19c ORACLE Database

Continuous Availability

Application Continuity for the Oracle Database

ORACLE white paper / August 27, 2020

ORACLE

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 an Oracle Database deployed using MAA best practices ensures that your applications are continuously available.

The following checklist is useful for preparing your applications and databases, even if you are not yet using Application Continuity. The points discussed here provide significant value for preparing your systems to support continuous availability, reducing possible downtime during planned maintenance activities and during unplanned outages should they occur.

FEATURE SET FOR KEEPING YOUR APPLICATIONS 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 events, unplanned outages and load imbalances. You can think of these features as an insurance policy protecting your applications from service interruptions. 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-proofing.

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. Idle sessions are released gradually. Active sessions are drained when the work executing in that session completes. Draining of sessions is in wide use with Oracle connection pools and mid-tiers configured for Fast Application Notification (FAN). Starting with Oracle Database 18c, the database itself drains sessions when PDBs and instances are stopped or relocated. Draining is always the best solution for hiding planned maintenance. Failover solutions such as Application Continuity are the fallback when work will not drain in the time allocated.

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 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
I don't know how the application is implemented	Yes	No	No	Yes
My application does transactions	Yes	Yes	No for unplanned Transactional disconnect only	Yes
My application uses Oracle state (temp lobs, PL/SQL, temp tables.)	Yes	Yes No for static mode	No	Yes
My application does not use connection pools	Yes	No	Yes	Yes
My application has side effects (such as file transfers)	Yes Side effects are not replayed	Customizable	No	Yes
My app needs Initial State Restored	Yes and custom	Yes and custom	Yes and custom	Yes
Future proofed for application changes	Yes	No	No	Yes

Transparent Application Continuity with Draining is the recommended solution for Continuous Availability. If you are using a 12c driver, or customization is required for initial states or side effects, then you should use Application Continuity. TAF continues to be available and is fully supported.

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 Connection String (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 outages such as node and network failures, fast failover of the application does not happen if the client is not interrupted 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 pre-12c, you will manually configure FAN.)

4. Before maintenance starts, drain your work from the instances or nodes targeted for maintenance. Enable FAN with Oracle Connection Pools or Connection tests (or both). Oracle connection pools with FAN are the best solution as pools provides full life cycle of session movement. 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_ADMIN`. Sessions that do not drain within the DRAIN_TIMEOUT will be failed over.

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

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

   Use Transparent Application Failover (TAF) if your application is read only and does 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 for sites and databases.
- a unit for availability: resources are recovered quickly, 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:

From ATP-D, 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	Yes	Yes	

	Medium Priority			
LOW	Reporting or Batch Lowest Priority	Yes	Yes	

Configure the Connection String or URL for High Availability

All Oracle-supplied connect strings will conform to the following 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.ora` file or the URL to allow connection requests to wait for the service and connect successfully. Connection attempts and retries are managed by Oracle Database Net Services.

Set `CONNECT_TIMEOUT` to a high value to prevent login storms such as 90s or 120s. 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.

Use this Connection String for ALL Oracle driver version 12.2 or higher:

```
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=standby-scan)(PORT=1521)))
 (CONNECT_DATA=(SERVICE_NAME = YOUR SERVICE)))
```

For JDBC connections in 12.1 or earlier the following should be used. This Connection String 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.

```
Alias (or URL) =
(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=standby-scan)(PORT=1521)))
(CONNECT_DATA=(SERVICE_NAME = YOUR SERVICE)))
```

FAST APPLICATION NOTIFICATION (FAN)

FAN must be used. FAN is a required component for interrupting the application to failover. Without FAN, applications can hang on TCP/IP timeout following hardware and network failures, and omit to rebalance when resources resume. All Oracle pools and all Oracle application servers use FAN. Third-party JAVA application servers can use UCP to enable FAN. No application changes are required to use FAN. These are configuration changes only.

For continuous service during planned maintenance, use FAN with:

- Oracle pools or
- UCP with third-party JDBC application servers or
- The latest Oracle client drivers

For continuous service during unplanned outages, use FAN with

- Application Continuity or
- Transparent Application Continuity

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 Data Guard Broker
- Oracle JDBC Universal Connection Pool or Driver for both JDBC thin and OCI interfaces
- ODP.NET Connection Pool for Unmanaged and Managed Providers
- Oracle Tuxedo
- SQL*Plus
- PHP
- Global Data Services
- Third party JDBC application servers using Oracle JDBC Universal Connection Pool
- Listeners

To enable FAN in the client:

Use the TNS alias or URL shown in the preceding discussion. This connection string is used to auto-configure the Oracle Notification Service (ONS) subscription at the client for FAN-event receipt when using an Oracle Database 12c or later client driver. For older drivers, refer to the FAN white paper in the Appendix for configuration details. ONS provides a secure communication path between the database tier and the client-tier allowing the client to be notified of service availability (components stopping or starting) as well as runtime load balancing advice for better work placement during normal operation.

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. Depending on the client, enable FAN in the application configuration properties as follows

Universal Connection Pool or JDBC thin driver (starting 12.2)

Set the property `FastConnectionFailoverEnabled`

WebLogic Active GridLink for Oracle RAC

FAN and Fast Connection Failover are enabled by default

Oracle WebLogic Server, IBM WebSphere, IBM Liberty, Apache Tomcat, Red Hat WildFly (JBoss), JDBC Applications

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 if using ODP.Net 12.1 or earlier

Oracle Call Interface (OCI) clients and OCI-based drivers

Oracle Call Interface (OCI) clients without native settings can use `oraacces.xml` 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` add the entry `oci8.events=on`

SQL*Plus enables FAN by default

3. You can configure ONS to use 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-D 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\nwalletpass
word=myWalletPassword");
```

ii) Set declaratively using an UCP XML configuration file:

ii) For Oracle Call Interface (OCI) applications Oracle client drivers 12.2 or later are required:

a) Add the following to the <default_parameters> section of an oraaccess.xml file:

```
<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>
```

The <wallet_location> path should be the name of the directory containing the wallet.

Other parameters may be set in the ons section of oraaccess.xml, including <hosts>, <max_connections>, and <subscription_wait_timeout>.

Drivers that support native event setting controls may omit the <events> section and use the driver setting instead.

By default connections will be established to the database even if ONS fails. If you prefer connections to fail in this scenario, you can add a section to the same level as <events> and <ons>:

```
<fan>
   <subscription_failure_action>
       error
   </subscription_failure_action>
</fan>
```

Place the oraaccess.xml file in the same directory as the tnsnames.ora and sqlnet.ora network files. For example, when using Oracle Instant Client these files might be in the default directory network/admin. Alternatively, all network configuration files can be put in another accessible directory. Then set the environment variable TNS_ADMIN to that directory name.

You are now ready to configure your application for planned maintenance.

MAINTENANCE WITHOUT IMPACTING YOUR APPLICATIONS

Drain Sessions Before Maintenance

For planned maintenance the recommended approach is to provide time for current work to complete before maintenance is started. 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 Database also drains work directly.

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 operations include a drain_timeout and stopoption 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 clearing idle sessions and marking active sessions to be released when the request completes or 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 stopoption applies and the services are stopped (using 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.

Configure for Planned Draining

By using one or both approaches, planned maintenance is hidden from applications connected to the Oracle Database. Before planned maintenance, drain or failover 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, the Oracle pool marks all connections connected to 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 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. 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 19c Draining or the Oracle Driver 19c Draining

If you cannot use an Oracle pool with FAN, the Oracle 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.

Oracle Driver 19c Draining with 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 and connection tests to close the connection safely. The driver uses connection tests that do not use SQL, PING-based such as `connection.isValid`(), `CheckConStatus`, or `OCI_ATTR_SERVER_STATUS`. Using request boundaries relies on 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.

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.

JDBC Driver connection Tests

> When using a connection pool `connection.isValid()` or SQL connection tests can be set in the property file. `connection.isValid()` is a local call starting with 12.2, and is recommended to be enabled by default for use when borrowing a connection from all Java-based connection pools, and also with job schedulers, or similar, before submitting a request on a connection to the database.

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 OCI_ATTR_SERVER_STATUS before borrow and after return to the pool, and drops the session 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 is used to enable checking of 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 four 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;
```

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. Examples -

By default, the driver uses ping tests. There is no need to enable for the driver. If you want the database to use to 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_admin.enable_connection_test(dbms_app_cont_admin.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_admin.add_sql_connection_test('select * from dual;',[SERVICE]);
```

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.

Transparent Application Continuity Coverage

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

It is strongly recommended to use the latest client drivers. Oracle Database 19c client drivers and later provide full support for TAC.

- 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
- OCI Session Pool 19c or later
- SQL*Plus 19c or later
- ODP.NET Unmanaged Provider 18c or later
- Oracle Call Interface based applications using the 19c OCI driver 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 Oracle WebLogic Server, IBM WebSphere, IBM Liberty, Apache Tomcat, Red Hat WildFly (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 Driver 18c or later or
- from the OCI-based applications using 19c or later

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 uses lower memory, compared to holding connections, 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 temporary 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 Database 19c 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. Capture is re-enabled automatically at the next discovered or explicit request boundary. Capture is automatically re-enabled, when possible, for Java 18c replay driver and OCI 19c driver by TAC as it supports implicit request boundaries. If you want the side effects replayed, use Application Continuity that offers this flexibility.

RESTRICTIONS

See the documentation for the most current list of restrictions.

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
- Applications and language drivers using Oracle Call Interface Session Pool 12c Release 2 or later
- SQL*Plus 19c or later
- ODP.NET Unmanaged Provider 12c Release 2 or later (“Pooling=true”;“Application Continuity=true” default in 12.2 and 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 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 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 explicit 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 WANT TO REPEAT SIDE EFFECTS DURING REPLAY

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. If you do not wish to replay all side effects, use Transparent Application Continuity.

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

Starting with Oracle Database 12.2, at failover, Transparent Application Failover (TAF) obtains a new connection and, if 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 in later 19c versions, `V$SERVICE_STATS`. These statistics are saved in the Automatic Workload Repository and are available in Automatic Workload Repository reports.

```
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
```

Tip: To report the protection level by PDB or by using historic data, see the Appendix for example SQL to use.

The output is similar to the following:

TAC or AC are enabled and protecting your application when

- Cumulative user calls in request = cumulative user calls protected
- And the above numbers are not equal to zero

This protection level is measured inside the database. The client may need to use an `ORDER BY` clause in queries and preset the initial session state if this contains state not covered by `FAILOVER_RESTORE` to achieve this level of protection.

The example above shows an increasing number of Begin and End requests. The number itself will depend on how frequently your application checks out and checks in to the connection pool or what request boundaries the database can discover when using TAC. The rate of increase of these values will depend on the rate your requests are being submitted. You can compute the percentage of user calls being protected using:

Percentage of Protected Calls = cumulative user calls protected / cumulative user calls in request * 100

It is possible that the percentage of protected calls is less than 100%? You may be using JDBC concrete classes, side effects are disabled, unrecoverable state may be being used, or the application may choose to disable application continuity for certain requests. If your application is not 100% protected, the `ORAchk` component `acchk` can be used, at your own site, to show where in your application coverage is below 100%. Your management can decide whether to follow the advisor or take no action by evaluating the impact.

CONFIGURING YOUR CLIENTS

JDBC-thin Driver Checklist

1. Ensure 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:

   `setConnectionFactoryClassName("oracle.jdbc.replay.OracleDataSourceImpl");` or

   `setConnectionFactoryClassName("oracle.jdbc.replay.OracleXADataSourceImpl");` (for XA)

3. Use JDBC Statement Cache for Coverage and Performance

For best coverage and performance, use the JDBC driver statement cache in place of an application server statement cache. This allows the driver to know that statements are closed and 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 10 and 100 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

For JDBC applications, Oracle 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 Fast Connection Failover (FCF)

For client drivers 12c and later

- Use the recommended URL for auto-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, UCP is recommended
- Open port 6200 for ONS (6200 is the default port, a different port may have been chosen)

For client drivers prior to 12c use the addresses provided:

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

OCI (Oracle Call Interface) Driver Checklist

1. Ensure all recommended patches are applied at the client. Refer to the MOS Note *Client Validation Matrix for Application Continuity (Doc ID 251148.1)*
2. Replace `OCIStmtPrepare` with `OCIStmtPrepare2`. `OCIStmtPrepare()` has been deprecated since 12.2. All applications should use `OCIStmtPrepare2()`. TAC and AC allow OCIStmtPrepare but do not replay this statement.

   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:

- ATP-D presets aq_ha_notifications on the services
- Use the recommended Connection String for auto-ons
- Set `auto_config`, `events`, and `wallet_location` (optional) in `oraaccess.xml` (See Appendix)
- 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)

For client drivers prior to 12c use the addresses provided in `oraccess.xml`.

ODP.NET Unmanaged Provider Driver Checklist

1. Ensure all recommended patches are applied at the client. Refer to the MOS Note Client Validation Matrix for Application Continuity (Doc ID 251148.1)
2. To use FAN for OCI-based applications, do the following:

- ATP-D presets `aq_ha_notifications` on the services
- Use Recommended Connection String for auto-ons
- Set `onsConfig` and `wallet_location` (optional) in `oraaccess.xml` (See Appendix)
- 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;"
```

CONCLUSION

The Oracle Database 19c is configured and managed for high availability on your behalf. No additional configuration or management is required by you.

There are a few simple steps to achieving Continuous Availability for your applications:

- Select the database service that is appropriate for your SLA's
- Configure Fast Application Notification (FAN)
- Use the recommended connection string for your applications
- Use application best practices to optimize for draining
- Use Transparent Application Continuity or Application Continuity for continuous service

By following these five simple steps, planned maintenance activities will no longer require outages and unplanned events will no longer result in failed transactions and interruptions to service for users for most cases.

APPENDIX: CONFIGURING YOUR SERVICES

When using Oracle Database, 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 add as service named GOLD to illustrate this:

Transparent Application Continuity

```
$ srvctl add service -db mydb -service GOLD -preferred inst1 -available
serv2 -failover_restore AUTO -failoverretry 1 -failoverdelay 3 -
commit_outcome TRUE -failovertype AUTO -replay_init_time 600 -retention
86400 -notification TRUE -drain_timeout 300 -stopoption IMMEDIATE
```

Or if an service with more than one active instance is needed (and in this case no *available* instance)

```
$ srvctl add service -db mydb -service GOLD -preferred inst1,inst2 -
failover_restore AUTO -failoverretry 1 -failoverdelay 3 -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 SILVER -preferred inst1 -available
inst2 -failover_restore LEVEL1 -failoverretry 1 -failoverdelay 3 -
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 inst1 -available
inst2 -failover_restore LEVEL1 -failoverretry 1 -failoverdelay 3 -
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 inst1 -available
inst2 -failover_restore AUTO -failoverretry 1 -failoverdelay 3 -
commit_outcome TRUE -failovertype AUTO -replay_init_time 600 -retention
86400 -notification TRUE   -role PHYSICAL_STANDBY -drain_timeout 300 -
stopoption IMMEDIATE
```

If the recommended Database connect string or URL (described earlier in this paper) is used, retry attempts are managed by Oracle Database Net Services

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>
```

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>
```

Start a service named GOLD on an instance named *inst1* (a given instance)

```
srvctl start service –db myDB –service GOLD –instance inst1
```

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]];
```

APPENDIX: USING ACCHK TO CHECK FOR CONCRETE CLASSES

This check applies to Java applications only. It is used to determine whether Java applications use deprecated Oracle JDBC concrete classes.

To use Application Continuity with Java, replace the deprecated Oracle JDBC concrete classes. For information about the deprecation of concrete classes including actions to take if an application uses them, see My Oracle Support Note 1364193.1.

To know if the application is using concrete classes, use Application Continuity checking (called `acchk` in Oracle ORAchk. Verify the application in advance while planning for high availability for your application.

For JDBC driver version 12.2.0.2 and below, Application Continuity is unable to replay transactions that use `oracle.sql` deprecated concrete classes of the form `ARRAY`, `BFILE`, `BLOB`, `CLOB`, `NCLOB`, `OPAQUE`, `REF`, or `STRUCT` as a variable type, a cast, the return type of a method, or calling a constructor.

For JDBC driver version 18c and above, Application Continuity is unable to replay transactions that use `oracle.sql` deprecated concrete classes of the form `OPAQUE`, `REF`, or `STRUCT` as a variable type, a cast, the return type of a method, or calling a constructor.

Deprecated Java should be removed for Application Continuity to protect the application.

There are four values that control the Application Continuity checking for Oracle concrete classes. Set these values either on the command line, or through shell environment variables, or mixed. The values are as follows:

TABLE 2-1 APPLICATION CONTINUITY CHECKING FOR CONCRETE CLASSES

Command-Line Argument	Shell Environment Variable	Usage
`-asmhome jarfilename`	`RAT_AC_ASMJAR`	This must point to a version of `asm-all-XXX.jar` that you download from ASM Home Page.
`-javahome JDK8dirname`	`RAT_JAVA_HOME`	This must point to the `JAVA_HOME` directory.
`-appjar dirname`	`RAT_AC_JARDIR`	To analyze the application code for references to Oracle concrete classes, this must point to the parent directory name for the code. The program analyzes `.class` files, and recursively `.jar` files and directories.
`-jdbcver`	`RAT_AC_JDBCVER`	Target version for the coverage check

EXAMPLE APPLICATION CONTINUITY CONCRETE CLASS CHECKS SUMMARY

The following command checks the Application Continuity checking for Oracle concrete classes.

```
$ ./orachk -acchk -asmhome /path/orachk/asm-5.0.3/lib/all/asm-all-5.0.3.jar -javahome
/usr/lib/jvm/jre-1.8.0-openjdk.x86_64 -jdbcver 19.1 -appjar /scratch/nfs/tmp/jarfiles
```

Example Output: Application Continuity Summary

Outage Type	Status	Message
Concrete class checks		**Total : 114 Passed : 110 Warning : 0 Failed : 2 (Failed check count is one per file)**
	FAILED	[ac/workload/lobsanity/AnydataOut][[CAST]desc=oracle/sql/ANYDATAmethodname=getDataInfo,lineno=38]

APPENDIX: USING ACCHK FOR APPLICATION CONTINUITY COVERAGE

Destructive testing is a good thing to do. However, introducing failures is non-deterministic. The application can fail over in all the tests, and then in production a failure occurs elsewhere and unexpectedly some requests do not fail over.

Using AC Check Coverage Analysis averts this situation by reporting in advance the percentage of requests that are fully protected by Application Continuity, and for the requests that are not fully protected, which they are and where. Use the coverage check before deployment, and after application changes. Developers and management know how to protect an application release from failures of the underlying infrastructure. If there is a problem, then it can be fixed before the application is released or waived knowing the level of coverage.

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 database user trace directory in user trace files. Then, pass this directory as input to Oracle ORAchk to report the coverage for the application functions. As this check uses Application Continuity, the database and client must be above 12*c*. The application need not necessarily be released with Application Continuity. The check is to help you before release.

The following is a summary of the coverage analysis.

- If a round trip is made to the database server and returns while Application Continuity' capture is enabled during capture phase, then it is counted as a protected call.
- If a round trip is made to the database server while Application Continuity' capture is disabled (not in a request, or following a restricted call or a disable replay API was called), then it is counted as an unprotected call.
- Round trips ignored by capture and replay are ignored in the protection-level statistics.

At the end of processing each trace file, a level of protection for the calls sent to the database is computed.

For each trace: PASS (>= 75), WARNING (25 <= value <75), and FAIL (< 25)

Running the Coverage Report

Turn on tracing at database level.

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 event='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.

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.

Using Application Continuity Checking for Protection Level

EXAMPLE COVERAGE REPORT

```
$ ./orachk -acchk -javahome /tmp/jdk1.8.0_40 -apptrc $ORACLE_BASE/diag/rdbms/$ORACLE_SID/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`

READING THE COVERAGE REPORT

The coverage check produces a directory named `orachk_`*`uname_date_time`*. This report summaries coverage and lists trace files that have WARNINGS or FAIL status. To ensure all requests PASS (Coverage(%) = 100), check the PASS report, `acchk_scorecard_pass.html` under the `reports` directory.

The output includes the database service name, the module name (from `v$session.program`, which can be set on the client side using the connection property on Java, for example, `oracle.jdbc.v$session.program`), the `ACTION` and `CLIENT_ID`, which can be set using `setClientInfo` with `OCSID.ACTION` and `OCSID.CLIENTID` respectively.

EXAMPLE OUTPUT: FOUND IN ORACHK_.....HTML#ACCHK_SCORECARD

Coverage checks		**TotalRequest = 1088** **PASS = 1082** **WARNING = 1** **FAIL = 5**
	FAIL	Trace file name = orcl1_ora_30467.trc Line number of Request start = 1409 Request number = 6 SERVICE NAME = (srv_auto_pdb1) MODULE NAME = (SQL*Plus) ACTION NAME = () CLIENT ID = () Coverage(%) = 12 Protected Calls = 1 Unprotected Calls = 7
	WARNING	Trace file name = ATPCDB12_ora_321597.trc Line number of Request start = 653 Request number = 1 SERVICE NAME = (HRPDB1_tp.atp.oraclecloud.com) MODULE NAME = (JDBC Thin Client) ACTION NAME = () CLIENT ID = () Coverage(%) = 25 Protected Calls = 1 Unprotected Calls = 3

	FAIL	Trace file name = ATPCDB12_ora_292714.trc Line number of Request start = 1598 Request number = 7 SERVICE NAME = (HRPDB1_tp.atp.oraclecloud.com) MODULE NAME = (SQL*Plus) ACTION NAME = () CLIENT ID = () Coverage(%) = 16 Protected Calls = 1 Unprotected Calls = 5
	FAIL	Trace file name = ATPCDB12_ora_112022.trc Line number of Request start = 1167 Request number = 3 SERVICE NAME = (HRPDB1_tp.atp.oraclecloud.com) MODULE NAME = (JDBC Thin Client) ACTION NAME = () CLIENT ID = () Coverage(%) = 0 Protected Calls = 0 Unprotected Calls = 1
	FAIL	Trace file name = ATPCDB12_ora_112022.trc Line number of Request start = 1689 Request number = 5 SERVICE NAME = (HRPDB1_tp.atp.oraclecloud.com) MODULE NAME = (JDBC Thin Client) ACTION NAME = () CLIENT ID = () Coverage(%) = 0 Protected Calls = 0 Unprotected Calls = 1
	PASS	Report containing checks that passed: [Full Path]_060219_184513/reports/acchk_scorecard_pass.html

APPENDIX: SQL TO REPORT PROTECTION BY PDB, SERVICE AND HISTORY

To report protection by PDB, use the following example:

```
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;
```

To report protection by service, use the following example:

```
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;
```

To report protection history over last three days, use the following example:

```
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 %"

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' a
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;
```

ADDITIONAL WHITEPAPERS

Fast Application Notification

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

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)

GRACEFUL APPLICATION SWITCHOVER IN RAC WITH NO APPLICATION INTERRUPTION

My Oracle Support (MOS) Note: Doc ID 1593712.1

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

August 2020August 2020
Author: Troy Anthony, Carol Colrain
Contributing Authors: Burt Clouse, Ian Cookson

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