# Implementation einer Dataguard-Umgebung unter Windows (Cluster)

**Peter Wehner, Dirk Buchhorn**
**Fink & Partner Media Services GmbH**
**Pohlandstr. 19, 01309 Dresden**

**Schlüsselworte:**

Oracle Dataguard, Startup/Shutdown, Physical Standby, Switchover, Oracle JDBC 11g, Windows Services, Windows Cluster, Java Service Wrapper

## Einleitung

Die Inbetriebnahme von Oracle Dataguard (DG) und DG-Broker unter Windows sieht sich einer Reihe offener Fragen gegenüber. Wie startet man eine bestimmte Datenbankrolle (Primary/Standby) automatisiert über einen Windows-Service (WS)? Wie verfährt man mit der Verwendung dynamischer DNS und IP in einer DG-Broker Konfiguration? Wie bindet man die Client-Anwendungen transparent über das Netzwerk an die Datenbank in der Primary-Rolle an?

Der Vortrag präsentiert einen Ansatz, der die neuen Funktionalitäten von Oracle JDBC 11g im Bereich der privilegierten Sessions nutzt. Diese Funktionalitäten, eine Datenbank hoch- und herunterfahren, öffnen, schließen, Managed Recovery starten usw., werden in einem separaten DG-Wrapper-Service (DG-WS) implementiert, der den eigentlichen Oracle-WS umschließt. Damit werden typische Hilfskonstrukte, die die dateibasierte Konfiguration des DG-Broker durch fehleranfällige, statuslose Skripte modifizieren, vermieden. Mithin wird die Management-Schicht des DG-Broker schließlich abgelöst, womit der Ansatz auch in geclusterten Windows-Umgebungen, in denen die Oracle-WS an dynamische DNS und IP gebunden werden müssen, anwendbar wird.

Mit Hilfe definierter Abhängigkeiten zwischen WS wird zudem eine geordnete Ausführung der jeweils notwendigen Schritte, z.B. erst die Instanz und dann die Datenbank starten, realisiert. Der DG-WS beinhaltet außerdem Funktionalitäten, die ein laufendes Monitoring und Logging des DG-Status sowie gfs. ein Alerting via eMail erlauben.

Das Client-Failover nutzt bekannte Techniken in DG-Umgebungen [3] und kann auch hier zum Einsatz kommen.

## Oracle Dataguard und Dataguard Broker

Oracle Dataguard (DG) dient zum Betrieb von Standby-Datenbanken und wird von Oracle seit der Version 7.3 angeboten [3]. Eine Standby-Datenbank, hier wird nur die Physical Standby Database betrachtet, stellt eine identische Kopie einer Quelldatenbank, der Primary-Datenbank, zur Verfügung und wird für hoch verfügbare Umgebungen vorgesehen. Prinzipiell werden dazu einfach die Änderungen auf der Primary-Datenbank, in Form von Redo-Log- (LGWR) oder Archive-Log-Informationen, laufend an die Standby-Datenbank übertragen und angewendet (Managed Recovery). Ziel ist es, potentielle Fehlerszenarien, die die Primary-Datenbank betreffen können, z.B. der Ausfall von Hardware-Komponenten oder Fehler in Administration und Betrieb, durch die Standby-Datenbank abzufedern. Dabei soll der Wechsel zwischen Primary- und Standby-Datenbank für den Endanwender

von Applikationen transparent sein, unabhängig davon, ob ein geplanter Wechsel der Datenbankrollen, Switchover, oder ein ungeplanter Wechsel der Datenbankrollen, Failover, vorgenommen wird. Zudem kann eine Standby-Datenbank, neben der Sicherstellung von Hochverfügbarkeit, auch die Effektivierung von Wartungsarbeiten in einer DG-Umgebung oder ein Read-Only-Reporting abseits der Primary-Datenbank abdecken.

Der Dataguard Broker (DG-Broker) kann als Management-Schicht über DG verstanden werden und steht seit der Oracle Version 9i, nur in der Enterprise Edition, zur Verfügung [2]. DG-Broker erleichtert die Erstellung, die Administration und das Monitoring von DG-Umgebungen erheblich. Auf dieser Basis sind neben dem Command Line Interface (CLI) mittlerweile auch grafische Verwaltungswerkzeuge im Rahmen des Oracle Enterprise Manager oder Grid Control verfügbar.

Für einen Betrieb von DG-Umgebungen sind die DG-Funktionalitäten, wie Redo-Transport, Redo-Apply und Role-Management (FAL, RFS, MRP), die über Eintragungen in der spfile-Datei der Datenbank angesprochen und parametrisiert werden, hinreichend. Ein DG-Broker hält zur Verwaltung einer DG-Umgebung eigene Konfigurationsdateien vor.

**Dataguard unter Windows**

Das automatische Starten und Stoppen einer Oracle Datenbank sowie weiterer Prozesse, wie z.B. des Listener, wird unter Windows im Rahmen eines Windows Service (WS) realisiert. Prinzipiell wird zum Starten immer zuerst eine benannte Instanz gestartet (`oracle.exe`) und schließlich eine Datenbank angehangen (`oradim.exe`, zusammen mit einer entsprechenden `spfile`-Datei). Entsprechendes gilt, in umgekehrter Weise, auch für das Stoppen. Dieses Standardprinzip kann über Eintragungen in der Windows Registry (`HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE`) leicht modifiziert werden. Beispielsweise kann bestimmt werden, ob eine Datenbank automatisch starten oder welcher Typ eines Shutdown vorgenommen werden soll.

Für den Einsatz des DG ist das Standardprinzip, zzgl. möglicher Parametrisierung, nicht hinreichend, da eine DG-Datenbank, je nach Rolle, Primary oder Standby, verschiedene Start-Optionen erfordert. Am wichtigsten erscheint dabei der Umstand, dass eine Standby-Datenbank unter allen Umständen nicht geöffnet, sondern in den Modus des Managed Recovery versetzt werden muss[1]. Eine Ermittlung der notwendigen Start-Optionen fällt mit den verfügbaren Schnittstellen, Skripting über die DOS-Shell und `sqlplus`, allerdings äußerst schwer. Einerseits existiert kein definierter Weg, DOS-Skripte in WS zu integrieren. Andererseits kann, sollte es doch wenigstens semi-automatisiert gelingen, von keiner verlässlichen Programmiermöglichkeit zur Behandlung von Ausnahmen während des stufenweisen Start-Prozesses ausgegangen werden.

In der Tat besteht durch den Einsatz von DG-Broker (`dg_broker_start=true`), der den aktuellen Zustand der DG-Umgebung, u.a. die Datenbank-Rollen, in eigenen Konfigurationsdateien speichert und auf alle beteiligten Datenbanken propagiert, ein gewisser Ausweg aus der eben beschriebenen Problemstellung. Allerdings zeigte sich bei Tests, dass sich DG-Broker nicht in Umgebungen einsetzen lässt, die wie Windows Cluster mit einer dynamischen Vergabe von virtuellen DNS und IP arbeiten. DG-Broker verwendet beim Anlegen einer Konfiguration immer die direkten DNS und IP des aktuellen Host. Ein Umstand, der die Anwendung von DG-Broker in einem Windows Cluster Verbund unmöglich macht, da damit z.B. nach dem Schwenken einer Oracle Datenbank Ressourcengruppe keine DG-Broker-basierte Kommunikation mehr möglich ist.

---

1 Mit Oracle 10g führt ein einfaches Startup auf einer Standby-Datenbank, ausgehend von dem Auslesen der Control-Datei, a priori zu einem Open Read Only, so dass kein eigenes Redo geschrieben wird.

**Konzept eines DG-Wrapper-Service (DG-WS)**

Von den vorangegangenen Problemstellungen ausgehend wurde ein Konzept zur Implementierung einer Management-Schicht für DG unter Windows (Cluster) erarbeitet, das DG-Broker in benötigten Teilfunktionalitäten ablöst. Das Konzept basiert auf der Idee, die Management-Schicht in einem weiteren Windows Service (WS) vorzuhalten, der alle benötigten Oracle-WS über definierte Abhängigkeiten umschließt. Damit kann ein stufenweises, von der jeweiligen Datenbank-Rolle abhängiges, automatisches Starten und Stoppen einer Datenbank erfolgen, wobei das oben genannte Standardprinzip des Betriebes von Oracle-Datenbanken unter WS erhalten bleibt. Nicht zuletzt kann eine Oracle damit wie gehabt über `Computerverwaltung->Dienste` administriert werden, die Integration von Oracle Ressourcen in Gruppen eines Windows Cluster bedarf keiner grundlegenden Anpassung. Die folgende Abbildung veranschaulicht die Konzeptidee an Hand eines typischen stufenweisen Startup-Prozesses unter Ermittlung der Datenbank-Rolle nach einem Mount.

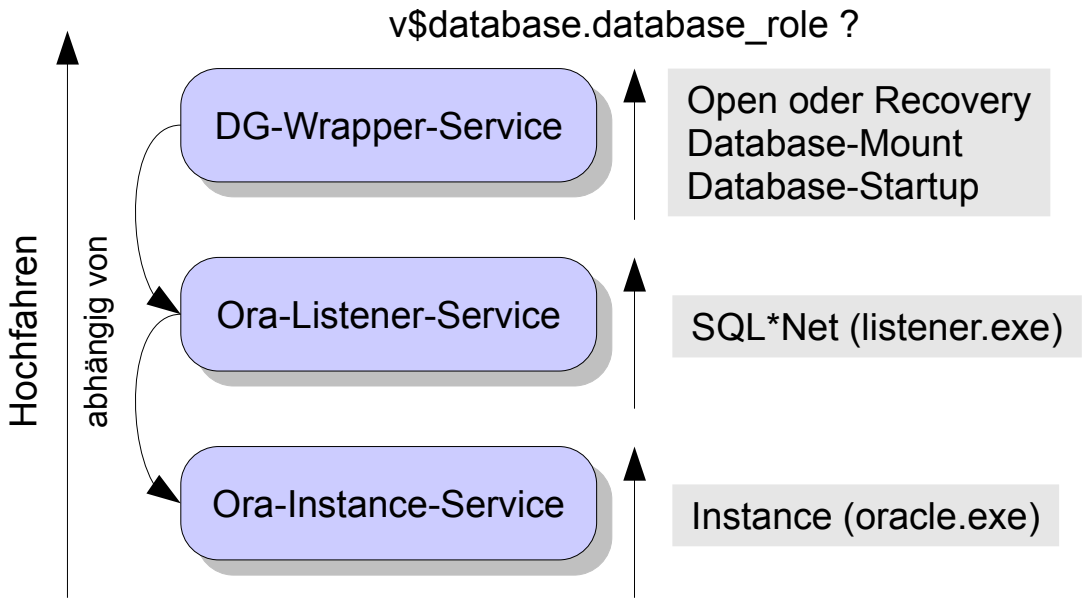


*Abb. 1: Funktionalitäten und Abhängigkeiten Oracle-Services und DG-Wrapper-Service (DG-WS)*

**Implementierung: Privilegierte Sessions mit JDBC 11g**

Die Implementierung des DG-WS nutzt eine Reihe von Modulen und Schnittstellen von Oracle und Drittanbietern. Hervorzuheben ist dabei die seit der Oracle 11gR1 verfügbare Möglichkeit, mit JDBC eine privilegierte Session als `SYSDBA/SYSOPER` zur Datenbank aufzubauen[2] und über das Interface `oracle.jdbc.OracleConnection` Befehle wie `startup` oder `shutdown` auszuführen. Eine `OracleConnection` muss dazu mit dem Property `INTERNAL_LOGON` belegt werden. Weiterhin muss für den Einsatz des JDBC Thin Driver der Parameter `REMOTE_LOGIN_PASSWORDFILE` auf `EXCLUSIVE` gesetzt werden, was die Nutzung einer externen Passwort-Datei impliziert.

Die `OracleConnection`-Methoden `startup` oder `shutdown` können weiter in ihrer Funktion spezifiziert werden. Beispielsweise kann ein `startup` in der Form `FORCE`, `RESTRICT` oder

2 Privilegierte Sessions waren über die C-Schnittstelle auch schon vor Oracle 11g möglich. Allerdings erweist sich C als nicht-4GL-Programmiersprache in modernen Umgebungen mittlerweile als unhandlich.

NO_RESTRICTION erfolgen. Ohne weiter ins Detail gehen zu wollen schließlich noch der Hinweis, dass die Methoden startup und shutdown nicht den gleichnamigen Befehlen aus sqlplus entsprechen. Ein OracleConnection.startup startet die Datenbank tatsächlich nur im Nomount-Modus und muss dafür mit dem Property PRELIM_AUTH belegt werden. Weitere alter database mount bzw. alter database open müssen in einer separaten Session folgen. Das folgende Listing umfasst eine komplette Java-Klasse zum Hochfahren einer Datenbank [4].

```
import oracle.jdbc.OracleConnection;
import oracle.jdbc.pool.OracleDataSource;
import java.sql.Statement;
import java.sql.SQLException;
import java.util.Properties;

public class JDBCStartup {
 public static void main(String[] args) {
  try {
   // Connection properties. Required: SYSDBA, PRELIM_AUTH = true
   Properties prop = new Properties();
   prop.setProperty("user","sys");
   prop.setProperty("password","password");
   prop.setProperty("internal_logon","sysdba");
   prop.setProperty("prelim_auth","true");
   OracleDataSource ods = new OracleDataSource();
   ods.setConnectionProperties(prop);
   ods.setURL("jdbc:oracle:thin:@//ora1:1521/orcl");
   OracleConnection con = (OracleConnection)ods.getConnection();
   // startup the database
   con.startup(OracleConnection.DatabaseStartupMode.NO_RESTRICTION);
   con.close();
   ods.close();
   // At this time the instance is only started. The database is
   // not mounted or opened. You must reconnect as SYSDBA without
   // the PRELIM_AUTH
   prop.clear();
   prop.setProperty("user","sys");
   prop.setProperty("password","password");
   prop.setProperty("internal_logon","sysdba");
   ods = new OracleDataSource();
   ods.setConnectionProperties(prop);
   ods.setURL("jdbc:oracle:thin:@//ora1:1521/orcl");
   con = (OracleConnection)ods.getConnection();
   // mount and open the database
   Statement stmt = con.createStatement();
   stmt.execute("alter database mount");
   stmt.execute("alter database open");
   stmt.close();
   con.close();
   ods.close();
  } catch(SQLException e) { System.out.println(e.getMessage()); }
 }
}
```

**Implementierung: Datenbank-Status und-Rolle via `v$instance` bzw. `v$database`**

Der DG-WS muss, angesteuert über `Computerverwaltung->Dienste` oder ein Schwenken von Gruppen in einem Windows Cluster, jederzeit in der Lage sein, den Status und die Rolle einer Datenbank in einer DG-Umgebung zu ermitteln um die jeweils richtigen JDBC-Befehle abzusetzen oder bekannte Ausnahmeszenarien zu kommunizieren[3].

Zur Ermittlung von Datenbank-Status und -Rolle kommt dabei ein relativ einfaches Verfahren zum Einsatz, das in einer geordneten Folge `select`-Befehle auf die Fixed-Views `v$instance` und `v$database` ausführt und schließlich `v$database.database_role` ausliest. Dabei wird vorausgesetzt, dass die Oracle Instanz und der Listener, die durch die Windows Standardservices gestartet werden, bereits laufen (siehe Kaskaden von Windows Services weiter unten). Die folgende Tabelle stellt die `select`-Befehle dem jeweiligen Datenbank-Status gegenüber.

| select | Status: nothing | Status: nomount | Status: mount |
|---|---|---|---|
| `select * from v$instance` | scheitert | ok | ok |
| `select * from v$database` | scheitert | scheitert | ok |

Mit diesem Verfahren ist es demnach ohne Nebeneffekte möglich, von dritter Seite den Status oder die Rolle einer Datenbank zu ändern, ohne die Integrität des DG-WS zu beeinflussen. Naheliegend und von hoher praktischer Relevanz sind dabei natürlich Anwendungen wie Switchover oder Failover in einer DG-Umgebung.

**Implementierung: Java Service Wrapper**

Während die vorangegangenen Ausführungen schon ein funktionsfähiges Java-Programm zum Starten und Stoppen einer Oracle Datenbank umfassen, muss die Funktionalität nun an die Schnittstelle von Windows Services angebunden werden. Mit dem Java Service Wrapper [1] steht dafür ein bewährtes und robustes Werkzeug zur Verfügung, welches über die Implementierung des Interface `org.tanukisoftware.wrapper.WrapperListener` die Behandlung der durch die WS-API beschriebenen Ereignisse `Start`, `Stop`, `Pause` und `Restart` ermöglicht.

Die tatsächliche Einbindung des DG-WS als Service übernimmt ebenfalls Java Service Wrapper, indem vorbereitete Konfigurations- und Administrationsdateien angepasst bzw. ausgeführt werden. Die nachfolgenden Listings stellen Auszüge aus der Konfigurationsdatei des DG-WS dar, die z.B. die Hauptklasse der Anwendung angeben:

```
# Java Main class.
wrapper.java.mainclass=de.fup.dataguard.DataguardService
```

ggfs. notwendige Parameter der Hauptklasse zulassen:

```
# Application parameters. Add parameters as needed starting from 1
wrapper.app.parameter.1=e:/FuP_dg-Service/conf/dg.properties
wrapper.app.parameter.2=e:/FuP_dg-Service/conf/dg.user.properties
```

3 Beispielsweise führt ein Scheitern eines `startup` und `shutdown` unter bestimmten Umstanden zum Blockieren aller weiteren Datenbankverbindungen durch den Listener.

den Namen des WS vergeben:

```
# Name of the service
wrapper.ntservice.name=FuP_dg-Service_DGP
```

oder den Starttyp des WS vereinbaren:

```
# Mode in which the service is installed. AUTO_START or DEMAND_START
wrapper.ntservice.starttype=AUTO_START
```

Schließlich wird der DG-WS durch den Aufruf einer Administrationsdatei installiert:

```
E:\FuP_dg-Service\jsw-3.2.3\bin>create-FuP_dg-Service_DGP.bat
wrapper  | FuP DataGuard Service (DGP) installed.
```

und ist ad hoc über `Computerverwaltung->Dienste` verfügbar.

**Konfiguration: Kaskaden von Windows Services (WS) für Host- und Cluster-Umgebungen**

In Abb. 1 wurden beispielhaft die Abhängigkeiten der Oracle Standardservices für Instanz, Listener und DG-WS dargestellt. Um diese geordnete Folge von Ausführungen zu definieren, stellt die WS-API Methoden zur Verfügung, die je nach Platform, Single-Host- oder Cluster-Umgebung, unterschiedlich angesprochen werden müssen.

Für die Single-Host-Umgebung können Abhängigkeiten mit dem sc-Kommando, dem Service Control Manager, vorgegeben werden. Die notwendigen Aufrufe lauten wie folgt[4]:

```
E:\Programme\FuP_dg-Service\jsw-3.2.3\bin>sc config
OracleOraDb10g_home1TNSListenerLISTENER_DGS
depend= "OracleServiceDGS"
[SC] ChangeServiceConfig SUCCESS

E:\Programme\FuP_dg-Service\jsw-3.2.3\bin>sc config
FuP_dg-Service_DGS
depend= "OracleOraDb10g_home1TNSListenerLISTENER_DGS"
[SC] ChangeServiceConfig SUCCESS

E:\Programme\FuP_dg-Service\jsw-3.2.3\bin>sc config
FuP_dg-Service_DGP
depend= "OracleOraDb10g_home1TNSListenerLISTENER_DGP"
[SC] ChangeServiceConfig SUCCESS
```

In einer Cluster-Umgebung kann über eine grafische Oberfläche, dem Windows Cluster Manager, gearbeitet werden. Abb. 2 zeigt einen entsprechenden Screenshot.

Im Ergebnis dieser Konfiguration kann die gesamte Kaskade jeweils durch ein Start-Ereignis auf den DG-Service `FuP DataGuard Service (DGS)` hochgefahren oder, umgekehrt, durch ein Stop-Ereignis auf den Service `OracleServiceDGS` heruntergefahren werden. Natürlich ist es auch möglich, nur den Service `FuP DataGuard Service (DGS)` durchzustarten, was einem

4 Zu beachten ist hier das Leerzeichen hinter `depend=`, das aus unerfindlichen Gründen gesetzt werden muss.

`shutdown immediate` und anschließendem `startup` unter `sqlplus` gleichkommt.

Nicht unerwähnt soll bleiben, dass die unter `HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE` befindlichen Registryeinträge `ORA_<SID>_AUTOSTART` und `ORA_<SID>_SHUTDOWN` auf `FALSE` zu setzen sind, da sich der DG-WS nunmehr um diese Belange kümmert.

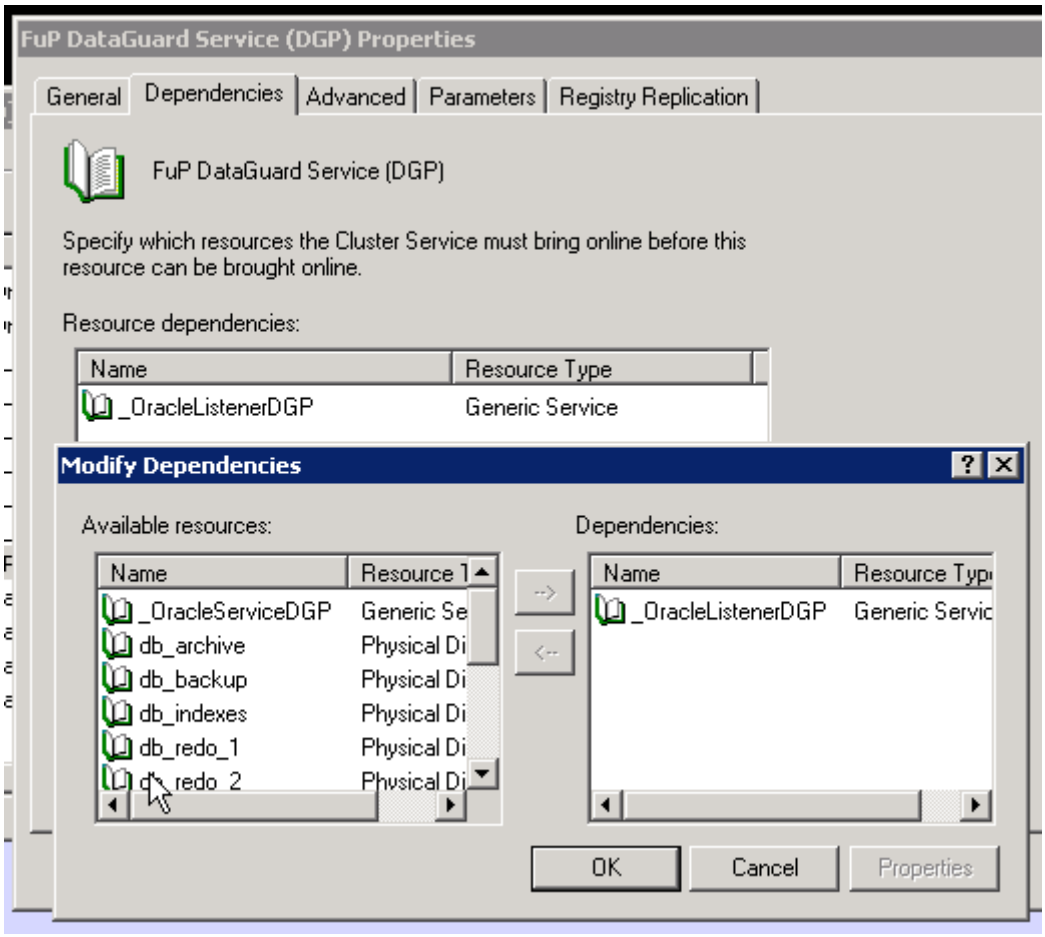


*Abb. 2: Vorgabe von Abhängigkeiten im Windows Cluster Manager*

**Implementierung: Monitoring Redo-Transport und Redo-Apply**

Neben der Implementierung der Start- und Stop-Logik zzgl. der Entgegennahme der entsprechenden WS-Ereignisse beinhaltet der DG-WS in einem eigenen Thread auch Überwachungsfunktionalitäten für DG-Umgebungen. Dafür werden laufend und je nach Rolle der verwalteten Datenbanken die typischen `v$log`-, `v$dest`- und `v$standby`- Fixed-Views ausgelesen. Auf diese Weise lassen sich Fehlerzustände und so genannte Log-Gaps sowohl im Transport als auch im Apply feststellen und kommunizieren[5].

5 Dafür ist nahezu mehr Programmieraufwand notwendig als für die Start- und Stop-Logik im DG-WS, da sich die Daten in den genannten Views, je nach Konfiguration der DG-Umgebung, z.B. Einsatz des LGWR mit Standby-Redo oder des ARCH für den Transport, teilweise stark unterscheiden.

Da die externe Passwort-Datei für JDBC-Verbindungen verwendet wird, kann durch einen lokalen DG-WS sogar der Status einer DG-WS-Datenbank auf einem entfernten Host ermittelt und auswertet werden. Weiterführende Aktivitäten auf entfernten DG-WS-Datenbanken, wie z.B. automatisches Failover, wurden bisher allerdings nicht betrachtet, da eine dbzgl. Funktinalität, vergleiche den bekannten Dataguard Observer, tatsächlich auf einem Drittsystem vorgehalten werden muss.

**Implementierung: Logging, Alerts und Debugging**

Die bereits mächtige Gesamtfunktionalität des DG-WS, die mithin auf nichttrivialen DG-Umgebungen aufsetzen muss, erfordert einerseits eine aussagekräftige Protokollierung auf Standardkanäle und andererseit auch eine Möglichkeit zur detaillierten Einsichtnahme in Prozeßstati und Kennwerte.

Alle relevanten Log-Einträge werden, wie für Windows Systeme üblich, in die Typen Fehler, Warnung und Information unterteilt und in `Ereignisanzeige->Anwendung` geschrieben. Damit ist die Evaluierung einer Resource in einem Windows Cluster erst möglich und auch die Anbindung von dritten Monitoring-Systemen über SNMP o.ä. gegeben.

Neben `Ereignisanzeige->Anwendung` wird ausserdem ein zirkuläres Datei-Logging gefahren, welches als Obermenge[6] der `Ereignisanzeige` ausführliche Infromationen enthält. Sämtliche Fehlerzustände und Log-Gaps werden über eMail an bekannte Adressaten kommuniziert.

Zur detaillierten Einsichtnahme in Prozeßstati und Kennwerte einer definierten DG-Umgebung wurde im DG-WS ein Jetty Servlet Container [6] implementiert, der eine feste html-Statusseite ausliefert. Hier werden die schon aus anderen Gründen erhobenen Überwachungsinformationen übersichtlich dargestellt und ggfs. kommentiert. Wie bereits erwähnt, kann ein lokaler DG-WS auch auf entfernte DG-WS-Datenbanken zugreifen, so dass in der Übersicht auch nichtlokale Informationen zur Anzeige kommen. Die folgende Abbildung zeigt eine voll funktionstüchtige und fehlerfreie DG-Umgebung mit zwei Datenbanken.

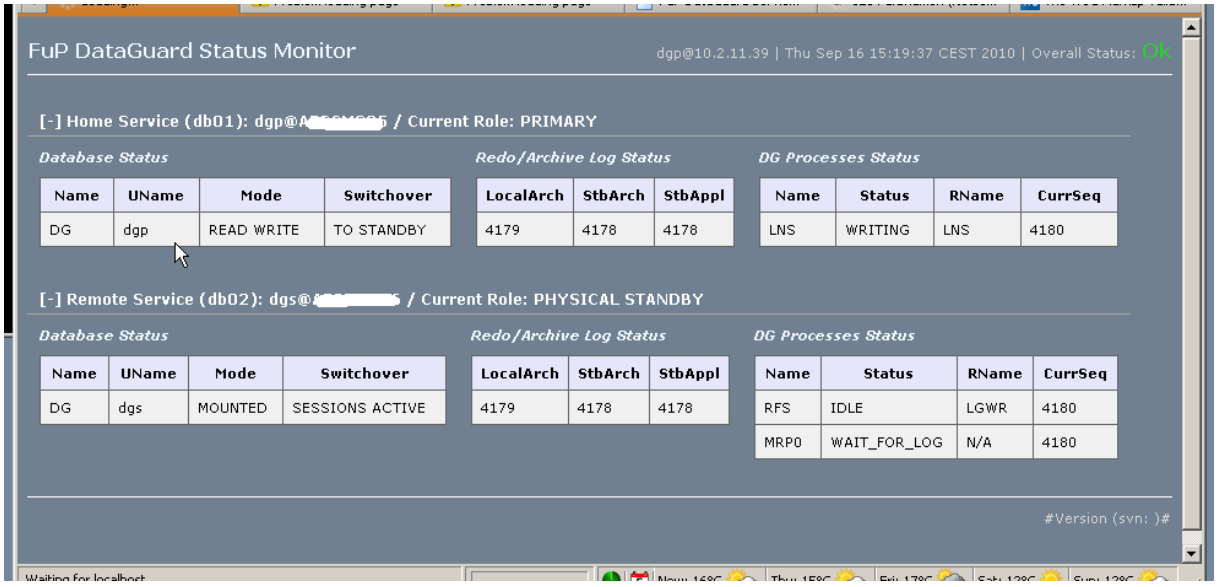


*Abb. 3: DG-WS-Statusseite (Jetty Servlet) einer DG-Umgebung mit zwei Datenbanken*

6 Die einzelnen Kanäle werden aus dem DG-WS über log4j [5] und die entsprechenden Log-Level angesprochen.

**Zusammenfassung und Ausblick**

Der automatisierte Betrieb von DG-Umgebungen unter Windows ist nicht ohne weiteres praktikabel. Das gilt insbesondere für die Windows Cluster Platform, die erweiterte Anforderungen an Service-basierte Programme stellt. Eine programmiertechnische Erweiterung ist an dieser Stelle möglich, wenn sich die Lösung in die gegebene Architektur integriert und die verfügbaren Schnittstellen nutzt. Der DG-Wrapper-Service (DG-WS) passt sich als eigener Windows Service (WS) nahtlos in die gegebene Architektur ein, wobei die Oracle Standardservices für Instanz und Listener umschlossen werden.

In der Tat profitiert der DG-WS von einer Reihe von Neuerungen bei JDBC 11g und professionellen Drittmodulen wie Java Service Wrapper, log4j und Jetty. Andererseits baut auch ein großer Teil der implementierten Logik auf bewährten Oracle Schnittstellen, wie Fixed Views auf.

Der DG-WS ist bereits seit einiger Zeit in produktiven Single-Host- oder Cluster-Umgebungen unter Windows im Einsatz und zeichnet sich durch ein verlässliches, unauffälliges Laufzeitverhalten aus. Im weiteren ist geplant, Funktionen zum Switchover bzw. Failover auch über das Jetty Servlet anzubieten.

**Literatur**


[1] http://wrapper.tanukisoftware.org/doc
[2] Held, Andrea: Oracle 10g Hochverfügbarkeit, Addison-Wesley Verlag, 2005
[3] Metalink Note 740029.1: Step By Step Guide On How to Configure And Test Client-Failover For Dataguard Switchover and Failover
[4] http://www.oracledistilled.com/java/database-startup-and-shutdown-through-jdbc
[5] http://logging.apache.org/log4j
[6] http://jetty.codehaus.org/jetty


**Kontaktadresse:**


**Dr. Peter Wehner, Dirk Buchhorn**
Fink & Partner Media Services GmbH
Pohlandstr. 19
D-01309 Dresden

Telefon: +49 (0) 351-314 0600
Fax: +49 (0) 351-314 06001
E-Mail: {peter.wehner, dirk.buchhorn}@finkundpartner.de
Internet: www.finkundpartner.de