ORACLE DATABASE 12c

Fast Application Notification (FAN)

Includes fanWatcher:

A utility to subscribe to ONS and view FAN events

ORACLE WHITE PAPER | DECEMBER 2016

ORACLE

Contents

Purpose and Audience

This white paper discusses Oracle 12*c* *Fast Application Notification* (FAN)[1] in Real Application Clusters (RAC) and Oracle Data Guard environments. FAN is a critical component towards solving the poor experience that end users can encounter when planned maintenance, unplanned outages, and load imbalances occur that make database instances unavailable or unresponsive. FAN enables end-to-end, lights-out recovery of applications and load balancing at runtime based on real transaction performance.

With FAN, the continuous service and continuous connections built into Real Application Clusters and Data Guard are extended to applications and application servers. When the state of database services change, (for example, up, down, or unresponsive), the new status is posted to interested subscribers through FAN events. FAN provides rapid notification about state changes for database services, instances, the databases themselves, and the nodes that form the cluster, and starting with Oracle Database 12c with Global Data Services, distributed database systems.

Oracle drivers and Oracle pools use FAN events to achieve the following:

- Draining of work during planned maintenance with no errors whatsoever returned to applications,
- Very fast detection of failures so that recovery of applications can occur in real time
- Load balancing of incoming work at runtime when performance imbalances occur and also following instances leaving and joining the system and resources becoming available.
- Affinity advice for incoming work so that related conversations, for example successive web sessions, are routed together for best performance.

This paper:

1. Outlines the benefits of enabling FAN events for your database system

2. Dissects the FAN events and their published fields

3. Describes how to view FAN events using `FANwatcher`

4. Steps through how to integrate and enable FAN events for Oracle and non-Oracle applications

The target audience includes RAC database administrators, Data Guard database administrators, and application integrators who need rapid notification of planned maintenance and unplanned outages integrated with their applications or application servers, monitoring consoles or internal business workflow systems. It is assumed that the reader is familiar with the concepts presented in the following references:

Oracle Clusterware Administration and Deployment Guide 12c Release 1 (12.1) Part number E48819-07

Oracle Real Application Clusters Administration and Deployment Guide 12c Release 1 (12.1) Part Number E48838-09

[1] FAN was introduced with Oracle Database 10g. This paper is relevant to all releases since then. Some information may be specific to Oracle Database 12*c* (command syntax, for example).

Benefits of Using FAN Events

Why is it important to use FAN?

Applications can waste time in a number of critical ways:

- Waiting for TCP/IP time-outs for minutes when a node fails without closing sockets, and for every subsequent connection while that IP address is down.
- Attempting to connect when services are down.
- Not connecting when services resume.
- Processing the last result at the client when the server goes down.
- Attempting to execute work on slow, hung, and dead nodes.

Still worse, these sources of poor performance are not captured and reported by the standard database performance methodologies. When a node fails without closing sockets, all sessions that are blocked in an IO (read or write) wait for TCP keepalive. This wait status is the typical condition for an application using all databases. Sessions processing the last result are even worse off, not receiving an interrupt until the next data is requested. Using FAN events eliminates applications waiting on TCP time-outs, time wasted processing the last result at the client after a failure has occurred, and time wasted executing work on slow, hung, or dead nodes.

Last century, client or mid-tier applications connected to the database relied on connection timeouts, out-of-band polling mechanisms, or other custom solutions to realize that a system component had failed, triggering actions to mitigate the impact of that failure. This approach had severe implications in (1) application availability because downtimes were extended and noticeable (due to polling intervals and timeout length), and (2) management of enterprise environments, because of the exponential growth in the number of server-side components, both horizontally (across all nodes) and vertically (across node components such as listeners, instances and application services) caused a major increase in the sheer number of polling actions.

FAN was released with Oracle Database 10g RAC. It solved these problems by integrating database events with listeners, applications and application servers, and IT management consoles (including Oracle Enterprise Manager, trouble ticket loggers and e-mail/texting servers). FAN posts events as soon as a change in the system is detected, resulting in both immediate actions at the incident, and overall improved resource usage as it is unnecessary for remote application components to implement status polling.

FAN with Oracle Database 12c Release 1

Starting with Oracle Database 12c, there are three important enhancements for FAN –

- FAN is default, configured and enabled out of the box with Oracle Real Application Clusters
- All Oracle clients use the Oracle Notification System (ONS) as the transport for FAN
- FAN is posted by Global Data Services (GDS) to allow FAN events to span data centers

FAN is on by Default

For Oracle Database 12*c*, no changes are needed to configure FAN. On the database side, FAN is configured for Oracle Grid Infrastructure during installation. `srvctl` does offer an interface if a particular site needs to use different ports, but this is the exception, not the norm. Starting with Oracle Database 12*c*, the goal is no configuration for all Oracle clients. When using Oracle Database 12*c* with Oracle Grid Infrastructure 12*c* or Oracle GDS and Oracle

Database client 12*c*, FAN is auto-configured. This means that when the client starts, it queries the databases for the ONS end-points automatically. The automatic configuration spans data centers. The client automatically receives an ONS configuration from each database listed in the connection URL. No configuration of ONS is required at the client other than enabling FAN.

FAN 12c Standardizes on ONS as the Transport

From Oracle Database 12c, all Oracle 12c FAN clients use the Oracle Notification System to receive FAN events. The reason for standardizing on ONS as the transport for FAN is "the failed component cannot tell you that it has failed". ONS runs outside of the database and across the system with no database dependencies. When a database is stopped or fails, FAN posts the status change events immediately, and ONS delivers them immediately. To support clients from earlier releases and new clients using older database versions, events are sent over both transport methods, ONS and AQ automatically.

Global Data Services (GDS)

From Oracle Database 12c, FAN is posted by Global Data Services (GDS) for spanning data centers, particularly for Oracle Active Data Guard farms that need runtime load balancing. GDS takes into account the service placement attributes, automatically performs an inter-database service failover to another available database for planned maintenance that involves a data center change, and if unplanned outages occur, notifies failures of an entire database. The Oracle clients and connection pools are interrupted on failure events and notified when a global service has been newly started.

How to use FAN

Using FAN requires no code changes whatsoever. The best and by far the easiest way to use FAN is to use an Oracle connection pool, an Oracle client driver or an Oracle Application that is configured for FAN.

FAN events are integrated with:

- Oracle Fusion Middleware and Oracle WebLogic Server (*Oracle® Fusion Middleware Administering JDBC Data Sources for Oracle WebLogic Server 12c (12.1.3)* Part Number E41864-02)
- Oracle Data Guard Broker *(Oracle® Data Guard Broker 12c Release 1 (12.1)* Part Number E48241-06)
- Oracle Enterprise Manager Cloud Control (*Cloud Control Basic Installation Guide 12c Release 4 (12.1.0.4)*)
- Oracle JDBC Universal Connection Pool for both JDBC thin and OCI interfaces (*Oracle® Universal Connection Pool for JDBC Developer's Guide 12c Release 1 (12.1)* Part Number E49541-01)
- ODP.NET (*Oracle® Database Development Guide 12c Release 1 (12.1)* Part Number E41452-06)
- SQLPLUS (*SQL*Plus® User's Guide and Reference Release 12.1* Part Number E18404-12)
- PHP (*Oracle® Database 2 Day + PHP Developer's Guide 12c Release 1 (12.1)* Part Number E18554-05)
- Global Data Services (*Oracle® Database Global Data Services Concepts and Administration Guide 12c Release 1 (12.1)* Part Number E22100-10)

When using 3rd party clients such as IBM WebSphere and Apache TomCat, FAN is supported by IBM and Apache respectively by replacing the default connection pool with the Oracle Universal Connection Pool. Oracle's Universal Connection Pool (UCP) has the capability to hide planned maintenance when used with FAN, and to hide unplanned outages when used with Oracle Database 12*c* and Application Continuity. UCP is the proven and recommended client solution for Java-based applications when using Oracle RAC, Oracle Data Guard, or Active Data Guard.

How do you use FAN for Fast Failover

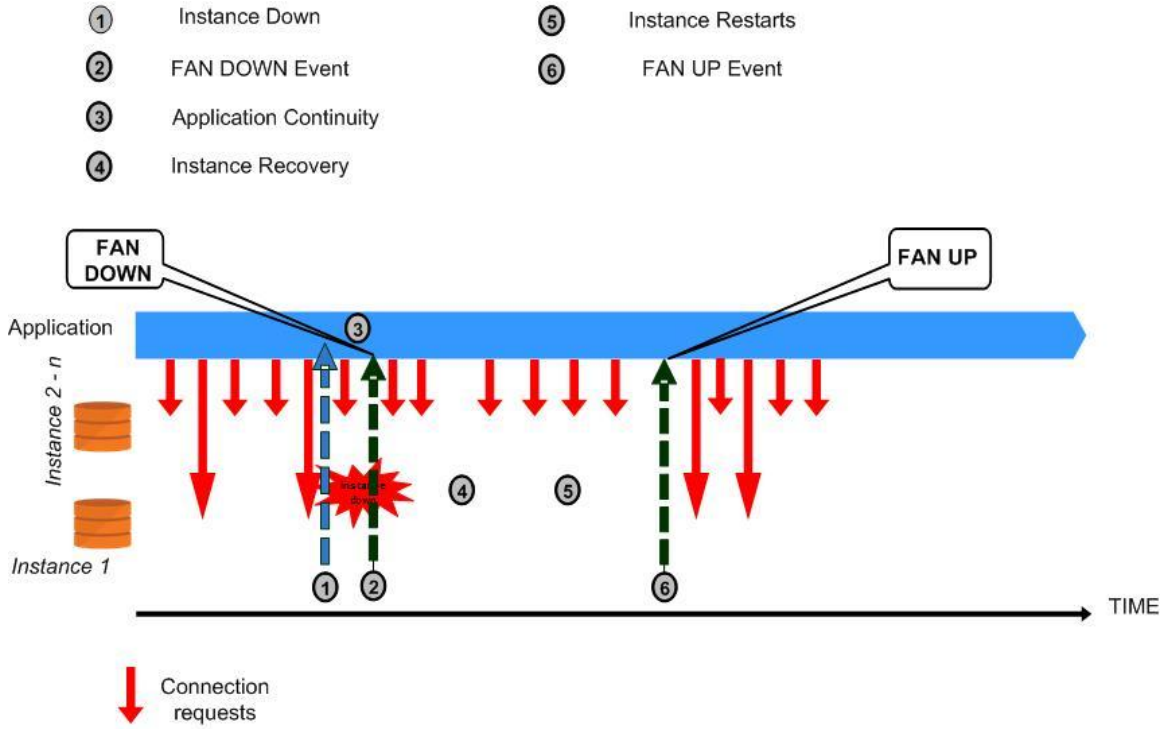

Figure 1: Timeline showing FAN activity during Instance Failure

Figure 1 shows a timeline for Instance Failure. The application has FAN enabled, for example UCP with Fast Connection Failover enabled, connected to a dynamic database service that is being offered on multiple instances of a RAC database in a UNIFORM configuration, or in a configuration where more than one instance is *preferred*. The application has active sessions on all instances. The following occurs during unplanned outages. This example uses instance failure:

- At Step 1: The instance crashes.
- The FAN planned `DOWN` event, delivered at Step 2, clears idle sessions from the connection pool immediately.

  Existing connections on other instances remain usable, and new connections are opened to these instances if needed.
- For those pools that are configured to use Application Continuity, active sessions are restored on a surviving instance and recovered by Application Continuity, masking the outage from users and applications.

  If not protected by Application Continuity, any sessions in active communication with the instance will receive an error, as shown in Step 3. The application can test if this is a recoverable error and could initiate a reconnection attempt at this time. If, as in our example, the service is available on another instance they would be able to get a new connection.

- As this is a RAC database, a surviving instance will perform recovery for the failed instance, as shown in Step 6, and then the instance is restarted by the Grid Infrastructure (Step 7).
- The FAN `UP` event for the service informs the connection pool that a new instance is available for use, allowing sessions to be created on this instance at next request submission.

How do you use FAN for Transparent Planned Maintenance

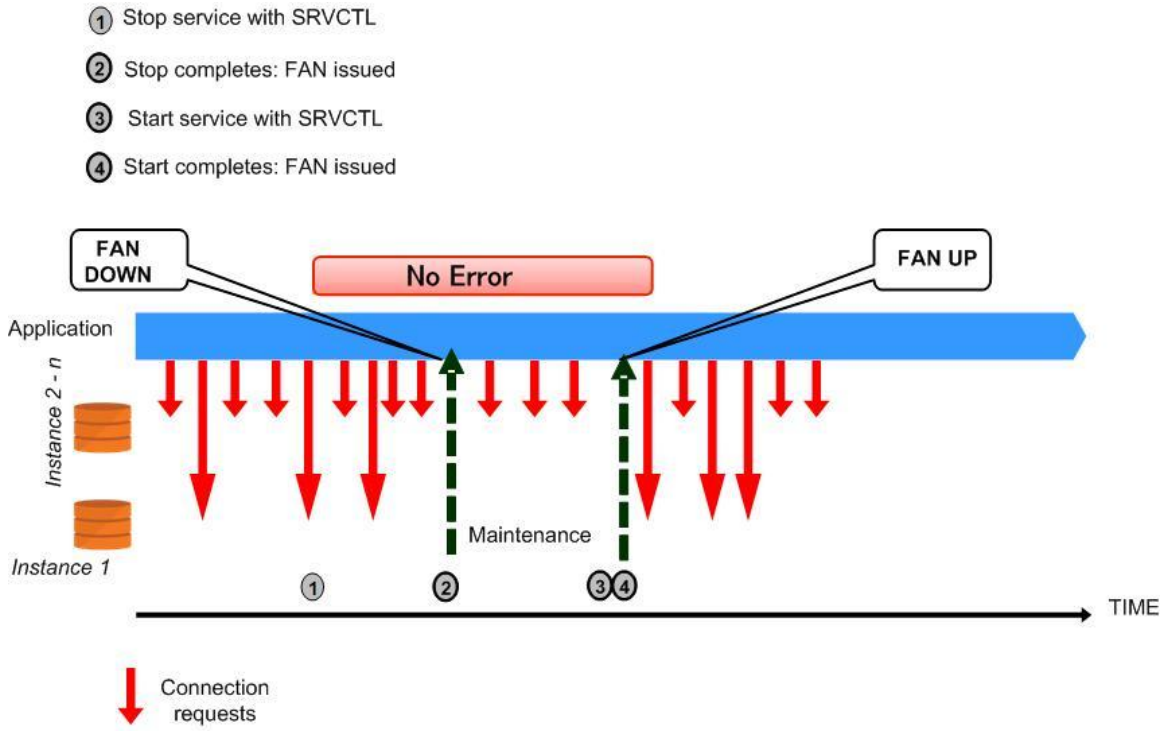

Figure 2: Timeline showing FAN activity during Planned Maintenance

Figure 2 shows a timeline for Planned Maintenance, applying a database patch for example. The application has FAN enabled, for example UCP with Fast Connection Failover enabled, connected to a dynamic database service that is offered on multiple instances of a RAC database in a UNIFORM configuration, or in a configuration where more than one instance is *preferred*. The application has active sessions on all instances. To manage a planned outage with no application interruption the patch application rolls across RAC:

- At Step 1: Use `srvctl` to shut down the service on an instance as we are using a UNIFORM service. If the service was not UNIFORM we could relocate the service to another instance. Do not use the `-force` flag with any of these commands. The connection pool automatically releases a connection at a request boundary, typically when the connection is checked in to the pool.
- The FAN planned `DOWN` event, delivered at Step 2, clears idle sessions from the connection pool immediately and marks active sessions to be released at the next return to the pool. These FAN actions drain the sessions from the instance without disrupting the application or users.

Existing connections on other instances remain usable, and new connections are opened to these instances if needed.

- Not all sessions, in all cases, will check their connections into the pool. It is best practice to have a timeout period after which the instance is forcibly shut down, evicting any remaining client connections.

  For those pools that are configured to use Application Continuity these remaining sessions are recovered by Application Continuity, masking the outage from users and applications.

- Once the upgrade, patch, or repair is complete, restart the instance and the service on the original node using `srvctl` (Step 3).

- The FAN `UP` event for the service informs the connection pool that a new instance is available for use, allowing sessions to be created on this instance at next request submission.

What do FAN events look like?

Oracle FAN events consist of header and payload information delivered as a set of name-value pairs describing the name, type and nature of the event, and the time that the event occurred. Based on this payload, the event recipient, normally an Oracle Client, takes actions such as refreshing stale connection references for Oracle connection pools, automatically recovering the in-flight work when that client is configured to use Application Continuity, determining the last committed outcome and returning this to the user when that client is configured to use Transaction Guard, logging a service request or sending a text to the database administrator when that client is Enterprise Manager.

A FAN client uses the information on the connection, called the connection signature, to know which connections to operate on when an event arrives. (Table 1)

Table 1 – FAN Connection Signature

FAN Parameter	Matching Database Signature
Service	`sys_context('userenv', 'service_name')`
Database unique name	`sys_context('userenv','db_unique_name')`
Instance	`sys_context('userenv', 'instance_name')`
Node name	`sys_context('userenv', 'server_host')`

FAN High Availability Events

Oracle FAN events are based on database services as the preferred database connection point for all applications. Database services decouple any hardwired mapping between a connection request and a database instance, and provide value-add functionality on their own, such as connection load balancing, dynamic workload management, service levels and priorities, connection pool reorganizations, and Application Continuity.

The event types that are important for Oracle FAN are:

- Service events for dynamic database services.
- Node events, which includes cluster membership states and native join/leave operations for nodes and networks that are used by all clients to group operations
- Instance events that are used by OCI based clients to group operations. (JDBC includes this level in a future release.)
- Runtime load balancing and affinity advice that are used by connection pool clients to direct incoming database requests to the best place to run based on prior affinity and load advice.

In addition to application services, FAN standardizes on the generation, presentation, and delivery of events related to managed database resources for high availability and for runtime load balancing. The node and VIP events

require Oracle Grid Infrastructure. The other events are provided by both Grid Infrastructure and GDS. This means that you can use Oracle RAC, Oracle RAC One Node, Oracle Restart and Oracle Data Guard with Grid Infrastructure for all event types, and GDS across data centers.

Table 2 – FAN High Availability Event Types

Event Type	Description
`event_type=NODE`	Oracle cluster node or network event
`event_type=INSTANCE`	Oracle Database Instance event
`event_type=DATABASE`	Oracle Database event
`event_type=SERVICEMEMBER`	Application service on a particular instance event
`event_type=SERVICE`	Application service event

Table 3 – FAN High Availability Event Status

The *event status* for each of these managed resources include:

Event Status	Description
`status=up`	Resource has started
`status=down`	Resource has stopped
`status=nodedown`	Node or public network has gone offline
`status=not_restarting`	Resource has failed more than 3 times in 30 minutes so is not being restarted

Table 4 – FAN High Availability Event Reasons

The event status for each managed resource is associated with an *event reason*. The reason further describes what triggered the event:

Event Reason	Activity Type	Event triggered by
`reason=USER`	Planned	User-initiated commands by `srvctl`, `sqlplus, or gdsctl`
`reason=FAILURE`	Unplanned	A failure has been detected for that resource

reason=member_leave	Unplanned or Planned	A node has gone offline
reason=public_nw_down	Unplanned or Planned	Public network has gone offline
reason=BOOT	Unplanned	When a node starts, DATABASE, INSTANCE or SERVICEMEMBER events will start with reason=BOOT if they were online before the node left the cluster

Table 5 – FAN High Availability Event Payload Fields

Additional event payload fields further describe the unique resource whose status is being monitored and published. These additional fields include:

Event Resource Identifier	Description
VERSION=<n.n>	Event payload version (currently 1.0)
timestamp=<eventDate> <eventTime>	Server-side date and time when the event was detected
service=<serviceName.dbDomainName>	Fully-qualified name of the database service
database=<dbName>	Database unique name
instance=<SID>	Name of the database instance
host=<hostname>	Name of the cluster node (as returned by Grid Infrastructure)
card=<n>	Service membership cardinality
timezone=<+|- GMT>	Difference from GMT
incarn	Cluster Incarnation number. Only visible in NODE DOWN event for guaranteed ordering
vip_ips	The IP address(es) of any VIP(s) that has/have gone down due to a public network failure. Only visible in Node DOWN events with reason=public_nw_down.

Example FAN High Availability Events

The result is a FAN event with one of the following payload structures:

Node Events

Down

```
VERSION=1.0 event_type=NODE host=rachost_724 incarn=316152110 status=nodedown
reason=member_leave timestamp=2015-01-21 19:13:24 timezone=-08:00
```

Public Network Down

```
VERSION=1.0 event_type=NODE host=rachost_724 incarn=0 status=nodedown
reason=public_nw_down vip_ips=10.10.8.249 timestamp=2015-01-21 19:13:13 timezone=-
08:00
```

Note: `incarn` is always 0 when the `reason=public_nw_down`

Instance Events

Instance Down

```
VERSION=1.0 event_type=INSTANCE service=testy.us.oracle.com instance=TESTY_2
database=testy db_domain=us.oracle.com host=rachost_723 status=down reason=FAILURE
timestamp=2015-01-21 19:32:43 timezone=-08:00
```

Instance Up

```
VERSION=1.0 event_type=INSTANCE service=testy.us.oracle.com instance=TESTY_2
database=testy db_domain=us.oracle.com host=rachost_723 status=up reason=FAILURE
timestamp=2015-01-21 19:33:10 timezone=-08:00
```

Service Events

Servicemember Down

```
VERSION=1.0 event_type=SERVICEMEMBER service=testy_pdb_srv.us.oracle.com
instance=TESTY_2 database=testy db_domain=us.oracle.com host=rachost_723
status=down reason=FAILURE timestamp=2015-01-21 19:32:43 timezone=-08:00
```

Servicemember Up

```
VERSION=1.0 event_type=SERVICEMEMBER service=testy_pdb_srv.us.oracle.com
instance=TESTY_2 database=testy db_domain=us.oracle.com host=rachost_723 status=up
card=3 reason=BOOT timestamp=2015-01-21 19:33:15 timezone=-08:00
```

Service Down

```
VERSION=1.0 event_type=SERVICE service=testy_pdb_srv.us.oracle.com database=testy
db_domain=us.oracle.com host=rachost_723 status=down reason=USER timestamp=2015-
01-21 19:25:52 timezone=-08:00
```

Service Up

```
VERSION=1.0 event_type=SERVICE service=testy_pdb_srv.us.oracle.com database=testy
db_domain=us.oracle.com host=rachost_723 status=up reason=USER timestamp=2015-01-
22 17:57:13 timezone=-08:00
```

Database Events

Database Down

```
VERSION=1.0 event_type=DATABASE service=testy.us.oracle.com database=testy
db_domain=us.oracle.com host=rachost_725 status=down reason=USER timestamp=2015-
03-24 20:09:17 timezone=-08:00
```

Database Up

```
VERSION=1.0 event_type=DATABASE service=testy.us.oracle.com database=testy
db_domain=us.oracle.com host=rachost_725 status=up reason=USER timestamp=2015-03-
24 21:09:27 timezone=-08:00
```

FAN Load Balancing Advisory Events

Runtime Load Balancing events have been available since Oracle Database 10g Release 2. These events guide subscribers as to where to place incoming load. By enabling runtime load balancing at the service level, FAN assists the application servers in directing work requests to where the request can be best satisfied based on the current response times, throughput and capacity. Runtime load balancing works together with affinity advice to keep requests in the same conversation together and to move work away from slow, hung and unresponsive nodes. Combined with affinity advice, runtime load balancing's advice purpose is predictable response time or throughput across the system, based on the service goal.

Runtime Load Balancing is provided out of the box for WebLogic Server Active GridLink, Oracle JDBC Universal Connection Pool, ODP.NET unmanaged and managed providers, PHP, and OCI Session Pool. All that is needed is to enable Runtime Load Balancing for the service, and to enable FAN and for ODP.NET, load balancing at the client. Load Balancing Advisory events are posted over ONS for all 12c clients, and over Advanced Queueing (AQ) for pre-12c OCI clients and all clients using pre-12c databases.

Table 5 – FAN Runtime Load Balancing Advisory Event

Event Resource Identifier	Description
Version	Version of the event payload.
Event type	SERVICE_METRICS
Service	Matches the service in DBA_SERVICES.
Database unique name	The unique database supporting the service. Matches the initialization parameter value for db_unique_name, which defaults to the value of the initialization parameter DB_NAME.
Timestamp	Date and time stamp (local time zone) to order events
Repeated for each event	
Instance	The name of the instance supporting the service. Matches the ORACLE_SID
Percent	The percentage of work requests to send to this database and instance
Service Quality	Weighted moving average of the service quality and bandwidth, based on the goal

	for the service (elapsed time or throughput)
Flag	Indication of the service quality relative to the service goal – Values are GOOD, VIOLATING, NO DATA, UNKNOWN Note: flag=UNKNOWN will occur if application work is not active on the instance (for the listed service) flag=NO DATA will display if the instance misses consecutive updates on load data (for the listed service)

Load Balancing Advisory Example

```
VERSION=1.0 database=TESTY service=testy_pdb_srv.us.oracle.com {
{instance=TESTY_2 percent=34 flag=UNKNOWN aff=FALSE}{instance=TESTY_3
percent=33 flag=UNKNOWN aff=FALSE}{instance=TESTY_1 percent=33 flag=UNKNOWN
aff=FALSE} } timestamp=2015-01-22 18:41:41
```

Viewing FAN events

FAN events are posted automatically when using 12c Grid Infrastructure. When using an Oracle Database 12*c* client, there is no further configuration other than enabling FAN at the client.

As an administrator, you may wish to see the FAN events flowing. It's also valuable to keep a record of FAN high availability events posted by your system. FAN callouts can provide an audit trail and timing information for what events occurred and when.

A sample callout to log FAN events on a Linux/UNIX system could look like the following:

```
#!/bin/bash
echo "`date` : $@" >> [your_path]/admin/`hostname`/callout_log.log
```

Refer to the section *Server-side Fast Application Notification Callouts* for more information on callouts.

`fanWatcher`

`fanWatcher` is a java program that connects to an ONS (Oracle Notification Service) daemon and subscribes to FAN events. The utility displays a portion of the FAN event header information as well as the event payload.

The source code for fanWatcher can be obtained from the *The WebLogic Server Blog* written by Stephen Felts at https://blogs.oracle.com/WebLogicServer/entry/fanwatcher_sample_program . A copy is included as an Appendix to this paper.

`fanWatcher` can run on an Oracle client installation, on any node in a Grid Infrastructure cluster, or on any mid-tier node that has an ONS daemon installed (for example, a node supporting WebLogic Server Active Gridlink for RAC)[2].

[2] Version 1.5 of FANwatcher does not support Auto-config of ONS nor remoteONS.

If you are using Oracle Database Release 12c Releas 1 (12.1.0.2), or Oracle WebLogic Server 12.1.3 and have compiled the `fanWatcher.java` sample code with the database jar files `ons.jar` and `ojdbcN.jar` that are provided with these releases, the utility can take advantage of the ONS auto-configuration capabilities offered by Oracle RAC. For auto-configuration of ONS to

The ONS daemon that is being subscribed to must be configured such that it is in communication with the ONS daemon running as part of Grid Infrastructure as the Grid Infrastructure ONS daemon is the one propagating Oracle events. Refer to the section in this paper on *ONS Configuration* for more information. The Grid Infrastructure may be local or remote, and the utility can be used whether your Oracle clients are using JDBC, OCI or ODP.Net managed and unmanaged providers.

Installing FANwatcher

Obtain a copy of the fanWatcher source code (from The WebLogic Server Blog or from the Appendix in this paper).

On a Linux/Unix machine: create a file named `fanWatcher.java` which contains the fanWatcher source code..

Save this file and compile the `fanWatcher.java` program. The CLASSPATH has to include the `ons.jar` and `ojdbcX`.jar file distributed with Oracle Database, Grid Infrastructure or Oracle WebLogic Server. `ojdbc8.jar` is the one supplied with Oracle Database 12*c*.

This example shows compiling and running the `fanWatcher` program on the server hosting Oracle Grid Infrastructure

```
$ vi fanWatcher.java
    add source code to this file

$ export CLASSPATH=Grid_Home/jdbc/lib/ojdbc8.jar:Grid_Home/opmn/lib/ons.jar:.

$ Grid_Home/jdk/bin/javac fanWatcher.java
Note: fanWatcher.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

$ Grid_Home/jdk/jre/bin/java –Doracle.ons.oraclehome=Grid_Home fanWatcher crs
```

The directory path described by the ”Grid_Home” designation in the example variable is that specified as ORACLE_HOME when installing an Oracle client, or Grid Infrastructure, or Oracle WebLogic server. The `fanWatcher` program is looking for the pathname `$ORACLE_HOME/opmn/conf/ons.config` used by the ONS daemon to start.

If you have installed `fanWatcher` on a WebLogic Server (WLS) midtier the WLS environment, including CLASSPATH, is set by `setWLSEnv` and only has to have the local directory (where fanWatcher is installed) included:

```
$ WLS_HOME/bin/setWLSEnv

$ export CLASSPATH=$CLASSPATH:.

$ javac fanWatcher.java
Note: fanWatcher.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

$ java fanWatcher ”nodes=racNode1:6200,racNode2:6200”
```

`fanWatcher` must be run as a user that has privilege to access the Oracle home specified above. This privilege correlates to the user who installed the Oracle client, or Grid Infrastructure (for example, `oracle`) or a member of the group to which this user belongs (for example, `oinstall`).

Using the fanWatcher program

The arguments passed to fanWatcher are dictated by the environment in which it is installed. The generic format for the command line is:

```
<Path to JRE>/bin/java [options] fanWatcher <config type> [event type]
```

where:

options is typically:

- `-Doracle.ons.oraclehome=<PATH to Grid Home>`
    - For example

`-Doracle.ons.oraclehome=/u01/app/oracle/product/12.1/grid`

- -classpath <list of class directories>
    - Can be set as the UNIX environment variable CLASSPATH or on the command line as in the following example

-classpath /u01/app/fanWatcher:`/u01/app/oracle/product/12.1/grid/jdbc/lib/ojdbc8.jar:/u01/app/oracle/product/12.1/grid/opmn/lib/ons.jar:.`

`event type`

- Optional. If omitted ALL events are displayed. If set, the subscriber will only return limited events. This is a very simple example, refer to the WebLogic Server Blog for a more detailed discussion:

`%"eventType=database/event/servicemetrics/<serviceName> "`

To run on the Grid Infrastructure server:

```
<Path to JRE>/bin/java –Doracle.ons.oraclehome=Grid_Home fanWatcher crs
```

where

Grid_Home is the install directory from which the ONS daemon is running. The Grid Home in 12*c* or the ORACLE_HOME for a 10.2 Oracle RAC environment, or the client ORACLE_HOME has a local ONS daemon running

To run on a WebLogic Server using an explicit node list

```
# Set the WLS environment using wlserver*/server/bin/setWLSEnv

$ export CLASSPATH="$CLASSPATH:."

$ /u01/app/12.1/jdk/jre/java –
Doracle.ons.oraclehome=/u01/app/12.1/grid fanWatcher
”nodes=racHost1:6200,racHost2:6200”
```

The node list is a string of one or more values of the form `name=value` separated by a newline character (\n).

This format is available for all versions of ONS. The following names may be specified.

`nodes` – This is required. The format is one or more `host:port` pairs separated by a comma.

`walletfile` – Oracle wallet file used for SSL communication with the ONS server.

`walletpassword` – Password to open the Oracle wallet file.

The node list will be extended in a later release of Oracle Database to allow for multiple unique topologies of remote ONS servers to be specified. This is not supported in Oracle Database 12*c* Release 1 (12.1.0.2).

Running with auto-configured ONS

Auto-configuration of ONS was released with Oracle Database 12*c* Release 1 (12.1.0.2) and only works for a RAC database running under Grid Infrastructure. Oracle Database Restart does not support auto-configuration of ONS.

Auto-configuration allows the client to obtain ONS information from the database server once a database session has been established. For the fanWatcher program to obtain this information it requires a `url` that points to a RAC database,and a `user` and `password` to connect to the database.

The `url`, `user`, and `password` can be specified as environment variables or the `fanWatcher.java` program can be modified to include them.

For example:

```
$ export password=mypassword
$ export url='jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=\
(ADDRESS=(PROTOCOL=TCP)(HOST=racHost1)(PORT=1521))\
(ADDRESS=(PROTOCOL=TCP)(HOST=racHost2)(PORT=1521)))\
(CONNECT_DATA=(SERVICE_NAME=myServiceName)))'

$ export user=myuser

$ export CLASSPATH="$CLASSPATH:."

$ $ORACLE_HOME/jdk/jre/bin/java fanWatcher autoons
```

With fanWatcher running events would then be received, as shown by the following example:

Client Node	GI Node

```
$ /u01/app/12.1/jdk/jre/java -
Doracle.ons.oraclehome=/u01/app/12.1/grid fanWatcher crs
```

```
$ srvctl start service -d testy -s
testy_pdb_srv
```

```
** Event Header **
Notification Type: database/event/service
Delivery Time: Tue Jan 27 16:22:39 PST 2015
Generating Node: rac2.oracle.com
Event payload:
VERSION=1.0 event_type=SERVICEMEMBER service=testy_pdb_srv.us.oracle.com
instance=TESTY_3 database=testy db_domain=us.oracle.com host=rachost_725
status=up card=1 reason=USER timestamp=2015-01-27 16:22:39 timezone=-08:00

** Event Header **
Notification Type: database/event/service
Delivery Time: Tue Jan 27 16:22:39 PST 2015
Generating Node: rac2.oracle.com
Event payload:
VERSION=1.0 event_type=SERVICE service=testy_pdb_srv.us.oracle.com
database=testy db_domain=us.oracle.com host=rachost_725 status=up
reason=USER
timestamp=2015-01-27 16:22:39 timezone=-08:00

** Event Header **
Notification Type: database/event/service
Delivery Time: Tue Jan 27 16:22:40 PST 2015
Generating Node: rac2.oracle.com
Event payload:
VERSION=1.0 event_type=SERVICEMEMBER service=testy_pdb_srv.us.oracle.com
instance=TESTY_1 database=testy db_domain=us.oracle.com host=rachost_723
status=up card=2 reason=USER timestamp=2015-01-27 16:22:40 timezone=-08:00

** Event Header **
Notification Type: database/event/service
Delivery Time: Tue Jan 27 16:22:41 PST 2015
Generating Node: rac2.oracle.com
Event payload:
VERSION=1.0 event_type=SERVICEMEMBER service=testy_pdb_srv.us.oracle.com
instance=TESTY_2 database=testy db_domain=us.oracle.com host=rachost_724
status=up card=3 reason=USER timestamp=2015-01-27 16:22:40 timezone=-08:00
```

Server-side Fast Application Notification Callouts

FAN callouts provide a simple yet powerful integration mechanism available with RAC that can be deployed with minimal programmatic efforts. A FAN callout is a wrapper shell script or pre-compiled executable written in any programming language that is executed each time a FAN event occurs. The purpose of the FAN callout is for simple logging, filing tickets and taking external actions. The purpose of the callout is not for integrated client failover – The FAN client failover is Fast Connection Failover in the next section. With the exception of node and network events, a FAN callout executes for the FAN events that are generated locally to each node and thus only for actions affecting resources on that node.

Configuring the FAN callout directory

Each FAN event posted by Grid Infrastructure results in the execution of each callout deployed in the standard Grid Infrastructure callout directory (on Linux it is `${Your_GI_Home}/racg/usrco`).

The order in which these callouts are executed is non-deterministic, and Grid Infrastructure guarantees that all callouts are invoked once for each FAN event, in an asynchronous fashion. You can install as many callout scripts or programs as your system requires. FAN callouts whose executions must be performed in a particular order, must be invoked from the same callout. Carefully test each callout for execution performance and correctness before production deployment.

Note: Ensure that the callout directory has write permissions only to the system user who installed Grid Infrastructure, and that each callout executable or script contained therein has execute permissions only to the same Grid Infrastructure owner.

Implementing FAN callouts

Writing FAN callouts involves the following steps:

1. Parsing FAN payload
2. Filtering incoming FAN events
3. Executing event-handling programs

Parsing callout argument list

The first step in a FAN callout is the parsing of the FAN payload. As described in the FAN event taxonomy section, there is a set of payload arguments for each FAN event type.

The event type is displayed as the first argument in the FAN payload. For example:

```
SERVICEMEMBER VERSION=1.0 service=testy_pdb_srv.us.oracle.com database=testy
instance=TESTY_1 host=rachost_723 status=down reason=USER timestamp=2015-01-27
17:56:08 timezone=-08:00 db_domain=us.oracle.com
```

A BASH shell script to parse the event could be as follows:

```
#!/bin/bash
# Scan and parse HA event payload arguments:
#
NOTIFY_EVENTTYPE=$1 # Event type is handled differently
for ARGS in $*; do
PROPERTY=`echo $ARGS | $AWK -F"=" '{print $1}'`
VALUE=`echo $ARGS | $AWK -F"=" '{print $2}'`
```

```
case $PROPERTY in
VERSION|version) NOTIFY_VERSION=$VALUE ;;
SERVICE|service) NOTIFY_SERVICE=$VALUE ;;
DATABASE|database) NOTIFY_DATABASE=$VALUE ;;
INSTANCE|instance) NOTIFY_INSTANCE=$VALUE ;;
HOST|host) NOTIFY_HOST=$VALUE ;;
STATUS|status) NOTIFY_STATUS=$VALUE ;;
REASON|reason) NOTIFY_REASON=$VALUE ;;
CARD|card) NOTIFY_CARDINALITY=$VALUE ;;
VIP_IPS|vip_ips) NOTIFY_VIPS=$VALUE ;; #VIP_IPS for public_nw_down
TIMESTAMP|timestamp) NOTIFY_LOGDATE=$VALUE ;; # catch event date
TIMEZONE|timezone) NOTIFY_TZONE=$VALUE ;;
??:??:??) NOTIFY_LOGTIME=$PROPERTY ;; # catch event time (hh24:mi:ss)
esac
done
```

Filtering incoming events

You can filter the incoming events based on payload information. For example, you may decide to apply the following filtering criteria for each target event handler to be invoked from the callout:

Example Target Event Handler	Target Services to Filter	Event Type (and Status) to filter
Start or stop local applications	All	SERVICE (up or down) SERVICEMEMBER (up or down) NODE (node down)
Log a service request in help system	FIN_APAC, PROD	SERVICE (down) DATABASE (down) NODE (down)
Email or message IT Supervisor	All	NODE (nodedown) SERVICE (up or down)
Forward SNMP traps to remote management console	WEB_GL	all
Notify local vendor components	Custom remove applications	NODE (down)

Table 5 – Example filtering criteria for specific FAN event handlers

In the following BASH shell script example, a trouble ticket system (using the second filtering example in Table 5) is invoked only when the Oracle Grid Infrastructure framework posts a SERVICE, DATABASE or NODE event type, with status either "`down`", "`nodedown`", or "`public_nw_down`" and only for two application service names: `HQPROD` and `FIN_APAC`:

```
#!/bin/bash
# FAN events with the following conditions will be inserted
# into the critical trouble ticket system:
# NOTIFY_EVENTTYPE => SERVICE | DATABASE | NODE
# NOTIFY_STATUS => down | public_nw_down | nodedown
# NOTIFY_DATABASE => HQPROD | FIN_APAC
#
if ((( [ $NOTIFY_EVENTTYPE = "SERVICE" ] ||
[ $NOTIFY_EVENTTYPE = "DATABASE" ] || \
[ $NOTIFY_EVENTTYPE = "NODE" ] \
) && \
( [ $NOTIFY_STATUS = "down" ] || \
[ $NOTIFY_STATUS = "public_nw_down" ] || \
 [ $NOTIFY_STATUS = "nodedown " ] \
)) && \
( [ $NOTIFY_DATABASE = "HQPROD" ] || \
[ $NOTIFY_DATABASE = "FIN_APAC" ] \
))
then
<< CALL TROUBLE TICKET LOGGING PROGRAM AND PASS RELEVANT NOTIFY_*
ARGUMENTS >>
fi
```

This is an example of what can be done with callouts. Sample code is available through OTN and Oracle Support to perform many tasks: relocate services on a preferred node when an instance starts, remove client connections immediately when a service stops, and so forth.

An example of a FAN callout that would log all events generated on a given node could be constructed by:

1. Create a file in the callout directory:

    a. `vi $ORA_CRS_HOME/racg/usrco/log_callouts.sh`

2. Add the following lines to the file (creating a BASH script):

    a.
    ```
    #!/bin/bash
    echo "`date` : $@" >> [your_path]/admin/`hostname`/callout_log.log
    ```

3. Set the execute permission on the file

    a. `chmod +x $ORA_CRS_HOME/racg/usrco/log_callouts.sh`

4. Copy this file to all nodes in the GI cluster and aggregate `callout_log.log` files to see all events on the system.

A sample `callout_log.log` entry may look like:

```
Tue Jan 31 17:56:08 PST 2015: SERVICEMEMBER VERSION=1.0
service=testy_pdb_srv.us.oracle.com database=testy instance=TESTY_1 host=rachost_723
status=down reason=USER timestamp=2015-01-27 17:56:08 timezone=-08:00
db_domain=us.oracle.com
```

More information on FAN and FAN Callouts can be found in Chapter 5 Workload Management with Dynamic Database Services of the *Oracle® Real Application Clusters Administration and Deployment Guide 12c Release 1 (12.1),* Part Number E48838-09

Configuring Applications to use FAN with Fast Connection Failover

Fast Connection Failover (FCF) is the pre-configured and proven FAN client-side integration. There is no need whatsoever to modify application code to receive and use FAN events. You should always use an FCF-capable client.

FCF is the FAN client solution for all application stacks, whether Oracle or third party. An FCF client automatically subscribes to and acts on FAN events. FCF clients understand which database sessions are impacted by an event, by using the FAN connection signature shown in Table 1. By matching the payload in the FAN event with that in the connection signatures, FCF clients know which sessions to act on when a FAN event is received. FCF operates as follows based on the event type (table 2), event status (table 3), and reason (table 4):

Down – Events with `status=down` and `reason=FAILURE` are caused by failures at the database server. The FCF client aborts the related connections immediately so that the application does not hang on TCP/IP timeouts, but rather receives an interrupt fast. The dead connections are removed from pooled FCF clients. When the application is configured to use Application Continuity or TAF, the application is automatically failed over.

Planned Down – Events with `status=down` and `reason=PLANNED` are posted when the DBA stops an instance or service or database to start planned maintenance using `srvctl` or `gdsctl or Data Guard Broker`. The FCF client drains the sessions ahead of the planned maintenance by allowing the active work to complete and then closing the session. For planned draining by FAN and FCF, there is no application interruption whatsoever. For Oracle Database *12c* UCP and JDBC clients, the system property `oracle.ucp.PlannedDrainingPeriod` allows for a time period to be set over which the draining occurs.

Up – Events with `status=up` are posted when a service starts for the first time or resumes. The FCF client re-allocates the sessions so that load is balanced across the database server. Rebalancing is a gradual process so as not to interrupt performance.

Load % - When runtime load balancing is enabled, the FAN events carry an advised percentage to distribute load across the service. The FCF client uses this advice to balance sessions locally when using RAC and globally when using GDS.

Affinity - When runtime load balancing is enabled, the FAN events carry advice as to when to keep the client conversation locality. This advice is applied for repeated borrows and returns from the same client.

Load % and Affinity are applicable to pooled FCF clients (see Table 7). All other actions apply to all FCF clients

FCF is provided with the following clients beginning with the releases listed in Table 7. Table 7 also shows the transport over which FAN is received by release.

Table 6. Fast Connection Failover by Oracle Database Release

	Database Version		
FCF Client	10g	11g	12*c*
JDBC Implicit Connection Cache (ICC)	ONS	ONS	ICC deprecated
JDBC Universal Connection Pool (UCP)		ONS	ONS
WebLogic Server Active GridLink		ONS	ONS
3rd Party Application Servers using UCP: Apache TomCat, IBM WebSphere		ONS	ONS
ODP.Net Unmanaged (OCI)	AQ	AQ	ONS
ODP.Net Managed (C#)	ONS	ONS	ONS
OCI Session Pool	AQ	AQ	ONS
PHP	AQ	AQ	ONS
SQL*Plus	AQ	AQ	ONS
Tuxedo		ONS	ONS
JDBC Thin Standalone Client		ONS	ONS
OCI/OCCI Driver	AQ	AQ	ONS
Net and SCAN Listeners	ONS	ONS	ONS

Beginning with Oracle Database 12c Release 1 (12.1), ONS is the transport when using an Oracle Database 12c Release 1 (12.1) and a server Oracle Database 12c Release 1 (12.1) FAN AQ HA notification feature is deprecated and maintained only for backward compatibility when there is an old client (pre-Oracle Database 12c Release 1 (12.1)) or old server (pre-Oracle Database 12c Release 1 (12.1)).

How to Configure the Oracle Notification Service for 12c Grid Infrastructure

FAN uses the Oracle Notification Service (ONS) for event propagation to all clients from Oracle Database 12*c* onwards and for JDBC, Tuxedo and listener clients before 12c. ONS is installed as part of Oracle Grid Infrastructure on a cluster, in an Oracle Data Guard installation, and when Oracle WebLogic is installed. ONS is responsible for propagating FAN events to all other ONS daemons it is registered with. ONS comes pre-configured and enabled with Oracle Grid Infrastructure. It is not necessary to manually configure ONS for it to operate.

Grid Intrastructure comes pre-configured with FAN. No steps are needed to configure or enable FAN at the server-side with one small exception: OCI FAN and ODP FAN require `-notification` is set to `TRUE` for the service by `srvctl` or `gdsctl`. For variations on the server side, refer to the appendix.

For FCF clients, ONS is installed as part of WebLogic Server, and as part of the Oracle client installation. With FAN auto-configuration at the client, ONS must be on the CLASSPATH or in the ORACLE_HOME dependent on your client. Transparently, the FCF client spawns an ONS thread to subscribe to interested FAN events.

Refer to the section *Appendix A: Configuring ONS* in this paper for more information on ONS configuration.

General Steps for Configuring FCF Clients

Follow these steps before progressing to driver specific instructions:

- Use a dynamic database service
- Use the Oracle notification service
- How many ONS connections are needed?
- How to pass ONS through Firewalls
- Use a NET connection that provides high availability
- Enable connection checking

Use a Dynamic Database Service

Using FAN requires that the application connects to the database using a dynamic database service. This is a service created using srvctl, if RAC, or gdsctl, if using global database services. Do not connect using the database service or PDB service – these are for administration only and are not supported for FAN. The TNSnames entry or URL must use the service name syntax and follow best practice by specifying a dynamic database service name. Refer to the examples later in this section.

Use the Oracle Notification Service

When you use FAN with JDBC thin or Tuxedo or Oracle Database 12c Release 1 (12.1.0.1) OCI or ODP.Net clients, FAN is received over ONS. When you use Oracle Database10g or Oracle Database 11g OCI or ODP.NET unmanaged provider FCF clients, FAN is received over AQ.

The target for FAN clients of Oracle Database 12*c* and later is no configuration. Accordingly, in Oracle Database 12*c* ONS FAN auto-configuration is introduced such that FCF clients discover the server-side ONS networks and self configure. FAN is automatically enabled when ONS libraries or jars are present. In Oracle Database 12*c*, it is still necessary to enable FAN on most FCF clients. Listeners and SQL*Plus require no client-side configuration.

FAN auto-configuration removes the need to list the Grid Infrastructure servers that an FCF client needs. Listing server hosts is incompatible with location transparency and causes issues with updating clients when the server configuration changes. Clients already use a TNS address string or URL to locate the remote listeners. FAN auto-configuration uses the TNS addresses to locate the databases and then asks each server database for the ONS server-side addresses. When there is more than one database, for example, FAN auto-configuration contacts each and obtains an ONS configuration for each one.

When using Oracle Database 12*c*, the ONS network is discovered from the URL. An ONS node group is automatically obtained for each address list when `LOAD_BALANCE` is off across the address lists. This is the default shown here. If `LOAD_BALANCE` is TRUE or ON across all address lists, only one set of ONS end points is created. (Note that this is the LOAD_BALANCE value across the `ADDRESS_LIST`. The `LOAD_BALANCE` inside each address list is to expand the SCAN address.)

How many ONS connections are needed?

By default the FCF client maintains three hosts for redundancy in each node group in the ONS configuration.. Each node group corresponds to each Grid Infrastructure cluster or each GDS data center. For example, if there is a primary database and several Oracle Data Guard standbys there is by default, 3 ONS connections maintained at each node group. The node groups are discovered when using FAN auto-configuration, or if before Oracle Database 12*c* use `ons.configuration`. With node_groups defined by FANauto-configuration, `load_balance=false` (the default), more ONS end points are not required. If you want to increase the number of end points you can do this by increasing `maxconnections`. This applies to each node group. Increasing to 4 in this example, maintains four ONS connections at each node. Increasing this value consumes more sockets.

```
oracle.ons.maxconnections=4
```

ONS Node groups: `oracle.ons.nodes`

If the client is to connect to multiple clusters, and receive FAN events from both, for example in the RAC with Data Guard situation, then multiple ONS node groups are needed. FAN auto-configuration creates these node groups using the URL or TNS names for 12*c* client and 12*c* database. If not using auto-ons, specify the node groups in the `ig` or `oraaccess.xml` configuration files.

Consider the situation with two clusters of 8 nodes each. Specify two node lists (one for each cluster):

```
AUTOONS creates a separate node group for each addresslist in the TNS connection
descriptor:

oracle.ons.nodes.001=node1a:6250,node1b:6250,node1c:6250,node1d:6250,node1e:6250,nod
e1f:6250,node1g:6250,node1h:6250

oracle.ons.nodes.002=node2a:6250,node2b:6250,node2c:6250,node2d:6250,node2e:6250,nod
e2f:6250,node2g:6250,node2h:6250
```

Leaving `oracle.ons.maxconnections` at the default of 3, for every active node list, results in the ONS client trying to maintain 6 total connections in this case.

How to pass ONS through Firewalls

When a firewall is present between the ONS node groups and the FCF client, a port needs to be opened to ons traffic. The recommended approaches are –

1) Add `ons.exe` to the firewall exceptions list
2) Manually set the port to use in the ONS configuration, then add the port to the firewall's exceptions list

Use a NET Connection that provides High Availability

For OCI and ODP.NET unmanaged provider clients use the following TNS names structure:

```
(DESCRIPTION = (CONNECT_TIMEOUT=90) (RETRY_COUNT=30)(RETRY_DELAY=3)
(TRANSPORT_CONNECT_TIMEOUT=30)
(ADDRESS_LIST =
   (LOAD_BALANCE=on)
   ( ADDRESS = (PROTOCOL = TCP)(HOST=primary-SCAN)(PORT=1521)))
(ADDRESS_LIST =
   (LOAD_BALANCE=on)
   ( ADDRESS = (PROTOCOL = TCP)(HOST=secondary-SCAN)(PORT=1521)))
(CONNECT_DATA=(SERVICE_NAME = gold-cloud)))
```

For JDBC thin clients use the following URL structure through to 12.1.0.2:

```
jdbc:oracle:thin = (CONNECT_TIMEOUT=4) (RETRY_COUNT=30)(RETRY_DELAY=3)
   (ADDRESS_LIST =
     (LOAD_BALANCE=on)
     ( ADDRESS = (PROTOCOL = TCP)(HOST=primary-SCAN)(PORT=1521)))
   (ADDRESS_LIST =
      (LOAD_BALANCE=on)
      ( ADDRESS = (PROTOCOL = TCP)(HOST=secondary-SCAN)(PORT=1521)))
 (CONNECT_DATA=(SERVICE_NAME = gold-cloud)))
```

Note that after Oracle Database 12*c* Release 1 (12.1.0.2), JDBC and OCI align and you should use the OCI version for all connection descriptions.

Best Practices for the NET Connections

ALWAYS use dynamic database services to connect to the database. These are created using `srvctl` or `gdsctl`.

Do not use the database service or PDB service – these are for administration only, not for application usage and do not provide FAN and many other features because they are available at mount.

For JDBC, use the current client driver (Oracle Database 12*c* Release 1) with current or older RDBMS

Use one `DESCRIPTION` in the TNS names entry or URL – using more causes long delays connecting when RETRY_COUNT and RETRY_DELAY are used.

Set `CONNECT_TIMEOUT=90` or higher to prevent logon storms for OCI and ODP clients. Use a lower setting for JDBC clients, `CONNECT_TIMEOUT=4`, as a temporary measure until `TRANSPORT_CONNECT_TIMEOUT` is available.

Do not also set JDBC property `oracle.net.ns.SQLnetDef.TCP_CONNTIMEOUT_STR` as it overrides `CONNECT_TIMEOUT`

Set `LOAD_BALANCE=ON` per address to expand SCAN names starting with Oracle Database 11g Release 2 (11.2.0.3) for OCI and Oracle Database 12*c* Release 1 (12.1.0.2) for JDBC

Do not use Easy*Connect syntax (EZConnect) – it has no High Availability capabilities.

Enable Connection Checks

Depending on the outage, applications may receive stale connections when connections are borrowed before 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(boolean) – Specifies whether or not connections are validated when the connection is borrowed from the connection pool. The method enables validation for every connection that is borrowed from the pool. The default value is false. Set the value to true so validation is performed.

OCI Connections

To verify that the connection to the server is terminated by either FAN or an OCI_ERROR, an application can check the value of the attribute OCI_ATTR_SERVER_STATUS in the server handle. If the value of the attribute is OCI_SERVER_NOT_CONNECTED, then the connection to the server has been terminated and the user session should be reestablished.

OCI_ATTR_SERVER_STATUS is set to OCI_SERVER_NOT_CONNECTED for both FAN unplanned down and FAN planned down use cases. When using FAN to report unplanned down, the application receives an error immediately. When using FAN to report planned maintenance, a custom OCI pool can check OCI_ATTR_SERVER_STATUS before borrow and after return to the pool, and drop the session at only these safe places only. Dropping closed connections before borrow and after return leads to a good user experience with no application errors received during planned maintenance.

ODP.NET Provider

CheckConStatus is on by default.

This property checks the status of the connection before putting the connection back into the ODP.NET connection pool. This registry entry is not created by the installation of ODP.NET. However, the default value 1 is used.

For custom ODP.NET programs, test ConnectionState.Open. For example -

```
if(OracleConnection.State!=ConnectionState.Open)
    OracleConnection.Open()
```

How to Configure FAN for 12c Java Clients

Using Universal Connection Pool

The best way to take advantage of FCF with the Oracle Database JDBC thin driver is to use either the Universal Connection Pool (UCP) or WebLogic Server Active GridLink. Setting the pool property `FastConnectionFailoverEnabled` on the Universal Connection Pool enables Fast Connection Failover (FCF). Active GridLink always has FCF enabled by default. Third party application servers including IBM WebSphere and Apache Tomcat support UCP as a connection pool replacement. For more information on embedding UCP with other web servers refer to these white papers:

*Design and Deploy WebSphere Applications for Planned, Unplanned Database Downtimes and Runtime Load Balancing with UCP (*http://www.oracle.com/technetwork/database/application-development/planned-unplanned-rlb-ucp-websphere-2409214.pdf) and

*Design and deploy Tomcat Applications for Planned, Unplanned Database Downtimes and Runtime Load Balancing with UCP (*http://www.oracle.com/technetwork/database/application-development/planned-unplanned-rlb-ucp-tomcat-2265175.pdf).

Follow these configuration steps to enable Fast Connection Failover:

1. The connection URL of a connection factory must use the service name syntax and follow best practice by specifying a dynamic database service name and the JDBC URL structure (above and also below). All other URL format do not provide high availability. The URL may use JDBC thin or JDBC OCI.
2. If wallet authentication has not previously been established or the cluster is running a version earlier than Oracle Grid Infrastructure 12.1.0.2 then remote ONS configuration is needed. This can be done through the pool property `setONSConfiguration` which can be set through a property file as shown in the following example:

```
PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();

pds.setConnectionPoolName("FCFSamplePool");
pds.setFastConnectionFailoverEnabled(true);
pds.setONSConfiguration("propertiesfile=/usr/ons/ons.properties");
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL("jdbc:oracle:thin@((CONNECT_TIMEOUT=4)(RETRY_COUNT=30)(RETRY_DELAY=3) "+
           " (ADDRESS_LIST = "+
             " (LOAD_BALANCE=on) "+
             " ( ADDRESS = (PROTOCOL = TCP)(HOST=RAC-SCAN)(PORT=1521))) "+
           " (ADDRESS_LIST = "+
              " (LOAD_BALANCE=on) "+
              "( ADDRESS = (PROTOCOL = TCP)(HOST=DG-SCAN)(PORT=1521)))"+
         "(CONNECT_DATA=(SERVICE_NAME=service_name)))");
```

The property file specified must contain an `oracle.ons.nodes` property and optionally, properties for `oracle.ons.walletfile` and `oracle.ons.walletpassword`. An example of an `ons.properties` file is shown here:

```
oracle.ons.nodes=racnode1:4200,racnode2:4200
oracle.ons.walletfile=/oracle12/onswalletfile
```

3. Ensure the pool property `setFastConnectionFailoverEnabled=true` is set.
4. The `CLASSPATH` must contain `ons.jar`, `ucp.jar`, and the jdbc driver jar file, for example `ojdbc7.jar`.
5. The following is not required and is an optimization to force re-ordering of the SCAN IP addresses returned from DNS for a SCAN address:

```
oracle.jdbc.thinForceDNSLoadBalancing=true
```

6. If you are using JDBC thin with Oracle Database 12c Application Continuity can be configured to failover the connections after FAN is received.
7. If the database is earlier than 12c or if the configuration needs different ONS end points to those auto-configured, the ONS end points can be enabled as per the following example:

```
pds.setONSConfiguration("nodes=mysun05:6200,mysun06:6200,
mysun07:6200,mysun08:6200");
```

Or in the situation with multiple clusters and when using autoons – autoons would gerate node lists similar to the following -

```
oracle.ons.nodes.001=node1a:6250,node1b:6250,node1c:6250,node1d:6250,node1e:6250,nod
e1f:6250,node1g:6250,node1h:6250

oracle.ons.nodes.002=node2a:6250,node2b:6250,node2c:6250,node2d:6250,node2e:6250,nod
e2f:6250,node2g:6250,node2h:6250
```

`oracle.ons.maxconnections` is set to 3 by default for EVERY active nodelist, so there is no need to explicitly set this. This example will result in the ONS client trying to maintain 6 total connections.

How to Configure FAN for ODP.Net Clients, Managed and Unmanaged Providers

To take advantage of Fast Connection Failover (FCF) with Oracle Database unmanaged provider or ODP.NET managed provider, the application uses the ODP.Net Connection Pool. FCF is supported for the connection pools and for Oracle Services for Microsoft Transaction Server (OraMTS).

Configuration for Oracle Database 12*c* ODP.Net Unmanaged and Managed Provider Clients with 12c Database Server

1. To enable FAN with ODP.NET at the client side, specify HA events in the connect string:

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

2. To enable Runtime Load balancing with ODP.NET at the client side, also specify load balancing in the connect string:

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

3. ODP.Net needs to be able to find the ONS listeners. Using 12c FAN autoons, ONS configuration information is passed directly to ODP.Net from the database server itself. Autoons uses the TNSnames to find the end points. Set the OCI TNS address best practice with full high availability capabilities

```
myAlias=(DESCRIPTION=
            (CONNECT_TIMEOUT=90)(RETRY_COUNT=30)(RETRY_DELAY=3)
            (TRANSPORT_CONNECT_TIMEOUT=3)
          (ADDRESS_LIST=(LOAD_BALANCE=ON)
            (ADDRESS=(PROTOCOL=TCP)(HOST=RAC-scan)(PORT=1521)))
          (ADDRESS_LIST=(LOAD_BALANCE=ON)
            (ADDRESS=(PROTOCOL=TCP)(HOST=DG-Scan)(PORT=1521)))
         (CONNECT_DATA=(SERVICE_NAME=service_name)))
```

Note that Transparent Application Failover (TAF) can be used with FCF and ODP.Net unmanaged provider. After FAN has interrupted the session, TAF will execute the failover using the `FAILOVER_TYPE` specified – `BASIC` or `SELECT`. Application Continuity will be available in a future release.

Which Steps Differ if Using ODP.Net UnManaged Provider 11g or ODP.Net UnManaged Provider with Oracle Database Release 10g or Release 11g?

When using an earlier database or client version, than Oracle Database 12*c* with unmanaged provider the steps are the same as 12c client and 12c database with one exception. FAN events are delivered over AQ, which requires `notification` to be set on the database service, similar to the following example:

```
srvctl modify service -db EMEA -service GOLD -notification TRUE
```

Which steps differ if you want to custom configure ONS end points with 12c:

For a customized ONS configuration

Edit `oraaccess.xml` located in `$ORACLE_HOME/network/admin`.

```
<onsConfig mode=”remote”>
   <ons database=”db1”>
      <add name=”nodeList” value=”racnode1:6700, racnode2:6700” />
   </ons>
   <ons database=”db2”>
      <add name=”nodeList” value=”racnode3:6700, racnode3:6700” />
   </ons>
</onsConfig>
```

In case of custom ONS configuration, the application specifies the <host>:<port> values for every potential database that it can connect to. The <host>:<port> value pairs represent the ports on the the different Oracle RAC nodes where the ONS daemons are talking to their remote clients.

Additional information on configuring ONS is available in the Appendix attached to this paper.

Refer to the following for FAN and ODP.Net Provider Oracle® Data Provider for .NET Developer's Guide 12c Release 1 (12.1) Part Number E17732-11: Real Application Clusters and Global Data Services.

How to Configure FAN for OCI Clients

OCI clients embed FAN at the driver level so that all clients can use them regardless of the pooling solution. Starting with Oracle database and client 12*c*, OCI clients use ONS for the FAN transport. Both the server and the client must use version 12*c*. If either the client or server is using version 11.2 or earlier, then the FAN transport used is advanced queues.

Configuration for SQL*Plus and PHP

1. Set notification for the service

```
srvctl modify service -db EMEA -service GOLD -notification TRUE
```

2. For PHP clients only, add oci8.events=On to php.ini:

```
php.ini:

oci8.events=on
```

Important

If oraccess.xml is present with events=-false or events not specified, this disables the usage of FAN. To maintain FAN with SQL*Plus and PHP when oraccess.xml is in use, set events=-true

3. On the client side and using a 12*c* client and Oracle Database 12*c* database, enable FAN in `oraaccess.xml`.

```
<oraaccess> xmlns="http://xmlns.oracle.com/oci/oraaccess"
    xmlns:oci="http://xmlns.oracle.com/oci/oraaccess"
    schemaLocation="http://xmlns.oracle.com/oci/oraaccess
    http://xmlns.oracle.com/oci/oraaccess.xsd">
  <default_parameters>
    <events>true</events>
  </default_parameters>
</oraaccess>
```

Configuration for 12c OCI Clients when using 12c Database Server

1. Tell OCI where to find ONS Listeners

Starting with 12*c*, the client install comes with ONS linked into the client library. Using ons auto-config, the ONS end points are discovered from the TNS address. This automatic method is the recommended approach. Like ODP.Net, manual ONS configuration is also supported using `oraaccess.xml`.

2. Enable FAN high availability events for the OCI connections

To enable FAN requires editing the OCI file `oraaccess.xml` specify the global parameter `events`. This file is located in `$ORACLE_HOME/network/admin`:

```
<oraaccess> xmlns="http://xmlns.oracle.com/oci/oraaccess"
    xmlns:oci="http://xmlns.oracle.com/oci/oraaccess"
    schemaLocation="http://xmlns.oracle.com/oci/oraaccess
    http://xmlns.oracle.com/oci/oraaccess.xsd">
  <default_parameters>
    <events>true</events>
  </default_parameters>
</oraaccess>
```

3. Tell OCI where to find ONS Listeners

   Starting with 12*c*, the client install comes with ONS linked into the client library. Using ons auto-config, the ONS end points are discovered from the TNS address. This automatic method is the recommended approach. Like ODP.Net, manual ONS configuration is also supported using `oraaccess.xml`.

```
myAlias=(DESCRIPTION=
           (CONNECT_TIMEOUT=90)(RETRY_COUNT=30)(RETRY_DELAY=3)
           (TRANSPORT_CONNECT_TIMEOUT=3)
         (ADDRESS_LIST=(LOAD_BALANCE=ON)
           (ADDRESS=(PROTOCOL=TCP)(HOST=RAC-SCAN-address)(PORT=1521)))
         (ADDRESS_LIST=(LOAD_BALANCE=ON)
           (ADDRESS=(PROTOCOL=TCP)(HOST=DG-SCAN-address)(PORT=1521)))
         (CONNECT_DATA=(SERVICE_NAME=myServiceName)))
```

4. Enable FAN at the server for all OCI clients

   Continuing with Oracle Database 12*c*, it is still necessary to enable FAN at the database server for all OCI clients (including SQL*Plus).

```
srvctl modify service -db EMEA -service GOLD -notification TRUE
```

Which steps differ if you want to custom configure ONS end points with Oracle Database 12*c*

If you want to custom configure ONS, use syntax similar to the following example. This is not the recommended method.

```
<oraaccess xmlns="http://xmlns.oracle.com/oci/oraaccess"
  xmlns:oci="http://xmlns.oracle.com/oci/oraaccess"
  schemaLocation="http://xmlns.oracle.com/oci/oraaccess
  http://xmlns.oracle.com/oci/oraaccess.xsd">
  <default_parameters>
    <fan>
     <!-- only possible values are "trace" or "error"  -->
      <subscription_failure_action>
        error
      </subscription_failure_action>
    </fan>
    <ons>
      <subscription_wait_timeout>
        5
      </subscription_wait_timeout>
      <auto_config>true</auto_config>
      <!—The following provides the manual configuration for ONS.
          Note that this functionality should not need to be used
          as auto_config can normally discover this information. -->
      <servers>
        <address_list>
          <name>pacific</name>
          <max_connections>3</max_connections>
          <hosts>10.228.215.121:25293, 10.228.215.122:25293</hosts>
        </address_list>
        <address_list>
          <name>Europe</name>
          <max_connections>3</max_connections>
          <hosts>myhost1.mydomain.com:25273,
            myhost2.mydomain.com:25298,
            myhost3.mydomain.com:30004</hosts>
        </address_list>
      </servers>
    </ons>
    <events>true</events>
  </default_parameters>
</oraaccess>
```

OCI and Oracle Database 11g

If either the client or the database is using Oracle Database 11g or earlier, then FAN events are sent over Advanced Queues (AQ).

For an Oracle Database 11g application, the application itself must:

- Initialize the OCI Environment in OCI_EVENTS mode
- Link with a thread library
- Set –notification for the service using srvctl or gdsctl

```
srvctl modify service -db EMEA -service GOLD -notification TRUE
```

Refer to *Oracle® Call Interface Programmer's Guide 12c Release 1 (12.1) E49886-05, Chapter 10 More OCI Advanced Topics* for more information.

Conclusion

FAN - Tells the applications when actions are required

To achieve fast recovery and mitigate flow-on effects after a failure of the hardware or the software supporting the application session, the session must be interrupted immediately when a failure is detected.

For service configuration changes, Grid Infrastructure and GDS post FAN events immediately when a state change occurs for services in the system. Instead of waiting for the application server or driver to timeout and detect a problem, using FAN the application server or driver receives these events and acts immediately.

For unplanned down events, the disruption to the application is minimized as connections to the failed instance or node are terminated and sockets are closed. In-flight requests are terminated and the application user is notified immediately, or if using TAF for OCI or Application Continuity for Java, the user experiences a slight disruption while the session is re-established at a functioning service. Not-borrowed sessions are cleaned up immediately, and application users requesting connections are directed to instances offering functioning services only.

For planned down events the disruption to the application is minimized by dropping the session at a safe place when returned to the connection pool, where the application receives no error whatsoever. Same as unplanned, not-borrowed sessions are cleaned up immediately, and application users requesting connections are directed to instances offering functional services only.

For Up events, when services are started, new connections are created automatically so the application can immediately take advantage of the extra resources.

The FANwatcher utility described in this paper enables you to determine the ONS topology that has been constructed and whether an event can be received by a subscribing client.

Appendix A Configuring ONS

ONS is installed as part of Oracle Grid Infrastructure on a cluster, in an Oracle Data Guard installation, and when Oracle Web Logic Server Active Gridlink is installed. ONS is responsible for propagating FAN events to all other ONS daemons it is registered with. It is not necessary to manually configure ONS for it to operate as part of a Grid Infrastructure cluster. ONS will also be installed as part of an Oracle client installation and may also be spawned by a standalone client application through a process named Remote ONS configuration.

Oracle application clients will subscribe, through an ONS daemon, to events in which it, the client, is interested.

ONS Configuration File

The ONS configuration file, `$ORACLE_HOME/opmn/conf/ons.config`, defines how ONS behaves. Information in this file is stored as a set of key-value pairs as shown in Table 6.

Parameter	Definition
`localport`	The port number on the local host that ONS will use to talk to local clients. Default in Grid Infrastructure is `localport=6100`
`remoteport`	The port number used by ONS to talk to other ONS daemons. The default value in Grid Infrastructure is `remoteport=6200`
`nodes`	A comma-separated list of nodes and ports indicating other ONS daemons to talk to. The port value is the remoteport entry that each ONS daemon will be listening on. In a Grid Infrastructure configuration all nodes in the cluster will be named. An example would look like nodes=myhost1.example.com:6500,myhost2.example.com:6500,
`logcomp` (optional parameter)	An optional parameter that specifies what subcomponents to log.The format is `<component>[<subcomponent>,…]` Exclusion of subcomponents is also allowed by preceding the subcomponent by an exclamation mark (!). For example, to log `all` components, except for `secure` specify `logcomp=ons[all,!secure]`
`logfile` (optional parameter)	The location of the log file used for logging messages. The default value is `logfile=$ORACLE_HOME/opmn/logs/ons.log`
`walletfile (optional parameter)`	Specifies the wallet file used by the Oracle Secure Sockets Layer (SSL) to store SSL certificates. If a wallet file is specified to ONS, then it uses SSL when communicating with other ONS instances and require SSL certificate authentication from all ONS instances that try to connect to it. In 12.1.0.2 a Grid Infrastructure installation will have the

	walletfile set by default, thus enforcing the use of SSL for all ONS connections. Existing UCP users will need to verify that they are using a walletfile, and ensure the contents are the same as the wallet on the server.
useocr (optional GI-SERVER ONLY parameter)	This parameter is only to be used on a Grid Infrastructure node. It indicates whether ONS should store all GI configuration in the Oracle Cluster Registry (OCR). To store information in OCR use useocr=on
allowgroup (optional parameter)	Specifies the ONS setting to indicate the user group connecting to the localport. When set to true, ONS allows users within the same OS group to connect to its local port. When set to false, ONS only allows the user who started the ONS daemon to access its local port. The default value of this parameter is false. When using remote ONS configuration, there is no need to set this parameter.

Note that the modification of these parameters in a Grid Infrastructure installation is done using srvctl with the modify nodeapps command. The following help screen excerpt shows the relevant parameters:

```
$ srvctl modify nodeapps -h

Modifies the configuration for a node application.

Usage: srvctl modify nodeapps {[-node <node_name> -address
{<vip_name>|<ip>}/<netmask>[/if1[|if2...]] [-skip]] | [-subnet
<subnet>/<netmask>[/if1[|if2|...]]]} [-nettype {STATIC|DHCP|AUTOCONFIG|MIXED}]
[-emport <em_port>] [ -onslocalport <ons_local_port> ] [-onsremoteport
<ons_remote_port> ] [-remoteservers <host>[:<port>][,<host>[:<port>]...]]
[-clientdata <file>] [-pingtarget "<pingtarget_list>"] [-verbose]
    -node <node_name>              Node name
…
    -onslocalport  <ons_local_port> ONS listening port for local client
connections
    -onsremoteport <ons_remote_port> ONS listening port for connections from
remote hosts
    -remoteservers                 <host>[:<port>][,<host>[:<port>]...]  List
of remote host/port pairs for ONS daemons outside this cluster
…
    -clientdata <file>             file with wallet to import, or empty string
to delete wallet used for SSL to secure ONS communication
…
    -verbose                       Verbose output
    -help                          Print usage
```

Client-side ONS Configuration

CLIENT-SIDE ONS IS NOT RECOMMENDED PRACTICE. For applications, use AUTO-ONS or remote ONS.

For Oracle clients that require an ONS daemon to be running, it is necessary to edit the `ons.config` file directly and to then start the ONS daemon.

The default location for an Oracle Client installation is `$ORACLE_HOME/opmn/conf/ons.config`, although this may vary for other Oracle product installations.

A sample Oracle Client ons.config file could look like:

```
# This is an example ons.config file
#
# The first three values are required
localport=4100
remoteport=4200
nodes=racnode1.example.com:6200,racnode2.example.com:6200
```

As indicated in this example it is necessary to specify the cluster nodes on which RAC instances will run, so that this client will receive FAN events for which it is interested. It is not necessary to specify all of the cluster nodes, as ONS will discover daemons running within the topology it constructs. However if only one, or a subset of nodes is specified in the `nodes=…` list the risk is taken that this node may be down when the ONS topology is being discovered and constructed.

The `$ORACLE_HOME/opmn/bin/onsctl` utility can then be used to `start` the ONS daemon.

The `onsctl` command

`onsctl` can be used to start, shutdown, reconfigure and monitor the ONS daemon. Available command options to `onsctl` are:

Command	Action	Output
`start`	Start the ONS daemon	`onsctl: ons started`
`shutdown`	Stop the ONS daemon	`onsctl: shutting down ons daemon`
`reload`	Re-read the `ons.config` file and update settings (ONS daemon not stopped)	
`ping [max-retry]`	Verifies the local ONS daemon is running. Will attempt `max-retry` times	`ons is running`
`debug`	Print debug information for the local	

	ONS daemon.	
`usage`	Print a detailed help screen.	
`help`	Print simple usage information	

Note that ONS is managed as part of Grid Infrastructure if it is running in a cluster. The `srvctl` utility allows for ONS start and stop through the `nodeapps` option as shown by this help screen:

```
$ srvctl stop nodeapps -h

Start the node applications running on a node.

Usage: srvctl start nodeapps [-node <node_name>] [-adminhelper | -onsonly]
[-verbose]
    -node <node_name>              Node name
    -adminhelper                   Start Administrator helper only
    -onsonly                       Start ONS only
    -verbose                       Verbose output
    -help                          Print usage
```

Validating ONS Topology

The `onsctl debug` command produces output that can be useful in determining the topology being used by the ONS daemons. It shows the server:port combinations on which ONS daemons are present, and thus capable of receiving FAN events.

An edited sample of the `onsctl debug` output from a 4-node Grid Infrastructure cluster is shown here:

```
== rac1.oracle.com:6200 5844 15/01/28 17:50:50 ==

Listener:

  TYPE                 BIND ADDRESS                PORT  SOCKET
-------- ---------------------------------------- ----- ------
Local                                        ::1  6100      6
Local                                  127.0.0.1  6100      7
Remote                                       any  6200      8
Remote                                       any  6200      -


Connection Topology: (4)

                IP                       PORT   VERS  TIME
---------------------------------------- ----- ----- --------
                          10.10.10.247  6200     4 54c5a3e3
                           **                            10.10.10.244 6200
                           **                            10.10.10.245 6200
                           **                            10.10.10.246 6200
                          10.10.10.246  6200     4 54c5a3e3
                           **                            10.10.10.244 6200
                           **                            10.10.10.245 6200
                           **                            10.10.10.247 6200
                          10.10.10.245  6200     4 54c5a3e3
                           **                            10.10.10.244 6200
                           **                            10.10.10.247 6200
                           **                            10.10.10.246 6200
                          10.10.10.244  6200     4 54c5a3e3=
                           **                            10.10.10.247 6200
                           **                            10.10.10.245 6200
                           **                            10.10.10.246 6200

Server connections:

   ID            CONNECTION ADDRESS              PORT  FLAGS  SENDQ REF WSAQ
-------- ---------------------------------------- ----- ------ ----- --- ----
       0                          10.10.10.245  6200 010405 00000 001
       1                          10.10.10.246  6200 010405 00000 001
       2                          10.10.10.247  6200 010405 00000 001
```

The section titled `Listener:` shows the `localport` and `remoteport` Port numbers being communicated over. The section titled `Connection Topology: (4)` shows the IP addresses and port numbers where an ONS daemon is running (that the daemon that generated this output is aware of). In this case the daemon running on **`rac1.oracle.com:6200`** is communicating with ONS daemons at the IP address:Port combinations: `10.10.10.247:6200, 10.10.10.246:6200, 10.10.10.245:6200` and `10.10.10.244:6200` (which is the local daemon running on `rac1.oracle.com`). Daemons that each of these ONS processes is aware of are shown as sub entries.
The section titled `Server connections:` shows all ONS daemons other than the local daemon that are visible.

Remote ONS Configuration

UCP for JDBC supports ONS configuration through the `ONSConfiguration` pool property. This property is used to set the remote ONS configuration. The parameter string closely resembles the content of the ONS configuration file (`ons.config`). The string contains a list of name=value pairs separated by a newline character (`\n`). The name can be one of `nodes, walletfile,` or `walletpassword`.

The parameter string should at least specify the ONS configuration nodes attribute as a list of host:port pairs separated by a comma. SSL would be used when the `walletfile` attribute is specified as an Oracle wallet file.

A JDBC code example of setting the ONS configuration string on a `PoolDataSource` is:

```
PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();

pds.setConnectionPoolName("FCFSamplePool");
pds.setFastConnectionFailoverEnabled(true);
pds.setONSConfiguration("nodes=racnode1:4200,racnode2:4200\nwalletfile=
/oracle11/onswalletfile");
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL("jdbc:oracle:thin@((CONNECT_TIMEOUT=4)(RETRY_COUNT=30)(RETRY_DELAY=3) "+
           " (ADDRESS_LIST = "+
             " (LOAD_BALANCE=on) "+
             " ( ADDRESS = (PROTOCOL = TCP)(HOST=RAC-SCAN)(PORT=1521))) "+
           " (ADDRESS_LIST = "+
              " (LOAD_BALANCE=on) "+
              "( ADDRESS = (PROTOCOL = TCP)(HOST=DG-SCAN)(PORT=1521)))"+
         "(CONNECT_DATA=(SERVICE_NAME=service_name)))");
```

If your application is using Oracle Database 12c Release 1 (12.1.0.1) UCP and does not require an ONS wallet or keystore it is no longer necessary to use the `setONSConfiguration` method. Your application can then use auto-configuration of ONS.

It is also possible to set ONS configuration using a Java properties file. In this case the name=value passed to `setONSConfiguration` can only be `propertiefile=<path to ons.properties file>`:

```
PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();

pds.setConnectionPoolName("FCFSamplePool");
pds.setFastConnectionFailoverEnabled(true);
pds.setONSConfiguration("propertiesfile=/usr/ons/ons.properties");
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL("jdbc:oracle:thin@((CONNECT_TIMEOUT=4)(RETRY_COUNT=30)(RETRY_DELAY=3) "+
           " (ADDRESS_LIST = "+
             " (LOAD_BALANCE=on) "+
             " ( ADDRESS = (PROTOCOL = TCP)(HOST=RAC-SCAN)(PORT=1521))) "+
           " (ADDRESS_LIST = "+
              " (LOAD_BALANCE=on) "+
              "( ADDRESS = (PROTOCOL = TCP)(HOST=DG-SCAN)(PORT=1521)))"+
         "(CONNECT_DATA=(SERVICE_NAME=service_name)))");
```

The property file specified must contain an `oracle.ons.nodes` property and optionally, properties for `oracle.ons.walletfile` and `oracle.ons.walletpassword`. An example of an `ons.properties` file is shown here:

```
oracle.ons.nodes=racnode1:4200,racnode2:4200
oracle.ons.walletfile=/oracle11/onswalletfile
```

Auto-configuration of ONS

Auto-configuration of ONS, often referred to as auto-ONS, allows the client to retrieve the ONS server configuration information when an initial connection is made to the database. The ONS configuration can then be used to construct a remote subscription the ONS daemon.

No client –side configuration of ONS is required. The ONS daemon does not have to run locally.

Appendix B Troubleshooting FAN

- The following checklist can be used in the diagnosis of FAN delivery and retrieval related problems: Is one or both Oracle Clusterware Grid Infrastructure being used or Global Data Services ? FAN requires Oracle Clusterware or GDS for posting. You can use these with Oracle Restart, RAC, RAC One, DG, and ADG. The important point is the database is being monitored.
- Is a dynamic database service being used? That is a service created and operated using one of srvctl or gdsctl.
- Does the client connect using one of the recommended connect strings discussed in this paper? See examples in the sections titled *General Steps for Configuring FCF clients,* or *How to Configure FAN (*for your particular client type).
- Are FAN events being generated at the database tier? Install `FANWatcher` on any of the database nodes and confirm FAN events can be generated and received.
- Are FAN events being generated at the client or mid-tier? Install `FANWatcher` on the client or mid-tier node(s) and confirm FAN events can be received
    - Ensure that the events being received at the client or mid-tier are for the DATABASE or SERVICE you are connected to. Examine the FAN event payload for this information
- Have you set the appropriate settings on the client side for FAN?
    - For Universal Connection Pool with JDBC thin driver, ensure the bolean pool property `FastConnectionFailoverEnabled = true` is set when using Universal Connection Pool with the JDBC thin driver

    http://st-doc.us.oracle.com/database/121/JJDBC/urls.htm#r7c1-t7

    - For ODP.Net, ensure that `pooling=true; HA events=true` is set in the connect string
    - For OCI clients, refer to the section *How to configure FAN for OCI clients* where you will find examples of how to:
        - Set "`<events>true</events>`" in `oraaccess.xml` and enable notifications on the dynamic database service "`srvctl modify service –notification TRUE …`
    - For WebLogic Active Grid Link, FAN is on by default. This is visible in the admin console. For WebLogic also set `ons.configuration` at the admin console for the ons end points.
    - For JDBC OCI clients using Universal Connection Pool set `fastConnectionFailover=true`
- Check for required patches – WebLogic Active GridLink needs 19033547, 20907322
- You should also check whether your application is frequently returning connections to the pool that it is using. This is required for Fast Connection Failover functionality including planned draining, runtime load balancing, affinity routing, and is also required by Application Continuity.

Appendix C fanWatcher sample code

The following sample code is reproduced from the blog: *The WebLogic Server Blog* by Stephen Felts - https://blogs.oracle.com/WebLogicServer/entry/fanwatcher_sample_program

Save the sample code printed below as `fanWatcher,java` and compile with the jar files (`ons.jar, ojdbcXX.jar`) as per the description in the blog or that reproduced in this paper:

```
/* Beginning of sample Code for fanWatcher program */
/*
 * Copyright (c) 2015 by Oracle. All Rights Reserved
 */
import oracle.ons.ONS;
import oracle.ons.Subscriber;
import oracle.ons.Notification;
import java.util.Date;
import java.nio.ByteBuffer;
import java.sql.DriverManager;
import java.util.Properties;
import oracle.jdbc.internal.OracleConnection;

public class fanWatcher
{
  private static boolean debug = false;
  private static Subscriber s;

  public static void main(String args[]) {
    String subType =  "";
    if (args.length < 1) {
      System.out.println("Usage: java fanWatcher config [events ...]");
      System.out.println("Set config to  'crs' to use CRS");
      System.out.println("Set config to  'autoons' to use Auto-ONS; set user,
password, and url in the environment to connect to the database");
      System.out.println("Set config to the configuration string otherwise, e.g.,
nodes=host:port,...");
      return;
    }
    String config = args[0];
    for (int i = 1; i < args.length; i++) {
      System.out.println("Subscribing to events of type: " + args[i]);
      subType = args[i];
      subType = "%\"" + subType + "\"";
      System.out.println("SubType: " + subType);
    }
    fanWatcher onc_s = new fanWatcher(config, subType);
    while (onc_s.receiveEvents());
  }
```

```
public fanWatcher(String config, String eventType) {
  if (config.equals("autoons")) { // auto-ONS
    try {
      Class.forName("oracle.jdbc.OracleDriver");
      String user = System.getenv("user");
      String password = System.getenv("password");
      String url = System.getenv("url");
      if (url == null || user == null || password == null) {
        System.out.println("Environment variables for user, password, and url must be set");
        System.exit(1);
      }
      java.util.Properties p = new java.util.Properties();
      p.put("url", url);
      p.put("user", user);
      p.put("password", password);
      OracleConnection oc = (OracleConnection)
        DriverManager.getConnection(p.getProperty("url"), p);
      Properties props = oc.getServerSessionInfo();
      config = props.getProperty("AUTH_ONS_CONFIG");
      if (config == null || config.equals("")) {
        System.out.println("Failed to get Auto-ONS configuration; maybe an older release");
        System.exit(1);
      }
      System.out.println("Auto-ONS configuration="+config);
      oc.abort();
      oc.close();
    } catch (Exception e) {
      System.out.println("Failed to connect to database");
      e.printStackTrace();
      System.exit(1);
    }
  }
  if (config.equals("crs")) {
    s = new Subscriber(eventType, ""); // subscribe to service events only
  } else {
    ONS ons = new ONS(config.trim());
    if (ons == null) {
      System.out.println("Failed to get ONS server");
      System.exit(1);
    }
    System.out.println("Opening FAN Subscriber Window ...\n\n\n");
    s = ons.createNewSubscriber(eventType, "");
  }
  if (s == null) {
    System.out.println("Failed to get subscriber");
    System.exit(1);
  }
  if (debug) {
     System.out.println("FANWatcher starting");
  }
}
```

```
public boolean receiveEvents() {
  if (debug) {
    System.out.println( "** In receiveEvents. Creating Notification now ...");
  }
  if (s == null) {
    System.out.println("Failed to get ONS server");
    System.exit(1);
  }
  Notification e = s.receive(true);  // blocking wait for notification receive

  // print event header to std out. Make debug only eventually
  if (debug) {
    System.out.println( "** HA event received -- Printing header:" );
    e.print();
  }

  // Print the header details
  printEvtHeader(e);

  if (debug) {
    System.out.println( "** Body length = " + e.body().length);
    System.out.println( "** Event type = " + e.type());
  }

  /* Test the event type to attempt to determine the event body format.

    Database events generated are "free-format" events -
    the event body is a string. It consists of space delimited
    key=value pairs.

    The following test only looks for database events.
    Other events will be received, but their bodies will not be displayed.
  */
  if (e.type().startsWith("database") ) {
    if (debug) { System.out.println( "Printing event"); }
    evtPrint(e);
  } else {
    System.out.println("Unknown event type. Not displaying body");
  }

  try {
    if (e.type().equals("onc/shutdown")) {
      if (debug) {
        System.out.println("Shutdown event received.");
        System.out.println(" ONC subscriber exiting!");
      }
      s.close();
      return false; // don't continue
    } else {
      java.lang.Thread.currentThread().sleep(100);
      if (debug) {
        System.out.println("Sleep and retry.");
      }
    }
  } catch (Exception te) {
    te.printStackTrace();
  }
  return true;
}
```

```
  public void printEvtHeader(Notification e) {
    System.out.println( "\n** Event Header **" );
    System.out.println("Notification Type: " + e.type());
    System.out.println("Delivery Time: " + new Date (e.deliveryTime()));
    System.out.println("Creation Time: " + new Date (e.creationTime()));
    System.out.println("Generating Node: " + e.generatingNode() );
  }

  // Print free format event
  public void evtPrint(Notification e) {
    if (debug)  {
       System.out.println("De-coding a free-format event");
       ByteBuffer ffbuf = ByteBuffer.wrap(e.body());
       showBufferData(ffbuf,"ffbufName");
    }

    if (debug) {
       System.out.println( "** About to generate Body Block **" );
    }
    // convert the byte array event body to a String
    System.out.println("Event payload:\n" + new String(e.body()));
  }

  private void showBufferData(ByteBuffer buf, String name) {
    //Displays byte buffer contents
    int pos = buf.position();
    buf.position(0);
    if (buf.hasArray()) System.out.println("There is an array!");
    System.out.println("Raw Data for " + name);
    while(buf.hasRemaining()) {
      System.out.print( buf.get() + " ");
    }
    System.out.println();
    //Restore position and return
    buf.position(pos);
  }
}

/* End of fanWatcher Sample code */
```

Appendix D Sample Callout program (PERL based)

The following sample code will attempt to start a SERVICE when an INSTANCE UP event is received.

Note that this program is a PERL script and requires modification before it can be run. This script must be fully tested on your system so that its behavior is understood

Save the sample code as a file, place the file in the `Grid_Home/racg/usrco` directory and ensure it has execute permissions for the Grid Infrastructure owner

```
#!/usr/bin/perl -w

# Callout program that will, on an INSTANCE UP event start any services defined against
# this database. This is to address the issue of INSTANCE STOP setting non-uniform service state
# to OFFLINE.
# Note: Running services will not be relocated.
#
#       CHANGE History:  13-JUN-2006  TANTHONY Created

use strict;

# Replace the following variables with appropriate values
my $CRS_HOME="FULL PATH TO CRS HOME";
my $ORACLE_HOME="FULL PATH TO ORACLE HOME";
my $GetHOST = "/bin/hostname";
# TMP refers to the log location only
my $TMP = "/tmp";
#
#
# Logging enabled
my $LOGFILE = "$TMP/SRV_co.log";

#
my $instance;
my $database;
my $host;
my $service;
my $state;
my $reason;
my $card;
my $status;
my ($key,$value) = "";
my $myHost = "";
my ($myServ) = "";

#

# Open a logfile
local *LOG_FILE;
open (LOG_FILE, ">>$LOGFILE") or do {
   print "Cannot open $LOGFILE\n";
   exit(1);
};

# MORE CODE BEYOND THIS POINT
```

```
# Determine this host
system("$GetHOST > $TMP/myhost");
local *TEMP_FILE;
open ( TEMP_FILE,"$TMP/myhost") or do {
   print "Cannot determine hostname\n";
   exit(1);
};
while (<TEMP_FILE>) {
   chomp;
   $myHost = $_;
};
close(TEMP_FILE);

# Uncomment these lines if only interested in specific events

if ($ARGV[0] ne "INSTANCE") { exit(0); };
#if ($ARGV[0] ne "SERVICEMEMBER") { exit(0); };
#if ($ARGV[0] ne "SERVICE") { exit(0); };
#if ($ARGV[0] ne "NODE") { exit(0); };

for (my $i=0; $i <= $#ARGV; $i++) {
   #print "$i $ARGV[$i]\n";
   if ($ARGV[$i] =~ m#=#) {
      ($key,$value) = (split /=/, $ARGV[$i]);
      #print "Key = $key  Value = $value\n";
      if ($key eq "service") {
         $service = $value;
      } elsif ($key eq "instance") {
         $instance = $value;
         $ENV{ORACLE_SID} = $value;
      } elsif ($key eq "database") {
         $database = $value;
      } elsif ($key eq "host") {
         $host = $value;
      } elsif ($key eq "card") {
         $card = $value;
      } elsif ($key eq "status") {
         $status = $value;
      } elsif ($key eq "reason") {
         $reason = $value;
      }
   }
}

# MORE CODE BEYOND THIS POINT
```

```
# The following function will set service state such that they will restart
#
if ($host eq "$myHost") {
   if ($status eq "up" && $ARGV[0] eq "INSTANCE") {

#      print LOG_FILE "Attempting set of service state for database: $database\n";

       # Determine services associated with this database
       srvMap($database, $instance);
   } else {

   }

} else {
   #print LOG_FILE "Event generated on a different node\n";
}

# Sub routine to start services defined against a particular database

sub srvMap {

   my ($dbIn, $instanceIn) = @_;
   local*SRVFILE;

   #print LOG_FILE "In srvMap subroutine for $dbIn\n";

   #system("date +'%D %H:%M:%S.%N' >> /tmp/SRV_co.log") ;
#
#  Identify services defined for this database

   system("$ORACLE_HOME/bin/srvctl config service -d $dbIn > $TMP/serviceMap-$dbIn.out");
   open (SRVFILE,"$TMP/serviceMap-$dbIn.out") or do {
      print "Cannot open SRVFILE\n";
      exit(1);
   };
   while (<SRVFILE>) {
      chomp;
      ($myServ) = ($_ =~ m#^([\w]+) #);
      # print LOG_FILE "Starting service $myServ for database $database\n";

# MORE CODE BEYOND THIS POINT
```

```
#
#     Only one of the following two lines needs to be active. The first line will attempt to
#     start each service somewhere in the system. Depending on the system configuration, this
#     may cause other instances to start.
#     The second method will ONLY start the service on the instance that just started.
#     Neither method will affect currently running services.

      system("$ORACLE_HOME/bin/srvctl start service -d $dbIn -s $myServ");
#     system("$ORACLE_HOME/bin/srvctl start service -d $dbIn -s $myServ -i $instanceIn");
   };
   #system("date +'%D %H:%M:%S.%N' >> /tmp/SRV_co.log") ;
   print LOG_FILE "Routine complete\n";

# END OF CALLOUT PROGRAM
```

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

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

Fast Application Notification
April 2016
Author: Troy Anthony and Carol Colrain

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together