19c ORACLE Database

Application Continuity

MAA Checklist for Preparation

ORACLE white paper / August 27, 2020

ORACLE

INTRODUCTION

The following checklist is useful for preparing your environment for using the Oracle Database feature Application Continuity. 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.

Follow these steps:

- 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
- Verify Protection Levels
- Configure Clients
- Align Application and Server Timeouts

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. You must use services to use Application Continuity. This cannot be the default database service or the default PDB service.

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 Active Data Guard. Services start and stop automatically at a site based on their role.

To create a service to use Application Continuity use a command similar to the following commands:

Note: `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.

Transparent Application Continuity

```
$ srvctl add service -db mydb -service TACSERVICE -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
```

Application Continuity

```
$ srvctl add service -db mydb -service ACSERVICE -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
```

CONFIGURE URL OR CONNECTION STRING FOR HIGH AVAILABILITY

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

Set `RETRY_COUNT`, `RETRY_DELAY`, `CONNECT_TIMEOUT` and `TRANSPORT_CONNECT_TIMEOUT` parameters in the `tnsnames.ora` file or in the URL to allow connection requests to wait for service availability and connect successfully. Use values that allow for your RAC and Data Guard failover times.

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.

Always set `RETRY_DELAY` when using `RETRY_COUNT`.

Beginning with Oracle Database 12c Release 2 (12.2) you can specify the wait time units in either centiseconds (cs) or milliseconds (ms). The default unit is seconds (s). If you are using JDBC driver for Oracle Database Release 2, you will need to apply the patch for Bug `26079621`.

Do not use Easy Connect Naming on the client as EZCONNECT prevents FAN auto-configuration capabilities.

Maintain your TNS/URL in a central location such as LDAP or tnsnames.ora. Do not scatter the tns/url in property files or private locations as doing so makes them extremely difficult to maintain. Using a centralised 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:

```
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)))
```

For JDBC connections in Oracle Database 12c Release 1 (12.1) the following should be used, ,specific values may be tuned. The patch for bug 19154304 (Patch Request 18695876) must be applied for RETRY_COUNT functionality.

```
(DESCRIPTION =
(CONNECT_TIMEOUT= 15)(RETRY_COUNT=50)(RETRY_DELAY=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 needs to be enabled. FAN is a mandatory 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:

- FAN is default, auto-configured and enabled out of the box with Oracle Real Application Clusters (RAC). When an application connects, FAN reads the URL or TNS connect string and auto-configures itself at the client. Using the URL format shown above is important for auto-configuration of FAN (using a different format prevents FAN from being auto-configured)

Starting with Oracle Database 18c and Oracle client 18c, there are two important enhancements:

- FAN is sent in-band, directly to the drivers, for planned maintenance (DOWN events)
- The Oracle Database and Oracle client drivers drain on connection tests and at request boundaries

FAN will already be enabled on the database and cluster server by using services.

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. 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.

Client Steps for Draining

THE RECOMMENDED APPROACH

Using a FAN-aware, Oracle connection pool is the recommended solution for hiding planned maintenance. Oracle pools provide full lifecycle: draining, reconnecting and rebalancing across the MAA system. As the maintenance progresses and completes, sessions are moved and rebalanced. There is no impact to users when your application uses an Oracle Pool with FAN and returns connections to the pool between requests. No application changes whatsoever are needed to use FAN.

Best practice for application usage is to check out connections for 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, and for rebalancing work at runtime and during maintenance and failover events.

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, Red Hat WildFly (JBoss), Spring, and 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.

ALTERNATIVE APPROACHES – DRAINING BY THE ORACLE DATABASE 19C OR THE DRAINING BY THE ORACLE DRIVER 19C

If you cannot use an Oracle pool or you do not use FAN, the Oracle Database 18c and later and the Oracle client drivers 18c and later drain the sessions. When services are relocated or stopped, or there is a switchover to Oracle Data Guard, the Oracle Database and Oracle client drivers look for safe places to release connections according to the following rules:

- Standard application server connection tests for connection validity
- Custom SQL tests for connection validity
- Request boundaries are in effect and the current request has ended

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
```

Refer to client configuration for client-specific configuration for using FAN with Oracle pools or connection tests.

Server 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.

Relocate all services by database, node or pdb on RAC

```
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
```

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>
```

Stop a service named GOLD on all instance(s)

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

ENABLE APPLICATION CONTINUITY OR TRANSPARENT APPLICATION CONTINUITY

Application Continuity is setup 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.

	TAC	AC
I DON'T KNOW HOW THE APPLICATION IS IMPLEMENTED	YES	NO
MY APPLICATION IS STATELESS SUCH AS REST, APEX OR MICROSERVICES	YES	YES – recommended
MY APPLICATION DOES TRANSACTIONS	YES	YES
MY APPLICATION USES ORACLE SESSION STATE (TEMP LOBS, TEMP TABLES, PL/SQL GLOBAL VARIABLES)	YES	YES NO for STATIC mode
MY APPLICATION DOES NOT USE ORACLE CONNECTION POOLS	YES	NO
MY APPLICATION HAS SIDE EFFECTS (EXTERNAL ACTIONS, FOR EXAMPLE FILE TRANSFERS OR EMAIL)	YES Side effects will not be replayed	YES Customizable as to whether side effects are replayed or not

Note that there are some restrictions on the ability of Application Continuity and Transparent Application Continuity to replay database requests. For details on these restrictions please refer to the Oracle documentation[1].

STEPS FOR USING APPLICATION CONTINUITY

Developers should work through these steps and request assistance from the Database Administrators for database configuration.

Return Connections to the Connection Pool

The application should return the connection to the Oracle 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.

When using an Oracle connection pool, such as Universal Connection Pool or OCI Session Pool, or ODP.Net Unmanaged Provider or when using WebLogic Active GridLink, following this practice also 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.

[1] Refer to most recent Oracle documentation, for version 19c see https://docs.oracle.com/en/database/oracle/oracle-database/19/racad/ensuring-application-continuity.html#GUID-2400FAAD-0BB2-48AF-B1F6-358EBA724028

Transparent Application Continuity will also 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)**

It is best practices to clean session state between database requests. Refer to the section below.

Configure FAILOVER_RESTORE on the Service

The attribute `FAILOVER_RESTORE` should be set on your database service.

If session state is set intentionally on connections outside requests, and requests expect this state, replay needs to re-create this state before replaying.

Most common session states are restored automatically by setting `FAILOVER_RESTORE` on the service. Starting with Oracle Database 19c RU 6, all modifiable system parameters are restored at failover using a wallet (refer to Ensuring Application Continuity in the Real Application Clusters Administration and Deployment Guide in the Oracle documentation). An example of an additional parameter that would need wallets is `USE_STORED_OUTLINES` if set through an `ALTER SESSION` command. If the parameter is specified in the `init.ora` or through a custom method such as a logon trigger, there is no action required.

Use `FAILOVER_RESTORE=LEVEL1` for AC or `FAILOVER_RESTORE=AUTO` for TAC. To configure 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
- Universal Connection Pool 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`, `SYS_GUID`, and `sequence.NEXTVAL`. 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` mutables for SQL.

If you need mutables for PL/SQL, or you are using a database version before Oracle Database 19c, then 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 to scott on mysequence.myobject;
```

Side Effects

When a database request includes an external call such as sending MAIL or transferring a file then this is a side effect.

Side effects are important to understand because they are external actions. External actions are not transactional, they do not roll back. When replay occurs, there is a choice as to whether side effects should be replayed. Many applications choose to repeat side effects such as journal entries and sending mail. For Application Continuity side effects are replayed unless the request or user call is explicitly disabled for replay. Conversely, as Transparent Application Continuity is on by default, TAC uses a conservative approach and does not replay side effects. The capture is disabled, and reenables at next implicit boundary created by TAC.

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. If your application is setting state, 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.

Cleaning your session state ensures that TAC can discover boundaries.

Use `DBMS_SESSION.RESET_PACKAGE` 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. Doing this will avoid state leakage and fetching from cursors by later usage of the connection pool.

If your application is stateless, such as REST, APEX or Microservices, it is still best practice to clear state.

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. Note that AS OF is not used in transactions or within PL/SQL.

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 steps:

1. FAN is always enabled for SQL*Plus. Use the above TNS 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. (Check release notes for this restriction removed.)
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

Application Continuity statistics and ACCHK offer separation by services and also 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, DBMS_APPLICATION_INFO, because the API's are local driver calls so provide higher performance.

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. (For statistics when using Oracle Database 18c, set _request_boundaries=3).

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 report query will result in the following example:

```
Statistic                          Total              per Second       per Trans
--------------------------------  ------------------  --------------  -------------
cumulative begin requests          1,500,000           14,192.9             2.4
cumulative end requests            1,500,000           14,192.9             2.4
cumulative user calls in request   6,672,566           63,135.2            10.8
cumulative user calls protected    6,672,566           63,135.2            10.8
```

Running the acchk Coverage Report

The acchk feature of Orachk provides a report detailing your protection level. To use acchk do the following steps:

Turn on tracing for the database.

Before running the workload, run the following statement as DBA on a test Oracle Database server so that the trace files include the needed information.

```
SQL> alter system set events='10602 trace name context forever, level 28:
trace[progint_appcont_rdbms]:10702 trace name context forever, level 16';
```

Run through the application functions. To report on an application function, the application function must be run. The more application functions run, the better the information that the coverage analysis provides.

To turn off the events when complete you can use:

```
SQL> alter system set events='10602 trace name context forever, off:
trace[progint_appcont_rdbms] off : 10702 trace name context forever, off';
```

Use Oracle ORAchk to analyze the collected database traces and report the level of protection, and where not protected, reports why a request is not protected. Use the following command:

```
./orachk -acchk -javahome /scratch/nfs/jdk1.8.0_171  –apptrc $ORACLE_BASE/diag/rdbms/myDB/myDB1/trace
```

Command-Line Argument	Shell Environment Variable	Usage
-javahome JDK8dirname	RAT_JAVA_HOME	This must point to the JAVA_HOME directory.
-apptrc dirname	RAT_AC_TRCDIR	To analyze the coverage, specify a directory name that contains one or more database server trace files. The trace directory is generally, $ORACLE_BASE/diag/rdbms/{DB_UNIQUE_NAME}/$ORACLE_SID/trace

CONFIGURE CLIENTS

Application Continuity and JDBC Applications

1. Ensure that all recommended patches are applied at the client. Refer to the MOS Note *Client Validation Matrix for Application Continuity (Doc ID 2511448.1)*
2. 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)
3. 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.
4. 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.
5. 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}"
```

6. 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`

7. 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` (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
```

Application Continuity and OCI-based Clients

OCI-based clients include Node.js, Python, SODA and others starting Oracle 19c.

OCI (ORACLE CALL INTERFACE) DRIVER CHECKLIST

1. Ensure that all recommended patches are applied at the client. Refer to the MOS Note *Client Validation Matrix for Application Continuity (Doc ID 2511448.1)*
2. 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

3. 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)
    <events>
        True
    </events>
    <ons>
        <auto_config>true</auto_config>
        <wallet_location>/path/onswallet</wallet_location>
    </ons>
</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 by setting:

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.

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 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, also in the connection string -

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

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 and replay 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 recommended feature 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 retry floods and premature aborts.

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`), (`FastStartFailoverThreshold` + `FAST_START_MTTR_TARGET` + TIME TO OPEN)

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/18/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*)*

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).

Using Universal Connection Pool with JBoss AS (https://blogs.oracle.com/dev2dev/using-universal-connection-pooling-ucp-with-jboss-as)

Fast Application Notification

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

May 2020
Author: Carol Colrain, Troy Anthony, Ian Cookson
Contributing Authors:

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