6
High Availability

This chapter explains how to design high availability into the database and database applications.

Topics:

6.1 Transparent Application Failover (TAF)

This section describes what Transparent Application Failover (TAF) is, how to configure TAF, and using TAF callbacks to notify the application of events as they are generated.

Topics:

6.1.1 About Transparent Application Failover

Transparent Application Failover (TAF) is a client-side feature of Oracle Call Interface (OCI), OCCI, Java Database Connectivity (JDBC) OCI driver, and ODP.NET designed to minimize disruptions to end-user applications that occur when database connectivity fails because of instance or network failure. TAF can be implemented on a variety of system configurations including Oracle Real Application Clusters (Oracle RAC), Oracle Data Guard physical standby databases, and on a single instance system after it restarts during a recovery.

TAF enables client applications to automatically (transparently) reconnect to a preconfigured secondary instance, creating a fresh connection, but identical to the connection that was established on the first original instance. That is, the connection properties are the same as that of the earlier connection, regardless of how the connection was lost. In this case, the active transactions roll back. Also, all statements that an application attempts to use after a failure attempt also failover.

See Also:

- *Oracle Call Interface Programmer's Guide* for more information about OCI TAF
- *Oracle C++ Call Interface Programmer's Guide* for more information about OCCI TAF
- *Oracle Database JDBC Developer's Guide* for more information about JDBC TAF
- *Oracle Data Provider for .NET Developer's Guide for Microsoft Windows* for more information about ODP.NET TAF
- *Oracle Database Net Services Reference* for more information about client-side configuration of TAF (Connect Data Section)
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the server-side configuration of TAF (`DBMS_SERVICE`)

6.1.2 Configuring Transparent Application Failover

TAF can be configured on both the client side and server side with the server side taking precedence if both client and server sides are configured. On the client side, you configure TAF by including the `FAILOVER_MODE` parameter in the `CONNECT_DATA` portion of a connect descriptor. On the server side, you configure TAF by modifying the target service with the `DBMS_SERVICE.MODIFY_SERVICE` packaged procedure.

See Also:

- About Transparent Application Failover for more information about configuration

6.1.3 Using Transparent Application Failover Callbacks

TAF callbacks are callbacks that are registered in case of failover and called during failover to notify the application of events as they are generated. They are called several times while reestablishing the user's session.

As the application developer you may want to inform the user that failover is in progress because there is a slight delay as failover proceeds. The first call to the callback carries out that function. Also, when failover is successful and the connection is reestablished, you may want to inform the user that failover has happened and then you may want to replay `ALTER SESSION` commands because these commands are not automatically replayed on the second instance. A subsequent call to the callback performs that function. Also, if failover is unsuccessful, then you want to inform the application that failover cannot occur. A third call to the callback performs this function as well.

Using TAF callbacks makes possible:

- Notifying users of the status of failover throughout the failover process; when failover is underway, when failover is successful, and when failover is unsuccessful
- Replaying of `ALTER SESSION` commands when that is needed

- Reauthenticating a user handle besides the primary handle for each time a session begins on the new connection. Because each user handle represents a server-side session, the client may want to replay `ALTER SESSION` commands for that session.

> **See Also:**
>
> Configuring Transparent Application Failover for specific callback registration information for each interface

6.2 Oracle Connection Manager in Traffic Director Mode

This feature allows the Oracle database Connection Manager (CMAN) to be configured in Traffic Director mode to serve clients connecting to different database services, with HA and performance features configurable at the router level, useful to all the clients connected.

Oracle Connection Manager in Traffic Director mode furnishes support for:

- Transparent performance enhancements and connection multiplexing
  - With multiple CMAN in Traffic Director mode instances, applications get increased scalability through client-side connection-time load balancing or with a load balancer (BIG-IP, NGINX, and others)
- Zero application downtime including: planned database maintenance or pluggable database (PDB) relocation and unplanned database outages for read-mostly workloads.
- High Availability of CMAN in Traffic Director mode to avoid a single point of failure.
- Security and isolation:
  - Database Proxy supporting transmission control protocol/transmission control protocol secure (TCP/TCPS) and protocol conversion
  - Firewall based on the IP address, service name, and secure socket layer/transport layer security (SSL/TLS) wallets
  - Tenant isolation in a multi-tenant environment
  - Protection against denial-of-service and fuzzing attacks
  - Secure tunneling of database traffic across Oracle Database On-premises and Oracle Cloud

Related Topics

- *Oracle Database Security Guide*
- *Oracle Database Net Services Administrator's Guide*
- *Oracle Database Net Services Administrator's Guide*
- *Oracle Database Net Services Administrator's Guide*

6.3 About Fast Application Notification (FAN)

An important component of high availability is a notification mechanism called Fast Application Notification (FAN). FAN notifies other processes about configuration and service level information that includes service status changes, such as UP or DOWN events. Applications

can respond to FAN events and take immediate action. FAN UP and DOWN events can apply to instances, services, and nodes.

FAN provides the ability to immediately terminate an active transaction when an instance or server fails. FAN integrated Oracle clients receive the events and respond. Applications can respond either by propagating the error to the user or by resubmitting the transactions and masking the error from the application user. When a DOWN event occurs, FAN integrated clients immediately clean up connections to the terminated database. When an UP event occurs, the FAN integrated clients create new connections to the new primary database instance.

Oracle has integrated FAN with many of the common Oracle client drivers. Therefore, the easiest way to use FAN is to use one of the following integrated Oracle clients:

- OCI session pools
- Universal Connection Pool for Java
- Thin JDBC Driver (12.2 and later)
- ODP.NET managed and un-managed providers
- All WebLogic server data sources, and Oracle Tuxedo

The overall goal is to enable applications to consistently obtain connections to the available primary database at anytime.

FAN events are published using Oracle Notification Service. The publication mechanisms are automatically configured as part of an Oracle RAC installation. Here, an Oracle RAC installation means any installation of Oracle Clusterware with Oracle RAC, Oracle RAC One Node, Oracle Data Guard (fast-start-failover), or Oracle Data Guard single instance with Oracle Clusterware). Beginning with Oracle Database 12*c* Release 1 (12.1), ONS is the primary notification mechanism for a new client (Oracle Database 12*c* Release 1 (12.1)) and a new server (Oracle Database 12*c* Release 1 (12.1)), while the AQ HA notification feature is deprecated and maintained only or backward compatibility when there is an older OCI or ODP.NET unmanaged client (pre-Oracle Database 12*c* Release 1 (12.1)) or old server (pre-Oracle Database 12*c* Release 1 (12.1)).

When you use JDBC or Oracle Database 12 *c* Release 1 (12.1.0.1) OCI or ODP.NET clients, the Oracle Notification Service is automatically configured using your TNS. When you use OCI-based clients, set HA notifications (`-notification = TRUE`) for your services and set EVENTS in `oraccess.xml`.

See Also:

- *Oracle Real Application Clusters Administration and Deployment Guide* for more information about FAN
- *Oracle Database Administrator's Guide* for information about enabling FAN events in an Oracle Restart environment
- *Oracle Call Interface Programmer's Guide* for more information about receiving runtime connection load balancing advisory FAN events to balance application session requests in an Oracle RAC environment with Oracle Clusterware set up and enabled
- *Oracle C++ Call Interface Programmer's Guide* for more information about runtime load balancing of the stateless connection pool by use of service metrics distributed by the Oracle RAC load-balancing advisory
- *Oracle Database JDBC Developer's Guide* for more information about fast connection failover

6.3.1 About Receiving FAN Event Notifications

Starting from Oracle Database 12c Release 2 (12.2), the Oracle RAC FAN APIs provide an alternative for taking advantage of the high-availability (HA) features of Oracle Database, if you do not use Universal Connection Pool or Oracle WebLogic Server with Active Grid Link (AGL). This feature depends on the Oracle Notification System (ONS) message transport mechanism.

This feature requires configuring your system, servers, and clients to use ONS. For using Oracle RAC Fast Application Notification, the `simplefan.jar` file must be present in the `CLASSPATH`, and either the `ons.jar` file must be present in the `CLASSPATH` or an Oracle Notification Services (ONS) client must be installed and running in the client system.

See Also:

Oracle Database JDBC Developer's Guide for more information about Oracle RAC FAN APIs.

6.4 About Fast Connection Failover (FCF)

In a configuration with a standby database, after you have added Oracle Notification Services (ONS) to your Oracle Restart configurations and enabled Oracle Advanced Queuing (AQ) HA notifications for your services, you can enable clients for Fast Connection Failover (FCF). The clients then receive FAN events and can relocate connections to the current primary database after an Oracle Data Guard failover. Beginning with Oracle Database 12*c* Release 1 (12.1), ONS is the primary notification mechanism for a new client (Oracle Database 12*c* Release 1 (12.1)) and a new server (Oracle Database 12*c* Release 1 (12.1)), while the AQ HA notification feature is deprecated and maintained only for backward compatibility when there is an old client (pre-Oracle Database 12*c* Release 1 (12.1)) or old server (pre-Oracle Database 12*c* Release 1 (12.1)).

For databases with no standby database configured, you can still configure the client FAN events. When there is an outage (planned or unplanned), you can configure the client to retry the connection to the database. Because Oracle Restart restarts the failed database, the client can reconnect when the database restarts.

You must enable FAN events to provide FAN integrated clients support for FCF in an Oracle Data Guard or standalone environment with no standby database.

FCF offers a driver-independent way for your Java Database Connectivity (JDBC) application to take advantage of the connection failover facilities offered by Oracle Database. FCF is integrated with Universal Connection Pool (UCP) and Oracle RAC to provide high availability event notification.

OCI clients can enable FCF by registering to receive notifications about Oracle Restart high availability FAN events and respond when events occur. This improves the session failover response time in OCI and removes terminated connections from connection and session pools. This feature works on OCI applications, including those that use Transparent Application Failover (TAF), connection pools, or session pools.

See Also:

- *Oracle C++ Call Interface Programmer's Guide* for more information about runtime load balancing of the stateless connection pool by use of service metrics distributed by the Oracle RAC load-balancing advisory
- *Oracle Database JDBC Developer's Guide* for more information about fast connection failover
- *Oracle Universal Connection Pool for JDBC Java API Reference*
- *Oracle Database Administrator's Guide* for information about enabling FCF for JDBC clients
- *Oracle Database Administrator's Guide* for information about enabling FCF for OCI clients
- *Oracle Database Administrator's Guide* for information about enabling FCF for ODP.NET clients

6.5 About Application Continuity

Application Continuity masks planned or unplanned outages (that cause database session unavailability) by attempting to rebuild the database sessions transactional and non-transactional states, so the outage appears to the user as nothing more than a delayed execution.

Transparent Application Continuity (TAC) transparently records the database session and transactional state so the database can be recovered following recoverable outages. This is done safely and with no reliance on application knowledge or application code changes, thus allowing TAC to be enabled as a standard. TAC is enabled by default starting Oracle Database Release 21c.

Oracle Database Release 21c introduces the `RESET_STATE` service attribute, which you can use to reset state in a session. You can set the `RESET_STATE` service attribute in the `DBMS_SERVICE.CREATE_SERVICE` procedure. It is executed at the end of a request before the next processing occurs in that request. You can use this service attribute when a session is

returned to a connection pool, so that the session state does not leak from one session usage to the next.

The RESET_STATE attribute can have the following values:

- NONE: If you set the RESET_STATE attribute to NONE, then the session is not cleaned at the end of the request. This is the default value of this attribute.
- BASIC: If you set the RESET_STATE attribute to BASIC, then the session states, which cannot be restored, are reset.
- LOGIN: If you set the RESET_STATE attribute to LOGIN, then the session states set by an application in a request is restored to the session state as it was at the end of LOGON when each request ends.

See the RESET_STATE topic below for more information.

Application Continuity supports recovering any outage that is due to database unavailability against a copy of a database with the same DBID (forward in time) or within an Active Data Guard farm. This may be Oracle RAC One, Oracle Real Application Clusters, within an Active Data Guard, Multitenant using PDB relocate with Oracle RAC or across RACs or across to Active Data Guard (ADG).

See Also:

- *Oracle Real Application Clusters Administration and Deployment Guide* for more information about Application Continuity
- CREATE_SERVICE Procedure in *Oracle Database PL/SQL Packages and Types Reference* for more information about the CREATE_SERVICE and other DBMS_SERVICE subprograms

6.5.1 Reset Database Session State to Prevent Application State Leaks, Use RESET_STATE

When you use the RESET_STATE service attribute, the session state set in a request by an application is cleared when the request ends.

The RESET_STATE service attribute (RESET_STATE) is recommended for all stateless applications to prevent the leakage of a session state to later reuses.

Note:

As of Oracle Database 23ai, RESET_STATE works independent of Application Continuity.

Stateless Applications

A stateless application is an application program that does not use the session state of one request, such as context and PL/SQL states that were set by a prior usage of that session, in another web request or similar connection pool usage. The necessary state to handle the

request is contained within the request, as a part of the request itself; in the URL, query-string parameters, request body, or headers.

In a cloud environment, it is preferable for applications to be stateless for scalability and portability. Being stateless enables greater scalability because the server does not have to maintain, update, or communicate a session state. For example, load balancers do not have to consider session affinity for stateless systems. Most modern web applications are stateless.

Prevent Application State Leaking to Later Usages

Setting the session state in a request leaves the session with the current state, meaning that subsequent usages of that session can see the current session state. For example, when an application borrows and returns a connection to a connection pool, if the sessions state is not cleared, the next usage of that connection can see the session state of the previous usage.

`RESET_STATE` is an attribute of the database service. When you use `RESET_STATE`, the session state set by an application in a request is cleared when the database request ends. When `RESET_STATE` is used, an application can depend on the state being reset at the end of a request. Without `RESET_STATE`, application developers must cancel their cursors and clear the session state that has been set before returning their connections to a pool for reuse.

`RESET_STATE` is used with applications that are stateless between requests. These types of applications use the session state in a request, and do not rely on that session state in the later requests. The necessary session state that the request needs is contained within the request itself. REST, Oracle REST Data Services (ORDS), Oracle Application Express (APEX) are examples of stateless applications.

`RESET_STATE` is available for all applications that use Oracle or third-party connection pools with request boundaries and is supported for all Oracle clients, such as Oracle Call Interface (OCI) session pools or Java Database Connectivity (JDBC) universal connection pools. Setting `RESET_STATE` enables automatic resetting of session states at an explicit end of request. `RESET_STATE` does not apply to implicit request boundaries, such as those with DRCP implicit statement caching or Transparent Application Continuity (TAC).

OPTIONS FOR RESET_STATE

RESET_STATE = BASIC

When you use `RESET_STATE` at `BASIC`, the database clears complex session states when each request ends.

Using `RESET_STATE BASIC` has the following impact at the end of a request:

- Cursors are canceled.
- PL/SQL global variables are cleared.
- Temporary tables that have a session-based duration are truncated.
- Temporary LOBs that have a session-based duration are cleared.
- Session local sequences are reset.
- `DBMS_OUTPUT` buffer is cleared.

The order of processing for resetting session states at `BASIC` is the following:

1. Borrow from a connection pool or other request explicitly marked
2. Label Callback
3. Begin Request

4. User request runs
5. End Request
6. `RESET_STATE BASIC` is applied automatically by the server

RESET_STATE = LOGIN

When you use `RESET_STATE` at `LOGIN`, the session state set by an application in a request is restored to the session state as it was at the end of LOGON when each request ends.

Setting `RESET_STATE` to `LOGIN` enables automatic resetting of session states back to the session state that was captured at the end of the logon and before the first request in the session is run.

Using `RESET_STATE LOGIN` has the following impact at the end of a request:

- Cursors are canceled. PL/SQL global variables are cleared.
- Temporary tables that have a session-based duration are truncated.
- Temporary LOBs that have a session-based duration are cleared.
- Session local sequences are reset.
- `DBMS_OUTPUT` buffer is cleared.
- `DBMS_OUTPUT` enabled or disabled is as it was at Logon.
- Session-modifiable parameters are restored as they were at the end of Logon.
- Local `SYS_Context` are restored as they were at the end of Logon.
- Non-password Protected Roles are restored as they were at the end of Logon.
- Dedicated database links are closed.
- `SQL_Trace` is restored as it was at the end of Logon.
- `CLIENT_INFO` and `CLIENT_ID` are restored as they were at the end of Logon.

The order of processing for resetting session states at `LOGIN` is the following:

1. Environment variable changed in client side
2. Alter session happened and client caches the new values
3. Logon trigger
4. `RESET_STATE BASIC` clears complex session states
5. `RESET_STATE` Session LOGON Template is captured
6. Label Callback
7. Begin Request
8. User request runs
9. End Request
10. `RESET_STATE LOGIN` is applied automatically by the server

`RESET_STATE` is a very important database feature because developers can rely on these listed session states being reset when a session is returned to a connection pool with request boundaries. This can be an Oracle connection pool or a custom connection pool with added request boundaries.

`RESET_STATE` also improves your protection when using TAC, because the session state is clean at the beginning of a new request.

6.6 About Transaction Guard

Transaction Guard is a reliable protocol and interface that returns the commit outcome of the current in-flight transaction when an error, or a time-out is returned to the client. Applications can leverage the Transaction Guard interface to code graceful recoverable error handling. Providing unambiguous message during an outage greatly improves the user experience.

Transaction Guard introduces the concept of *at-most-once transaction semantics*, also referred to as transaction idempotence. When an application opens a connection to the database using this service, the logical transaction ID (LTXID) is generated at authentication and stored in the session handle at the database and a copy at the client driver. This is a globally unique ID that identifies the database transaction from the application perspective. Applications use the Transaction Guard interface to obtain a known commit outcome following a recoverable error.

When there is an outage, an application using Transaction Guard can retrieve the LTXID from the previous failed session's handle and use it to determine the outcome of the transaction that was active prior to the session failure. If the LTXID is determined to be `UNCOMMITTED`, then the application can return the `UNCOMMITTED` outcome to the user to decide what action to take, or optionally, the application can replay an uncommitted transaction. If the LTXID is determined to be `COMMITTED`, then the transaction is committed and the application can return this outcome to the end user and might be able to take a new connection and continue. Transaction Guard also reports whether the last user call not only `COMMITTED`, but also whether it completed changing needed non-transactional states - see `USER_CALL_COMPLETED`.

> **See Also:**
>
> Using Transaction Guard

6.7 About Service and Load Management for Database Clouds

The database cloud is a self-contained system of databases integrated by the service and load management framework that ensures high performance, availability and optimal utilization of resources. This framework provides effective balancing of processing workload across distributed databases that maintain multiple synchronized replicas both locally and in geographically disparate regional data centers. Replicas may be instances in an Oracle RAC environment, or single instances interconnected using Oracle Data Guard, Oracle Golden Gate, or any combination that supports replication technology. Thus, the service and load management framework provides dynamic load balancing, failover, and centralized service management for these replicated databases.

A global service is a database service provided by multiple databases synchronized through some form of data replication that satisfies quality of service requirements for the service. This allows a client request for a service to be forwarded to any database that provides that service.

A database pool within a database cloud consists of all databases that provide the same global service that belong to the same administrative domain. The database cloud is partitioned into multiple database pools to simplify service management and to provide higher levels of security by allowing each pool to be administered by a different administrator.

A global service manager (GSM) is a software component that provides service-level load balancing and centralized management of services within the database cloud. Load balancing is done at connection and runtime. Other capabilities provided by GSM include creation,

configuration, starting, stopping, and relocation of services and maintaining global service properties such as cardinality and region locality. A region is a logical boundary known as a data center that contains database clients and servers that are considered close enough to each other so as to reduce network latency to levels required by applications accessing these data centers.

The GSM can run on a separate host, or can be colocated with a database instance. Every region must have at least one GSM instance running. For high availability, Oracle recommends that you deploy multiple GSM instances in a region. A GSM instance belongs to only one particular region; however, it manages global services in all database pools associated with this region.

From an application developer's perspective, a client program connects to a regional global service manager (GSM) and requests a connection to a global service. The client need not specify which database or instance it requires. GSM forwards the client's request to the optimal instance within a database pool in the database cloud that offers the service.

Beginning with Oracle Database 12*c* Release 1 (12.1.0.1), the DBA can configure client-side connect strings for database services in a Global Data Services (GDS) framework using an Oracle Net string.

Introduced in Oracle Database 12*c* Release 1 (12.1.0.1), the logical transaction ID (LTXID) is initially generated at authentication and stored in the session handle and used to identify a database transaction from the application perspective. The logical transaction ID is globally unique and identifies the transaction within a highly available (HA) infrastructure.

Using the HA Framework, a client application (JDBC, OCI, and ODP.NET) supports fast application notification (FAN) messages. FAN is designed to quickly notify an application of outages at the node, database, instance, service, and public network levels. After being notified of the outage, an application can reestablish the failed connection on a surviving instance.

Beginning with Oracle Database 12*c* Release 1 (12.1.0.1), the DBA can configure server-side settings for the database services used by the applications to support Application Continuity for Java and Transaction Guard.

See Also:

- *Oracle Database Concepts* for an overview of global service management and description of the physical and logical components of the service and load management framework
- Introduction to Global Data Services in *Oracle Database Global Data Services Concepts and Administration Guide* for more information about global service management in a database cloud
- Installation and Configuration in *Oracle Database Global Data Services Concepts and Administration Guide* for more information about configuring database clients for connectivity to the Global Data Services (GDS) framework.
- *Oracle Call Interface Programmer's Guide* for more information about OCI, Application Continuity, and Transaction Guard
- *Oracle Database JDBC Developer's Guide* for more information about JDBC, Application Continuity, and Transaction Guard