19c ORACLE Database

Continuous Availability

MAA Checklist for Applications for the Oracle Database

ORACLE WHITE PAPER | MARCH 2019

Table of Contents

Introduction

Applications achieve continuous service when planned maintenance, unplanned outages and load imbalances of the database tier are hidden. A combination of application best practice, simple configuration changes and the Oracle Database will ensure that your applications are continuously available.

The following checklist is useful for preparing your application and database for achieving continuous availability, even if you are not yet using Application Continuity. The points discussed here provide significant value in preparing your systems to support Continuous Availability.

Feature Set for Keeping Your Application Continuously Available

Applications achieve continuous availability when planned maintenance, unplanned outages, and load imbalances of the database tier are hidden. Oracle provides a set of features that you can choose from to keep your application available during planned and unplanned outages and load imbalances. You can think of these features as an insurance policy should an outage occur in the future. The best features are those that are fully transparent to your application so your application developers can focus on building functionality, not infrastructure, and that continue to protect the application when it changes in the future. We call this future-proofed.

Start with the feature set:

Draining and Rebalancing Sessions for Planned Maintenance

When planned maintenance starts sessions that need to be drained from an instance, PDB, or database are marked to be drained. Immediately idle sessions are released. Active sessions are drained when the work executing in that session completes. Draining of sessions is in wide use with Fast Application Notification (FAN) aware clients and Oracle connection pools and mid-tiers. Starting with Oracle Database 18c, the autonomous Oracle database itself drains sessions when PDBs and instances are stopped. Draining is always the best solution for hiding planned maintenance. Failover solutions are the fallback when work will not drain in the time allocated and when unplanned outages occur.

Transparent Application Failover (TAF)

TAF is a feature dating back to Oracle8*i*. Following an instance failure, TAF creates a new session and, when using SELECT mode, on demand, replays queries back to where they were before the failure occurred. Starting with Oracle Database 12.2, TAF offers `FAILOVER_RESTORE,` matching Application Continuity, to restore the initial session state before queries are replayed. Cursors are replayed using the state re-established initially. Applications using TAF must not change session state later in the session, (for example PLSQL, temp tables, temp lobs, sys context) as this session state is not restored.

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. 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-style pooled applications.

Transparent Application Continuity (TAC)

Starting with Oracle Database18c, 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 safely and 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.

Which solution should I use?

	TAC	**AC**	**TAF**	**Draining**
My application uses Oracle state (temp lobs, PL/SQL, temp tables.)	Yes	Yes excludes static mode	No	
My application has transactions	Yes	Yes	No for unplanned Transactional disconnect only	
My application uses connection pools	Yes	Yes	Yes	
My application has side effects	Yes Side effects are not replayed	Custom You decide to repeat or not	No	Yes
My application is not pooled	Yes (Java only) OCI (19.3)	Thin Java or SQL*Plus only	Yes	
My app needs Initial State Restored	Yes and custom	Yes and custom	Yes and custom	Yes
My application does not use session states	Yes	Yes	Yes	Yes
Future proofed	Yes	No	No	Yes

Instructions for Application Configuration

Follow these instructions when implementing your solution.

1. Use an Oracle Clusterware-managed service that is not the default database service (the default service has the same name as the database or PDB). The services that you create provide location transparency and HA features.

2. Use the recommended TNS (explained later in this paper) with built in timeouts, retries and delay so incoming connections do not see errors during outages.

3. Fast Application Notification (FAN) is a mandatory component to initiate draining, to break out of failures, and to rebalance sessions when services resume and when load imbalances occur. For hard outages, fast failover does not happen if the outage is not broken out by FAN. FAN applies to all failover solutions. When configuring FAN use Auto-configuration of ONS. Use the recommended TNS format strictly, do not alter this format. Exception: if your client is 11.2, you will manually configure FAN.

4. Before maintenance starts, drain your work from the instances or nodes targeted for maintenance. Enable FAN with Oracle Pools or Connection tests (or both). FAN with Oracle pools is the best solution as it provides full life cycle. That is, draining and rebalancing of work as maintenance progresses. When using FAN, return your connections to the pool. If you are using server draining (the alternate plan explained later in this paper) and your test is not a standard test, add your test to the server using `DBMS_APP_CONT`. Sessions that do not drain will use a failover solution.

5. The standard solution for failing over applications is Transparent Application Continuity (TAC).

   Use Application Continuity (AC): if you are using Oracle Database 12c Release 2, you want to customize with side effects or callbacks, or have an application that uses state such as temp tables and never cleans up.

   Use Transparent Application Failover (TAF): if your application is read only, or you are using OCI and are not pooled and would like a new connection and do not need the inflight work failed over. When choosing TAF, you must not change Oracle session state in the session after the initial setup.

Building Blocks for Continuous Availability of your Application

Use Services

Service is a logical abstraction for managing work. Services allow applications to benefit from the reliability of the redundant parts of the MAA system. The services hide the complexity of the underlying system from the client by providing a single system image for managing work.

The service is -

- a unit for management: a handful of services are manageable, many nodes, instances, listeners, and network interfaces are not manageable. The service provides location transparency.
- a unit for availability: resources are recovered fast, independently and in parallel for each service and without the need to start entire software stacks; and
- a unit for performance: work is routed transparently across the MAA system according to service quality and priority. Services are measured against service level thresholds and violations are reported to management with advised solutions in AWR.

FAN, connection identifier, TAC, AC, switchover, consumer groups, and many other features and operations are predicated on the use of services. Do not use the default database service as this cannot be disabled, relocated, or restricted and so has no high availability support. The services you use are associated with a specific primary or standby role in a Data Guard environment. Do not use the initialization parameter `service_names` for application usage.

An Example of Services to follow:

From ATP, each PDB database is supplied with 5 pre-configured services to choose from. All provide FAN and draining.

Service Name	Description	Draining	FAN	TAC
TPURGENT	OLTP Highest Priority	Yes	Yes	Yes
TP	OLTP General Priority Recommended to be used as main service	Yes	Yes	Yes
HIGH	Reporting or Batch Highest Priority	Yes	Yes	
MEDIUM	Reporting or Batch Medium Priority	Yes	Yes	
LOW	Reporting or Batch Lowest Priority	Yes	Yes	

The following help choosing the batch service:
HIGH: Queries run with a Degree of Parallelism equal to CPU_COUNT. There is a limit of 3 concurrent queries after which statement queuing comes in to affect.

MEDIUM: Queries run with a Degree of Parallelism of 4. The maximum number of queries that can be run is (CPU_COUNT*1.25).

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

If you wish to use TAF, use the older client side configuration for TAF in your connection string.

```
        (FAILOVER_MODE=(TYPE=select)(METHOD=basic)(RETRIES=3)(DELAY=5))
```

Modifying your Service on ADB-Serverless

ADB-Serveless provides draining on all services through the use of database draining and in-band FAN. ADB-Serverless will provide a PL/SQL procedure to modify your service to use Application Continuity.

Configure TNS

Configure the TNS or URL for High Availability

All Oracle-supplied connect strings will conform to the following strong recommendations. There is no need to do anything if you use the Oracle-supplied wallet. The following TNS/URL configuration is recommended for use for connecting at failover, switchover, fallback and basic startup.

Set `RETRY_COUNT`, `RETRY_DELAY`, `CONNECT_TIMEOUT` and `TRANSPORT_CONNECT_TIMEOUT` parameters in the TNSnames or the URL to allow connection requests to wait for the service and connect successfully.

Set `CONNECT_TIMEOUT` to a high value to prevent login storms. Low values can result in frenzied login-attempts due to the application or pool cancelling and retrying connection attempts.

Do not set `(RETRY_COUNT+1)*RETRY_DELAY` or `CONNECT_TIMEOUT` larger than your response time SLA. The application should either connect or receive an error within the response time SLA.

These are general recommendations for configuring the connections for high availability. Do not use Easy Connect Naming on the client as EZCONNECT has no high availability capabilities.

Note that the standby-scan specified below refers to the SCAN address available on the STANDBY site specified in your (Active) Data Guard configuration. Attempt will be made to connect to the PRIMARY site first, and if the service is not available, attempt to connect to this service at the standby. Once the location of the service is known, Oracle drivers 12.2 and later remember the address_list with that service offered and chooses this first, until the service next moves.

Adding the *standby-scan* to TNS connection descriptor to transparently fail over to the *standby-scan* is optional. Failing over to a standby database within the same region will have acceptable performance in most cases versus failing over to a standby database in a different region where additional network latency may result in unacceptable response time performance. In the latter case, a site failover operation will be required which involves DNS failover to another region containing mid-tier resources and standby database.

This is the recommended TNS for ALL Oracle drivers for 12.2 and higher:

```
Alias (or URL) = (DESCRIPTION =
(CONNECT_TIMEOUT= 120)(RETRY_COUNT=20)(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=standby-scan)(PORT=1521)))
 (CONNECT_DATA=(SERVICE_NAME = MY-SERVICE)))
```

For JDBC connections in 12.1 or earlier the following should be used. This TNS uses a lower `CONNECT_TIMEOUT` because `TRANSPORT_CONNECT_TIMEOUT` is not available for thin Java drivers until Oracle Database 12.2. RETRY_DELAY requires a patch for 11.2.0.4 clients.

```
(DESCRIPTION =
(CONNECT_TIMEOUT= 15)(RETRY_COUNT=20)(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=standby-scan)(PORT=1521)))
(CONNECT_DATA=(SERVICE_NAME = MY-SERVICE)))
```

Use Fast Application Notification (FAN)

FAN must be used. FAN is a required component for interrupting the application to failover. Without FAN applications will hang on TCP/IP timeout following hardware and network failures. FAN is used with all of the failover solutions.

The format of the address list described above is important for a number of reasons, one being that this format allows for auto-configuration of ONS. ONS is used to propagate FAN events to mid-tier pools and clients. When a database connection is made, database-tier ONS information is sent back to the mid-tier, allowing the mid-tier to establish ONS communication paths automatically. This means that configuration can be dynamic and need not be maintained at the mid-tier. ONS connections will be made from both the primary and standby sites.

FAN Coverage

FAN events are integrated with:

- Oracle Fusion Middleware and Oracle WebLogic Server (*Oracle® Fusion Middleware Administering JDBC Data Sources for Oracle WebLogic Server 12c (12.1.3)* Part Number E41864-02) or higher

- Oracle Data Guard Broker *(Oracle® Data Guard Broker 12c Release 1 (12.1)* Part Number E48241-06) or higher
- Oracle Enterprise Manager Cloud Control (*Cloud Control Basic Installation Guide 12c Release 4 (12.1.0.4)*) or higher
- Oracle JDBC Universal Connection Pool for both JDBC thin and OCI interfaces (*Oracle® Universal Connection Pool for JDBC Developer's Guide 12c Release 1 (12.1)* Part Number E49541-01) or higher
- ODP.NET (*Oracle® Database Development Guide 12c Release 1 (12.1)* Part Number E41452-06) or higher
- SQLPLUS (*SQL*Plus® User's Guide and Reference Release 12.1* Part Number E18404-12) or higher
- PHP (*Oracle® Database 2 Day + PHP Developer's Guide 12c Release 1 (12.1)* Part Number E18554-05) or higher
- Global Data Services (*Oracle® Database Global Data Services Concepts and Administration Guide 12c Release 1 (12.1)* Part Number E22100-10) or higher

To enable FAN in the client:

1. Use the TNS alias or URL provided. This connect string will auto-configure ONS (auto-ONS) subscription at the client for FAN event receipt when using a 12c driver or later. For older drivers, refer to the FAN white paper.

2. Then, depending on your client, enable FAN using the following:

Universal Connection Pool

Set the pool property `FastConnectionFailoverEnabled`

WebLogic Active GridLink for RAC

FAN and Fast Connection Failover are enabled by default

IBM WebSphere, Apache Tomcat, Red Hat JBoss, 3rd party Application Servers

Use Universal Connection Pool as a connection pool replacement

ODP.Net clients, Managed and Unmanaged Providers

Set "`HA events = true;pooling=true`" in the connect string

OCI clients

OCI clients using `oraacces.xml` (excluding `SQL*Plus`) set `events` to `true`

PHP

In `php.ini` add the entry `oci8.events=on`

3. In the ATP environment, ONS uses TLS (wallet-based) authentication.

For JDBC applications:
a) Ensure the following JAR files are present in your application's CLASSPATH : (ons.jar,osdt_cert.jar,osdt_core.jar,oraclepki.jar). The Oracle ATP service always uses TLS

Set declaratively using an UCP XML configuration file:

```
<?xml version="1.0"?>
<ucp-properties>
<connection-pool
connection-pool-name="UCP_pool1"
user="dbuser"
password="dbuserpasswd"
connection-factory-class-name="oracle.jdbc.pool.OracleDataSource"
initial-pool-size="10"
min-pool-size="5"
max-pool-size="15"
url="jdbc:oracle:thin:@(RECOMMENDED TNS)
fastConnectionFailoverEnabled="true"
onsConfiguration="nodes=primary-scan:6200,secondary-
scan:6200\nwalletfile=/net/host/path/onswallet\nwalletpassword=myWalletPasswd">
</connection-pool>
</ucp-properties>
```

b) Specify the wallet for FAN do one of the following:

- To use AUTO-ONS with wallets an application must set the following Java system properties:
  `“-Doracle.ons.walletfile=/replace_this_with_host_path/onswallet”` and
  `“-Doracle.ons.walletpassword=myONSwalletPassword”`

  This cannot be set on a per-pool or per-connection basis

- To use explicit ONS configuration (instead of AUTO-ONS) do one of the following:
  i) Programmatically within UCP, call
  `setONSConfiguration()`, for example

```
pds.setONSConfiguration(“nodes=primary-scanhost:6200,secondary-
scanhost:6200\nwalletfile=/replace_this_with_host_path/onswallet\nwalletpassword=
myWalletPassword”);
```

ii) Set declaratively using an UCP XML configuration file:

ii) For OCI applications (using 12.2 and more recent client drivers):

a) Change `oraaccess.xml` file to include the following in the `<default_parameters>` section:

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

b) Place `oraaccess.xml` in the same directory as the network files `tnsnames,ora`, `sqlnet.ora`.

<u>You are now ready to configure your application for planned maintenance.</u>

What is Planned Draining

For planned maintenance the recommended approach is to provide time for current work to complete. You can do this by draining requests initiated by FAN for Oracle connection pools and Oracle drivers, or third parties using these pools. The Oracle Autonomous Database also drains work directly.

Services connected to the Oracle Autonomous Database are pre-configured with draining tests and a drain_timeout specifying how long to allow for draining, and the stop option, IMMEDIATE, that applies after draining. The stop, relocate and switchover operations include a drain_timeout and stop_option to override the set values if needed.

When planned maintenance starts, a FAN planned-down event is posted to all applications subscribing to FAN. On receipt of this FAN event, the FAN-aware pool reacts by immediately clearing idle sessions and active sessions are marked to be released when the request completes or, at driver level, when the next connection check occurs. The FAN event causes sessions to drain from the instance without disrupting work for the period specified by drain_timeout. If not all sessions have checked in and the drain timeout has been reached, the stop mode applies and the services are stopped (using STOP IMMEDIATE). Starting with Oracle database 18c, the database itself marks sessions to be drained. Once marked, the database applies rules to find a safe place to drain sessions where the application is not disturbed. Draining continues through the drain_timeout period.

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 attempt to recover sessions that did not drain in the allocated time. There is never a need to restart application servers when planned maintenance follows best practice.

Planned Draining Checklist

By following this checklist planned maintenance is hidden from applications connected to Oracle's Autonomous Database. Before planned maintenance, drain or fail over database sessions at the database instance so application work is not interrupted.

The Recommended Approach

Using a FAN aware Oracle connection pool is the recommended solution for hiding planned maintenance. The 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.

When the FAN event is received by an Oracle pool, Oracle pools mark all connections at the instance to be drained. Immediately, checked-in connections are closed so that they are not re-used. In-use connections remain marked to be closed when they are next checked-in to the connection pool. As they are returned to the pool, these connections are released gradually.

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 the rebalancing or 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 IBM WebSphere, Apache Tomcat, Red Hat JBoss, Spring, and Hibernate, and others. Whitepapers from both Oracle and other vendors such as IBM describe how to use UCP with these application servers. 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. Many customer studies can be found at `oracle.com` under the Technology pages describing "Application Continuity" and "JDBC".

Alternative Approaches – Oracle Database 18c Draining or the Oracle Driver 18c In-Band FAN Draining

If you cannot use an Oracle pool with FAN, beginning with Oracle Database 18c, the database itself drains the sessions. The database starts to look for safe places to release connections. The database uses an extensible set of rules and heuristics to detect when to take the database session away. When draining starts, the database session persists at the database until a rule is satisfied. The rules include the following:

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

For the Connection Tests, it is standard practice for application servers, pooled applications, job schedulers, and so on to test connections when borrowed from connection pools, when returned to the pool and, at batch commits. During the drain period, the database intercepts the connection test, closes the connection and returns a failed status for the test. The application layer issuing the connection test is ready to handle a failed return status and issues a further request, to obtain a different connection. The application is not disturbed.

Connection tests

Ping and end request:

Depending on the outage, applications may receive stale connections in a race window when connections are borrowed before Fast Connection Failover (FCF) is processed. This can occur, for example, on a clean instance down when sockets are closed coincident with incoming connection requests. To prevent the application from receiving any errors, connection checks should be enabled at the connection pool.

JDBC Universal Connection Pool

`setValidateConnectionOnBorrow`() – Specifies whether or not connections are validated when the connection is borrowed from the connection pool. The method enables validation for every connection that is borrowed from the pool. The default value is false. Set the value to true so validation is performed.

OCI Connections

To verify that the connection to the server is terminated by either FAN or an OCI_ERROR, an application should code checking the value of the attribute OCI_ATTR_SERVER_STATUS in the server handle.

In response to the FAN event the OCI layer sets the OCI_ATTR_SERVER_STATUS attribute to OCI_SERVER_NOT_CONNECTED if the service is down. For scheduled maintenance this status is set without removing the connection to allow a grace period for the work to complete.

When using FAN to report planned maintenance, the application checks codes OCI_ATTR_SERVER_STATUS before borrow and after return to the pool, and drops the session at only these safe places only. Dropping closed connections before borrow and after return leads to a good user experience with no application errors received during planned maintenance. When using FAN to report unplanned down, the application receives an error immediately.

This is the only connection test that requires code. All other connection tests are configurable.

ODP.NET Provider

`CheckConStatus` is on by default.

This property checks the status of the connection before putting the connection back into the ODP.NET connection pool.

SQL Connection Tests:

Every application server has a feature to test the validity of the connections in their respective connection pools, which is set either by a configuration property or at the administrative console. The purpose of the test is to prevent vending an unusable connection to an application, and when an unusable connection is detected, to remove it when released to the pool.

Across the various application servers, the tests have similar names. The tests offered use various approaches, the most common being a SQL statement. Oracle recommends that Java application servers use the standard Java call `connection.isValid`(). Beginning with Oracle Database 18c, all tests are used to drain the database. Also beginning with Oracle Database 18c, the database drains sessions without using FAN by inspecting sessions for safe draining points. Inspection starts automatically when the service is stopped or relocated. Server draining supports all drivers.

There are 4 SQL connection tests added for every database service and pluggable database service by default, so if an application uses these SQL tests they are already covered:

```
SELECT 1 FROM DUAL;

SELECT COUNT(*) FROM DUAL;

SELECT 1;

BEGIN NULL;END;
```

In-Band FAN:

Starting with Oracle Database 18c together with Oracle Database Drivers 18c and later, the drivers receive FAN in-band events for planned down, enabling draining. The drivers look for end of request status to close the connection safely. This does rely on request boundaries, provided by Oracle Database 12c pools and later, or Java Pools that are using JDK9 or later. Your application must follow best practices and return connections to these pools when your request is completed.

Configuring More Connection Tests for Draining at the Server

You can add, delete, enable or disable, connection tests to a service, a pluggable database, or non-container database. Use the view DBA_CONNECTION_TESTS to see those added and enabled for you. Examples -

If you want to run any test that uses ping such as isValid, isUsable, OCIping, or connection.status, then use a SQL statement similar to the following:

```
SQL> execute dbms_app_cont.enable_connection_test(dbms_app_cont.ping_test);
```

To add a new server-side SQL connection test for a pluggable database or non-container database, log on to the non-container database and use a SQL statement similar to the following:

```
SQL> execute dbms_app_cont.add_sql_connection_test(select * from dual;');
```

Next choose your failover solution.

Transparent Application Continuity

Transparent Application Continuity is a mode of Application Continuity beginning with Oracle Database 18c that transparently tracks and records session and transactional state so that a database session can be recovered following recoverable outages. This is done safely and with no need for a DBA to have any knowledge of the application or to make application code changes. Transparency is achieved by using a state-tracking infrastructure that categorizes session state usage as an application issues calls to the database. The use of state tracking future proofs applications using Transparent Application Continuity as applications or environments change.

TAC is enabled when you select the appropriate service for the Autonomous Database.

Transparent Application Continuity Coverage

Transparent Application Continuity for Oracle Autonomous Database supports the following clients:

- Oracle JDBC Replay Driver 18c or later. This is a JDBC driver feature provided with Oracle Database 18c for Application Continuity.
- Oracle Universal Connection Pool (UCP) 18c or later.
- Oracle WebLogic Server 18c, and third-party JDBC application servers using UCP
- Java connection pools or standalone Java applications using Oracle JDBC - Replay Driver 12c or later with Request Boundaries
- Oracle Tuxedo with JDBC-thin or OCI 19c drivers
- OCI Session Pool 18c or later
- SQL*Plus 18c or later (use "-ac" command line switch)
- ODP.NET Unmanaged Provider 18c or later

If using a third party, Java-based application server, the most effective method to achieve high availability is to replace the data source with UCP. This approach is supported by many application servers including IBM WebSphere and Apache Tomcat, Red Hat JBoss, Spring, and Hibernate, and others. Whitepapers from both Oracle and other vendors such as IBM describe how to use UCP with these application servers. 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.

Steps for using Transparent Application Continuity

Use a supported client (see coverage above)

Return Connections to the Connection Pool

You do not need to make any changes to your application for identifying request boundaries, if the application uses connections from the Oracle connection pools, or from the `oracle.jdbc.replay.OracleConnectionPoolDataSourceImpl` data source. For the connection pool to function as described, the application must get connections when needed, and release connections when not in use. This scales better and provides request boundaries transparently. It is best practice that an application checks-out a connection only for the time it needs it. Holding a connection when not in use is not good practice.

Request boundaries are discovered and advanced transparently when using TAC with JDBC-thin driver release 18c and for OCI-based applications starting with 19c. This means lower resource usage and faster recovery because request boundaries advance automatically, and statements that do not contribute to transactions and session state are purged when no longer needed. It is still best practice to use an Oracle pool and to return your connections to the Oracle pool between requests.

Use FAILOVER_RESTORE

TAC automatically restores preset states by setting `FAILOVER_RESTORE` to `AUTO` on your service.

If you find that you need preset session states in addition to the standard set, then you can register a callback, or UCP label, to restore these states. If you find you need complex session states, such as temp tables or sys_context, restored then use a callback with Application Continuity that offers this flexibility.

Enable Mutable Use 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 Autonomous Database automatically `KEEPs` mutables for SQL.

If you need mutables for PL/SQL then configure mutable objects 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 | KEEP SYSGUID] … to USER
```

```
SQL> GRANT KEEP SEQUENCE mySequence to myUser on sequence.object
```

Side Effects are disabled

During normal runtime, Transparent Application Continuity detects side effects and does not replay them. A side effect is an external action such as sending mail, transferring files, using TCP. The type of side effect is distinguished between those that relate to an application's logic and those that are internal, relating to database housekeeping. For applications that use statements that have side effects, capture is disabled when the statement is running. Once a new request starts, capture is re-enabled automatically. Capture is also automatically reenabled for Java as soon as the request is recoverable as TAC with Java has implicit request boundaries. If you would like side-effects replayed, use Application Continuity that offers this flexibility.

Application Continuity

Application Continuity is customizable, allowing you to choose to replay side-effects, for example mail, or to add complex callbacks at failover that TAC does not allow. Use AC, if are on 12.2, you want to customize with side effects or callbacks, or have an application that uses state such as session duration temp tables and does not clean up.

Application Continuity Coverage

Application Continuity for Oracle Database 18c supports the following clients:

- Oracle JDBC Replay Driver 12c or later. This is a JDBC driver feature provided with Oracle Database 12c for Application Continuity
- Oracle Universal Connection Pool (UCP) 12c or later
- Oracle WebLogic Server 12c, and third-party JDBC application servers using UCP
- Java connection pools or standalone Java applications using Oracle JDBC - Replay Driver 12c or later with Request Boundaries
- Oracle Tuxedo with JDBC-thin or OCI
- OCI Session Pool 12c Release 2 or later

- SQL*Plus 12c release 2 or later (use "-ac" command line switch)
- ODP.NET Unmanaged Provider 12c Release 2 or later

If using a third party, Java-based application server, the most effective method to achieve high availability is to replace the data source with UCP. This approach is supported by many application servers including IBM WebSphere and Apache Tomcat, Red Hat 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.

Steps for using Application Continuity

Use a supported client (see coverage above)

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 it needs it. Holding a connection when not in use is not good practice. 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 subsequent use by other threads, or your thread when needed again. Following this practice also embeds request boundaries that Application Continuity uses to identify safe places to resume and end capture. AC does not provide additional discovered request boundaries as TAC does.

Use FAILOVER_RESTORE

Some applications and mid-tiers configure connection pools such that all connections are, for example, in a preset language or time zone. 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 states are restored automatically by setting `FAILOVER_RESTORE` to `LEVEL1`. If you preset session states in addition to the standard set, then you will need to register a callback, or UCP label, to restore these states. Refer to Application Continuity whitepaper listed in Additional Materials below.

If you need additional session states, use one of these options

- FAILOVER_RESTORE=LEVEL1 set on the service and
- Connection Initialization Callback for Java or the (older) TAF Callback for OCI or
- Universal Connection Pool or WebLogic Server Connection Labeling

Enable Mutable Use in the Application

See TAC section on Mutables.

Decide if you wish to repeat side effects

Side effects are replayed unless the application specifies otherwise. Applications that use external actions should be reviewed to decide if requests with side effects make sense to replay or not. For example, does the application want to send email again or to transfer a file again? Frequently it is desirable to replay the side effect. However, sometimes it may be better not to. If a request has an external action that should not be replayed, that request can use a connection that does not have Application Continuity enabled, or replay can be disabled for that request using the `disableReplay` API for Java or `OCIRequestDisableReplay` for OCI. All other requests continue to be replayed. You could also use Transparent Application Continuity that does not replay side-effects.

Transparent Application Failover

Transparent Application Failover (TAF) is a client-side feature of OCI, OCCI, Java Database Connectivity (JDBC) OCI driver, and ODP.NET that failover connections or `SELECT` statements for applications that do not alter Oracle state after the initial setup of a connection. Set your `FAILOVER_TYPE` to `BASIC` for connections to failover and `SELECT` for reconnecting and replaying `SELECT` statements. TAF pre-connect is not a recommended option because it will cause delays at failover due to your application not being pre-connected in real situations.

Transparent Application Failover for Oracle Database supports the following clients:

- Oracle Call Interface (OCI)
- Oracle C++ Call Interface (OCCI)
- Oracle JDBC OCI driver (thick-driver is not recommended in general)
- ODP.Net Unmanaged Provider
- Oracle Tuxedo with OCI
- OCI Session Pool

Steps for using Transparent Application Failover

Use a supported client (see coverage above)

Use FAILOVER_RESTORE

From Oracle Database 12.2 onwards check if the application is presetting values on connections. Some applications and mid-tiers configure connection pools such that all connections are, for example, in a preset language or time zone. 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 states are restored automatically by setting `FAILOVER_RESTORE` to `LEVEL1`. If you preset session states in addition to the standard set, then you will need to register a TAF callback, to restore these states

Use the following options together:

`FAILOVER_RESTORE=LEVEL1` set on the service

TAF Callback for OCI

Starting with RDBMS 12.2, `FAILOVER_RESTORE=LEVEL1` is the recommended method if you do not already have a callback.

TAF with Transaction Guard

At failover, Transparent Application Failover (TAF) starting with Oracle Database 12.2 obtains a new connection and when Transaction Guard is enabled invokes Transaction Guard to force the commit outcome. If Transaction Guard returns committed and completed, TAF continues and the application sees no errors. If Transaction Guard returns uncommitted or committed but not completed, TAF returns a TAF error to the application. TAF maintains the new connection.

When TAF and Transaction Guard are both used, developers can use the TAF error codes (ORA-25402, ORA-25408, ORA-25405) to decide to resubmit transactions or to return a message indicating uncommitted to the user. It is ONLY safe to resubmit or to return uncommitted on these errors codes when BOTH TAF and Transaction Guard are enabled. It is not safe to resubmit or to return uncommitted if only TAF is enabled. Application Continuity does the resubmission for you.

Refer to the Transaction Guard whitepaper referred below.

Knowing your Protection Level when using TAC or AC

Protection-Level Statistics

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, when service statistics are enabled, in V$SERVICE_STATS. These statistics are saved in the Automatic Workload Repository and are available in Automatic Workload Repository reports.

The output is similar to the following:

TIP: To enable protection-level statistics for Oracle Database 18c, use `_request_boundaries = 3`. This is the default in later releases.

```
Statistic                                    Total             /Second        /Trans
--------------------------------          ------------------ -------------- ---
cumulative begin requests                     50,000           241.0           3.3
cumulative user calls in request             458,095         2,208.4          29.9
cumulative user calls protected              458,095         2,208.4          29.9
```

JDBC Steps for using TAC or AC

Ensure all recommended patches are applied at the client. Refer to the AC page on OTN (http://www.oracle.com/goto/ac).

Enable JDBC Statement Cache

For JDBC based applications, always use the JDBC statement cache for performance. When using TAC or AC, it is important to disable a statement cache at the application server level (for example Tomcat, WebLogic, WebSphere etc.) if used, as described in the JDBC reference documentation:

If a statement cache at the application server level is enabled (for example, the WebLogic or third-party application server statement cache), this must be disabled when the replay is used. Instead, configure the JDBC statement cache, which performs better because it is optimized for JDBC and Oracle and because it supports Application Continuity.

Use `oracle.jdbc.implicitstatementcachesize=[open cursors]`

Setting `nnn` to a positive value enables the Implicit Statement Cache.

Tune Garbage Collector

For many apps the default Garbage Collector tuning is correct. For an application with extreme performance requirements it may be necessary to perform JVM tuning. Our recommendation for extreme performance is to add the following to the Java command line (note that both attributes should be set to the same value). For applications that return and keep a lot of data you may need to use higher values such as 2G or more as in the example:

```
java -Xms3072m -Xmx3072m
```

COMMIT

Application Continuity supports `COMMIT` of all styles: top-level (`OCOMMIT` or `COMMIT()`) , `AUTOCOMMIT` and `COMMIT` embedded in PLSQL. If your application is using a top-level commit that is standalone, then there is full support for replay including when using `SESSION_STATE_CONSISTENCY=STATIC` mode (12.2.0.1) and with Transparent Application Continuity (TAC). If your application is using `COMMIT` embedded in PLSQL or `AUTOCOMMIT`, it may not be possible to replay for cases where Application Continuity detects that the call including the `COMMIT` did not run to completion. Application Continuity will make the correct decision. It is recommended practice to use a top-level commit.

For JDBC applications, if the application does not need to use `AUTOCOMMIT`, you disable `AUTOCOMMIT` either in the application itself or in the connection properties. This is particularly important when `UCP` is embedded in 3rd party application servers such as Apache Tomcat, IBM WebSphere and Red Hat JBoss.

Set autoCommit to false through UCP PoolDataSource' connection properties --

```
connectionProperties="{autoCommit=false}"
```

JDBC CONCRETE CLASSES

For applications using Oracle JDBC Driver, there is no support for deprecated oracle.sql concrete classes BLOB, CLOB, BFILE, OPAQUE, ARRAY, STRUCT or ORADATA. (See MOS note 1364193.1). Use ORAchk -acchk "Clean Up Concrete Classes for Application Continuity" to know if an application passes. The list of restricted concrete classes for JDBC Replay Driver is reduced to the following starting with 18c and later:

```
oracle.sql.OPAQUE

oracle.sql.STRUCT

oracle.sql.ANYDATA
```

OCI (Oracle Call Interface) Steps for Using FAN

Applications must connect to an Oracle RAC instance to enable HA event notification. Furthermore, these applications must:

- Initialize the OCI Environment in `OCI_EVENTS` mode (before Oracle Database 12c)
- Connect to a service that has notifications enabled (use the `srvctl` to set `AQ_HA_NOTIFICATIONS` to `TRUE`)
- Link with a thread library

Client-side deployment characteristics can be set in the file `oraaccess.xml including OCI_EVENTS`.

Appendix: Operating Your Cloud on Premise

Configuring your Service for Cloud on Premise

When using Oracle Database on premise, you can create services on Oracle RAC that use Transparent Application Continuity, or Application Continuity, or TAF. You can use roles to distinguish whether the services are active on Active Data Guard or the primary database. The following examples illustrate this:

Transparent Application Continuity

```
$ srvctl add service -db mydb -service GOLD -preferred serv1 -available serv2 -
failover_restore AUTO -failoverretry 30 -failoverdelay 10 -commit_outcome TRUE -
failovertype AUTO -replay_init_time 1800 -retention 86400 -notification TRUE -
drain_timeout 300 -stopoption IMMEDIATE
```

Application Continuity

```
$ srvctl add service -db mydb -service SILVER -preferred serv1 -available serv2
-failover_restore LEVEL1 -failoverretry 30 -failoverdelay 10 -commit_outcome
TRUE -failovertype TRANSACTION -replay_init_time 1800 -retention 86400 -
notification TRUE -drain_timeout 300 -stopoption IMMEDIATE
```

Transparent Application Failover

```
$ srvctl add service -db mydb -service BRONZE -preferred serv1 -available serv2
-failover_restore LEVEL1 -failoverretry 30 -failoverdelay 10 -commit_outcome
TRUE -failovertype SELECT -retention 86400 -notification TRUE -drain_timeout 300
-stopoption IMMEDIATE
```

To add with the Data Guard role, here is the TAC example:

```
$ srvctl add service -db mydb -service GOLD -preferred serv1 -available serv2 -
failover_restore AUTO -failoverretry 30 -failoverdelay 10 -commit_outcome TRUE -
failovertype AUTO -replay_init_time 1800 -retention 86400 -notification TRUE   -
role PHYSICAL_STANDBY -drain_timeout 300 -stopoption IMMEDIATE
```

Planned Draining – server-side operation

Many enterprises run a large number of services, whether it be many services offered by a single database or instance, or many databases offering a few services running on the same node. Starting with Oracle Database 12c Release 2, you no longer need to run SRVCTL commands for each individual service but need only specify the node name, database name, pluggable database name, or the instance name for all affected services.

Both `DRAIN_TIMEOUT` and `STOPOPTION` are service attributes that you can define when you add the service or modify it after creation. You can also specify these attributes using `SRVCTL`, which will take precedence over what is defined on the service. The largest `DRAIN_TIMEOUT` for a group of services is applied across the group operation. Refer to the following examples:

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>

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

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>
```

Start/Stop everything at a node

```
srvctl stop service -node <node_name> -drain_timeout <timeout> -stopoption <stop_option>

srvctl stop service -db <db_unique_name>  [-node <node_name> | -instance <inst_name>] -
drain_timeout <timeout> -stopoption <stop_option>

srvctl stop service -db <db_unique_name>  [-node <node_name> | -instance <inst_name>] -
pdb <pluggable_database> -drain_timeout <timeout> -stopoption <stop_option>
```

Data Guard Switchover

```
switchover to <db_resource_name> [wait [xx]];
```

Application Continuity Protection Report

Protection-Level Report

Using Application Continuity's Protection Report provides in advance of any outages, the percentage of requests that are fully protected by AC or TAC, and for those are not fully protected, which they are and where. Executing the coverage check is rather like using SQL_TRACE. First run the application in a representative test environment with Application Continuity trace turned on at the server side. The trace is collected in the standard RDBMS user trace directory in user trace files. Set RDBMS event trace[progint_appcont_rdbms]. Then, pass the RDBMS trace directory as input to Oracle ORAchk to report the coverage for the application functions that were run. Refer to the Application Continuity whitepaper for more details.

Performance Tips for Database Administrators

Transaction History Table

The transaction history table (LTXID_TRANS) is created by default in the `SYSAUX` tablespace at database creation and upgrade. New partitions are added when instances are added, using the storage of the last partition.

If the location of this tablespace is not optimal for performance, the DBA can move partitions to another tablespace.

To move the history table, use an `alter table move partition` command for each partition.

For example

```
SQL> ALTER TABLE LTXID_TRANS move partition LTXID_TRANS_4
       tablespace FastPace
       storage ( initial
       100M  next 100M minextents 20 maxextents 121 );
```

Table Locks

Disable table locks on the transaction history table by issuing the ALTER TABLE command. If you need to add a new instance, enable table locks and add the new partition in your designated tablespace. Then disable table locks again. During runtime table locks incur significant database CPU usage.

Note: When adding a new RAC instance enable table locks, create a partition for the `LTXID_TRANS` table in the new tablespace and then disable table locks again.

```
SQL> ALTER TABLE LTXID_TRANS DISABLE TABLE LOCK;
```

Additional whitepapers available on OTN

Fast Application Notification

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

Embedding UCP with other web servers:

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) and

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)

Application Continuity

*(*http://www.oracle.com/technetwork/database/options/clustering/applicationcontinuity/overview/application-continuity-wp-12c-1966213.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)

Transparent Application Failover

https://docs.oracle.com/en/database/oracle/oracle-database/18/adfns/high-availability.html#GUID-96599425-9BDA-483C-9BA2-4A4D13013A37

Transaction Guard

Transaction Guard (http://www.oracle.com/technetwork/database/database-cloud/private/transaction-guard-wp-12c-1966209.pdf)

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

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle

facebook.com/oracle

twitter.com/oracle

oracle.com

Integrated Cloud Applications & Platform Services

White Paper: Application Checklist for Continuous Service on the Autonomous Database
March 2019
Author: Carol Colrain, Troy Anthony and Ian Cookson

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