Global Data Services with Oracle GoldenGate

Cookbook

OCTOBER 2017

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. 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 for Oracle's products remains at the sole discretion of Oracle.

Oracle Global Data Services (GDS) in a nutshell

Typically Oracle customers protect their mission critical and business important production databases by deploying Active Data Guard or Oracle GoldenGate both in local or remote data centers. In order to fully utilize these replicated environments, customers often use Active Data Guard to offload their READ ONLY and READ MOSTLY production workloads to sets of standby databases. In the case of Oracle GoldenGate, they can also distribute READ WRITE workloads across Active-Active configurations.

Oracle Global Data Services (GDS) is a feature of Oracle Database 12c that provides connect-time and run-time load balancing, region affinity, replication lag tolerance based workload routing, and enables inter-database service failover over a set of replicated databases. GDS maximizes performance and availability of the applications and it allows database workloads to be managed across distributed replicas with a single unified framework

Oracle GDS extends the familiar RAC-Style connect-time and run-time load balancing, service failover and management capabilities – so far applicable only to a single database, to a set of replicated databases, be it within or across datacenters. With a newly created concept called Global Service, Oracle GDS framework extends the notion of a database Service to a set of replicas running on a combination of Single Instance, Oracle RAC, Oracle Engineered systems, Active Data Guard and Oracle GoldenGate.

GDS sits between the application tier and the database tiers of the stack. It orchestrates the Service high availability, Service level load balancing and routing. Global Services run on the databases but are managed by GDS. GDS algorithms take into account, DB instance load, network latency between data centers and the workload management policies (region affinity, load balancing goals, DB cardinality, DB role, replication lag) that the customers can configure. These workload management policies are enabled via the attributes of a given Global Service.

Cookbook Overview

The objective of this cookbook is to give you a walk through of the deployment and configuration of Global Data Services (GDS). You will learn the following:

- Installation of Global Service Managers
- Creation of GDS catalog
- Specify the metadata and configure GDS
- Configure global services for the following GDS features
    - Inter-database global service failover
    - Connect-time and Run-time load balancing
    - Locality based workload routing
- Obtain familiarity with the GDSCTL interface.

The exercises are designed from the top down so that each step builds upon the previous exercise. It is essential that you follow the exercises in order.

Environment Overview

In this cookbook, we will use the following components:

- Two Global Service Managers (GSMs)
- One GDS Catalog
- Two GDS Regions
- A GDS Pool with two databases using Oracle GoldenGate in bi-directional replication. One database in each of the Regions. The "SFO" database is in the West region and the "CHI" database is in the East region.

Here is the cookbook topology :

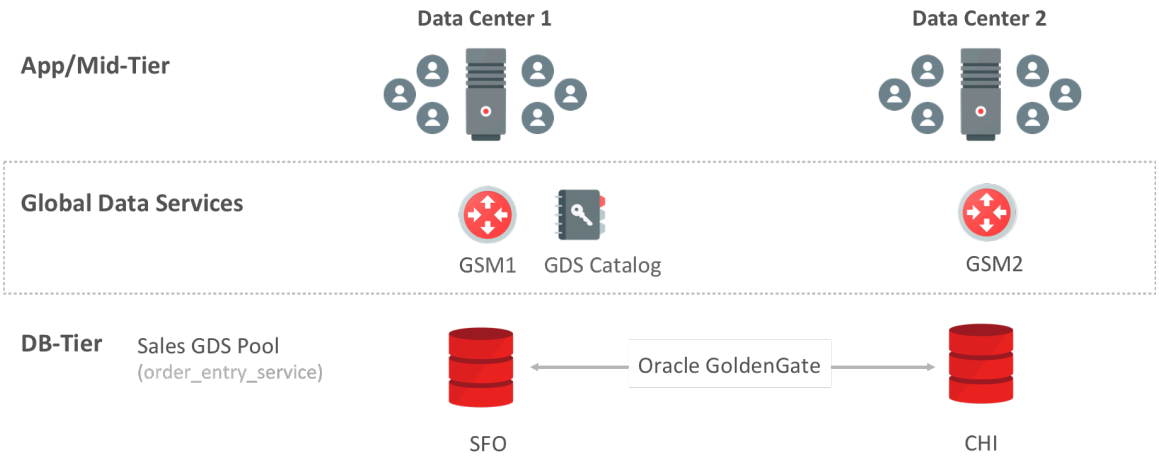

Figure 1. GDS with Oracle GoldenGate - Demo Topology

In this lab, we are leveraging the two servers to host not only the databases SFO and CHI, but also, the Global Service Managers and GDS Catalog. In production environment, you would host GSMs and GDS Catalog on separate VMs or servers. The layout used for this lab is shown below.

Machine	Component	Oracle_Home
ServerA	gdscat	/u01/app/oracle/product/12.2.0/dbhome_1
ServerA	gsm1	/u01/app/oracle/product/12.2.0/gsmhome_1
ServerA	SFO	/u01/app/oracle/product/12.2.0/dbhome_1
ServerB	CHI	/u01/app/oracle/product/12.2.0/dbhome_1
ServerB	gsm2	/u01/app/oracle/product/12.2.0/gsmhome_1

Figure 2. GDS Components Layout – For the Lab

Note: For production deployments, it is highly recommended to configure Data Guard for the GDS catalog database and three GSMs per region.

High-Level Deployment Steps

The high-level steps of GDS deployment are:

1. Install GSM software on GSM Nodes
2. Setup GDS Administrator accounts & privileges
3. Configure GDS (Create GDS Catalog, Add GSMs, Regions, Pools, Databases and Global Services) using GDSCTL commands.
4. Setup client connectivity

Environment Prerequisites

Before the GDS is deployed, perform the following few pre-requisite steps:

A. Hardware and Software sizing for the labs
B. Download the database and gsm media
C. Install Oracle Database software on the database servers used for the GDS Pool databases
D. Create a Non-CDB database (which will be used to host the GDS catalog)
E. Install Global Service Managers (GSMs)

Important Pre-Requisites (MUST READ):

1. Each and every GDS Pool database must be able to reach (in both directions) each and every GSM's Listener and ONS ports (Default Listener Port of the GSM is 1522 (We are using port 1571 in this cookbook) and the default ONS ports are: 6123 for the localONS and 6234 for remoteONS - on most platforms). These GSM Listener ports and the ONS ports must also be opened to the Application/Client tier, all the GDS Pool databases, the GDS catalog and all other GSMs.
2. The TNS Listener port (Default: 1521) of each GDS Pool database must be opened (in both directions) to GSMs and GDS Catalog.
3. If GDSCTL is run from a separate machine, we also need a port opened (in both directions) from that machine directly to each GDS Pool database that is added.

Here are the detailed steps that cover the prerequisites:

A. Hardware and software sizing

You need two servers or VMs to perform lab exercises.

Here is the minimum hardware/software configuration that we recommend for the cookbook VMs:

CPUs - 2 Cores
Memory - 4G
Disk Space > 200G (based on the load that is planned to be executed)
Network - Low Latency GigE
OS – Oracle Enterprise Linux (OEL 64Bit)

B. Setup environment scripts

Under $HOME of your dedicated VMs, create various environment shell scripts to set your database env, GDS catalog env and Global Service Manager (GSM) env.

- sfo.sh (on ServerA) and chi.sh (on ServerB)
- gdscat.sh (for GDS Catalog – on ServerA)
- gsm1.sh (for GSM1 on ServerA) and gsm2.sh (for GSM2 ServerB)

On ServerA

Note: In this example, oracle OS user is using bash shell.
Save your current path: export SAVEPATH=$PATH

```
$ cd $HOME
$ more gdscat.sh
export ORACLE_SID=gdscat
export ORACLE_BASE=/u01/app/oracle
```

```
export ORACLE_HOME=/u01/app/oracle/product/12.2.0/dbhome_1
export LD_LIBRARY_PATH=$ORACLE_HOME/lib
export PATH=$SAVEPATH:$ORACLE_HOME/bin

$ more gsm1.sh
export ORACLE_BASE=/u01/app/oracle
export ORACLE_HOME=/u01/app/oracle/product/12.2.0/gsmhome_1
export LD_LIBRARY_PATH=$ORACLE_HOME/lib
export PATH=$SAVEPATH:$ORACLE_HOME/bin

$ more sfo.sh
export ORACLE_SID=sfo
export ORACLE_BASE=/u01/app/oracle
export ORACLE_HOME=/u01/app/oracle/product/12.2.0/dbhome_1
export LD_LIBRARY_PATH=$ORACLE_HOME/lib
export PATH=$SAVEPATH:$ORACLE_HOME/bin
```

On ServerB:
Save your current path: export SAVEPATH=$PATH

```
$ more chi.sh
export ORACLE_SID=chi
export ORACLE_BASE=/u01/app/oracle
export ORACLE_HOME=/u01/app/oracle/product/12.2.0/dbhome_1
export LD_LIBRARY_PATH=$ORACLE_HOME/lib
export PATH=$SAVEPATH:$ORACLE_HOME/bin

$ more gsm2.sh
export ORACLE_BASE=/u01/app/oracle
export ORACLE_HOME=/u01/app/oracle/product/12.2.0/gsmhome_1
export LD_LIBRARY_PATH=$ORACLE_HOME/lib
export PATH=$SAVEPATH:$ORACLE_HOME/bin
```

C. Download the Oracle Database, Oracle GoldenGate and Oracle GSM software

Download the Oracle Database and Oracle Database Global Service Manager (GSM) for the appropriate release/version of your choice, from https://edelivery.oracle.com
or
http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html

Copy the database software to both the servers and unzip it to /u01/stage/database.
Copy the gsm software to both the servers and unzip it to /u01/stage/gsm.

Note: Steps that cover the Installation and the configuration of Oracle GoldenGate replication are out of scope for this cookbook.

D. Install database software on ServerA and ServerB

This cookbook uses the Database software for the Linux x86-64. Install the database software on ServerA and ServerB on the Oracle Home (`/u01/app/oracle/product/12.2.0/dbhome_1`).

Run the Installer from /u01/stage/database directory

E. Create a Non-CDB database that hosts the GDS catalog

Note: Currently, only the Non-CDB databases are supported for the GDS catalog. (The GDS Pool databases can be CDBs). In this cookbook, you will be creating the gdscat on ServerA.

F. Create CDB databases – sfo on ServerA and chi on ServerB.

Note: The GDS Pool databases can be CDBs or Non-CDBs. In this cookbook, you will be creating the Pool databases as a Container Database with a PDB called mypdb.

1. Install GSM software

This cookbook uses the GSM software for the Linux x86-64. Install the GSM software on ServerA and ServerB on separate Oracle Home (`/u01/app/oracle/product/12.2.0/gsmhome_1`).

Run the Installer from /u01/stage/gsm directory

2. GDS Setup and Configuration

Add the following TNS entries of GDS catalog and pool databases to the GSM Home's tnsnames.ora.

Note: Repeat this step on all GSM Nodes. (In this cookbook, these TNS entries must be appended to the tnsnames.ora of the GSM Homes of both ServerA and ServerB)

Note for RAC Databases: If the Pool databases are RAC-enabled, update the following TNS entries so that it would use 3 address lines, where each address line resolves to one of the SCAN VIPs.

```
GDSCAT =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = ServerA)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = gdscat.us.oracle.com)
    )
  )

SFO,SFO.US.ORACLE.COM =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = ServerA)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = sfo.us.oracle.com)
    )
  )

CHI,CHI.US.ORACLE.COM =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = ServerB)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = chi.us.oracle.com)
    )
  )
```

3. Configure GDS

In this lab, you will perform the following steps:

- Execute grants and privileges used by the GSM and GDS administrator
- Create GDScatalog and GSMs

1. On the GDSCAT Env:

From the GDSCAT terminal, connect to the gdscat database on ServerA, using SQL*Plus :

```
$ cd $HOME
$ . ./gdscat.sh

$ env |grep ORA
ORACLE_SID=gdscat
ORACLE_BASE=/u01/app/oracle
ORACLE_HOME=/u01/app/oracle/product/12.2.0/dbhome_1
```

2. Grant Roles/Privileges (on accounts used by GSMs) on gdscat:

```
sqlplus / as sysdba
SQL> spool setup_grants_privs.lst
SQL> set echo on
SQL> set termout on
```

Unlock and set the password for the gsmcatuser schema. This schema is used by the GSM while connecting to the GDS catalog database.

```
SQL> alter user gsmcatuser account unlock;
SQL> alter user gsmcatuser identified by passwd_gsmcatuser;
```

Create the administrator schema (mygdsadmin) and give the privileges

```
SQL> create user mygdsadmin identified by passwd_mygdsadmin;
SQL> grant connect, create session, gsmadmin_role to mygdsadmin;
```

Enable tracing on the GDS catalog (for the cookbook only)

```
SQL> alter system set events 'immediate trace name GWM_TRACE level 7';
SQL> alter system set event='10798 trace name context forever, level 7'
SCOPE=spfile;

SQL> spool off
```

3. Grant Roles/Privileges (on accounts used by GSMs) on gdscat:

Make sure that the listeners & databases sfo, chi and gdscat are up.

Connect to the pool databases sfo and chi and unlock the GSM user (as shown below).

On a new terminal and name it GSM1 Terminal:

```
$ cd $HOME
$ . ./gsm1.sh
$ env |grep ORA
```

```
ORACLE_BASE=/u01/app/oracle
ORACLE_HOME=/u01/app/oracle/product/12.2.0/gsmhome_1


sqlplus system/oracle@sfo
SQL> spool unlock_gdsusers_sfo.lst append
SQL> show parameter db_unique
SQL> alter user gsmuser account unlock;
SQL> alter user gsmuser identified by passwd_gsmuser;
SQL> spool off;

sqlplus system/oracle@chi
SQL> spool unlock_gdsusers_chi.lst append
SQL> show parameter db_unique
SQL> alter user gsmuser account unlock;
SQL> alter user gsmuser identified by passwd_gsmuser;
SQL> spool off;
SQL> exit
```

4. Launch GDSCTL and create the GDS catalog:

Note: For all GDSCTL commands that span more than one line, they should not be copy pasted to terminal directly. Please copy to any other text editor, make them one line instead of multiple lines as given in doc and then paste in terminal.
The other option is to type the command by hand.

```
$ gdsctl
```

Create the gdscatalog

```
GDSCTL>create catalog -database serverA:1521:gdscat -user
mygdsadmin/passwd_mygdsadmin -region region1,region2

Catalog is created
```

Create and start the first GSM

```
GDSCTL> add gsm -gsm gsm1 -listener 1571 -catalog serverA:1521:gdscat -
pwd passwd_gsmcatuser -region region1 -trace_level 16

GSM successfully added


GDSCTL> start gsm -gsm gsm1

GSM is started successfully

GDSCTL> config gsm -gsm gsm1

Name: gsm1
Endpoint 1: (ADDRESS=(HOST=serverA)(PORT=1571)(PROTOCOL=tcp))
Local ONS port: 6123
Remote ONS port: 6234
```

```
ORACLE_HOME path: /u01/app/oracle/product/12.2.0/gsmhome_1
GSM Host name: serverA
Region: region1


Buddy
------------------------

GDSCTL> status gsm -gsm gsm1

Alias                     GSM1
Version                   12.2.0.1.0
Start Date                10-AUG-2017 13:52:29
Trace Level               support
Listener Log File         /u01/app/oracle/diag/gsm/serverA/gsm1/alert/log.xml
Listener Trace File
/u01/app/oracle/diag/gsm/serverA/gsm1/trace/ora_27385_139940518471104.trc
Endpoint summary          (ADDRESS=(HOST=serverA)(PORT=1571)(PROTOCOL=tcp))
GSMOCI Version            0.1.11
Mastership                Y
Connected to GDS catalog  Y
Process Id                27388
Number of reconnections   0
Pending tasks.     Total  0
Tasks in  process. Total  0
Regional Mastership       TRUE
Total messages published  0
Time Zone                 -07:00
Orphaned Buddy Regions:
     None
GDS region                region1
```

5. Add the second GSM :

Note: In this lab, we are hosting GSM2 on the serverB .

```
$ cd $HOME
$ . ./gsm2.sh

$ env |grep ORA
ORACLE_BASE=/u01/app/oracle
ORACLE_HOME=/u01/app/oracle/product/12.2.0/gsmhome_1


$ gdsctl

GDSCTL> add gsm -gsm gsm2 -listener 1571 -pwd passwd_gsmcatuser -catalog
serverA:1521:gdscat -region region2 -trace_level 16

GSM successfully added


GDSCTL> start gsm -gsm gsm2
GSM is started successfully
```

```
GDSCTL> config gsm -gsm gsm2

Name: gsm2
Endpoint 1: (ADDRESS=(HOST=serverB)(PORT=1571)(PROTOCOL=tcp))
Local ONS port: 6123
Remote ONS port: 6234
ORACLE_HOME path: /u01/app/oracle/product/12.2.0/gsmhome_1
GSM Host name: serverB
Region: region2


Buddy
------------------------


GDSCTL> status gsm -gsm gsm2

Alias                     GSM2
Version                   12.2.0.1.0
Start Date                10-AUG-2017 18:40:12
Trace Level               support
Listener Log File
/u01/app/oracle/diag/gsm/serverB/gsm2/alert/log.xml
Listener Trace File
/u01/app/oracle/diag/gsm/serverB/gsm2/trace/ora_303_140545174263232.trc
Endpoint summary
(ADDRESS=(HOST=serverB)(PORT=1571)(PROTOCOL=tcp))
GSMOCI Version            0.1.11
Mastership                N
Connected to GDS catalog  Y
Process Id                305
Number of reconnections   0
Pending tasks.     Total  0
Tasks in  process. Total  0
Regional Mastership       TRUE
Total messages published  0
Time Zone                 -07:00
Orphaned Buddy Regions:
     None
GDS region                region2
Network metrics:
   Region: region1 Network factor:0


Type "exit" to exit from GDSCTL
```

6. Create GDS Pool and add databases – sfo and chi to the Pool

As we need just one gdspool, we can either use the pre-defined one dbpoolora or create a new one called sales.

Note: You can stay in any GSM env from here onwards and be able to manage the GDS configuration.

On GSM1 Terminal (ServerA):

```
$ cd $HOME
$ . ./gsm1.sh

$ gdsctl

GDSCTL> set gsm -gsm gsm1

GDSCTL> connect mygdsadmin/passwd_mygdsadmin
```

Add a GDS Pool

```
GDSCTL> add gdspool -gdspool sales

GDSCTL>config gdspool
Name                           Broker
----                           ------
sales                          No
dbpoolora                      No
```

Execute the “add invitednode”

Notes: "The valid node checking for registration (VNCR) feature provides the ability to configure and dynamically update a set of IP addresses, host names, or subnets from which registration requests are allowed by the GSMs. Database instance registration with a GSM succeeds only when the request originates from a valid node. By default, the GDS Framework automatically adds a VNCR entry for the host on which a remote database is running each time “add database” is executed. The automation (called auto-VNCR) finds the public IP address of the target host, and automatically adds a VNCR entry for that IP. If the host has multiple public IP addresses, then the one on which the database registers may not be the same as the one which was added using auto-VNCR, as a result, registration many be rejected. If the target database host has multiple public IPs, it is advisable to configure VNCR manually for this host using the “add invitednode” or “add invitedsubnet” commands in GDSCTL.

If there are multiple net-cards on the target host (“/sbin/ifconfig” returns more than one public interface), use “add invitednode” to be safe (after finding out which interface will be used to route packets).

If there is any doubt about registration, then the user should simply check with “config vncr” and use “add invitednode” as necessary. There is no harm in doing this, if the node is added already, auto-VNCR will ignore it, and if the user tries to add it after auto-VNCR already added it, they will simply get a warning stating that it already exists.

```
GDSCTL> add invitednode <ServerA>
GDSCTL> add invitednode <ServerB>
GDSCTL> config vncr
Name                           Group ID
----                           ----- --
10.xx.yy.zzz
```

```
serverA
serverB
```

Important Notes:

==========

Before the databases are added to a GDS Pool, it is required that the ports used by the GSM listeners are opened for the databases to register with them. You can verify this using the following steps:

Can you do tnsping from the database nodes to the GSM listeners. This will confirm, if there is a problem in reaching the Listener port (e.g. 1571) on GSMs.

Here is an example (from one of the database nodes). Repeat this step (for all GSMs) from each of the databases that are being added to the pool.

Create " gdstest" tns alias (shown below) in the tnsnames.ora of the /u01/app/oracle/product/12.1.0.2/dbhome_1/network/admin/tnsnames.ora

```
gdstest=
  (ADDRESS = (PROTOCOL = TCP)(HOST = gsmhost1)(PORT = 1571))


tnsping gdstest

TNS Ping Utility for Linux: Version 12.1.0.2.0 - Production on 15-OCT-
2015 10:41:41
Copyright (c) 1997, 2014, Oracle.  All rights reserved.
Used parameter files:
/u01/app/oracle/product/12.1.0.2/dbhome_1/network/admin/sqlnet.ora

Used TNSNAMES adapter to resolve the alias
Attempting to contact (DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST =
gsmhost1)(PORT = 1571)) (CONNECT_DATA = (SERVER = DEDICATED)
(SERVICE_NAME = TEST_SERVICE)))
TNS-12543: TNS:destination host unreachable
```

If the tnsping fails, work with your Networking team to make sure that the GSM listener ports are reachable from the database servers.

==========

After the add invitednode command is executed, you now can add the databases to the sales GDS Pool

```
GDSCTL> add database -connect serverA:1521:sfo -gdspool sales -pwd
passwd_gsmuser -region region1

Catalog connection is established
DB Unique Name: sfo
```

```
GDSCTL> add database –connect –connect serverB:1521:chi –gdspool sales
–pwd passwd_gsmuser – region region2

DB Unique Name: chi
```

7. Check the configuration from any GSM :

```
GDSCTL>config database
Name                          Pool                          Status        Region
----                          ----                          ------        ------
sfo                           sales                         Ok            region1
chi                           sales                         Ok            region2

GDSCTL>config database -database sfo
Name: sfo
Pool: sales
Status: Ok
Region: region1
Connection string:
(DESCRIPTION=(ADDRESS=(HOST=serverA)(PORT=1521)(PROTOCOL=tcp))(CONNECT_
DATA=(SID=sfo)))
SCAN address:
ONS remote port: 0
Disk Threshold, ms: 20
CPU Threshold, %: 75


Supported services
------------------------
Name
Preferred Status
----                                                                -------
-- ------

GDSCTL>config database -database chi
Name: chi
Pool: sales
Status: Ok
Region: region2
Connection string:
(DESCRIPTION=(ADDRESS=(HOST=serverB)(PORT=1521)(PROTOCOL=tcp))(CONNECT_
DATA=(SID=chi)))
SCAN address:
ONS remote port: 0
Disk Threshold, ms: 20
CPU Threshold, %: 75


Supported services
------------------------
Name
Preferred Status
```

```
----                                                                  -------
-- ------
```

8. Observe that all databases are “Registered”:

```
GDSCTL> databases
Database: "chi" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1 Region: region2
   Registered instances:
     sales%11
Database: "sfo" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1 Region: region1
   Registered instances:
     sales%1
```

9. Add a global service that runs on a preferred database:

```
GDSCTL>add service -service order_entry_srvc -gdspool sales -preferred
sfo -available chi -pdbname mypdb
```

For RAC-enabled databases, do the following additional step:

```
GDSCTL>modify service -service order_entry_srvc -gdspool sales  -
database sfo -modify_instances -preferred sfo_n1, sfo_n2
```

```
GDSCTL>config service -service order_entry_srvc
Name: order_entry_srvc
Network name: order_entry_srvc.sales.oradbcloud
Pool: sales
Started: No
Preferred all: No
Locality: ANYWHERE
Region Failover: No
Role: NONE
Primary Failover: No
Lag: ANY
Runtime Balance: SERVICE_TIME
Connection Balance: LONG
Notification: Yes
TAF Policy: NONE
Policy: AUTOMATIC
DTP: No
Failover Method: NONE
Failover Type: NONE
Failover Retries:
Failover Delay:
Edition:
```

```
PDB: mypdb
Commit Outcome:
Retention Timeout:
Replay Initiation Timeout:
Session State Consistency:
SQL Translation Profile:


Databases
------------------------
Database                       Preferred Status
--------                       --------- ------
chi                            No        Enabled
sfo                            Yes       Enabled
```

10. Start the Global Service :

```
GDSCTL> start service -service order_entry_srvc -gdspool sales
```

11. Verify the status of the global services:

```
GDSCTL> status service

GDSCTL>status service
Service "order_entry_srvc.sales.oradbcloud" has 1 instance(s).
Affinity: ANYWHERE
   Instance "sales%1", name: "sfo", db: "sfo", region: "region1",
status: ready.

GDSCTL>services
Service "order_entry_srvc.sales.oradbcloud" has 1 instance(s).
Affinity: ANYWHERE
   Instance "sales%1", name: "sfo", db: "sfo", region: "region1",
status: ready.

GDSCTL>databases
Database: "chi" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1 Region: region2
   Service: "order_entry_srvc" Globally started: Y Started: N
            Scan: Y Enabled: Y Preferred: N
   Registered instances:
     sales%11
Database: "sfo" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1 Region: region1
   Service: "order_entry_srvc" Globally started: Y Started: Y
            Scan: Y Enabled: Y Preferred: Y
   Registered instances:
     sales%1
```

12. Check out the overall configuration:

```
GDSCTL>config

Regions
------------------------
Name                          Buddy
----                          -----
region1
region2

GSMs
------------------------
gsm1
gsm2

GDS pools
------------------------
dbpoolora
sales

Databases
------------------------
sfo
chi

Services
------------------------
order_entry_srvc

GDSCTL pending requests
------------------------
Command                       Object                        Status
-------                       ------                        ------

Global properties
------------------------
Name: oradbcloud
Master GSM: gsm1
```

4. Setup client connectivity

Add the TNS entries (based on GSM listeners) to the client's tnsnames.ora. Here is an example TNS entry for the "sales_reporting_srvc" application clients – where SERVICE_NAME is the name of the global service and the REGION is the region that the client is coming from.

Note: The connect descriptor in the tnsnames.ora in a GDS config will be using the GSM listener end points and not the RAC SCAN listeners or local listeners.

```
order_entry_srvc =
  (DESCRIPTION =
   (FAILOVER=ON)
    (ADDRESS_LIST =
     (LOAD_BALANCE=ON)
```

```
    (ADDRESS = (PROTOCOL = TCP)(HOST = serverA)(PORT = 1571))
   )
   (ADDRESS_LIST =
    (LOAD_BALANCE=ON)
    (ADDRESS = (PROTOCOL = TCP)(HOST = serverB)(PORT = 1571))
   )
   (CONNECT_DATA =
     (SERVICE_NAME = order_entry_srvc.sales.oradbcloud) (REGION=region1)
   )
 )
```

Note: 1571 is the GSM listener port of GSM1 and GSM2

GDS/Oracle GoldenGate Use cases

The following section describes the GDS use cases and the global service definition that supports the corresponding scenario.

Inter-database Global Service Failover scenario:

"Shutdown abort" the database sfo and observe that the service has failed-over to chi

```
SQL> show parameter db_unique_name

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_unique_name                       string      sfo
SQL> shutdown abort
ORACLE instance shut down.

GDSCTL>services
Service "order_entry_srvc.sales.oradbcloud" has 1 instance(s).
Affinity: ANYWHERE
   Instance "sales%11", name: "chi", db: "chi", region: "region2",
status: ready.
GDSCTL>databases
Database: "chi" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1 Region: region2
   Service: "order_entry_srvc" Globally started: Y Started: Y
            Scan: Y Enabled: Y Preferred: N
   Registered instances:
     sales%11
Database: "sfo" Registered: N State: Ok ONS: N. Role: N/A Instances: 0
Region: region1
   Service: "order_entry_srvc" Globally started: Y Started: N
            Scan: Y Enabled: Y Preferred: Y
```

Bring up the "sfo" database and then relocate the service from "chi" to "sfo" using the gdsctl relocate command

```
SQL> startup


SQL> select name, open_mode from v$pdbs;
NAME                      OPEN_MODE
------------------------  ----------
PDB$SEED                  READ ONLY
MYPDB                     MOUNTED
SQL> alter pluggable database mypdb open;
Pluggable database altered.
SQL> alter pluggable database mypdb save state;
Pluggable database altered.
```

Relocate Global Service:

While the GDS Demo App is running, execute the GDSCTL relocate command and observe that the application automatically connects to the new database where the global service has been relocated to.

```
GDSCTL>relocate service -gdspool sales -service order_entry_srvc -
old_db chi -new_db sfo
GDSCTL>services
Service "order_entry_srvc.sales.oradbcloud" has 1 instance(s).
Affinity: ANYWHERE
   Instance "sales%1", name: "sfo", db: "sfo", region: "region1",
status: ready.


GDSCTL>databases
Database: "chi" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1 Region: region2
   Service: "order_entry_srvc" Globally started: Y Started: N
            Scan: Y Enabled: Y Preferred: N
   Registered instances:
     sales%11
Database: "sfo" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1 Region: region1
```

```
   Service: "order_entry_srvc" Globally started: Y Started: Y
            Scan: Y Enabled: Y Preferred: Y
   Registered instances:
     sales%1
```

Connect and see that you got connected to "sfo" after the relocation of the global service:

Via Easy Connect Naming Method:

```
sqlplus sys/oracle@//serverB:1571/order_entry_srvc.sales.oradbcloud as
sysdba
```

Via TNSNAMES Connect Descriptor:

```
$ sqlplus sys/oracle@order_entry_srvc as sysdba

SQL> show parameter db_unique_name

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------
--------
db_unique_name                       string      sfo
```

We can query DBA_SERVICES and V$ACTIVE_SERVICES to learn more about the global services. So let's connect to the "sfo" where the global service is currently running on:

On "sfo" database:

```
SQL> column name format a30
SQL> column network_name format a40
SQL> column global format a10
SQL> set linesize 120

SQL> select name, network_name, global_service from dba_services;

NAME                           NETWORK_NAME                             GLO
------------------------------ ---------------------------------------- ----------
mypdb                          mypdb                                    NO
order_entry_srvc               order_entry_srvc.sales.oradbcloud        YES
```

Note: For the order_entry_srvc, the value of the column GLOBAL_SERVICE" is "Yes", denoting that it is a global service.

```
SQL> select name, network_name, global from v$active_services;
NAME                           NETWORK_NAME                             GLOBAL
------------------------------ ---------------------------------------- ----------
order_entry_srvc               order_entry_srvc.sales.oradbcloud        YES
mypdb                          mypdb                                    NO
```

Note: The query above shows that currently the order_entry_srvc global service is listed in the v$active_services because it has been started.

Connect-time and Run-time Load balancing:

If the objective is to load balance the workload between the two databases, the global service must be defined as follows.

```
GDSCTL>add service -service loadbal_srvc -gdspool sales -preferred_all
-pdbname mypdb -CLBGOAL LONG -RLBGOAL SERVICE_TIME

GDSCTL>config service -service loadbal_srvc -gdspool sales
Name: loadbal_srvc
Network name: loadbal_srvc.sales.oradbcloud
Pool: sales
Started: No
Preferred all: Yes
Locality: ANYWHERE
Region Failover: No
Role: NONE
Primary Failover: No
Lag: ANY
Runtime Balance: SERVICE_TIME
Connection Balance: LONG
Notification: Yes
TAF Policy: NONE
Policy: AUTOMATIC
DTP: No
Failover Method: NONE
Failover Type: NONE
Failover Retries:
Failover Delay:
Edition:
PDB: mypdb
Commit Outcome:
Retention Timeout:
Replay Initiation Timeout:
Session State Consistency:
SQL Translation Profile:


Databases
------------------------
Database                       Preferred Status
--------                       --------- ------
chi                            Yes       Enabled
sfo                            Yes       Enabled

GDSCTL> start service -service loadbal_srvc -gdspool sales
GDSCTL>services

Service "loadbal_srvc.sales.oradbcloud" has 2 instance(s). Affinity:
ANYWHERE
```

```
   Instance "sales%1", name: "sfo", db: "sfo", region: "region1",
status: ready.

   Instance "sales%11", name: "chi", db: "chi", region: "region2",
status: ready.

Service "order_entry_srvc.sales.oradbcloud" has 1 instance(s).
Affinity: ANYWHERE

   Instance "sales%1", name: "sfo", db: "sfo", region: "region1",
status: ready.

GDSCTL>databases

Database: "chi" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1 Region: region2

   Service: "loadbal_srvc" Globally started: Y Started: Y

            Scan: Y Enabled: Y Preferred: Y

   Service: "order_entry_srvc" Globally started: Y Started: N

            Scan: Y Enabled: Y Preferred: N

   Registered instances:

     sales%11

Database: "sfo" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1 Region: region1

   Service: "loadbal_srvc" Globally started: Y Started: Y

            Scan: Y Enabled: Y Preferred: Y

   Service: "order_entry_srvc" Globally started: Y Started: Y

            Scan: Y Enabled: Y Preferred: Y

   Registered instances:

     sales%1
```

The run-time load balancing capability over a set of replicas can be achieved by setting the –clbgoal and –rlbgoal global service attributes and employing the global service in the TNS entry of the clients. Configure the clients to subscribe to FAN events via GSM ONS ports and Enable FCF. (For details, refer to the GDS Documentation link shown in the Resources Section).

The –clbgoal can be set to either LONG or SHORT. For applications with long lived connections (e.g. connection pools), the –clbgoal is set to LONG. The –rlbgoal can be set to either SERVICE_TIME or THROUGHPUT.

Once the global services are created and are reflected in the TNS entries, run your application to spawn database connections and observe (with monitoring tools such as EM Cloud Control or oratop) that the database connections are well balanced based on load, network latency, clbgoal and other global service attributes.

Locality-based Routing:

The Locality based routing capability allows customers to maximize their application performance by avoiding latency overhead accessing databases in remote regions. With GDS, customers can choose to configure client connections to be always routed among a set of replicated databases in a local region. To exhibit this capability, let's create a new global service called sales_local_only_srvc and set the –locality attribute to LOCAL_ONLY.

```
GDSCTL>add service -service sales_reader_local_only_srvc -gdspool sales
-preferred_all –locality LOCAL_ONLY

GDSCTL>start service -service sales_reader_local_only_srvc -gdspool
sales


GDSCTL>services
Service "sales_reader_local_only_srvc.sales.oradbcloud" has 2
instance(s). Affinity: LOCALONLY
   Instance "sales%1", name: "sfo", db: "sfo", region: "west", status:
ready.
   Instance "sales%11", name: "chi", db: "chi", region: "east", status:
ready.

GDSCTL>databases
Database: "sfo" Registered: Y State: Ok ONS: N. Role: PH_STNDBY
Instances: 1 Region: west
   Service: "sales_reader_local_only_srvc" Globally started: Y Started:
Y
            Scan: N Enabled: Y Preferred: Y
   Registered instances:
     sales%1
Database: "chi" Registered: Y State: Ok ONS: N. Role: PRIMARY
Instances: 1 Region: east
   Service: "sales_reader_local_only_srvc" Globally started: Y Started:
Y
            Scan: N Enabled: Y Preferred: Y
   Registered instances:
     sales%11

GDSCTL>config service -service sales_reader_local_only_srvc -gdspool
sales
Name: sales_reader_local_only_srvc
Network name: sales_reader_local_only_srvc.sales.oradbcloud
Pool: sales
Started: Yes
Preferred all: Yes
Locality: LOCAL_ONLY
Region Failover: No
Role: NONE
Primary Failover: No
Lag: ANY
```

```
Runtime Balance: SERVICE_TIME
Connection Balance: LONG
Notification: Yes
TAF Policy: NONE
Policy: AUTOMATIC
DTP: No
Failover Method: NONE
Failover Type: NONE
Failover Retries:
Failover Delay:
Edition:
PDB:
Commit Outcome:
Retention Timeout:
Replay Initiation Timeout:
Session State Consistency: DYNAMIC
SQL Translation Profile:


Databases
------------------------
Database                       Preferred Status
--------                       --------- ------
chi                            Yes       Enabled
sfo                            Yes       Enabled
```

Add the following TNS entry to the client's tnsnames.ora file. In this example, we will add this TNS entry to the GSM1's (of Region1) tnsnames.ora.

```
sales_reader_local_only_srvc=
  (DESCRIPTION =
   (FAILOVER=ON)
    (ADDRESS_LIST =
     (LOAD_BALANCE=ON)
     (ADDRESS = (PROTOCOL = TCP)(HOST = serverA)(PORT = 1571))
    )
    (ADDRESS_LIST =
     (LOAD_BALANCE=ON)
     (ADDRESS = (PROTOCOL = TCP)(HOST = serverB)(PORT = 1571))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = sales_reader_local_only_srvc.sales.oradbcloud)
(REGION=region1)
    )
  )
```

And when we launch an SQLPLUS session from the HOME that has the TNS entry shown below, we will always be routed to a database in the GDS REGION Region1.

```
sqlplus sys/oracle@sales_reader_local_only_srvc as sysdba

SQL> show parameter db_unique
```

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ----------------------
--------
db_unique_name                       string      sfo

SQL> exit
```

In the above example, since the REGION attribute is set to Region1 in the client's TNS entry, the client is routed to the SFO database in the GDS region Region1.

If the TNS entry's REGION attribute is updated to Region2 (to mimic a client from the EAST region), the client will be routed to the CHI database in the GDS region Region2

```
sales_reader_local_only_srvc=
  (DESCRIPTION =
   (FAILOVER=ON)
    (ADDRESS_LIST =
     (LOAD_BALANCE=ON)
     (ADDRESS = (PROTOCOL = TCP)(HOST = serverA)(PORT = 1571))
    )
    (ADDRESS_LIST =
     (LOAD_BALANCE=ON)
     (ADDRESS = (PROTOCOL = TCP)(HOST = serverB)(PORT = 1571))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = sales_reader_local_only_srvc.sales.oradbcloud)
(REGION=region2)
    )
  )

sqlplus sys/oracle@sales_reader_local_only_srvc as sysdba

SQL> show parameter db_unique

NAME                                 TYPE        VALUE
------------------------------------ ----------- ----------------------
--------
db_unique_name                       string      chi
```

Conclusion

This concludes the exercises for the Oracle Database 12c Global Data Services (for Oracle GoldenGate) cookbook In this cookbook, you have explored and exercised some of the Global Data Services capabilities.

You have performed the following exercises:

- Installed Global Service Managers
- Deployed GDS Framework for your data cloud
- Configured global services for the following GDS features
    a. Inter-database global Service failover
    b. Connect-time and Run-time load balancing
    c. Locality based workload routing
- Obtained familiarity with the GDSCTL interface.

Resources

- GDS OTN Portal: http://www.oracle.com/goto/gds
- GDS Blog site: https://www.nageshbattula.com
- Oracle® Global Data Services 12c Documentation: http://docs.oracle.com/database/122/GSMUG/toc.htm

Appendix A: Sample Application for GDS

```
/*
 *
 This is a sample test program to demonstrate GDS automatic service failover

 - Modify userName, password and connURL in GDSBasicTest to point to your
   GDS setup

 - Compile:
   javac -cp .:$ORACLE_HOME/jdbc/lib/ojdbc8.jar:$ORACLE_HOME/ucp/lib/ucp.jar
GDSBasicTest.java

 - Run:
   java -cp
.:$$ORACLE_HOME/jdbc/lib/ojdbc8.jar:$ORACLE_HOME/ucp/lib/ucp.jar:$ORACLE_HOME/opmn/lib/on
s.jar GDSBasicTest

 The program will establish a UCP connection pool for the specified global
 service, then repeat the following operations every second for 100 iterations:
  Obtain a connection from the pool;
  Prints out the global instance name of database connected to;
  Release the connection back to the pool;
 As it is running, you can perform a service "relocate" and observe that the
 instance name of the obtained connections would change to the new database
 where the service has been relocated to.
 */

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.SQLException;

import oracle.jdbc.internal.OracleConnection;
import oracle.ucp.UniversalConnectionPoolException;
import oracle.ucp.common.UniversalConnectionPoolImpl;
import oracle.ucp.jdbc.PoolDataSource;
import oracle.ucp.jdbc.PoolDataSourceFactory;


public class GDSBasicTest {

    static final String userName="system";
    static final String password = "oracle";
    static final String poolName = "mypool";
    static final String connURL
        = "jdbc:oracle:thin:@"
        + "(DESCRIPTION="
        + " (FAILOVER=on)"
        + " (ADDRESS_LIST="
        + "  (LOAD_BALANCE=ON)"
        + "  (ADDRESS=(PROTOCOL=tcp)(host=ServerA)(port=1571)))"
        + " (ADDRESS_LIST="
        + "  (LOAD_BALANCE=ON)"
        + "  (ADDRESS=(PROTOCOL=tcp)(host=ServerB)(port=1571)))"
        + " (CONNECT_DATA="
        + "  (SERVICE_NAME=order_entry_srvc.sales.oradbcloud)))";

    Connection conn = null;

  public static void main(String[] args) throws Exception {
    testGDSDatabaseAccess();

  }

  private static void testGDSDatabaseAccess()
```

```
    throws SQLException, UniversalConnectionPoolException {

  // create UCP pool data source using the specified attributes
  final PoolDataSource pds = newUCPPoolDS();
  int numloops = 100;

  do {
    // Get a connection from the UCP pool
      Connection conn = pds.getConnection();

    // The connection now can be used to execute any query on the DB
    Statement stmt = conn.createStatement();
    ResultSet rs =
      stmt.executeQuery("select sys_context('USERENV', 'INSTANCE_NAME') "
                        + "from dual");

    rs.next();
    String instName = rs.getString(1);

    System.out.println("Connection obtained on instance = " + instName);
    rs.close();
    stmt.close();

    show("Close/return the connection to pool...");
    conn.close();
    show("Close Connection successful.");
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
       e.printStackTrace();
    }

  }while(--numloops>0);
}

public static String getUser() {
  return userName;
}

public static String getPassword() {
  return password;
}

public static void show(String message) {
  System.out.println(message);
}

public static void showError(Exception e) {
  e.printStackTrace();
}

public static String getURL() {
  return connURL;
}

public static PoolDataSource newUCPPoolDS() {
  PoolDataSource newPoolDS = null;
  try {

    final String url = getURL();
    final String user = getUser();
    final String pwd = getPassword();
    show("url:" + url);
    show("user/pwd:" + user + "/" + pwd);
    newPoolDS = PoolDataSourceFactory.getPoolDataSource();
    final String factoryName = "oracle.jdbc.pool.OracleDataSource";

    newPoolDS.setUser(user);
```

```
      newPoolDS.setPassword(pwd);
      newPoolDS.setURL(url);
      newPoolDS.setConnectionFactoryClassName(factoryName);
      newPoolDS.setInitialPoolSize(5);
      newPoolDS.setMinPoolSize(5);
      newPoolDS.setMaxPoolSize(20);
      newPoolDS.setFastConnectionFailoverEnabled(true);
      show("ucpPoolName:" + poolName);
      newPoolDS.setConnectionPoolName(poolName);
    } catch (Exception e) {
      show("getUCPConnection exception:" + e.getMessage());
    }
    return newPoolDS;
  }

}
```

Appendix B: Troubleshooting Tips

Alert Logs/Trace File Location:

GSM Alert Log Location: `/u01/app/oracle/diag/gsm/<GSM-node>/gsm1/trace/alert*.log`

GSM Listener Trace:

```
GDSCTL>status
```

GWM tracing on gdscatalog:

To get full tracing in the RDBMS, set GWM_TRACE level as shown below:

Note: Typically shared servers are used for many of the connections to the GDS catalog and GDS Pool databases and therefore the tracing is typically in a shared server trace file named `<sid>_s00*.trc` (for example).

The following command(s) activate GDS tracing for RDBMS:

For immediate tracing (Activates all GDS tracing immediately, but will be lost after an RDBMS re-start)

```
alter system set events 'immediate trace name GWM_TRACE level 7';
```

To continue tracing after re-start of RDBMS (Activates all GDS tracing permanently, but does not take effect until next RDBMS re-start)

```
ALTER SYSTEM SET EVENT='10798 trace name context forever, level 7'
SCOPE=spfile;
```

To be safe, just set both traces. To trace everything, you will need to set this on both the GDS catalog and all GDS Pool databases. The traces are written to the RDBMS session trace file for either the GDSCTL session (on the GDS catalog), or the session(s) created by the GSMs (on the GDS Pool databases).

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

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

Oracle Global Data Services with Oracle GoldenGate – Cookbook by Nagesh Battula