19c ORACLE Database

Application Checklist for Continuous Service with Autonomous Database on Shared Infrastructure

ORACLE Technical Brief / November 9, 2022

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 when using the Autonomous Database on Shared Infrastructure (ADB-S).

The steps can be staged, they are building blocks:

- Choose your Database Service
- Use the URL or Connection String for High Availability
- Use Recommended Practices that Support Draining
- Enable Application Continuity or Transparent Application Continuity
- Align Application Timeouts

You will need a minimum Oracle Database 12c client (or later) in order to use Application Continuity with an Oracle Database 19c database client extending this to support Transparent Application Continuity. However, you will get benefit from service usage, and draining practices for earlier Oracle clients.

The primary audience for this checklist is application developers and application owners using the Autonomous Database on Shared Infrastructure.

CHOOSE YOUR DATABASE SERVICE

Services provide transparency for the underlying ADB-S infrastructure. Draining, Transparent Application Continuity (TAC), Application Continuity (AC), consumer groups and many other features and operations are predicated on the use of services.

Oracle's Autonomous Database Shared (ADB-S) offers up to five preconfigured services to choose from. All provide in-band FAN and draining for maintenance. An API is available to enable TAC or AC settings on all preconfigured services.

Pre-configured Services offered by the Oracle Autonomous Database

SERVICE NAME	DESCRIPTION	DRAINING	INBAND FAN	TAC/ AC ALLOWED
TPURGENT	OLTP Highest Priority	Yes	Yes	Yes
TP	OLTP General Priority (Use as main service)	Yes	Yes	Yes
HIGH	Reporting or Batch (Highest Priority)	Yes	Yes	Yes
MEDIUM	Reporting or Batch (Medium Priority)	Yes	Yes	Yes
LOW	Reporting or Batch (Lowest Priority)	Yes	Yes	Yes

To help in choosing the service for batch work:

HIGH: Queries run with a Degree of Parallelism equal to `CPU_COUNT`. There is a limit of three concurrent queries after which statement queuing occurs.

MEDIUM: Queries run with a Degree of Parallelism of four. The maximum number of queries that can run simultaneously is (`CPU_COUNT`*1.25).

LOW: Queries run serially. Queueing starts when concurrent queries exceed (2*`CPU_COUNT`).

USE THE URL OR CONNECTION STRING PROVIDED FOR HIGH AVAILABILITY

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

You have received a ZIP file with the URL/TNS that you can customize. 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 there values to allow the application to pause reconnecting during RAC failovers and switchovers.

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.

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)))
(CONNECT_DATA=(SERVICE_NAME = [TPURGENT…])))
```

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 in decreasing order of value. Choose the one that best suits your application:

- Oracle Connection Pools (In-Band FAN is built into 19c drivers to tell the clients when to drain)
- Standard Driver-Side Connection tests
- Server-side with SQL Connection tests
- 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 did not drain in the allocated time.

Draining Method One : Use a Connection Pool

Use an Oracle Connection Pool

Using an Oracle connection pool is the recommended solution for hiding planned maintenance. There is no impact to users when your application uses an Oracle Pool with In-Band FAN 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 requests. Enabling connection tests is recommended.

Use UCP with a Third-Party Connection Pool or a Pool with 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, IBM Liberty, Apache Tomcat, Spring, Hibernate, 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. UCP may not be used for J2EE -based applications or with XA-based transactions.

If your application is using J2EE or Container Managed Transactions (CMT) with Red Hat JBoss, request boundaries are provided with version Red Hat JBoss 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 on 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.

Draining Method Two: Use Connection Tests to Drain your Application

If you cannot use an Oracle Pool, then the Oracle client drivers 19c (and later) will drain the sessions. 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 rules. Choose the one that best suits your application:

- Standard driver-based connection tests for connection validity at borrow or return from a connection pool
- Custom SQL tests for connection validity

USE STANDARD CONNECTION TESTS WITH THIN JAVA DRIVER

If you would like to use connection tests that are local to the driver:

- Enable `validate-on-borrow=true`
- Set the Java system properties
  - `-Doracle.jdbc.fanEnabled=false`
  - `-Doracle.jdbc.defaultConnectionValidation=SOCKET`

and use one of the following tests, `isValid()` is the preferred method:

- `java.sql.Connection.isValid(int timeout) or`
- `oracle.jdbc.OracleConnection.pingDatabase() or`
- `oracle.jdbc.OracleConnection.pingDatabase(int timeout) or`
- `a` HINT at the start of your test SQL:
  - `/*+ CLIENT_CONNECTION_VALIDATION */`

IMPORTANT: If using in-Band FAN with UCP, you will need the fix for bug 31112088, and your application should return your connections to the pool between requests. Doing so will drain at the end of the request.

Your connection tests is set in your connection pool property: See Standard Connection Tests for Some Common Application Servers

USE STANDARD CONNECTION TESTS WITH OCI DRIVER

If you would like 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");
```

Draining Method Three: Use SQL Connection Tests to the Oracle Database

If you cannot use either an Oracle Pool or use connection tests at the Oracle client drivers, the Oracle Database 19c (and later) can drain your sessions.

Use the view `DBA_CONNECTION_TESTS` to see the connection tests added and enabled. 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
dbms_app_cont_admin.enable_connection_test(dbms_app_cont_admin.sql_test,
                      'SELECT COUNT(1) FROM DUAL');
SQL> SET LINESIZE 120
SQL> SELECT * FROM DBA_CONNECTION_TESTS
```

Configure the same connection test that is enabled in your database (the same identical statement) at your connection pool or application server. Also disable flushing and destroying the pool on connection test failure, or set it to at least two times the maximum pool size or MAXINT.

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.

CHECK FOR DRAINING WITH ORACLE AUTONOMOUS DATABASE SHARED

Use the function `userenv` to determine whether your session is in draining mode. For example, use this function as a check to exit PLSQL when in a long running PL/SQL loop processing records. This feature is available starting Oracle Database 19c, release update 10 and later release updates (refer to Bug 32761229).

```
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
```

Alternate Method: Use Planned Failover with Transparent Application Continuity

Oracle Database 19c introduces the concept of 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. This failover includes long running and standalone requests that rely on implicit boundaries are discovered by Transparent Application Continuity. Planned failover is used by `SQL*PLUS` with TAC starting 19c, (TIP: do not set `SERVEROUTPUT`) and is beneficial for applications that mostly use SELECTS, INSERTS, UPDATES and DELETES..

This feature is available for OCI clients in Oracle Database 19c and JDBC thin clients 19RU12

Use Planned Failover with 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, and to drain `stopoption TRANSACTIONAL`. Sessions will automatically failover during the drain timeout at COMMIT boundaries.

ENABLE APPLICATION CONTINUITY OR TRANSPARENT APPLICATION CONTINUITY

Application Continuity is highly recommended for failover when your application will not drain, for planned failover, and for handing timeouts as well as for unplanned outages. It is not mandatory but adds significant benefits.

Application Continuity is enabled on the database service in one of two configurations, depending on the application:

Application Continuity (AC)

Application Continuity hides outages, starting with Oracle database 12.1 for thin Java-based applications, and Oracle Database 12.2.0.1 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 Transparent Application Continuity 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

Developers should work through these steps with the PDB Administrators for database configuration.

Enable Application Continuity on Your Service

You can change the failover type offered on your service by using the generic package `DBMS_APP_CONT_ADMIN`. Use this API to enable Application Continuity or Transparent Application Continuity, or to disable failover. New sessions will use the new failover type.

To use these procedures you must have been granted the role `PDBADMIN`. Use your FULL service name in these examples.

To enable Transparent Application Continuity for your service:

```
execute DBMS_APP_CONT_ADMIN.ENABLE_TAC('TPURGENT');
```

To enable Application Continuity for your service:

```
execute DBMS_APP_CONT_ADMIN.ENABLE_AC('TPURGENT');
```

To disable failover for your service:

```
execute DBMS_APP_CONT_ADMIN.DISABLE_FAILOVER('HIGH');
```

Return Connections to the Connection Pool

The application should return the connection to the Oracle connection pool on each request. Best practice for application usage is to check-out (borrow) connections for only the time that they are needed, and then check-in to the pool when complete for the current actions. This is important for best application performance at runtime, for rebalancing work at runtime and during maintenance and failover events. This practice is also important for draining.

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, following this practice embeds request boundaries that Application Continuity uses to identify safe places to resume and end capture. This is required for Application Continuity and is recommended for Transparent Application Continuity.

Transparent Application Continuity, in addition, will discover request boundaries if a pool is not in use or when replay is disabled. The conditions for discovering a boundary in Oracle Database 19c are:

- No open transaction
- Cursors are returned to the statement cache or cancelled
- No un-restorable session state exists (refer to Clean Session State between Requests in this paper)

FAILOVER_RESTORE on the Service

`FAILOVER_RESTORE` is set on your service to restore common session states at failover. All modifiable system parameters outside of (and including) this common set, starting with Oracle Database 19c RU8, are restored at failover by using a wallet with FAILOVER_RESTORE (refer to *Ensuring Application Continuity* in the Real Application Clusters Administration and Deployment Guide in the Oracle documentation). This is preconfigured for you when using Oracle Autonomous Database.

To configure additional custom values at connection establishment use:

- A logon trigger.
- Connection Initialization Callback or UCP label for Java or TAF Callback for OCI, ODP.Net or open source drivers
- UCP or WebLogic Server Connection Labeling

Enable Mutables Used in the Application

Mutable functions are functions that can return a new value each time they are executed. Support for keeping the original results of mutable functions is provided for `SYSDATE`, `SYSTIMESTAMP`, `CURRENT_TIMESTAMP`, `LOCALTIMESTAMP`, `SYS_GUID`, and `sequence.NEXTVAL`. 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 automatically `KEEPs` sequences for SQL. We recommend that you configure mutables using `GRANT KEEP` for application users, and the `KEEP` clause for a sequence owner. When `KEEP` privilege is granted, replay applies the original function result at replay.

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.

Side effects are external actions, they do not roll back. When configuring for replay, a choice can be made as to whether side effects should be replayed or not. Many applications choose to repeat side effects such as journal entries and sending mail. For Application Continuity (AC) side effects are replayed unless the request or user call is explicitly disabled for replay. Conversely, TAC does not replay side effects. The capture is disabled, and re-enables at the next implicit boundary created by TAC.

DEVELOPER BEST PRACTICES FOR CONTINUOUS AVAILABILITY

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 just by returning their connections to the pool.

Clean Session State between Requests

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 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, **`RESET_STATE`** is one of the most valuable developer features. `RESET_STATE` clears session state set by the application in a request with no code required. Setting the service attribute `RESET_STATE` to `LEVEL1` resets session states at explicit end of request. `RESET_STATE` does not apply to implicit request boundaries. When `RESET_STATE` is used, applications can rely on the state being reset at end of request. `RESET_STATE` is available for ADB-S Oracle Database 21c by using `DBMS_APP_CONT_ADMIN`.

Clearing session state improves your protection when using TAC, TAC can re-enable more often.

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 re-entrant. 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 output is lost.

If your application is using a top-level commit, then there is full support for Transparent Application Continuity (TAC) and Application Continuity (AC). 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 that 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 where `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 our 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:

- In-Band FAN is always enabled for SQL*Plus.
- 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
- SQL*Plus is supported for TAC starting Oracle 19c. For best results:
    a. set a large arraysize e.g. (`set arraysize 1000`).
    b. Avoid enabling `serveroutput` as this creates unrestorable session state. (Check release notes for this restriction removed.)

Restrictions

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

VERIFY PROTECTION LEVELS

Use the statistics for request boundaries and protection level to monitor the level of coverage. Application Continuity collects statistics from the system, the session, and the service, enabling you to monitor your protection levels. The statistics are available in `V$SESSTAT`, `V$SYSSTAT`, and in Oracle Database 19c, `V$SERVICE_STATS`. 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 %"

select con_id, service_name, total_requests,
total_calls,total_protected,total_protected*100/NULLIF(total_calls,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 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
order by a.snap_id,a.instance_number;
```

CONFIGURE CLIENTS

JDBC THIN DRIVER CHECKLIST

1. Ensure that all recommended patches are applied at the client. Refer to the MOS Note *Client Validation Matrix for Application Continuity* (Note 2511448.1)

JDBC THIN DRIVER CHECKLIST FOR APPLICATION CONTINUITY

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

    a. For Universal Connection Pool (UCP)
    Configure the Oracle JDBC Replay Data Source as a connection factory on UCP `PoolDataSource`:
    `setConnectionFactoryClassName("oracle.jdbc.replay.OracleDataSourceImpl");` Or

    `setConnectionFactoryClassName("oracle.jdbc.replay.OracleXADataSourceImpl");` Or preferred set these in the property file

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

c. Standalone Java applications or 3rd-party connection pools
Configure the Oracle JDBC 12c Replay Data Source in the property file or in the thin JDBC application:
`datasource=oracle.jdbc.replay.OracleDataSourceImpl` (for non-XA) or
`datasource=oracle.jdbc.replay.OracleXADataSourceImpl` (for XA)

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 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 and others starting Oracle 19c)

1. Ensure that all recommended patches are applied at the client. Refer to the MOS Note MOS Note *Client Validation Matrix for Application Continuity* (Note 2511448.1)

ODP.NET UNMANAGED PROVIDER DRIVER CHECKLIST

1. Ensure that all recommended patches are applied at the client. Refer to the MOS Note MOS Note *Client Validation Matrix for Application Continuity* (Note 2511448.1)

TRACKING YOUR GRANTS FOR MUTABLES

Use SQL similar to the following to know which grants for mutables 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 would show output similar to:

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

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



Tue Nov 16                                              page    1
                Date/Time Kept for Replay

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

ADDITIONAL MATERIALS

Oracle Technology Network (OTN) Home page for Application Continuity

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

Application Continuity

Continuous Availability, Application Continuity for the Oracle Database (https://www.oracle.com/technetwork/database/options/clustering/applicationcontinuity/applicationcontinuityformaa-6348196.pdf)

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

Application Continuity with Oracle Database12c Release 2 (http://www.oracle.com/technetwork/database/options/clustering/applicationcontinuity/overview/application-continuity-wp-12c-1966213.pdf)

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

ORACLE CORPORATION

Worldwide Headquarters
500 Oracle Parkway, Redwood Shores, CA 94065 USA

Worldwide Inquiries
TELE + 1.650.506.7000 + 1.800.ORACLE1
FAX + 1.650.506.7200
oracle.com

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/oracle

 facebook.com/oracle

 twitter.com/oracle

Integrated Cloud Applications & Platform Services

September 2022
Authors: Carol Colrain, Troy Anthony.

Contributing Authors: Ian Cookson

 Oracle is committed to developing practices and products that help protect the environment