ORACLE

Technical Brief

Application Checklist for Continuous Service for MAA Solutions

Version [1.0]

Public

ORACLE

Table of contents

ORACLE

Introduction

The following checklist is useful for preparing your environment for continuous availability for your applications. Even if Application Continuity is not enabled on your database service, or is not used by your applications, the points discussed here provide great value in preparing your systems to support Continuous Availability.

The steps can be staged, they are building blocks:

- Use Database Services
- Configure URL or Connection String for High Availability
- Enable Fast Application Notification (FAN)
- Use Recommended Practices that Support Draining
- Enable Application Continuity or Transparent Application Continuity

The primary audience for this checklist is application developers and application owners. Operation examples are included for your DBA's and PDB administrators.

Use Database Services

Service is a logical abstraction for managing work. Services hide the complexity of the underlying system from the client by providing a single system image for managing work. Your application must connect to a service to use the high availability features: FAN, Draining and Application Continuity. This cannot be the default database service or the default PDB service (the service with the same name as the database or PDB).

Server-Side Steps for Services

To create a service, use a command similar to the provided commands.

When using multiple sites, services should be created using the primary role for the primary site, and standby role for services that will be open on secondary sites managed by Active Data Guard. Services start and stop automatically at a site based on their role.

Basic Service Creation

```
$ srvctl add service -db mydb -service MYSERVICE -preferred inst1 -available
inst2 -pdb mypdb -notification TRUE -drain_timeout 300 -stopoption IMMEDIATE
-role PRIMARY
```

Transparent Application Continuity

```
$ srvctl add service -db mydb -service TACSERVICE -pdb mypdb -preferred inst1
-available inst2 -failover_restore AUTO -commit_outcome TRUE -failovertype
AUTO -replay_init_time 600 -retention 86400 -notification TRUE -drain_timeout
300 -stopoption IMMEDIATE -role PRIMARY
```

Application Continuity

```
$ srvctl add service -db mydb -service ACSERVICE -pdb mypdb -preferred inst1 -available
inst2 -failover_restore LEVEL1 -commit_outcome TRUE -failovertype TRANSACTION -
session_state dynamic -replay_init_time 600 -retention 86400 -notification TRUE -
drain_timeout 300 -stopoption IMMEDIATE -role PRIMARY
```

TAF Select Plus

```
$ srvctl add service -db mydb -service TAFSERVICE -pdb mypdb -preferred inst1 -
available inst2 -failover_restore LEVEL1 -commit_outcome TRUE -failovertype SELECT - -
notification TRUE -drain_timeout 300 -stopoption TRANSACTIONAL -role PRIMARY
```

Notes

failoverretry and failoverdelay are not required when RETRY_COUNT and RETRY_DELAY are set in the connection string as recommended and are not shown here.

Starting with 19c, a failback option (-failback) is available in srvctl for services with preferred and available instances. When the preferred instance becomes available, a service that is not running there will failback. This option is not recommended for high availability as the database sessions incur an unnecessary outage. Services should be location transparent. However, some deployments wanting services at preferred instances may find the failback option helpful.

ORACLE

Configure URL or Connection String for High Availability

Oracle recommends that your application uses the following connection string configuration for successfully connecting at basic startup, failover, switchover, and fallback.

Set `RETRY_COUNT`, `RETRY_DELAY`, `CONNECT_TIMEOUT` and `TRANSPORT_CONNECT_TIMEOUT` parameters to allow connection requests to wait for service availability and to connect successfully. Tune these values to allow failover across RAC node failures and across Data Guard role transitions depending on your MAA solution.

RULES: (see section: *Align Application and Server Timeouts* for more details)

- Always set `RETRY_DELAY` when using `RETRY_COUNT`.
- Set `(RETRY_COUNT +1) * RETRY_DELAY > MAXIMUM of RAC and Data Guard recovery times.`
- Set `TRANSPORT_CONNECT_TIMEOUT in the range 1-5 seconds unless using a slow wide area network.`
- Set `CONNECT_TIMEOUT` to a high value to prevent login storms. Low values can result in 'feeding frenzies' logging in due to the application or pool cancelling and retrying connection attempts.
- Do not use Easy Connect Naming on the client as EZCONNECT prevents FAN auto-configuration capabilities.

You can specify the wait time units in either centiseconds (cs) or milliseconds (ms). The default unit is seconds (s).

Maintain your Connect String or URL in a central location such as LDAP or `tnsnames.ora`. Do not scatter the connect string or URL in property files or private locations as doing so makes them extremely difficult to maintain. Using a centralized location helps you preserve standard format, tuning and service settings.

This is the recommended Connection String for ALL Oracle drivers 12.2 and later, specific values may be tuned but the values quoted in this example are reasonable starting points:

```
Alias (or URL) = (DESCRIPTION =
(CONNECT_TIMEOUT=
90)(RETRY_COUNT=50)(RETRY_DELAY=3)(TRANSPORT_CONNECT_TIMEOUT=3)
 (ADDRESS_LIST =
   (LOAD_BALANCE=on)
   (ADDRESS = (PROTOCOL = TCP)(HOST=primary-scan)(PORT=1521)))
 (ADDRESS_LIST =
   (LOAD_BALANCE=on)
   (ADDRESS = (PROTOCOL = TCP)(HOST=secondary-scan)(PORT=1521)))
 (CONNECT_DATA=(SERVICE_NAME = gold-cloud)))
```

Enable Fast Application Notification (FAN)

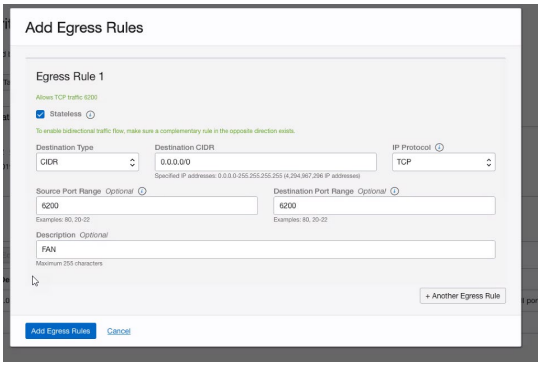
FAN is a required component for interrupting the application to failover. When a node or network fails, the application needs to be interrupted in real time. Failing to enable FAN will lead to applications hanging when HARD physical failures, such as node, network, or site failures occur.

Starting with Oracle Database 19c and Oracle client 19c, there are two important enhancements for FAN:

- FAN is sent in-band, directly to the drivers for planned events. For Java, you will need the fix for bug 31112088
- The Oracle Database and Oracle client drivers drain on connection tests and at request boundaries on receipt of FAN

Server Side

The FAN port must be opened on your RAC, RAC-One and ADG nodes. This step is very important not to miss including for ADB-D, EXA, and ExaCC, as shown in the example.



Client Side

There are no code changes to use FAN. Starting with Oracle Database 12c and Oracle client 12c, and Oracle Grid Infrastructure 12c, FAN is auto-configured and enabled out of the box. When connecting to the Oracle database, the Oracle database reads the URL or TNS connect string and auto-configures FAN at the client. Using the TNS format shown above is important for auto-configuration of FAN (using a different format prevents FAN from being auto-configured). To use FAN, you must connect to a database service (step one) and be able to receive events from the Oracle Notification Service (ONS).

To observe the receipt of FAN events, you can use the FANWatcher utility described in this paper:

http://www.oracle.com/technetwork/database/options/clustering/applicationcontinuity/learnmore/fastapplicationnotification12c-2538999.pdf

If the FANWatcher utility can subscribe to ONS using the auto-configuration method described above, and print events as they are received, this demonstrates that your application can as well. If FANWatcher is unable to perform this task you should:

- Check that the ONS port (on the Grid Infrastructure cluster) is not blocked by a firewall
- Confirm that a client subscription from your application has been created

Refer to the Section on *FAN DEBUGGING* for additional information.

Refer to the client configuration for additional client-specific steps. Note that application code changes are not required to use FAN.

Use Recommended Practices That Support Draining

There is never a need to restart application servers when planned maintenance follows best practice.

For planned maintenance, the recommended approach is to provide time for current work to complete before maintenance is started. You do this by draining work. Several methods for draining are available. Choose the one that best suits your application:

- Oracle Connection Pools
- Standard Driver-Side Connection tests
- Server-side Connection tests and more rules
- Planned failover with Transparent Application Continuity

Use draining in combination with your chosen failover solution for those requests that do not complete within the allocated time for draining. Your failover solution will try to recover sessions that are expected not to not drain in the allocated time.

Use an Oracle Connection Pool

Using an Oracle connection pool is the recommended solution for hiding planned maintenance. There is no impact to users during maintenance when your application uses an Oracle Pool and returns connections to the pool between requests. Supported Oracle Pools include UCP, WebLogic Active GridLink, Tuxedo, OCI Session Pool, and ODP.NET Managed and Unmanaged providers. No application changes whatsoever are needed to drain other than making sure that your connections are returned to pool between usages. Enabling connection tests is also recommended.

Use UCP with Third-Party Application Servers or a Connection Pool with no Request Boundaries

If you are using a third party, Java-based application server, the most effective method to achieve draining and failover is to replace the pooled data source with UCP. This approach is supported by many application servers including Oracle WebLogic Server, IBM WebSphere and IBM Liberty (XA and non-XA usage), Apache Tomcat, Hikari CP, and others. Using UCP as the data source allows UCP features such as Fast Connection Failover, Runtime Load Balancing and Application Continuity to be used with full certification.

If your application is using J2EE or Container Managed Transactions (CMT) with Red Hat JBoss, request boundaries are provided with version Red Hat JBoss EAP 7.4. This configuration supports draining with FAN (XA and non-XA usage) and Application Continuity (non-XA usage).

NOTE: Return Connections to the Connection Pool

The application should return the connection to the connection pool at end of each request. It is best practice that an application checks-out a connection only for the time that it needs it. Holding a connection instead of returning it to the pool does not perform. An application should therefore check-out a connection and then check-in that connection immediately the work is complete. The connections are then available for later use by other threads, or your thread when needed again. Returning connections to a connection pool is a general recommendation for good performance.

Use Connection Tests to Drain your Application

If you cannot use an Oracle Pool, then the Oracle client drivers 19c or Oracle Database 19c will drain the sessions for you. When services are relocated or stopped, or there is a switchover to a standby site via Oracle Data Guard, the Oracle Database and Oracle client drivers are notified to look for safe places to release connections according to the following:

- Standard connection tests for connection validity
- Predefined rules at the server for draining

- Custom SQL tests for connection validity

Do NOT use a connection test failing to decide to stop more processes. The connection test applies to that connection. The connection pool will close the connection and get another.

Use Standard Connection Tests with Thin JDBC Driver

To use connection tests with the JDBC driver:

- Set `ValidateConnectionOnBorrow = true`
- Set the Java system properties
    - `-Doracle.jdbc.fanEnabled=true (use false for ADB-S only)`
    - `-Doracle.jdbc.defaultConnectionValidation=SOCKET. (optional)`

and set the following test: `java.sql.Connection.isValid(int timeout)`

NOTE: Use *failing session option* only for your pool, disable *flushing and destroying the pool* on connection test failure.

Use Connection Tests with OCI Driver

To use the OCI driver directly, use `OCI_ATTR_SERVER_STATUS`. This is the only method that is a code change. In your code, check the server handle when borrowing and returning connections to see if the session is disconnected. When the service is stopped or relocated, the value `OCI_ATTR_SERVER_STATUS` is set to `OCI_SERVER_NOT_CONNECTED`. When using OCI session pool, this connection check is done for you.

The following code sample shows how to use `OCI_ATTR_SERVER_STATUS`:

```
ub4 serverStatus = 0
OCIAttrGet((dvoid *)srvhp, OCI_HTYPE_SERVER,
               (dvoid *)&serverStatus, (ub4 *)0, OCI_ATTR_SERVER_STATUS, errhp);
if (serverStatus == OCI_SERVER_NORMAL)
       printf("Connection is up.\n");
else if (serverStatus == OCI_SERVER_NOT_CONNECTED)
        printf("Connection is down.\n");
```

Use Connection Tests to the Oracle Database

If you cannot use an Oracle Pool, the Oracle Database 19c can drain your sessions.

Use the view `DBA_CONNECTION_TESTS` to see the connection tests and rules that are enabled for you. If using a SQL-based connection test, use the same SQL that is enabled in your database (the same identical statement) at your connection pool or application server.

If you need additional connection tests, you can add, delete, enable or disable connection tests for a service, a pluggable database, or non-container database. For example:

```
SQL> EXECUTE
     dbms_app_cont_admin.add_sql_connection_test('SELECT COUNT(1) FROM DUAL');
SQL> EXECUTE   n
SQL> SET LINESIZE 120
SQL> SELECT * FROM DBA_CONNECTION_TESTS
```

Note: For connection tests you will need the fix for Bug 31863118, which is applicable to all SQL draining, released with DBRU19.10 and later release updates.

Use `USERENV` to Drain PLSQL Workloads

If your application uses long-running PLSQL, use the function `userenv` to determine whether your session is in draining mode. For example, use this function as a check to exit the PLSQL block between batches when in a long running PL/SQL loop. This feature is available starting Oracle Database 19c, release update 10.

```
SQL> select SYS_CONTEXT('USERENV', 'DRAIN_STATUS') from dual ;

SYS_CONTEXT('USERENV','DRAIN_STATUS')
--------------------------------------------------------------------------------
DRAINING
SQL> select SYS_CONTEXT('USERENV', 'DRAIN_STATUS') from dual ;

SYS_CONTEXT('USERENV','DRAIN_STATUS')
--------------------------------------------------------------------------------
NONE
```

Use Planned Failover with Transparent Application Continuity

Oracle Database 19c introduces Planned Failover to Application Continuity. For applications that are discoverable by TAC, i.e. they close their cursors in fetch and clear or do not use Oracle complex PLSQL states, planned failover with TAC is an out of the box solution for failing over at planned and unplanned outages.

When maintenance is underway, planned failover occurs at the start of new requests and when implicit boundaries are detected by TAC. Planned failover is used by `SQL*PLUS`. It is beneficial for applications that mostly use SELECTS, INSERTS, UPDATES and DELETES. (TIP Oracle 19c only: do not set `SERVEROUTPUT`)

This feature is enabled for OCI clients in Oracle Database 19c and JDBC thin clients 19RU12 when using TAC and also AC.

Use TAF SELECT Plus

Some older OCI-based configurations may use pre-compilers (PRO*C, PRO*COBOL) or Oracle ODBC, and some may use OCI API's not yet covered by Application Continuity for OCI. For planned maintenance with older OCI-based applications, TAF SELECT PLUS may be good option to drain. To use TAF SELECT PLUS, create a separate service, with the following service attributes set: FAILOVER_TYPE=SELECT, FAILOVER_RESTORE=LEVEL1, COMMIT_OUTCOME=TRUE .

Server-Side Steps for Draining

Services connected to the Oracle Database are configured with connection tests and a `drain_timeout` specifying how long to allow for draining, and the `stopoption`, `IMMEDIATE`, that applies after the drain timeout expires. The stop, relocate, and switchover commands managed by `SRVCTL` include a `drain_timeout` and `stopoption` switch to override values set on the service if needed.

Maintenance commands are similar to the commands described below. Oracle tools, such as Fleet Patching and Provisioning (FPP) use these commands. Use these commands to start draining. Include additional options, if needed, as described in My Oracle Support (MOS) Note: Doc ID 1593712.1.

For example, to stop an instance to do RAC maintenance use the command below. Services that can relocate, will be relocated. CRS may start instances that aren't currently running but can run a service that requires that instance. Services that cannot be

1. relocated or do not need relocation, are stopped. If a singleton service is defined with no other "available" instances, then it may incur complete downtime which is expected behavior. It is better to have preferred instances always.

   Tip : If using these in scripts, you may find it helpful to include *wait = yes*.

2. After the RAC instance is restarted, no additional srvctl action is required because the clusterware service attribute will automatically determine where services will end up

For example, to stop an instance with draining:

```
srvctl stop instance -db <db_name> -node <node_name> -stopoption immediate -
drain_timeout <#> -force -failover

srvctl stop instance -db <db_name> -node <node_name> -stopoption immediate -
drain_timeout <#> -force -failover -role primary
```

to relocate all services by database, node, or PDB:

```
srvctl relocate service -database <db_unique_name> -oldinst <old_inst_name> [-
newinst <new_inst_name>] -drain_timeout <timeout> -stopoption <stop_option> -
force

srvctl relocate service -database <db_unique_name> -currentnode <current_node> [-
targetnode <target_node>] -drain_timeout <timeout> -stopoption <stop_option> -
force

srvctl relocate service -database <db_unique_name> -pdb <pluggable_database> {-
oldinst <old_inst_name> [-newinst <new_inst_name>] | -currentnode <current_node>
[-targetnode <target_node>]}  -drain_timeout <timeout> -stopoption <stop_option>
-force
```

to stop a service named *GOLD* on an instance named *inst1* (a given instance):

```
srvctl stop service -db myDB -service GOLD -instance inst1 -drain_timeout
<timeout> -stopoption <stop_option>
```

to switchover to Data Guard secondary site with a wait timeout of 60 seconds using Data Guard Broker:

```
SWITCHOVER TO dg_south WAIT 60
```

to switchover to Data Guard secondary site with a wait timeout from the services using Data Guard Broker:

```
SWITCHOVER TO dg_south WAIT
```

Enable Application Continuity or Transparent Application Continuity

Application Continuity is highly recommended for failover when your application will not drain, and for handing timeouts and unplanned outages. Application Continuity is enabled on the database service in one of two configurations:

Application Continuity (AC)

Application Continuity (AC) hides outages, starting with Oracle database 12.1 for thin Java-based applications, and Oracle Database 12.2 for OCI and ODP.NET based applications with support for open-source drivers, such as Node.js, and Python, beginning with Oracle Database 19c. Application Continuity rebuilds the session by recovering the session from a known point which includes session states and transactional states. Application Continuity rebuilds all in-flight work. The application continues as it was, seeing a slightly delayed execution time when a failover occurs. The standard mode for Application Continuity is for OLTP applications using an Oracle connection pool.

Transparent Application Continuity (TAC)

Starting with Oracle Database19c, Transparent Application Continuity (TAC) transparently tracks and records session and transactional state so the database session can be recovered following recoverable outages. This is done with no reliance on application knowledge or application code changes, allowing TAC to be enabled for your applications. Application transparency and failover are achieved by consuming the state-tracking information that captures and categorizes the session state usage as the application issues user calls.

Steps for Using Application Continuity (AC/TAC)

Return Connections to the Connection Pool

Request Boundaries are required for Application Continuity and are recommended for Transparent Application Continuity. When using an Oracle connection pool, such as Universal Connection Pool (UCP) or OCI Session Pool, or ODP.Net Unmanaged Provider or when using WebLogic Active GridLink and also RedHat JBoss, request boundaries are embedded for you. The application must return the connection to the Oracle connection pool on each request to obtain request boundaries.

Transparent Application Continuity, in addition, will discover request boundaries. The conditions for discovering a boundary in Oracle Database 19c are:

- No transaction in progress
- Cursors are not left in fetch
- No un-restorable session state exists (PLSQL globals, OJVM, populated temporary tables). Using RESET_STATE on the service will clear these for you.

FAILOVER_RESTORE on the Service

The attribute `FAILOVER_RESTORE` should be set on your database service. Use `FAILOVER_RESTORE=LEVEL1` for AC or `FAILOVER_RESTORE=AUTO` for TAC. All modifiable parameters are restored automatically by using a wallet with FAILOVER_RESTORE (refer to Ensuring Application Continuity in Oracle Real Application Clusters).

Wallets are enabled for ADBD and ADBS and are the same as those used for database links.

To configure additional custom values at connection establishment and failover use:

- A logon trigger
- Connection Initialization Callback or UCP label for Java or TAF Callback for OCI and ODP.NET
- UCP or WebLogic Server Connection Labeling

Restore Original Function Values

Support for keeping the original results of Oracle functions is provided for `SYSDATE`, `SYSTIMESTAMP`, `SYS_GUID`, `sequence.NEXTVAL`, `CURRENT_TIMESTAMP` and `LOCALTIMESTAMP`. Identity sequences are supported for owned sequences in SQL. If the original values are not kept and different values are returned to the application at replay, replay is rejected.

Oracle Database 19c `KEEPs` the original values for SQL automatically. If you are using PLSQL, then `GRANT KEEP` for application users, and the `KEEP` clause for a sequence owner.

For example:

```
SQL> GRANT KEEP DATE TIME to scott;

SQL> GRANT KEEP SYSGUID to scott;

SQL> GRANT KEEP SEQUENCE mySequence on mysequence.myobject to scott;
```

Side Effects

When a database request includes an external call such as sending MAIL or transferring a file then this is termed a side effect. When replay occurs, there is a choice as to whether side effects should be replayed. Many applications want to repeat side effects such as journal entries, and sending mail. For Application Continuity side effects are replayed (and may be disabled using disableReplay). Conversely, as Transparent Application Continuity is on by default, TAC does not replay side effects.

Monitoring

Application Continuity collects statistics to monitor your protection levels. These statistics are saved in the Automatic Workload Repository and are available in Automatic Workload Repository reports.

The following statistics are available for query:

```
Statistic
-----------------------------------------
cumulative begin requests
cumulative end requests
cumulative user calls in requests
cumulative user calls protected by Application Continuity
successful replays by Application Continuity
rejected replays by Application Continuity
cumulative DB time protected in requests
```

To report protection history by service for example you could run:

```
set pagesize 60
set lines 120
col Service_name format a30 trunc heading "Service"
break on con_id skip1
col Total_requests format 999,999,9999 heading "Requests"
col Total_calls format 9,999,9999 heading "Calls in requests"
col Total_protected format 9,999,9999 heading "Calls Protected"
col Protected format 999.9 heading "Protected %"
col time_protected format 999.999 heading "Time Prot"

select con_id, service_name, total_requests,
total_calls,total_protected,time_protected,total_protected*100/NULLIF(total_cal
ls,0) as Protected
from(
select * from
(select  a.con_id, a.service_name, c.name,b.value
  FROM   gv$session a, gv$sesstat b, gv$statname c
  WHERE  a.sid         = b.sid
  AND    a.inst_id     = b.inst_id
  AND    b.value      != 0
  AND    b.statistic# = c.statistic#
  AND    b.inst_id     = c.inst_id
  AND    a.service_name not in ('SYS$USERS','SYS$BACKGROUND'))
pivot(
  sum(value)
  for name in ('cumulative begin requests' as total_requests, 'cumulative end
requests' as Total_end_requests, 'cumulative user calls in requests' as
Total_calls, 'cumulative DB time protected in requests' as time_protected,
'cumulative user calls protected by Application Continuity' as total_protected)
))
order by con_id, service_name;
```

This would display output in the following format:

```
CON_ID Service              Requests Calls in requests Calls Protected Time Prot Protected %
---------- --------------- -------------- ------------- ----------------- --------------- ------

109   RDDAINSUH6U1OKC_TESTY_high.adb  11      7                                          63
      RDDAINSUH6U1OKC_TESTY_tp.adb.o   7      9                 9                       100
```

Reports could also be structured to show results for a PDB:

```
set lines 85
col Service_name format a30 trunc heading "Service"
break on con_id skip1
col Total_requests format 999,999,9999 heading "Requests"
col Total_calls format 9,999,9999 heading "Calls in requests"
col Total_protected format 9,999,9999 heading "Calls Protected"
col Protected format 999.9 heading "Protected %"

select con_id, total_requests,
total_calls,total_protected,total_protected*100/NULLIF(total_calls,0) as
Protected
from(
select * from
(select  s.con_id, s.name, s.value
  FROM   GV$CON_SYSSTAT s, GV$STATNAME n
  WHERE s.inst_id    = n.inst_id
  AND   s.statistic# = n.statistic#
  AND   s.value     != 0 )
pivot(
  sum(value)
  for name in ('cumulative begin requests' as total_requests, 'cumulative
end requests' as Total_end_requests, 'cumulative user calls in requests' as
Total_calls, 'cumulative user calls protected by Application Continuity' as
total_protected)
))
order by con_id;
```

Similar to:

```
CON_ID         Requests   Calls in requests Calls Protected Protected %
---------- ------------ ----------------- --------------- -----------
   854        70              283                 113            39.9
```

Or for a period of time. In this example 3 days:

```
set lines 85
col Service_name format a30 trunc heading"Service"
break on con_id skip1
col Total_requests format 999,999,9999 heading "Requests"
col Total_calls format 9,999,9999 heading "Calls in requests"
col Total_protected format 9,999,9999 heading "Calls Protected"
col Protected format 999.9 heading "Protected %"

select  a.instance_number,begin_interval_time, total_requests, total_calls,
total_protected, total_protected*100/NULLIF(total_calls,0) as Protected
from(
select * from
(select  a.snap_id, a.instance_number,a.stat_name, a.value
  FROM   dba_hist_sysstat a
  WHERE    a.value     != 0 )
pivot(
  sum(value)
  for stat_name in ('cumulative begin requests' as total_requests,
'cumulative end requests' as Total_end_requests, 'cumulative user calls in
requests' as Total_calls, 'cumulative user calls protected by Application
Continuity' as total_protected)
)) a,
dba_hist_snapshot b
where a.snap_id=b.snap_id
and a.instance_number=b.instance_number
and begin_interval_time>systimestamp - interval '3' day
```

Restrictions

Be aware of these restrictions and considerations when using Application Continuity (Restrictions and Other Considerations for Application Continuity).

Configure Clients

JDBC Thin Driver Checklist

1. Configure FAN for Java called Fast Connection Failover (FCF)
   For client drivers 12c and later
   Use the recommended URL for auto-configuration of ONS
   - Check that `ons.jar`, `simpleFan.jar` (when not using Oracle Pools) (plus optional WALLET jars, `osdt_cert.jar`, `osdt_core.jar`, `oraclepki.jar`) are on the `CLASSPATH`
   - Set the pool or driver property `fastConnectionFailoverEnabled=true`
   - For third party JDBC pools, Universal Connection Pool (UCP) is recommended
   - Open port 6200 for ONS (6200 is the default port, a different port may have been chosen)

   If you are not able to use the recommended connect string, configure your clients manually by setting:

   ```
   oracle.ons.nodes =XXX01:6200, XXX02:6200, XXX03:6200
   ```

JDBC Thin Driver Checklist For application continuity

1. Configure the Oracle JDBC Replay Data Source in the property file or on console:

For applications using 19c DBRU client jars, use `oracle.jdbc.replay.OracleDataSourceImpl` in a standalone manner, or configure them as connection factory class for a Java connection pool, such as UCP, or WebLogic AGL Server connection pool.

The 19c UCP documentation explains how to enable AC/TAC on UCP (one would configure the above JDBC driver data source class "`oracle.jdbc.replay.OracleDataSourceImpl`" as the connection factory class on the UCP data source PoolDataSourceImpl) --

https://docs.oracle.com/en/database/oracle/oracle-database/19/jjucp/application-continuity-using-ucp.html#GUID-EA541BD8-7B19-4A28-BF29-C4A623B41EEC

For applications using 21c DBRU client jars and later, use the data source that enables AC/TAC when on the service automatically. `datasource=`*`oracle.jdbc.datasource.impl.OracleDataSource (available since 21.1)`*

For WebLogic server, use the Oracle WebLogic Server Administration Console, choosing the local replay driver:
`Oracle Driver (Thin) for Active GridLink Application Continuity Connections`

2. Use JDBC Statement Cache
   Use the JDBC driver statement cache in place of an application server statement cache. This allows the driver to know that statements are cancelled and allows memory to be freed at the end of requests.
   To use the JDBC statement cache, use the connection property `oracle.jdbc.implicitStatementCacheSize` (`OracleConnection.CONNECTION_PROPERTY_IMPLICIT_STATEMENT_CACHE_SIZE`). The value for the cache size matches your number of `open_cursors`. For example:
   `oracle.jdbc.implicitStatementCacheSize=`*`nnn`* where *`nnn`* is typically between 50 and 200 and is equal to the number of open cursors your application maintains.

3. Tune the Garbage Collector
   For many applications the default Garbage Collector tuning is sufficient. For applications that return and keep large amounts of data you can use higher values, such as 2G or larger. For example:
   ```
   java -Xms3072m -Xmx3072m
   ```
   It is recommended to set the memory allocation for the initial Java heap size (`ms`) and maximum heap size (`mx`) to the same value. This prevents using system resources on growing and shrinking the memory heap.

4. Commit
   For JDBC applications, if the application does not need to use `AUTOCOMMIT`, disable `AUTOCOMMIT` either in the application itself or in the connection properties. This is important when `UCP` or the replay driver is embedded in third-party application servers such as Apache Tomcat, IBM WebSphere, IBM Liberty and Red Hat WildFly (JBoss).
   Set `autoCommit` to false through UCP `PoolDataSource` connection properties
   `connectionProperties="{autoCommit=false}"`

5. JDBC Concrete Classes – Applies to jars 12.1 and 12.2 driver ONLY
   For JDBC applications, Oracle Application Continuity does not support deprecated `oracle.sql` concrete classes BLOB, CLOB, BFILE, OPAQUE, ARRAY, STRUCT or ORADATA. (See MOS note `1364193.1` *New JDBC Interfaces*). Use `ORAchk -acchk` on the client to know if an application passes. The list of restricted concrete classes for JDBC Replay Driver is reduced to the following starting with Oracle JDBC-thin driver version 18c and later: `oracle.sql.OPAQUE`, `oracle.sql.STRUCT`, `oracle.sql.ANYDATA`

OCI (ORACLE CALL INTERFACE) DRIVER CHECKLIST (OCI-based clients include Node.js, Python, SODA in thick mode)

1. To use FAN for OCI-based applications, do the following:

- Set `aq_ha_notifications` on the services

Use the recommended Connection String for auto-configuration of ONS

Set `auto_config`, `events`, and `wallet_location` (optional) in `oraaccess.xml`

```
<default_parameters>
      (Other settings may be present in this section)
   <ons>
      <auto_config>true</auto_config>
      <wallet_location>/path/onswallet</wallet_location>
   </ons>
   <events>
      True
   </events>
</default_parameters>
```

- Many applications, including open source, will already be threaded. If not, link the application with the O/S client thread library
- Open port 6200 for ONS (6200 is the default port, a different port may have been chosen)

If you are not able to use the recommended connect string, configure your clients manually:

- Oracle Call Interface (OCI) clients without native settings can use an `oraacces.xml` file and set `events` to `true`

Python, Node.js and PHP have native options. In Python and Node.js you can set an events mode when creating a connection pool.

In PHP, edit `php.ini` adding the entry `oci8.events=on`.

SQL*Plus enables FAN by default.

OCI Driver Considerations for ApPlication continuity

Check the documentation for the complete list of supported statements. Replace OCIStmtPrepare with OCIStmtPrepare2. OCIStmtPrepare() has been deprecated since 12.2. All applications should use OCIStmtPrepare2(). TAC and AC allows OCIStmtPrepare() and other OCI APIs not covered but does not replay these statements.

https://docs.oracle.com/en/database/oracle/oracle-database/19/lnoci/high-availability-in-oci.html#GUID-D30079AC-4E59-4CC3-86E8-6487A4891BA2

https://docs.oracle.com/en/database/oracle/oracle-database/19/lnoci/deprecated-oci-functions.html#GUID-FD74B639-8B97-4A5A-BC3E-269CE59345CA

ODP.NET Unmanaged Provider Driver Checklist

1. Ensure that all recommended patches are applied at the client. Refer to the MOS Note Client Validation Matrix for Draining and Application Continuity (Doc ID 2511448.1)
2. To use FAN for OCI-based applications, do the following:

- Set `aq_ha_notifications` on the services
- Use Recommended Connection String for auto-configuration of ONS
- Set `onsConfig` and `wallet_location` (optional) in `oraaccess.xml`

Open port 6200 for ONS (6200 is the default port, a different port may have been chosen)

- Set FAN in the connection string:

```
"user id=oracle; password=oracle; data source=HA; pooling=true; HA
events=true;"
```

- (optional) Set Runtime Load Balancing, in the connection string:

```
"user id=oracle; password=oracle; data source=HA; pooling=true; HA
events=true;
load balancing=true;"
```

Use ACCHK to deep-dive into your protection

ACCHK is a database feature starting Oracle Database 19c RU11 (19.11). Database views and PL/SQL-based reports show you the level of protection for your applications for failover. If an application is not fully protected, then ACCHK identifies that application, finds out the reason why the application is not fully protected, and guides you how to increase the protection.

ACCHK uses Application Continuity data to collect coverage for a workload and provides detailed information as per your request. You must enable ACCHK to collect coverage before you execute a database workload. ACCHK also provides diagnostics for failover.

More information can be found in the Oracle *RAC Admin and Deployment Guide*.

Enable acchk

To use `acchk` connect to the database (with SQL*Plus for example) and run the following commands:

- Grant read access to the users, who will run the Application Continuity Protection Check report and views, using the `ACCHK_READ` role:

```
GRANT ACCHK_READ to myUser;
```

- Enable Application Continuity tracing for your applications using the `dbms_app_cont_admin.acchk_set` procedure:

```
EXECUTE dbms_app_cont_admin.acchk_set(true);
```

`acchk` is disabled after 600 seconds, by default. A different value can be set by providing a timeout value to the `acchk_set` procedure. For example, to disable after 300 seconds:

```
EXECUTE dbms_app_cont_admin.acchk_set(true, 300);
```

To manually disable acchk, run the following procedure:

```
EXECUTE dbms_app_cont_admin.acchk_set(false);
```

Note that when manualy disabled only new sessions are affected, tracing will not be disabled for the current sessions until the sessions are terminated.

With acchk enabled, exercise your application by running its functionality. It is not necessary to induce failures, run maintenance tasks and so on, regular runtime operation is sufficient. Once you have executed your application functions, disable acchk and then examine the in-built reports or the view-based data.

Examine the acchk report and VIEWS

After running your application's database operations, examine the acchk report:

```
EXECUTE dbms_app_cont_report.acchk_report(dbms_app_cont_report.SUMMARY)
```

Report levels are `FULL`, `WARNING`, `SUMMARY`. The default report is `SUMMARY`.

A sample report shows:

```
------------------------------------
------------ ACCHK Report ------------
------------------------------------
CON_ID SERVICE       FAILOVER PROTECTED_ PROTECTED_ REQUESTS AVG_CALLS/ PROTECTED_  AVG_TIME/  PROTECTED_TIME/ EVENT_  ERROR_ PROGRAM   MODULE            ACTION    SQL_ CALL      TOTAL
                             CALLS %   TIME %             REQUEST    CALLS/REQUEST REQUEST MS REQUEST MS       TYPE   CODE                                       ID
------ ------------ -------- --------- -------- -------- ---------- ------------- ---------- --------------- ------- ------ --------- ----------------- --------- ---- --------- ------
3      srv_tacr_pdb1 AUTO     98.734    98.432   117      9.453      9.333         2279.751   2244.014        DISABLE 41409  JDBC Thin AddCustNewOrder   Action-20      COMMIT    1

3      srv_tacr_pdb1 AUTO     98.734    98.432   117      9.453      9.333         2279.751   2244.014        REPLAY_ 41412  JDBC Thin InsertNewChecksum Action-1       SQL/PLSQL 1
                                                                                                               FAILED         Client                                     Execu
End of report.
```

Several views also exist, and can be queried to retrieve data. The views are `DBA_ACCHK_EVENTS`, `DBA_ACCHK_EVENTS_SUMMARY`, `DBA_ACCHK_STATISTICS`, and `DBA_ACCHK_STATISTICS_SUMMARY`

APPENDIX

Developer Best Practices for Continuous Availability

The following are best practices for developers when writing applications that will be highly available.

Return Connections to the Connection Pool

The most important developer practice is to return connections to the connection pool at the end of each request. This is important for best application performance at runtime, for draining work and for rebalancing work at runtime and during maintenance, and for handing failover events. Some applications have a false idea that holding onto connections improves performance. Holding a connection neither performs nor scales. One customer reported 40% reduction in mid-tier CPU and higher throughput by returning their connections to the pool.

Clean Session State between Requests

RESET_STATE is one of the most valuable developer features in Oracle Database.

When an application returns a connection to the connection pool, cursors in FETCH status, and session state set on that session remain in place unless an action is taken to clear them. For example, when an application borrows and returns a connection to a connection pool, next usages of that connection can see this session state if the application does not clean. At the end of a request, it is best practice to return your cursors to the statement cache and to clear application related session state to prevent leakage to later re-uses of that database session.

Prior to Oracle Database 21c, use `dbms_session.modify_package_state(dbms_session.reinitialize)` to clear PL/SQL global variables, use TRUNCATE to clear temporary tables, SYS_CONTEXT.CLEAR_CONTEXT to clear context and cancel your cursors by returning them to the statement cache.

With Oracle Database 21c, the service attribute, RESET_STATE, clears session state set by the application with no code required. Setting RESET_STATE to LEVEL1 on the service resets session states at explicit end of request. RESET_STATE does not apply to implicit request boundaries discovered by TAC. When RESET_STATE is used, applications can rely on the state being reset at end of request.

Clearing session state improves your protection when using TAC, TAC can re-enable more often. With Oracle Database 23c, **RESET_STATE** is available for all applications with and without TAC.

Do not embed COMMIT in PL/SQL and Avoid Commit on Success and Autocommit

It is recommended practice to use a top-level commit, (`OCOMMIT` or `COMMIT()` or `OCITransCommit`). If your application is using `COMMIT` embedded in PL/SQL or `AUTOCOMMIT or COMMIT ON SUCCESS`, it may not be possible to recover following an outage or timeout. PL/SQL is not reentrant. Once a commit in PL/SQL has executed, that PL/SQL block cannot be resubmitted. Applications either need to unpick the commit which is not sound as that data may have been read, or for batch use a checkpoint and restart technique. When using `AUTOCOMMIT or COMMIT ON SUCCESS`, the return message is lost.

If your application is using a top-level commit, then there is full support for Transparent Application Continuity (TAC), Application Continuity (AC), and TAF Select Plus (12.2). If your application is using `COMMIT` embedded in PLSQL or `AUTOCOMMIT or COMMIT ON SUCCESS`, it may not be possible to replay for cases where the call including the `COMMIT` did not run to completion.

Use ORDER BY or GROUP BY in Queries

Application Continuity ensures that the application sees the same data at replay. If the same data cannot be restored, Application Continuity will not accept the replay. When a `SELECT` uses `ORDER BY` or `GROUP BY` order is preserved. In a RAC environment the query optimizer most often uses the same access path, which can help in the same ordering of the results. Application Continuity also uses an `AS OF` clause under the covers to return the same query results everywhere `AS OF` is allowed.

Considerations for SQL*Plus

SQL*Plus is often our go to tool for trying things out. SQL*Plus of course does not reflect your actual application that will be used in production, so it is always better to use the real application test suite to test your failover plan and to measure your protection. SQL*Plus is not a pooled application so does not have explicit request boundaries. Some applications do use SQL*Plus for example for reports. To use SQL*Plus with failover check the following:

1. FAN is always enabled for SQL*Plus. Use the recommended connect string (described above) that auto-configures ONS end points for you.
2. When using SQL*plus the key is to minimize round trips to the database: https://blogs.oracle.com/opal/sqlplus-12201-adds-new-performance-features
3. SQL*Plus is supported for TAC starting Oracle 19c. For best results set a large arraysize e.g. (`set arraysize 1000`). Avoid enabling `serveroutput` as this creates unrestorable session state. This restriction is removed with Oracle Database 23c.
4. SQL*Plus is supported for AC starting Oracle 12.2. AC does not have implicit boundaries so does not reenable after your first commit.

End-to-End (e2e) Tracing

Enterprise Manager Top Consumers, TKPROF, Application Continuity statistics, ACCHK and more offer separation by services and by module and action. You will be using services. It is good practice for applications to use module and action to designate work. When using module and action tags, these are reported by EM top consumers, AWR, statistics and ACCHK. To set module and action use the driver provided API's rather than the PL/SQL package `DBMS_APPLICATION_INFO` - the API's are local driver calls so provide higher performance.

Align Application and Server Timeouts

If an application-level timeout is lower than timeouts provided for the detection and recovery times of the underlying system, then there is insufficient time available for the underlying recovery to complete. Misaligned timers can result in replay by Application Continuity starting before the system has recovered, potentially causing multiple replays to be attempted before success, or requests timing out and an error being returned to your applications or users.

Resource manager is the feature recommended by Oracle to stop, quarantine or demote long running SQL, and to block SQL from executing in the first place. If you wish to use `READ_TIMEOUT` for hung and dead systems, the value for `READ_TIMEOUT` should be above recovery timeouts. It is not advisable to use `READ_TIMEOUT` with low values. This can lead to unnecessary aborts to systems in recovery that have not failed.

Consider an application that uses `READ_TIMEOUT` or `HTTP_REQUEST_TIMEOUT` or a custom timeout, then the following guidelines apply:

`READ_TIMEOUT` > `EXADATA special node eviction (FDDN) (2 seconds)`

`READ_TIMEOUT` > `MISSCOUNT (default 30 sec, modifiable in 12c Grid Infrastructure)`

`READ_TIMEOUT` > Data Guard Observer: `FastStartFailoverThreshold` (default 30 sec, modifiable)

`FastStartFailoverThreshold` > `MISSCOUNT` (must be at least twice)

`READ_TIMEOUT` > `FAST_START_MTTR_TARGET (many systems choose 15 or 30 seconds)`

`READ_TIMEOUT` > Oracle SQL*NET level: (`RETRY_COUNT`+1) * `RETRY_DELAY`

`READ_TIMEOUT` < `Replay_Initiation_Timeout` (modifiable on the service, default 300 seconds)

To avoid premature cancelling of requests the application timeout should be larger than the maximum of:

(`MISSCOUNT` (or FDNN) + `FAST_START_MTTR_TARGET`), or

(FastStartFailoverThreshold + FAST_START_MTTR_TARGET + TIME TO OPEN)

ORACLE

Tracking your Grants for keeping original values

Use SQL similar to the following to know which grants for KEEP are set on your database.

ALTER SESSION SET CONTAINER=&PDB_NAME ;

```
set pagesize 60
set linesize 90

ttitle "Sequences Kept for Replay"
col sequence_owner format A20 trunc heading "Owner"
col sequence_name format A30 trunc heading "Sequence Name"
col keep_value format A10 trunc heading "KEEP"
break on sequence_owner

select sequence_owner, sequence_name, keep_value
from all_sequences, all_users
where sequence_owner = username
and   oracle_maintained = 'N'
order by sequence_owner, sequence_name;

ttitle "Date/Time Kept for Replay"
col grantee format A20 trunc heading "Grantee"
col PRIVILEGE format A20 trunc heading "Keep"
col ADMIN_OPTION format A8 trunc heading "Admin|Option"
break on grantee

select grantee, PRIVILEGE, ADMIN_OPTION
from dba_sys_privs, all_users
where
      grantee = username
and   oracle_maintained = 'N'
and   PRIVILEGE like '%KEEP%'
union
select distinct grantee, 'NO KEEP' PRIVILEGE, 'NO' ADMIN_OPTION
from dba_sys_privs l1, all_users
where
      grantee = username
and   oracle_maintained = 'N'
and   l1.grantee not in
       ( select l2.grantee
       from dba_sys_privs l2
       where PRIVILEGE like '%KEEP%' )
order by privilege, grantee;
```

Which should return data in a format similar to the following example:

```
Tue Nov 16                                                    page     1
                    Sequences Kept for Replay

Owner                Sequence Name                    KEEP
-------------------- -------------------------------- ----------
MOVIESTREAM          MDRS_1715A$                      Y
                     MDRS_17165$                      Y


Tue Nov 16                                                    page     1
                  Date/Time Kept for Replay

                                                 Admin
Grantee              Keep                        Option
-------------------- --------------------------- --------
ADMIN                NO KEEP                     NO
GGADMIN              NO KEEP                     NO
MOVIESTREAM          KEEP                        NO
RMAN$VPC             NO KEEP                     NO
```

Debugging FAST APPLICATION NOTIFICATION

FAN events are sent via two protocols:

- Oracle Notification Server (ONS) or full-FAN
- In-band FAN

In order to receive FAN events via ONS a subscription must be made to the ONS daemon which is publishing the events. In the simplest configuration this is the ONS daemon installed as part of Grid Infrastructure and running on the cluster which forms the database tier.

The default port for ONS is 6200, which can be confirmed with:

```
$ srvctl config nodeapps
```

The output will show the ONS configuration:

```
ONS exists: Local port 6100, remote port 6200, EM port 2016, Uses SSL true
ONS is enabled
ONS is individually enabled on nodes:
ONS is individually disabled on nodes:
```

The Remote Port is the port through which events are sent. This port needs to be open for TCP POSTS through any firewall that may be configured.

To confirm that a subscription to ONS has been opened by your client application, if you have access to the GI cluster you can examine connections to the ONS daemon at runtime.

```
$ onsctl debug
```

Which will show you the following information:

The remote Port and the local host IP address (6200 in this example)

The clients connected at runtime where a connected client's IP will be displayed

```
            IP ADDRESS     PORT    TIME    SEQUENCE  FLAGS
----------------------- ----- -------- -------- --------
            123.123.1.123  6200 6205af70 00000008 00000008

<<< Information removed >>>

Client connections: (5)

   ID            CONNECTION ADDRESS               PORT   FLAGS  SNDQ REF PHA SUB
-------- ---------------------------------------- ----- ------- ---- --- --- ---
       0                                 internal     0 000044a    0   1  IO   1
       3                                127.0.0.1 16576 040041a    0   1  IO   0
       d                                127.0.0.1 16598 040041a    0   1  IO   1
       e                                127.0.0.1 16600 040041a    0   1  IO   1
       f                  ::ffff:100.101.102.103 61838 04c042a    0   1  IO   1
```

Have you set the appropriate settings on the client side for FAN?

- For Universal Connection Pool with JDBC thin driver, ensure the Boolean pool property `FastConnectionFailoverEnabled = true` is set when using Universal Connection Pool with the JDBC thin driver
- For ODP.Net, ensure that `pooling=true; HA events=true` is set in the connect string
- For OCI clients
    - Set "`<events>true</events>`" in `oraaccess.xml` and
    - enable notifications on the dynamic database service `srvctl modify service -notification TRUE`
- For WebLogic Active Grid Link, FAN is on by default. This is visible in the admin console. Also set `ons.configuration` at the admin console for the ONS end points.

Additional Materials

Oracle Technology Network (OTN) Home page for Application Continuity

http://www.oracle.com/goto/ac

Application Continuity

Ensuring Application Continuity (https://docs.oracle.com/en/database/oracle/oracle-database/21/racad/ensuring-application-continuity.html#GUID-C1EF6BDA-5F90-448F-A1E2-DC15AD5CFE75)

Graceful Application Switchover in RAC with No Application Interruption
My Oracle Support (MOS) Note: Doc ID 1593712.1

Embedding UCP with JAVA Application Servers:

WLS UCP Datasource, https://blogs.oracle.com/weblogicserver/wls-ucp-datasource

*Design and Deploy WebSphere Applications for Planned, Unplanned Database Downtimes and Runtime Load Balancing with UCP (*http://www.oracle.com/technetwork/database/application-development/planned-unplanned-rlb-ucp-websphere-2409214.pdf)

Reactive programming in microservices with MicroProfile on Open Liberty 19.0.0.4
(https://openliberty.io/blog/2019/04/26/reactive-microservices-microprofile-19004.html#oracle)

*Design and deploy Tomcat Applications for Planned, Unplanned Database Downtimes and Runtime Load Balancing with UCP (*http://www.oracle.com/technetwork/database/application-development/planned-unplanned-rlb-ucp-tomcat-2265175.pdf).

Fast Application Notification

http://www.oracle.com/technetwork/database/options/clustering/applicationcontinuity/learnmore/fastapplicationnotification12c-2538999.pdf

ORACLE

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