 bytes), and for later database releases the maximum value is 2MB (2097152 bytes).

It is recommended that you use Oracle Net or Secure Sockets Layer (SSL) encryption for connections to the source database. For more information about SSL with Oracle Net, see the *Oracle Database Security Guide*.

https://docs.oracle.com/en/database/oracle/oracle-database/19/dbseg/configuring-secure-sockets-layer-authentication.html#GUID-6AD89576-526F-4D6B-A539-ADF4B840819F

Create a Dedicated Listener

It is recommended to create a separate listener for the sole purpose of the database migration.

Creating a separate TNS_ADMIN directory on the source database host ensures separation between the current and newly configured migration listener. This new TNS_ADMIN directory will contain the sqlnet.ora and listener.ora parameter files.

If the source database is enabled for Oracle Real Application Clusters (RAC), configure the migration listener on all cluster nodes running instances for the database. This way, if one instance goes down during the migration, the service can migrate to a surviving instance. If using SSL, configure the SSL wallet on all Oracle RAC nodes.

The following example `sqlnet.ora` and `listener.ora` files are configured with SSL and an increased SDU size for Oracle Database 12g Release 1 and later.

Example sqlnet.ora file

```
SQLNET.IGNORE_ANO_ENCRYPTION_FOR_TCPS = TRUE
SQLNET.WALLET_OVERRIDE = FALSE
SQLNET.EXPIRE_TIME = 10
WALLET_LOCATION = (SOURCE=(METHOD=FILE)(METHOD_DATA=(DIRECTORY=/u01/oracle/tcps_wallets)))
SSL_VERSION = 1.2

# Parameters required for Net encryption if not using SSL Authentication, replacing
# the above parameters:
# SQLNET.ENCRYPTION_SERVER = accepted
# SQLNET.ENCRYPTION_TYPES_SERVER= (AES256)

DEFAULT_SDU_SIZE = 2097152
```

Example listener.ora file

```
Migration =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (SDU = 2097152)
      (ADDRESS = (PROTOCOL = TCPS)(HOST = <source database host>)(PORT = 2484))
    )
  )

SID_LIST_Migration =
  (SID_LIST =
    (SID_DESC =
      (SDU = 2097152)
      (SID_NAME = <ORACLE_SID>)
      (ORACLE_HOME = <ORACLE_HOME>)
    )
  )
```

Set Environment Variables and Start the Listener

To start, stop, and get status on the migration listener, make sure the following environment variables are set, and then issue the `start`, `stop`, and `status` commands.

```
export ORACLE_HOME=<oracle home directory>
export PATH=$PATH:$ORACLE_HOME/bin
export TNS_ADMIN=<TNS admin directory>
```

Start the listener.

```
$ lsnrctl start Migration
```

Stop the listener.

```
$ lsnrctl stop Migration
```

Get the current status of the listener.

```
$ lsnrctl status Migration
```

2. Create an Oracle GoldenGate Database Administrator Account

If the source database is currently part of a GoldenGate configuration the GoldenGate administrator account may already exist. If no GoldenGate administrator user exists, the user must be created. The recommended username is `GGADMIN` for single tenant and Pluggable Database (PDB), and `C##GGADMIN` for a Container Database (CDB). If the user already exists, you need to confirm the permissions are correctly granted, and if not, grant them.

Use the following instructions to verify the GoldenGate administrator account existence and to create the account, if needed.

i. Verify the Existence of an Oracle GoldenGate Administrator Account

If you are using Oracle Multitenant, also check for all PDBs, using a statement similar to the following example:

```
SQL> SELECT name, username
        FROM cdb_goldengate_privileges a, v$pdbs b
        WHERE a.con_id = b.con_id

     UNION
     SELECT decode(a.con_id,1,'CDB ROOT'), username
       FROM cdb_goldengate_privileges a, v$pdbs b
       WHERE a.con_id=1;
```

For a single tenant database use the following example.

```
SQL> SELECT username FROM dba_goldengate_privileges;
```

These queries should return no rows if GoldenGate has never been configured on the database.

If a GoldenGate administrator user already exists, it should be used for the database migration GoldenGate configuration. Make sure the permissions listed below are granted to the current GoldenGate administrator account.

Throughout this white paper the database GoldenGate administrator account will always be named `GGADMIN`.

ii. Create a New GoldenGate Administrator Account

Use the following example to create a new GoldenGate administrator account.

NOTE: If the source database is a PDB, this account should be created on ALL the PDB's you are intending to replicate.

```
SQL> create user ggadmin identified by <password>
       default tablespace users temporary tablespace temp;
SQL> grant connect, resource to ggadmin;
SQL> grant select any dictionary to ggadmin;
SQL> grant create view to ggadmin;
SQL> grant execute on dbms_lock to ggadmin;
SQL> exec dbms_goldengate_auth.GRANT_ADMIN_PRIVILEGE('ggadmin');
```

If the source database is a PDB, a separate account must be created in the CDB.

```
SQL> create user c##ggadmin identified by <password>
       default tablespace users temporary tablespace temp;
SQL> grant connect, resource to c##ggadmin;
SQL> grant select any dictionary to ggadmin;
SQL> grant create view to c##ggadmin;
```

```
SQL> grant execute on dbms_lock to c##ggadmin;
SQL> exec dbms_goldengate_auth.GRANT_ADMIN_PRIVILEGE('c##ggadmin',container=>'all');
```

NOTE: If you are only interested in replicating data from a single PDB, replace '`all`' with the PDB name.

Configure the Target Database

If the target database will be instantiated using Oracle Data Pump with mv2oci or mv2adb, the following pre-requisite steps must be completed.

Create the Target Cloud Database

The empty target database must first be created.

For migrating to ATP-D, see the Autonomous Database creation documentation.

https://docs.cloud.oracle.com/iaas/Content/Database/Tasks/adbcreating.htm

For migrating to ExaCS, see "Creating Exadata DB Systems."

https://docs.cloud.oracle.com/iaas/Content/Database/Tasks/exacreatingDBsystem.htm

Be sure to size the system tablespace, undo tablespace, temporary tablespaces, and the online redo logs at least as large as the source database.

Set Database Initialization Parameters

To allow GoldenGate Replicat to apply changes to the target database, the initialization parameter `ENABLE_GOLDENGATE_REPLICATION` must be set, as shown here.

```
SQL> ALTER SYSTEM SET enable_goldengate_replication=TRUE scope=both;
```

This parameter must be set in the CDB.

Create the GoldenGate Administrator Database Account

The GoldenGate administrator account is created as part of the ATP-D database creation, but it needs to be unlocked using the following command.

```
SQL> ALTER USER ggadmin IDENTIFIED BY <password> ACCOUNT UNLOCK;
```

For ExaCS, create the GoldenGate administrator account in the PDB into which the source database will be migrated.

Create the GoldenGate administrator account with the following commands.

```
SQL> create user ggadmin identified by <password>
       default tablespace users temporary tablespace temp;
SQL> grant connect, resource, dba to ggadmin;
SQL> grant select any dictionary to ggadmin;
SQL> grant create view to ggadmin;
SQL> grant execute on dbms_lock to ggadmin;
SQL> exec dbms_goldengate_auth.GRANT_ADMIN_PRIVILEGE('ggadmin');
```

NOTE: The DBA role is required for DDL and Sequence support. If you are not replicating DDL or Sequences, then the DBA role is not required.

ORACLE GOLDENGATE HUB CONFIGURATION

Using the Oracle GoldenGate OCI Marketplace offering, the GoldenGate Hub is configured with the following software.

- Oracle Client Software for Oracle database versions 11gR2, 12g, 18c and 19c
- Oracle GoldenGate software for Oracle database versions 11gR2, 12g, 18c and 19c
- NGINX reverse proxy server used by the Oracle GoldenGate Microservices reverse proxy
- One or two GoldenGate deployments are created, one for the source and target databases

The following prerequisites are required to deploy Oracle GoldenGate Microservices using the OCI Marketplace.

- Oracle Cloud Account
- Access to assigned Oracle Cloud Tenant
- Compute node resources within Oracle Cloud Tenant
- Local SSH/RSA Key

For complete information about installing Oracle GoldenGate from the OCI Marketplace see

https://docs.oracle.com/en/middleware/goldengate/core/19.1/oggmp/deploying-oracle-goldengate-microservices-oracle-cloud-marketplace.html#GUID-EF6680D7-7571-41E7-BF2D-831BD79BCC15

OCI Marketplace Installation

When installing Oracle GoldenGate from the OCI Marketplace use the settings detailed in the following sections.

Network Settings

Do not select ”CREATE A NEW NETWORK”, instead select an existing Virtual Cloud Network (VCN) and a subnet that matches the ATP-D or ExaCS target environment, as shown below.

This requires that you have already created a Virtual Cloud Network (VCN).

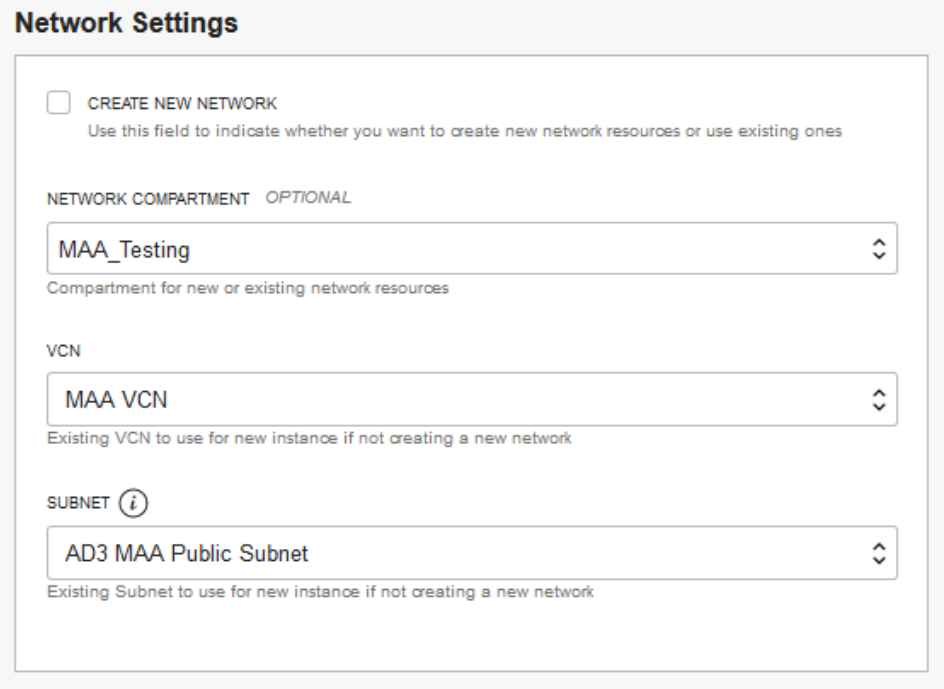

Figure 4: Example Network Settings

Instance Settings

Select the availability domain that matches the availability domain of the target ATP-D or ExaCS environment. The compute shape should be based on the following guidelines:

- VM.Standard2.4 - less than 1MB/sec peak redo rate
- VM.Standard2.8 - less than 15MB/sec peak redo rate
- VM.Standard2.16 or VM.Standard2.24 - more than 15MB/sec peak redo rate

Make sure to select 'Assign Public IP' so that the GoldenGate Hub can be accessed from the on-premises network.

The default storage sizes are the installation are shown below.

- Boot Volume Size – Default value is 50GB
- Swap Volume Size – Default value is 256GB
- Trails Volume Size – Default value is 512GB
- Deployments Volume Size – Default value is 128GB

If larger volume sizes are required, especially for the trail files, select 'Custom Volume Sizes' to increase them.

Example instance settings:

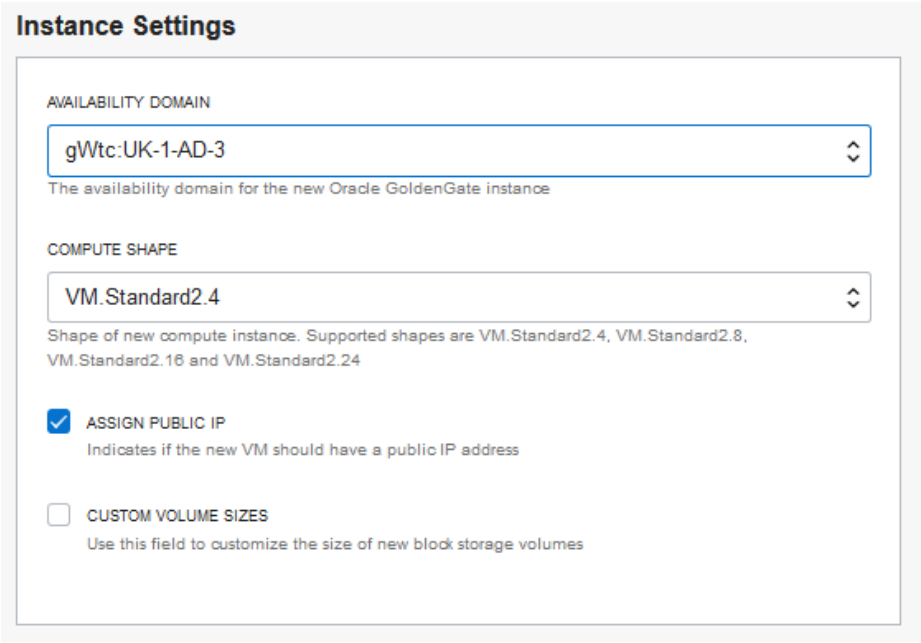

Figure 5: Example Instance Settings

Create OGG Deployments

If the source and target databases are the same, enter the deployment 1 name along with the database version. If the source and target database versions are different, also enter the deployment 2 name and database version.

Example deployment creation:

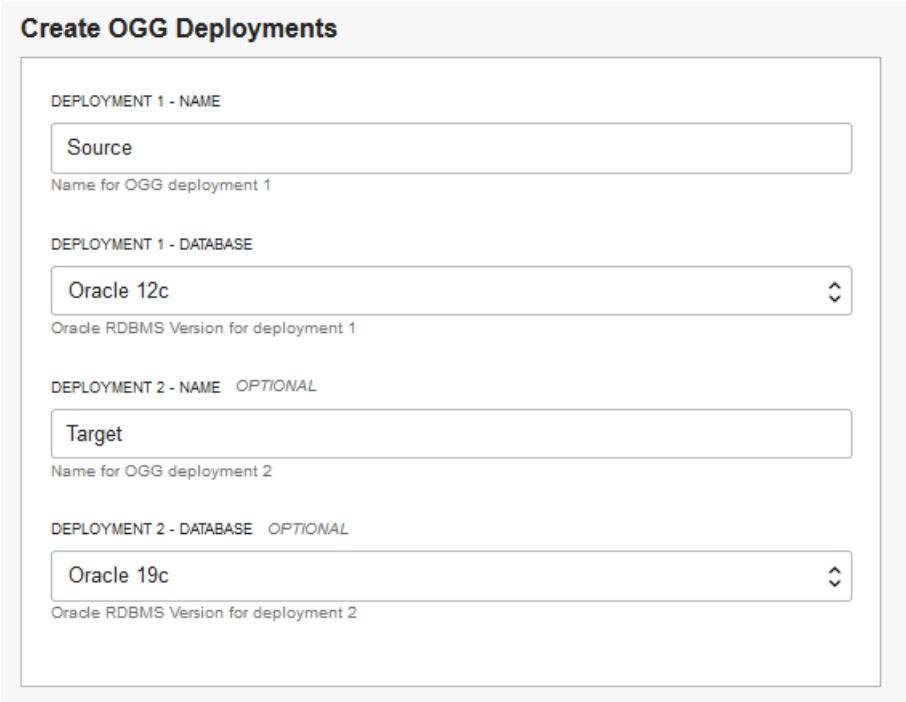

Figure 6: Example Create Oracle GoldenGate Deployments

Once the installation is complete, the GoldenGate deployments are automatically started.

Be sure to reset the Service Manager and Administration Server passwords as described in the documentation at

https://docs.oracle.com/en/middleware/goldengate/core/19.1/oggmp/getting-started-oracle-goldengate-microservices.html#GUID-7D2C4A9C-EFEA-4BE0-8AB0-92C8FC6B6258

Throughout this white paper, two GoldenGate deployments will be referenced, named SOURCE and TARGET.

The trail file storage volume created by the GoldenGate in OCI Marketplace is `/u02/trails`. This directory should be used for all trail files generated by GoldenGate.

Starting and Stopping the GoldenGate Deployments

After the GoldenGate deployments are created with the OCI Marketplace installation, and on subsequent compute instance reboots, the deployments are automatically restarted.

The GoldenGate Service Manager, which also starts the deployments, can be manually started and stopped using the following commands, when connected to the GoldenGate Hub compute instance with `ssh` as the administrator account user.

```
$ sudo systemctl start OracleGoldenGate

$ sudo systemctl status OracleGoldenGate

$ sudo systemctl stop OracleGoldenGate
```

Open the GoldenGate Hub Ingress Port

Once the GoldenGate software is configured on the hub, the ingress port 443 must be opened in the compute instance VCN security list to allow access to the GoldenGate deployments through the Nginx reverse proxy server.

Before adding a new ingress port security rule, connectivity to the GoldenGate Hub should be tested in case the default security rules are already configured to allow connectivity to the Nginx reverse proxy. To do this, retrieve the public IP address for the compute instance, like is shown below in figure 7 below.

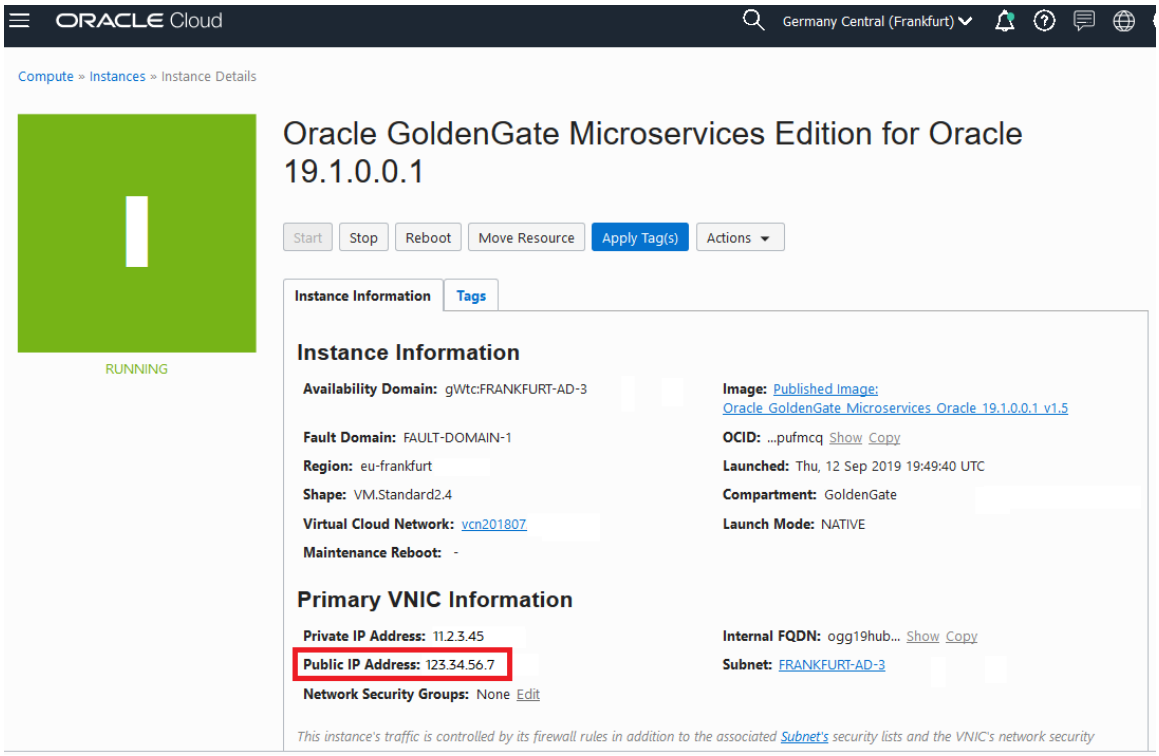

Figure 7: Retrieving GoldenGate public IP address

Using the Public IP Address, connect to the GoldenGate Service Manager from a web browser using the following URL.

```
https://<public_ip_address>
```

If the Service Manager logon page is not displayed because the ingress rule has not been added, follow the instructions below to create the ingress rule.

1. Open the navigation menu. Under **Compute**, click **Instances**.
2. Choose your **Compartment**. A list of instances is displayed.
3. Select the instance from the list and click on the instance's **Virtual Cloud Network**.
4. Locate the subnet in the list, and then click its default security list under **Security Lists**.
5. Click **Add Ingress Rules** and add an ingress rule with the following parameter values:
   - **Source Type -** Set to CIDR
   - **Source CIDR -**This should either be set to a named source IP address, for example 12.34.5.67/32 (recommended) or 0.0.0.0/0 to allow any machine access to the GoldenGate Hub.
   - **Protocol -** set to TCP
   - **Destination Port Range -** set to 443

   The following graphic shows the example rule configuration.

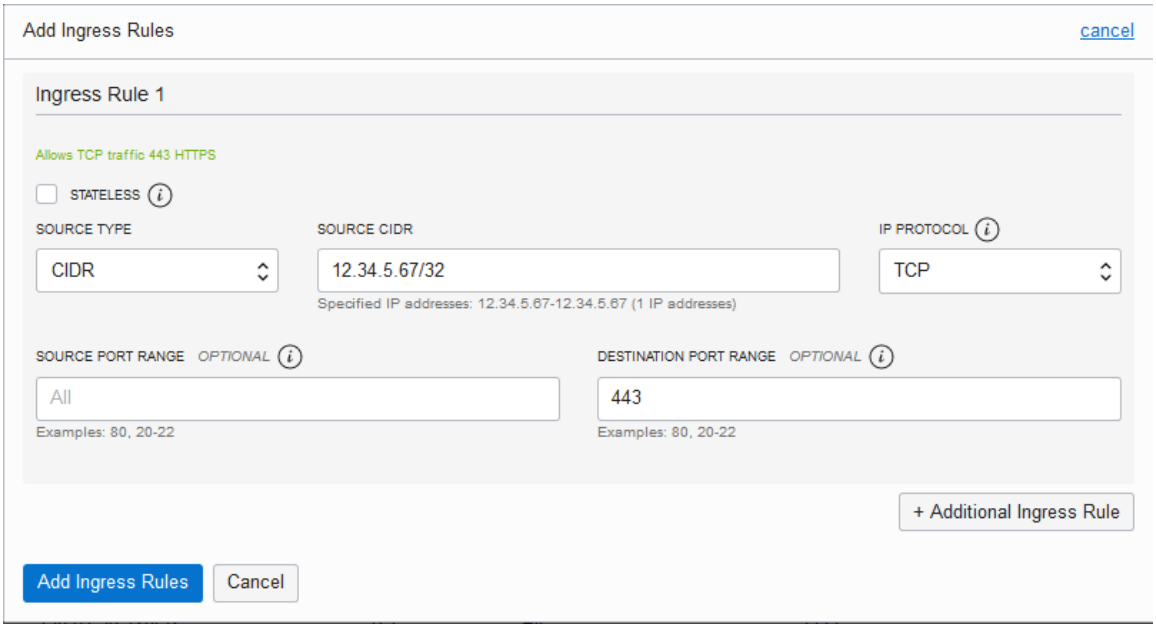

Figure 8 Add Ingress Rules example

6. Re-test the connectivity to the GoldenGate Hub using public IP address.

   For detailed information about creating or updating a security list, see https://docs.cloud.oracle.com/iaas/Content/Network/Concepts/securitylists.htm

NOTE: Because the IP address of the GoldenGate Hub is not registered with DNS (Domain Name Registration) you should create a local hosts file entry for the hub server.

For example,
123.34.56.7 gghub.example.com

7. Test Oracle GoldenGate Microservices Connectivity

A simple test is to query the health of the deployments using the following command. The command can be run from either the named IP address or any on-premises server with curl and python installed, that was specified for the source CDIR above (Open GoldenGate Hub Ingress Port).

```
$ curl -s -K access.cfg https://gghub.example.com/services/v2/config/health -XGET | python -m
json.tool
```

Sample output:

```
{
    "$schema": "api:standardResponse",
    "links": [
        {
            "href": "https:// gghub.example.com/services/v2/config/health",
            "mediaType": "application/json",
            "rel": "canonical"
        },
        {
```

```
...
    "response": {
        "$schema": "ogg:health",
        "criticalResources": [
            {
                "deploymentName": "source",
                "healthy": true,
                "name": "adminsrvr",
                "status": "running",
                "type": "service"
            },
            {
                "deploymentName": "target”,
                "healthy": true,
                "name": "adminsrvr",
                "status": "running",
                "type": "service"
            },
        ],
        "deploymentName": "ServiceManager",
        "healthy": true,
        "serviceName": "ServiceManager",
        "started": "2019-09-26T21:14:08.258Z"
    }
}
```

ORACLE GOLDENGATE CONFIGURATION

This section describes the Oracle GoldenGate configuration steps that must be done on the GoldenGate Hub before instantiating the target database.

Create the Source GoldenGate Credentials

The GoldenGate database credentials are required to allow GoldenGate processes to connect to the source and target databases. Using a GoldenGate credential store prevents clear text passwords from being stored in any of the Extract or Replicat parameter files. Because the target database has not yet been created, only the source database credential is created in this step. The target database credential is created in a later step.

On the GoldenGate Hub the `tnsnames.ora` and `sqlnet.ora` files will be located in each deployment client software database version directory. For example, if the source database is Oracle 12c the `TNS_ADMIN` directory is `/u01/app/client/oracle12/network/admin`. If the target database is Oracle 19c, the `TNS_ADMIN` directory is `/u01/app/client/oracle19/network/admin`.

Prepare the Oracle Net Configuration Files

The database credential uses the Oracle Easy Connect naming method, which does not require the use of a local `tnsnames.ora` file, unless the database version is earlier than release 19c and using SSL authentication. In those cases, Easy Connect naming cannot be used, and `tnsnames.ora` is required.

The following is an example `tnsnames.ora` entry for a pre-19c source Oracle RAC database with SSL authentication.

```
GGSOURCE =
  (DESCRIPTION=
    (SDU=2097152)    # Or set to 65536 for 11.2.0.4 database
    (CONNECT_TIMEOUT=10)(RETRY_COUNT=3)
    (ADDRESS=(PROTOCOL=TCPS)(HOST=<primary db scan address>)(PORT=2484))
    (CONNECT_DATA=(SERVICE_NAME=<db service name>))
  )
```

If the source database is using SSL authentication and the database is an earlier version than 19c, a `sqlnet.ora` file with the following parameters is required.

```
SSL_CLIENT_AUTHENTICATION=TRUE
SSL_SERVER_DN_MATCH=OFF
SQLNET.EXPIRE_TIME = 10
SQLNET.WALLET_OVERRIDE = FALSE
WALLET_LOCATION = (SOURCE=(METHOD=FILE)(METHOD_DATA=(DIRECTORY="/u01/oracle/network")))
SSL_VERSION = 1.2
DEFAULT_SDU_SIZE=2097152     # Change to 65536 for 11.2.0.4 database
```

NOTE: Make sure you copy the SSL wallet from the source database environment to the `DIRECTORY` directory on the GoldenGate Hub.

Lastly, if the source database uses Oracle Net encryption instead of SSL authentication, and the database is of any version, a `sqlnet.ora` file is required with the following parameters.

```
SQLNET.ENCRYPTION_CLIENT = required
SQLNET.ENCRYPTION_TYPES_CLIENT= (AES256)
# The SDU size is only required for databases earlier than 19c:
DEFAULT_SDU_SIZE=2097152     # Change to 65536 for 11.2.0.4 database
```

Oracle Database 19c enhanced the Easy Connect naming method, renaming it to Easy Connect Plus. Refer to the *Oracle Net Administrators Guide* for more information about the Easy Connect Plus naming method.

https://docs.oracle.com/en/database/oracle/oracle-database/19/netag/configuring-naming-methods.html#GUID-8C85D289-6AF3-41BC-848B-BF39D32648BA

Create the GoldenGate Database Credentials

If you are using a `tnsnames.ora` file, create the credential to the source database using the TNS alias.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/credentials/goldengate/source -X POST --data
'{"userid":"ggadmin@GGSOURCE", "password":"<password>"}' | python -m json.tool
```

If you are not using a `tnsnames.ora` file, but the database version is an earlier release than 19c, create the credential to the source database using the following Easy Connect naming example (with a SCAN listener).

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/credentials/goldengate/source -X POST --data
```

```
'{"userid":"ggadmin@database-host-scan:1521/database_service_name", "password":"<password>"}' |
python -m json.tool
```

The following example shows credential creation for a 19c source database that is configured with Oracle RAC and SSL authentication.

```
curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/credentials/goldengate/source -X POST --data
'{"userid":"ggadmin@tcps://database-host-scan:2484/db_service_name?sdu=2097152&
SSL_SERVER_DN_MATCH=NO&ssl_server_cert_dn='cn=common_name'&wallet_location='/u01/oracle/network'",
"password":"<password>"}' | python -m json.tool
```

The following is an example of creating a credential that is using Oracle Net encryption, enabled in the `sqlnet.ora` file, with an Oracle RAC database.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/credentials/goldengate/source -X POST --data
'{"userid":"ggadmin@tcp://database-host-scan:1521/db_service_name?sdu=2097152",
"password":"<password>"}' | python -m json.tool
```

For a multitenant database, separate credentials must be created for the CDB and the PDB being migrated using the guidelines above.

To show the GoldenGate credentials created, use the following commands.

1. Show the name of the GoldenGate credentials for a deployment

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/credentials/goldengate -XGET | python -m
json.tool | grep name

"name": "source_cdb"
"name": "source_pdb"
```

2. Show connection details for the credential.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/credentials/goldengate/source_cdb -XGET |
python -m json.tool

    "response": {
        "$schema": "ogg:credentials",
        "userid": "ggadmin@tcps://database-host-
scan:2484/db_service_name?sdu=2097152&SSL_SERVER_DN_MATCH=NO&ssl_server_cert_dn='cn=common_name'&wa
llet_location='/u01/oracle/network'"
    }
```

3. To delete a credential, use the following command.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/credentials/goldengate/source_cdb -X DELETE
| python -m json.tool
```

Create the Heartbeat Table on the Source Database

The GoldenGate heartbeat table is required to monitor the replication latency between the source and target databases. If the source database already contains GoldenGate heartbeat objects, you must use them.

1. Check if the heartbeat table already exists, use the source database credentials previously created, as shown below.

NOTE: For a multitenant database, the heartbeat table must be located in the PDB being replicated. Be sure to provide the source PDB credential for checking the heartbeat table.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/connections/goldengate.source_pdb/tables/hea
rtbeat -XGET | python -m json.tool
```

The following example output shows that heartbeat tables are NOT currently configured.

```
    "messages": [
        {
            "$schema": "ogg:message",
            "code": "OGG-08100",
            "issued": "2019-09-26T23:06:38Z",
            "severity": "INFO",
            "title": "Heartbeat table ggadmin.gg_heartbeat does not exist.",
        },
        {
            "$schema": "ogg:message",
            "code": "OGG-08100",
            "issued": "2019-09-26T23:06:38Z",
            "severity": "INFO",
            "title": "Heartbeat table ggadmin.gg_heartbeat_seed does not exist.",
        },
        {
            "$schema": "ogg:message",
            "code": "OGG-08100",
            "issued": "2019-09-26T23:06:38Z",
            "severity": "INFO",
            "title": "Heartbeat table ggadmin.gg_heartbeat_history does not exist.",
        }
    ]
```

The following example output shows that the GoldenGate heartbeat table already exists:

```
    "messages": [
        {
            "$schema": "ogg:message",
            "code": "OGG-08100",
            "issued": "2019-09-26T23:31:08Z",
            "severity": "INFO",
            "title": "HEARTBEAT table ggadmin.gg_heartbeat exists.",
        },
...
```

```
        {
            "$schema": "ogg:message",
            "code": "OGG-08100",
            "issued": "2019-09-26T23:31:08Z",
            "severity": "INFO",
            "title": "HEARTBEAT table ggadmin.gg_heartbeat_seed supplemental logging ENABLED.",
        },
...
    "response": {
        "$schema": "ogg:tablesHeartbeat",
        "addTrandata": true,
        "frequency": 300,
        "partitioned": false,
        "purgeFrequency": 1,
        "retentionTime": 30,
        "targetOnly": false
    }
```

If a heartbeat table is already in use, the same tables are used by the new migration GoldenGate Extract.

2. If the existing heartbeat table's frequency is not the default 60 seconds, note the current value and modify it to 60 seconds for the duration of the migration.

NOTE: Remember what the frequency value is BEFORE changing it so you can revert it after the migration is complete.

Modify the heartbeat update frequency.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/connections/goldengate.source_pdb/tables/hea
rtbeat -X PATCH --data '{"frequency":60}'| python -m json.tool
```

3. If the heartbeat tables do not exist, create the heartbeat tables in the source database.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/connections/goldengate.source_pdb/tables/hea
rtbeat -X POST
--data '{}'| python -m json.tool
```

NOTE: The target database heartbeat tables are created after the database is instantiated.

Prepare the Source Database Schemas for Instantiation

You must prepare for replication the source schemas for all of the database schemas that are part of the database migration and will be replicated.

NOTE: Preparing the database schemas for instantiation MUST be done before the GoldenGate Extract process is created.

1. Use the following command to instantiate each database schema.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/connections/goldengate.source_pdb/trandata/s
chema -X POST --data '{"operation":"add","schemaName":"soesmall", "prepareCsnMode":"nowait"}' |
python -m json.tool
```

2. Use the following example commands to check each prepared schema in the source database.

   Check single schema with the following command.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/connections/goldengate.source_pdb/trandata/s
chema -X POST --data '{"operation":"info","schemaName":"soesmall"}'| python -m json.tool
```

   Check for all database schemas (from which you can pull out the schemas you are replicating) with the following command.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/connections/goldengate.source_pdb/trandata/s
chema -X POST --data '{"operation":"info","schemaName":"*"}'| python -m json.tool
```

Create and Start the GoldenGate Extract Process

The following section describes how to verify the existence of the Extract, add an Extract, start and stop the process, and delete the Extract.

1. Check for Extract existence (in case it was previously created and not removed).

```
$ curl -s -K access.cfg https://gghub.example.com /SOURCE/adminsrvr/services/v2/extracts/EXT1 -X
GET | python -m json.tool
```

   The following example shows the response if the Extract does NOT exist.

```
    "messages": [
        {
            "$schema": "ogg:message",
            "code": "OGG-12029",
            "issued": "2019-09-26T04:52:41Z",
            "severity": "INFO",
            "title": "The extract with name 'EXT1' does not exist.",
        }
```

   The following example shows the response if the Extract does exist.

```
   "messages": [],
    "response": {
        "$schema": "ogg:extract",
        "begin": "now",
        "config": [
            "Extract EXT1",
            "ExtTrail aa",
            "UseridAlias source_cdb DOMAIN goldengate",
```

```
            "TRANLOGOPTIONS PERFORMANCEPROFILE HIGH",
            "TRANLOGOPTIONS _readaheadcount 64"
            "REPORTCOUNT EVERY 15 MINUTES, RATE",
            "STATOPTIONS REPORTFETCH",
            "DDL EXCLUDE ALL",
            "Table pdb1.soesmall.*;"
        ],
        "credentials": {
            "alias": "source_cdb",
            "domain": "goldengate"
        },
        "registration": {
            "containers": [
                "PDB1"
            ],
            "csn": 51162830
        },
        "source": {
            "tranlogs": "integrated"
        },
        "status": "stopped",
        "targets": [],
        "type": "Integrated"
    }
```

2. Create the Extract process.

   The following are the minimum recommended Extract parameters.

```
-- Replace with trail file naming standard

EXTTRAIL aa
TRANLOGOPTIONS PERFORMANCEPROFILE HIGH
TRANLOGOPTIONS _readaheadcount 64   -- Required for streaming protocol (bug fix 28849751)
DISCARDFILE APPEND
REPORTCOUNT EVERY 15 MINUTES, RATE
STATOPTIONS REPORTFETCH

-- It is recommended to prevent DDL on the source database during the database
-- migration:
DDL EXCLUDE ALL

-- Repeat TABLE command for each schema being replicated:
TABLE [ container. ]<schema_name>.*;

-- Repeat TABLEEXCLUDE command for each table that was highlighted as not supported
-- for GoldenGate replication:
TABLEEXCLUDE [ container. ]<schema_name>.<tablename>;
```

   If the source database is a PDB, the Extract must connect to the CDB, because it must read the entire database redo stream. The objects being replicated from the PDB are identified by the `TABLE` parameter.

   Create the Extract using the following command.

```
$ curl -s -K access.cfg https://gghub.example.com/SOURCE/adminsrvr/services/v2/extracts/EXT1 -X
POST --data '{"config":["Extract EXT1", "ExtTrail /u02/trails/aa","UseridAlias source_cdb DOMAIN
goldengate”,"TRANLOGOPTIONS PERFORMANCEPROFILE HIGH", "TRANLOGOPTIONS _readaheadcount 64",
"REPORTCOUNT EVERY 15 MINUTES, RATE","STATOPTIONS REPORTFETCH","DDL EXCLUDE ALL","Table
pdb1.soesmall.*;"],"source":{"tranlogs":"integrated"},
"credentials":{"alias":"source_cdb","domain":"goldengate"},"begin":"now","targets":[{"path":"/u02/t
rails","name":"aa","sizeMB":500}],"registration":{"containers":["pdb1"]}}'| python -m json.tool
```

NOTE: The `_readaheadcount` parameter is required after the patch for bug 28849751 is applied to the source on-premises database.

If you are extracting multiple schemas, use multiple `Table <Source PDB>.<Schema Name>.*;` parameters.

Create the Extract for a non-CDB database using the following command.

```
$ curl -s -K access.cfg https://gghub.example.com/SOURCE/adminsrvr/services/v2/extracts/EXT1 -X
POST --data '{"config":["Extract EXT1", "ExtTrail /u02/trails/aa","UseridAlias source_db DOMAIN
goldengate","TRANLOGOPTIONS PERFORMANCEPROFILE HIGH", "TRANLOGOPTIONS _readaheadcount 64",
"REPORTCOUNT EVERY 15 MINUTES, RATE","STATOPTIONS REPORTFETCH","DDL EXCLUDE ALL","Table
soesmall.*;"],"source":{"tranlogs": "integrated"},
"credentials":{"alias":"source_db","domain":"goldengate"},
"begin":"now","targets":[{"path":"/u02/trails","name":"aa","sizeMB":500}],"registration":"default"}
'| python -m json.tool
```

3. Confirm the Extract configuration.

```
$ curl -s -K access.cfg https://gghub.example.com/SOURCE/adminsrvr/services/v2/extracts/EXT1 -X GET
| python -m json.tool
```

4. Check status of the Extract.

   The Extract should not be started automatically, so the status should be stopped.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/extracts/EXT1/info/status -X GET | python -m
json.tool
```

The following example shows the expected result with Extract NOT started.

```
    "messages": [],
    "response": {
        "$schema": "ogg:extractStatus",
        "lag": 0,
        "lastStarted": null,
        "position": "0.0",
        "sinceLagReported": 208,
        "status": "stopped"
    }
```

5. Start the Extract.

```
$ curl -s -K access.cfg https://gghub.example.com/SOURCE/adminsrvr/services/v2/commands/execute -X
POST --data '{"name":"start","processName":"EXT1"}' | python -m json.tool
```

Wait 1-2 minutes and then check the status of the Extract.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/extracts/EXT1/info/status -X GET | python -m
json.tool
```

The output after a successful start should be similar to the following.

```
    "response": {
        "$schema": "ogg:extractStatus",
        "lag": 1,
        "lastStarted": "2019-09-26T19:09:45.524Z",
        "position": "0.51251093",
        "processId": 4702,
        "sinceLagReported": 2,
        "status": "running"
    }
```

Create Autostart Tasks

Autostart and autorestart profiles are needed to automatically start the GoldenGate processes when the deployment is started. By default GoldenGate processes are not part of an active profile to automatically start them.

1. Check to see if an autostart profile exists, and if one does, you can delete it or update it.

   Check for the existence of the autostart profile.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/config/types/ogg:managedProcessSettings/valu
es/ogg:managedProcessSettings:MIGRATE01 -XGET | python -m json.tool
```

The following example output displays when the profile doesn't exist.

```
    "messages": [
        {
            "$schema": "ogg:message",
            "code": "OGG-12029",
            "issued": "2019-09-26T17:22:43Z",
            "severity": "INFO",
            "title": "The value with name 'ogg:managedProcessSettings:MIGRATE01' does not exist.",
        }
    ]
```

When the profile does exist, you'll see the following output.

```
    "response": {
        "autoRestart": {
            "delay": 60,
            "disableOnFailure": true,
```

```
            "enabled": true,
            "onSuccess": false,
            "retries": 5,
            "window": 1200
        },
        "autoStart": {
            "delay": 60,
            "enabled": true
        }
    }
```

If the profile already exists, then it can either be deleted or updated with the settings recommended below.

To delete the profile:

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/config/types/ogg:managedProcessSettings/valu
es/ogg:managedProcessSettings:MIGRATE01 -XDELETE | python -m json.tool
```

To update the current profile with recommended settings:

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/config/types/ogg:managedProcessSettings/valu
es/ogg:managedProcessSettings:MIGRATE01 -XPUT --data '{"autoStart": {"enabled": true,"delay":
10},"autoRestart": {"delay": 30,"disableOnFailure": true,"enabled": true,"onSuccess":
false,"retries": 5,"window": 1200}}' | python -m json.tool
```

2. Create the following autostart and autorestart profiles for both the source and target deployments.

   If the profile doesn't currently exist, or was deleted in the previous step, create the profile.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/config/types/ogg:managedProcessSettings/valu
es/ogg:managedProcessSettings:MIGRATE01 -XPOST --data '{"autoStart": {"enabled": true,"delay":
60},"autoRestart": {"delay": 60,"disableOnFailure": true,"enabled": true,"onSuccess":
false,"retries": 5,"window": 1200}}' | python -m json.tool
```

3. Assign the autostart profile to the Extract.

   NOTE: You cannot yet assign the profile to the GoldenGate Replicat because it has not yet been created.

```
$ curl -s -K access.cfg https://gghub.example.com/SOURCE/adminsrvr/services/v2/extracts/EXT1 -X
PATCH --data '{"managedProcessSettings": "MIGRATE01"}' | python -m json.tool
```

Confirm that the Extract is configured with the new profile (partial output shown).

```
$ curl -s -K access.cfg https://gghub.example.com/SOURCE/adminsrvr/services/v2/extracts/EXT1 -X GET
| python -m json.tool | grep managedProcessSettings

   "managedProcessSettings": "MIGRATE01",
```

Monitor the Source Database for Long Running Transactions

Before the database can be used to instantiate the target database, it is important that you make sure any transactions that were active at the time the GoldenGate Extract was created have committed to ensure that GoldenGate replicates all transactional data.

For example, consider the following timeline.

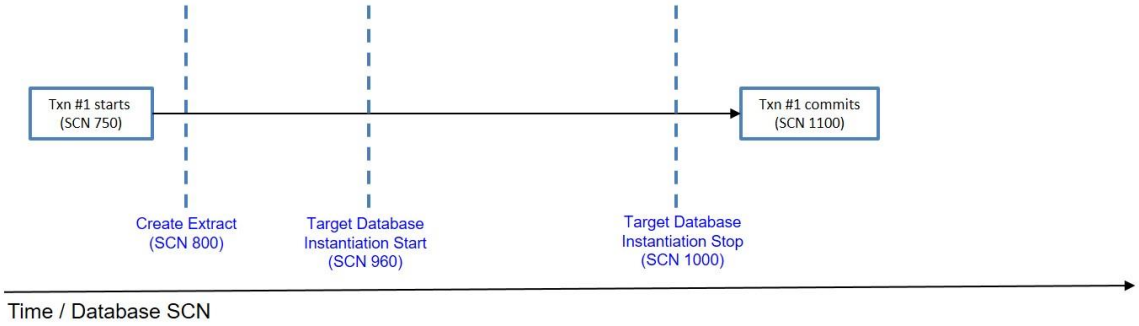

Figure 9: Long-running transaction timeline

Transaction #1 commits on the source database at SCN 1100, but it is not captured by Extract because Extract only processes complete transactions, and the process was created after the transaction had begun. Because SCN 750 is less than 800, this transaction is ignored.

Use the following query on the source database to determine when all of the transactions that were started before the Extract was created have committed or rolled back, making it safe to start instantiating the target database.

```
SQL> SELECT capture_name, c.start_scn, t.start_scn
FROM dba_capture c, v$transaction t WHERE t.start_scn < c.start_scn;
```

When the query returns no rows, it is safe to begin the target database instantiation.

TARGET DATABASE INSTANTIATION

Oracle Data Pump (mv2oci/mv2adb)

Oracle Data Pump provides a method to instantiate the target database when the source and target database versions are different, the hardware platform is different, or the database structure is different. For example, when migrating to a non-encrypted database to an encrypted database.

When the source database schemas are prepared for replication and the GoldenGate Extract is created (see Prepare the Source Database Schemas for Instantiation) the subsequent Data Pump Export dump file will contain table instantiation SCNs marking the SCN at which each table is consistent to. When the GoldenGate Replicat applies trail file data that was extracted before the instantiation SCNs, it is ignored. Once the instantiation SCN is reached, for each replicated table, Replicat applies the replicated transactions. This way we ensure zero data loss, and prevention of applying duplicate data.

Refer to the following My Oracle Support (MOS) notes for migrating data to the target database using mv2oci and mv2adb tools.

- For an ATP-D target database refer to MOS note 2463574.1 for using mv2adb.
- For an ExaCS target database refer to MOS note 2514026.1 for using mv2oci.

Note the following when using Data Pump export/import with mv2adb/mv2oci when migrating your on-premises database:

- Monitor the Data Pump export/import progress with the `V$SESSION_LONGOPS` data dictionary view.
- Because the objects being replicated by Oracle GoldenGate were already prepared in an earlier step, after the objects are imported into the target database, Replicat can determine which transactions from the trail files to apply without applying duplicate data. Because of this, use the mv2adb/mv2oci parameter `--goldengate` if not using the '`auto -netlink`' option.

Oracle RMAN Duplicate Database

Duplicating a source database using RMAN is an alternative method to copy an entire database if the source and destination platform endianness is compatible, and the Oracle Database versions are the same. This method of database instantiation cannot be used when migration to ATP-D. Complete details about using RMAN to duplicate the source database to migrated to ExaCS can be found in the Oracle Cloud Infrastructure documentation at

https://docs.cloud.oracle.com/iaas/Content/Database/Tasks/mig-rman-duplicate-active-database.htm

After the target database has been opened after duplicating it, there are some additional configuration steps that need to be done.

1. When the target database is opened after duplication, the SCN to which the database was recovered must be recorded. The SCN is used in a later step to start the Oracle GoldenGate Replicat process. You can get the SCN from the target database `alert.log` or from the data dictionary. Both examples are shown below.

   Get the SCN from the `alert.log` entry.

```
RESETLOGS after incomplete recovery UNTIL CHANGE 681543060 time 05/01/2019 15:05:20
```

   Get the SCN using a data dictionary query.

```
SQL> select RESETLOGS_CHANGE# - 1 from v$database;
```

   NOTE: Record this SCN for later use when you start the GoldenGate Replicat process.

2. Confirm that the database initialization parameter, `ENABLE_GOLDENGATE_REPLICATION`, is set to TRUE. If it is not, set it using the following command.

```
SQL> alter system set enable_goldengate_replication=TRUE scope=both;
```

COMPLETE THE ORACLE GOLDENGATE CONFIGURATION

This section describes the Oracle GoldenGate configuration required after the target database is instantiated.

Create the Target Database GoldenGate Credentials

If the target database was instantiated using Oracle Data Pump, the GoldenGate database administrator account should have already been created in a previous step, Create GoldenGate administrator database account.

The GoldenGate database credentials are required to allow the GoldenGate processes connectivity to the target database. The GoldenGate credential for the target database must connect to the PDB, and not the CDB.

The target GoldenGate credentials are created similar to the source database credentials specified in the earlier Create the Source GoldenGate Credentials with the following additional notes.

NOTE: If the target database is multitenant, the GoldenGate credential will connect to the PDB that was previously instantiated.

ATP-D Target Database

For ATP-D, the Oracle client credentials (wallet files) need to be downloaded from the Oracle Cloud Infrastructure console. For details on downloading the credentials, refer to the ATP-D documentation at https://docs.oracle.com/en/cloud/paas/atp-cloud/atpud/download-client-credentials.html#GUID-9F5AD1CB-5DAD-44C4-8978-C485575E23C2

The wallet files should be placed in a directory on the GoldenGate Hub and referenced below with the `wallet_location` parameter.

It is recommended to use the predefined database service name `tp_tls`. For example, using Oracle Easy Connect Plus naming method alias to create the target database credential:

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/credentials/goldengate/target -X POST --data
'{"userid":"ggadmin@tcps://database-host-scan:2484/atpd_tp_tls.atp.oraclecloud.com?sdu=2097152&
SSL_SERVER_DN_MATCH=NO&ssl_server_cert_dn='cn=common_name'&wallet_location='/u01/oracle/network'",
"password":"<password>"}' | python -m json.tool
```

ExaCS Target Database

For ExaCS, connectivity to the database is configured to use either the database host public IP address or the SCAN listener name that has been registered with your DNS administrator. The default listener port configured in ExaCS is port 1521 and the TCP protocol.

It is recommended to use Oracle Net encryption between the GoldenGate Hub and the target database in ExaCS.

Use the following parameters in the GoldenGate Hub `sqlnet.ora` file to enable Oracle Net encryption.

```
SQLNET.ENCRYPTION_CLIENT = required
SQLNET.ENCRYPTION_TYPES_CLIENT= (AES256)
```

Further information on connecting to the ExaCS database can be found at
https://docs.oracle.com/en/cloud/paas/exadata-cloud/csexa/connect-db-using-net-services.html

The following is an example of creating a credential that is using Oracle Net encryption, enabled in the `sqlnet.ora` file, to the first node of an ExaCS Oracle database.

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/credentials/goldengate/target -X POST --data
'{"userid":"ggadmin@tcp://<public_ip_address>:1521/db_service_name", "password":"<password>"}' |
python -m json.tool
```

To show the GoldenGate credentials created, use the following commands.

1. Show the name of the GoldenGate credentials for a deployment.

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/credentials/goldengate -XGET | python -m
json.tool | grep name

"name": "target"
```

2. Show connection details for the credential.

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/credentials/goldengate/target -XGET | python
-m json.tool
```

Create the Target Database GoldenGate Checkpoint Table

The checkpoint table is a required component of nonintegrated Parallel Replicat, which is recommended for use in this whitepaper.

A nonintegrated Replicat maintains its recovery checkpoints in the checkpoint table, which is stored in the target database. Checkpoints are written to the checkpoint table within the Replicat transaction. Because a checkpoint either succeeds or fails with the transaction, Replicat ensures that a transaction is only applied once, even if there is a failure of the process or the database.

1. If the target database was created using RMAN Duplicate, verify the existence of a checkpoint table, using the following command.

```
$ curl -s -K access.cfg https://gghub.example.com/TARGET/adminsrvr/services/v2/commands/execute -X
POST --data '{"name": "report","reportType": "checkpointTables","credentials": {"alias":
"target_atp1","domain": "goldengate"},"specification": "ggadmin.gg_checkpoints"}'| python -m
json.tool
```

NOTE: Replace the alias, domain, and specification of the checkpoint table to match the target database.

Example output if the checkpoint table already exists:

```
    "response": {
        "$schema": "ogg:commandResult",
        "tables": [
            "GGADMIN.GG_CHECKPOINTS"
        ]
    }
```

Example output if the checkpoint table does not exist:

```
    "response": {
        "$schema": "ogg:commandResult",
        "tables": []
    }
```

2. If the checkpoint table already exists, drop it before creating a new one.

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/connections/goldengate.target/tables/checkpo
int -X POST --data '{"operation": "delete","name": "ggadmin.gg_checkpoints"}'| python -m json.tool
```

3. Create the checkpoint table.

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/connections/goldengate.target/tables/checkpo
int -X POST --data '{"operation": "add","name": "ggadmin.gg_checkpoints"}'| python -m json.tool
```

4. Then check the table creation.

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/connections/goldengate.target/tables/checkpo
int -X POST --data '{"operation": "info","name": "ggadmin.gg_checkpoints"}'| python -m json.tool
```

Example return message:

```
  "$schema": "ogg:message",
  "code": "OGG-08100",
  "issued": "2019-06-26T17:18:09Z",
```

```
  "severity": "INFO",
  "title": "Checkpoint table ggadmin.gg_checkpoints created 2019-06-26 17:17:36.",
```

Create Heartbeat Tables on the Target Database

The GoldenGate heartbeat tables are required to monitor the replication latency between the source and target databases. If the target database already contains GoldenGate heartbeat objects, after being instantiated using RMAN Duplicate, the objects need to be dropped before being recreated.

1. Check to see if heartbeat objects currently exist in the target database.

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/connections/goldengate.target/tables/heartbe
at -XGET | python -m json.tool
```

2. If the heartbeat objects exist, use the following command to drop the target heartbeat table objects.

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/connections/goldengate.target/tables/heartbe
at -X DELETE| python -m json.tool
```

3. Create the target heartbeat table objects.

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/connections/goldengate.target/tables/heartbe
at -X POST --data '{"targetOnly":true}'| python -m json.tool
```

Create the Oracle GoldenGate Replicat

Before creating the Replicat, if the target database was instantiated using RMAN, the GoldenGate Extract that was previously created for the database migration should be unregistered and deleted.

Use the following command to unregister the Extract and delete it from the database.

```
$ curl -s -K access.cfg https://gghub.example.com/TARGET/adminsrvr/services/v2/extracts/EXT1 -X
DELETE | python -m json.tool
```

The creation of the Replicat process differs depending on how the target database was instantiated. Follow the instructions below for either RMAN or Data Pump instantiation.

When creating a Replicat using the REST API, the process parameter file is automatically created, so there is no need to manually create one.

Creating the Replicat for an RMAN Instantiation

The following example command creates the non-integrated parallel Replicat.

```
$ curl -s -K access.cfg https://gghub.example.com/TARGET/adminsrvr/services/v2/replicats/REP1 -X
POST --data '{"credentials":{"alias":"target","domain": "goldengate"},
"source": {"path":"/u02/trails","name":"aa"},"begin": {"sequence":0,"offset":0}, "checkpoint":
{"table":"ggadmin.gg_checkpoints"},
"mode":{"type": "nonintegrated","parallel":true},"config":["Replicat REP1", "UseridAlias target
```

```
DOMAIN goldengate","MAP_PARALLELISM 4","MIN_APPLY_PARALLELISM 2", "MAX_APPLY_PARALLELISM 50",
“REPORTCOUNT EVERY 15 MINUTES, RATE", "BATCHSQL", "Map pdb1.*.*"," Target PDB1.*.*;"]}' | python -m
json.tool
```

Confirm that the Replicat was created.

```
$ curl -s -K access.cfg https://gghub.example.com/TARGET/adminsrvr/services/v2/replicats/REP1 -X
GET | python -m json.tool
```

To start the Replicat, the SCN you recorded above in the Oracle RMAN Duplicate Database section is required. Using the SCN value in the above example, `681543060`, start the Replicat using the following example command.

```
$ curl -s -K access.cfg https://gghub.example.com/TARGET/adminsrvr/services/v2/commands/execute -X
POST --data '{"name":"start","processName":"REP1",”AT”:681543060}' | python -m json.tool
```

Check the status of the Replicat.

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/replicats/REP1/info/status -X GET | python -
m json.tool
```

Example output:

```
    "response": {
        "$schema": "ogg:replicatStatus",
        "lag": 0,
        "lastStarted": "2019-09-26T19:40:10.419Z",
        "position": {
            "name": "aa",
            "offset": 190639772,
            "path": "/u02/trails",
            "sequence": 5
        },
        "processId": 55184,
        "sinceLagReported": 8,
        "status": "running"
    }
```

Creating the Replicat for a Data Pump Instantiation

When the target database is instantiated with Oracle Data Pump, there is an additional parameter required when you create the Replicat process. The `ENABLE_INSTANTIATION_FILTERING` parameter instructs Replicat to filter out and not apply trail file data that was committed before the instantiation SCN of each table, recorded in the Data Pump export file. The instantiation SCN is stored in the target database data dictionary when the table is imported.

Create the Replicat.

```
$ curl -s -K access.cfg https://gghub.example.com/TARGET/adminsrvr/services/v2/replicats/REP1 -X
POST --data '{"credentials": {"alias":”target","domain": "goldengate"},"source":{"path":
"/u02/trails","name": "aa"},"begin": {"sequence": 0,"offset": 0}, "checkpoint":
{"table":"ggadmin.gg_checkpoints"}, "mode":{"type": "nonintegrated","parallel":
true},"config":["Replicat REP1", "UseridAlias target DOMAIN goldengate","MAP_PARALLELISM
```

```
4","MIN_APPLY_PARALLELISM 2", "MAX_APPLY_PARALLELISM 50", "REPORTCOUNT EVERY 15 MINUTES, RATE",
"BATCHSQL", "DBOPTIONS ENABLE_INSTANTIATION_FILTERING","Map pdb1.*.*"," Target PDB1.*.*;"]}' |
python -m json.tool
```

Confirm that the Replicat was created.

```
$ curl -s -K access.cfg https://gghub.example.com/TARGET/adminsrvr/services/v2/replicats/REP1 -X
GET | python -m json.tool
```

Start the Replicat.

```
$ curl -s -K access.cfg https://gghub.example.com/TARGET/adminsrvr/services/v2/commands/execute -X
POST --data '{"name":"start","processName":"REP1"}' | python -m json.tool
```

Check the status of the started Replicat.

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/replicats/REP1/info/status -X GET | python -
m json.tool
```

Assign the Autostart Task to Replicat

Now that the Replicat process is running, the autostart task previously in Create Auto-start Tasks should be assigned to the Replicat using the following example command.

```
$ curl -s -K access.cfg https://gghub.example.com/TARGET/adminsrvr/services/v2/replicats/REP1 -X
PATCH --data '{"managedProcessSettings": "MIGRATE01"}' | python -m json.tool
```

Confirm that the Replicat is configured with the new profile.

```
$ curl -s -K access.cfg https://gghub.example.com/TARGET/adminsrvr/services/v2/replicats/REP1 -X
GET | python -m json.tool | grep managedProcessSettings

   "managedProcessSettings": "MIGRATE01",
```

MONITOR GOLDENGATE REPLICATION

When the Extract and Replicat are both running, the GoldenGate end-to-end latency must be monitored until all outstanding transactional data generated during the target database instantiation has been applied, after which, you can switch users and applications over to the migrated database.

First, compare the trail file progress of the Extract and Replicat processes.

View Extract checkpoint progress:

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/extracts/EXT1/info/checkpoints -XGET |
python -m json.tool
```

Example output:

```
"current": {
   "name": "aa",
   "offset": 388314032,
```

```
   "path": "/u02/trails/",
   "sequence": 311,
   "sequenceLength": 9,
   "sequenceLengthFlip": false,
   "timestamp": "2019-10-07T21:43:36.660Z"
}
```

View Replicat checkpoint progress:

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/replicats/REP1/info/checkpoints -XGET |
python -m json.tool
```

Example output:

```
"current": {
   "name": "aa",
   "offset": 346955548,
   "path": "/u02/trails/",
   "sequence": 311,
   "sequenceLength": 9,
   "sequenceLengthFlip": false,
   "timestamp": "2019-10-07T21:48:12.906Z"
}
```

When the Replicat is reading from the same trail file that the Extract is writing to, with a similar checkpoint offset, the Replicat has applied all of the trail files generated during the target database instantiation.

Next, monitor the Oracle GoldenGate end-to-end latency using the heartbeat tables. Run the following command until the lag time is acceptably low (for example, less than 2 seconds).

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/connections/goldengate.target/tables/heartbe
at/REP1 -XGET | python -m json.tool
```

Example output:

```
        "heartbeats": [
            {
                "ageSeconds": 5.02,
                "lagSeconds": 1.27,
                "path": "EXT1 ==> REP1",
                "source": "DSGG:PDB1",
                "target": "GGT:PDB1"
            }
        ]
```

TESTING THE MIGRATED DATABASE

Suspend GoldenGate apply and then create a Guaranteed Restore Point (GRP) so that you can test the target database. When you are done testing, flashback the database to the GRP, the GoldenGate Replicat is restarted, and replication resumes. When all testing is complete, you can drop the GRP. The cycle of suspending replication, creating a GRP, testing, flashback database, and resuming replication can be done a number of times.

Due to the privilege restrictions in ATP-D, it is not possible to create a Guaranteed Restore Point. Therefore, the following instructions for read write testing of the migrated database will only work for ExaCS. When migrating to ATP-D, the database should be validated by read-only queries.

Suspend Replication

Suspend replication by stopping the GoldenGate Replicat process. Extract is left running so it will continue to extract data from the source database, creating new trail file data.

1. Stop Replicat.

```
$ curl -s -K access.cfg https://gghub.example.com/TARGET/adminsrvr/services/v2/commands/execute -X
POST --data '{"name":"stop","processName":"REP1","force":false}' | python -m json.tool
```

2. Check Replicat status.

```
$ curl -s -K access.cfg https://gghub.example.com/TARGET/adminsrvr/services/v2/replicats/REP1 -X
GET | python -m json.tool
```

It is not safe to continue until the Replicat status is stopped cleanly.

Example output:

```
"sinceLagReported": 14,
"status": "stopped"
```

Create a Guaranteed Restore Point (GRP)

If the target database is a multitenant database, you create the GRP from in the CDB.

Connect to the CDB as the SYSDBA user and run the following statement.

```
SQL> CREATE RESTORE POINT <GRP name> FOR PLUGGABLE DATABASE <PDB name> GUARANTEE FLASHBACK
DATABASE;
```

For a single tenant database, use the following command to create the restore point.

```
SQL> CREATE RESTORE POINT <GRP name> GUARANTEE FLASHBACK DATABASE;
```

Confirm GRP creation.

```
SQL> set linesize 140 pages 1000
SQL> col con_id format 9999
SQL> col pdb_restore_point format a20
SQL> col name format a20
SQL> col guarantee_flashback_database format a5
SQL> col scn format 999999999999
SQL> col time format a48
SQL> alter session set nls_date_format='dd-Mon-yyyy hh:mi:ss';

SQL> select con_id,name,scn,time,guarantee_flashback_database,pdb_restore_point
from v$restore_point;
```

Validate the Target Database

The target database can now be used for application validation and a quick sanity check, including DML/DDL against database tables. The test window should not normally exceed a couple of hours.

It is recommended that comprehensive performance and functional testing is completed before you start the migration. The risks of using the current migration window for such comprehensive testing are:

1. Extend the migration window.
2. Extend the time GoldenGate requires, and possibly makes it impossible to catch up replicating the transactions that are continually generated on the source database.
3. Cause issues that may compromise the target system and compromise the entire migration.

During testing, it is recommended that you continue monitoring GoldenGate lag and checkpoint differences (current Extract trail file vs. Replicat last trail file applied) as detailed in the Monitoring GoldenGate Replication section.

Flashback the Target Database

When testing is complete the target database can be closed, flashed back to the GRP, and reopened.

All of the following commands must be run while connected to the CDB as the SYSDBA user, if running a multitenant database.

1. Shut down the database.

   For single tenant:

```
SQL> shutdown immediate
```

   For multitenant:

```
SQL> ALTER PLUGGABLE DATABASE <PDB name> CLOSE IMMEDIATE;
```

2. Flashback the database.

   For single tenant:

```
SQL> FLASHBACK DATABASE TO RESTORE POINT <GRP name>;
```

   For multitenant:

```
SQL> FLASHBACK PLUGGABLE DATABASE <PDB name> TO RESTORE POINT <GRP name>;
```

3. Start the database.

   For single tenant:

```
SQL> ALTER DATABASE OPEN RESETLOGS;
```

   For multitenant:

```
SQL> ALTER PLUGGABLE DATABASE <PDB name> OPEN RESETLOGS;
```

After the database has been flashed back it is possible to do multiple cycles of testing and flashback to the same GRP. However, the longer that replication is suspended, the longer it takes for replication to catch up.

Drop the Guaranteed Restore Point

When the database is open and testing is complete, drop the GRP. If you want to do multiple cycles of testing, it is recommended that you resume replication and allow GoldenGate to catch up before re-iterating the testing cycle.

For single tenant:

```
SQL> DROP RESTORE POINT <GRP name>;
```

For multitenant:

```
SQL> DROP RESTORE POINT <GRP name> FOR PLUGGABLE DATABASE <PDB name>;
```

Resume Replication

Resume replication by restarting the GoldenGate Replicat process.

```
$ curl -s –K access.cfg https://gghub.example.com/TARGET/adminsrvr/services/v2/commands/execute -X
POST --data '{"name":"start","processName":"REP1"}' | python -m json.tool
```

Continue to monitor the GoldenGate checkpoints and lag, and when testing is complete, move to the final switchover.

SWITCH OVER TO THE MIGRATED DATABASE

When you are done testing, you can switch over to the target database.

Determine Whether GoldenGate Replication Lag is Acceptably Low

An acceptable low replication lag can vary greatly, depending on the source workload and network latency.

The lag should be monitored along with the current checkpoint position of the GoldenGate Extract and Replicat to make sure that both processes are operating within the same trail file, which commonly implies lower latency between Extract and Replicat. Use the commands specified in the Monitor GoldenGate Replication section to determine when Extract and Replicat have reached an acceptable lag.

Stop Transactions From Starting on the Source Database

To complete a zero data loss switchover, stop transactions from occurring on the source database by stopping the service for the source database, using SRVCTL.

After new connections and transactions are no longer allowed on the source database, monitor the database for all active transactions to complete.

For example:

```
SQL> select XIDUSN, XIDSLOT, XIDSQN, STATUS, USED_UBLK, USED_UREC from v$transaction where
RECURSIVE='NO';
```

Once this returns no rows, you can assume that the source database is idle.

Verify That Extract Has Completed Outstanding Transactions

When all transactions have stopped on the source database, monitor Extract to make sure it has caught up with extracting all outstanding transactions.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/extracts/EXT1/info/checkpoints -XGET |
python -m json.tool
```

The Extract output checkpoint position should not be moving much, only by the GoldenGate heartbeat table updates every 60 seconds.

Stop the Extract

When you can see the Extract no longer moving forwards, you can stop it.

```
$ curl -s -K access.cfg https://gghub.example.com/SOURCE/adminsrvr/services/v2/commands/execute -X
POST --data '{"name":"stop","processName":"EXT1","force":true}' | python -m json.tool
```

Check that the status is stopped.

```
$ curl -s -K access.cfg https://gghub.example.com/SOURCE/adminsrvr/services/v2/extracts/EXT1 -X GET
| python -m json.tool|grep status
```

Wait for Replicat to Apply All Trail File Data

After Extract has been stopped you must wait for Replicat to apply all outstanding trail file data. To do this, monitor the Replicat with the `LOGEND` command.

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/replicats/REP1/command -X POST --data
'{"command":"LOGEND"}' | python -m json.tool
```

Example output:

```
"replyData": {
    "$schema": "er:logEndResult",
    "allRecordsProcessed": true
}
```

When the Replicat has applied all trail file data the `allRecordsProcessed` value of “true” is returned.

Switch Over to the Target Database

Now that all of the replicated data is applied to the target database, you can stop the Replicat.

```
$ curl -s -K access.cfg https://gghub.example.com/TARGET/adminsrvr/services/v2/commands/execute -X
POST --data '{"name":"stop","processName":"REP1","force":true}' | python -m json.tool
```

With Replicat stopped, switch the database clients over to the target database.

Because both source and target databases are opened read-write during the migration process, enabling application database services and switching the application to use the targeted services is under the discretion of the system or database administrators. For read-only applications, switchover can happen immediately after GoldenGate Replicat has applied all outstanding source transactions and

stops, allowing for zero application downtime for those services. Read-write applications may require the assurance that all transactions have been applied on the target before switching the application over to ensure the most up to date data is being manipulated.

If desired, Oracle GoldenGate Veridata can be used to identify and repair any out-of-sync data before switching over to the migrated database. Refer to the Oracle GoldenGate Veridata documentation for more information.

https://docs.oracle.com/en/middleware/goldengate/veridata/12.2.1.2.0/books.html

REMOVING THE GOLDENGATE CONFIGURATION

After migration of the Oracle database, you can remove the GoldenGate configuration using the following commands.

Drop the GoldenGate Processes

```
$ curl -s -K access.cfg https://gghub.example.com/TARGET/adminsrvr/services/v2/replicats/REP1 -X
DELETE | python -m json.tool

$ curl -s -K access.cfg https://gghub.example.com/SOURCE/adminsrvr/services/v2/extracts/EXT1 -X
DELETE | python -m json.tool
```

Use the following command to remove the trail files.

```
$ curl -s -K access.cfg https://gghub.example.com/SOURCE/adminsrvr/services/v2/commands/execute -X
POST --data '{"name": "purge","purgeType": "trails","trails":[{"name":"aa"}],"useCheckpoints":
false,"keep": [{"type": "min","units": "files","value": 0}]}' | python -m json.tool
```

Remove the Autostart Tasks

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/config/types/ogg:managedProcessSettings/valu
es/ogg:managedProcessSettings:MIGRATE01 -XDELETE | python -m json.tool

$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/config/types/ogg:managedProcessSettings/valu
es/ogg:managedProcessSettings:MIGRATE01 -XDELETE | python -m json.tool
```

Drop the Heartbeat Tables on the Target Database

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/connections/goldengate.target/tables/heartbe
at -X DELETE| python -m json.tool
```

If the heartbeat table was created in the source database as part of the migration, then it should be also be dropped.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/connections/goldengate.source_pdb/tables/hea
rtbeat -X DELETE| python -m json.tool
```

Drop the Checkpoint Table

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/connections/goldengate.target/tables/checkpo
int -X POST --data '{"operation": "delete","name": "ggadmin.gg_checkpoints"}'| python -m json.tool
```

Remove the GoldenGate Credentials

Removing the Oracle GoldenGate credentials does not drop the user accounts from the database.

```
$ curl -s -K access.cfg
https://gghub.example.com/TARGET/adminsrvr/services/v2/credentials/goldengate/target -X DELETE |
python -m json.tool
```

The following example is for a multitenant source database.

```
$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/credentials/goldengate/source_cdb -X DELETE
| python -m json.tool

$ curl -s -K access.cfg
https://gghub.example.com/SOURCE/adminsrvr/services/v2/credentials/goldengate/source_pdb -X DELETE
| python -m json.tool
```

Once the GoldenGate credential has been dropped, the user can be dropped from the database using the following example SQL command.

```
SQL> drop user ggadmin cascade;
```

Remove GoldenGate Hub

If the GoldenGate Hub, along with the created deployments, are not needed for any additional database migrations the GoldenGate Instance should be removed from your Oracle Cloud environment. Use the following instructions to remove the Oracle GoldenGate Instance.

1. Log in to your Oracle Cloud Account.
2. Select Resource Manager, Stacks from the menu.

   You can get a list of stacks that are built in your compartment.
3. Click the link of the stack that you want to remove.
4. In the Stack, from the TerraForm Action drop-down list select Destroy.

   This permanently removes your Oracle GoldenGate Instance.
5. Delete the stack (optional)

CONNECT WITH US

Call +1.800.ORACLE1 or visit oracle.com.
Outside North America, find your local office at oracle.com/contact.

blogs.oracle.com facebook.com/oracle twitter.com/oracle

Migration to the Oracle Cloud with an Oracle GoldenGate Hub Configuration
April, 2020

Author: Stephan Haisley