Integrating RAID in High Availability and Disaster Recover environments

1. Overview of High Availability & Disaster Recovery

High Availability

The term high availability (HA), when applied to computer systems, means that the application or the service in question is available all the time, regardless of time of day, location and other factors that can influence the availability of such an application. In general, it is the ability to continue a service for extremely long duration without any interruptions. Typical technologies for HA include redundant power supplies and fans for servers, RAID (redundant array of inexpensive/independent disks) configuration for disks, clusters for servers, multiple network interface cards and redundant routers for networks. However, with these benefits has come an increasing dependence on that infrastructure. If a critical application becomes unavailable, then the business can be directly affected. Revenue and customers can be lost, penalties can be owed, and bad publicity can have a lasting effect on customers. It is important to examine the factors that determine how the data is protected and maximize availability to the users.

Disaster Recovery

A Disaster Recovery (DR) solution requires the replication of all data (database and file systems data) between 2 distinct data centers located usually in 2 distinct geographic points, the main site and the disaster recovery site.

- For the Oracle database replication, "Oracle Data Guard" can be used to ensure the database synchronization between the main site and the disaster recovery site.
- On the other hand, file system asynchronous replication can be ensured by several third party solutions depending on the host operating system, or when available, could be resolved at the hardware level by the storage system itself.

DR has two specific metrics that have to be defined: the Recovery Time Objective **(RTO)** and the Recovery Point Objective **(RPO):**

- **RTO** is the elapsed time from the service interruption until the service is restored. It answers the question: "How long can you be without service?"
- **RPO,** on the other hand, is the point of time represented by the data upon service resumption. It answers the question: "How old can the data be?"

Normally a DR site is only activated upon weighted decision involving IT teams up to executive representatives and there could be different modes to point to the DR site that is why the operationalization of a Switch Over to a DR site is something that needs to be well aligned with the customer.

What are HA clusters?

HA clusters are groups of computers that support server applications that can be reliably utilized with a minimum of down-time. Without clustering, if a server running a particular application crashes, the application will be unavailable until the crashed server is fixed. HA clustering remedies this situation by detecting hardware/software faults, and immediately restarting the application on another system without requiring administrative intervention, a process known as **failover**. As part of this process, clustering software may configure the node before starting the application on it. For example, appropriate file systems may need to be imported and mounted, network hardware may have to be configured, and some supporting applications may need to be running as well.
HA cluster implementations attempt to build redundancy into a cluster to eliminate single points of failure, including multiple network connections and data storage which is redundantly connected via storage area networks.

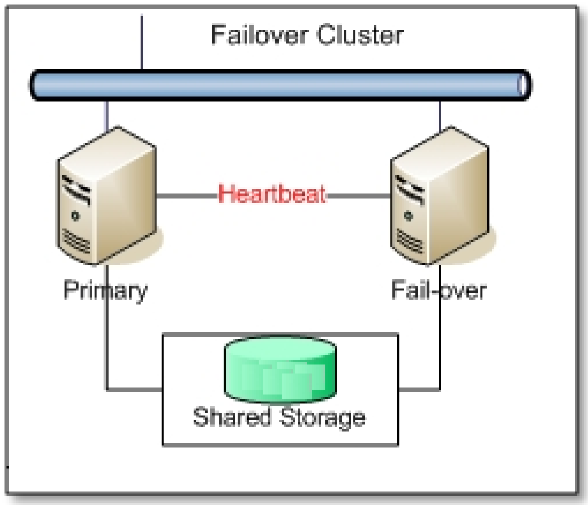


Figure 1 - Fail-over cluster with 2 nodes Diagram

2. RAID High Availability Architectures

RAID High Availability must be ensured **at the Application level** and **at the Database level**. HA Clusters will respond and support RAID HA.

2.1 RAID Application Level High Availability Architectures

RAID Active-Active Cluster

RAID Active-Active Cluster is not supported.

RAID Active-Passive Cluster

In an Active-Passive application cluster nodes, there are passive nodes that are in a standby state waiting for the eventual fail-over but still are well underused. That is why we do not recommend this kind of application cluster because there is no full profit of the node's resources.

A RAID Active-Passive cluster could also consist of more than two active cluster nodes supported by a lower number of standby cluster nodes (i.e.: 5 active nodes and 2 passive nodes) where each of the active nodes is able to fail over to any of the other standby nodes. It is important to note that the passive nodes must be correctly dimensioned regarding the applications that are failed over in order to be in accordance with predefined performance objectives.

RAID Active Passive Cluster Application Integration

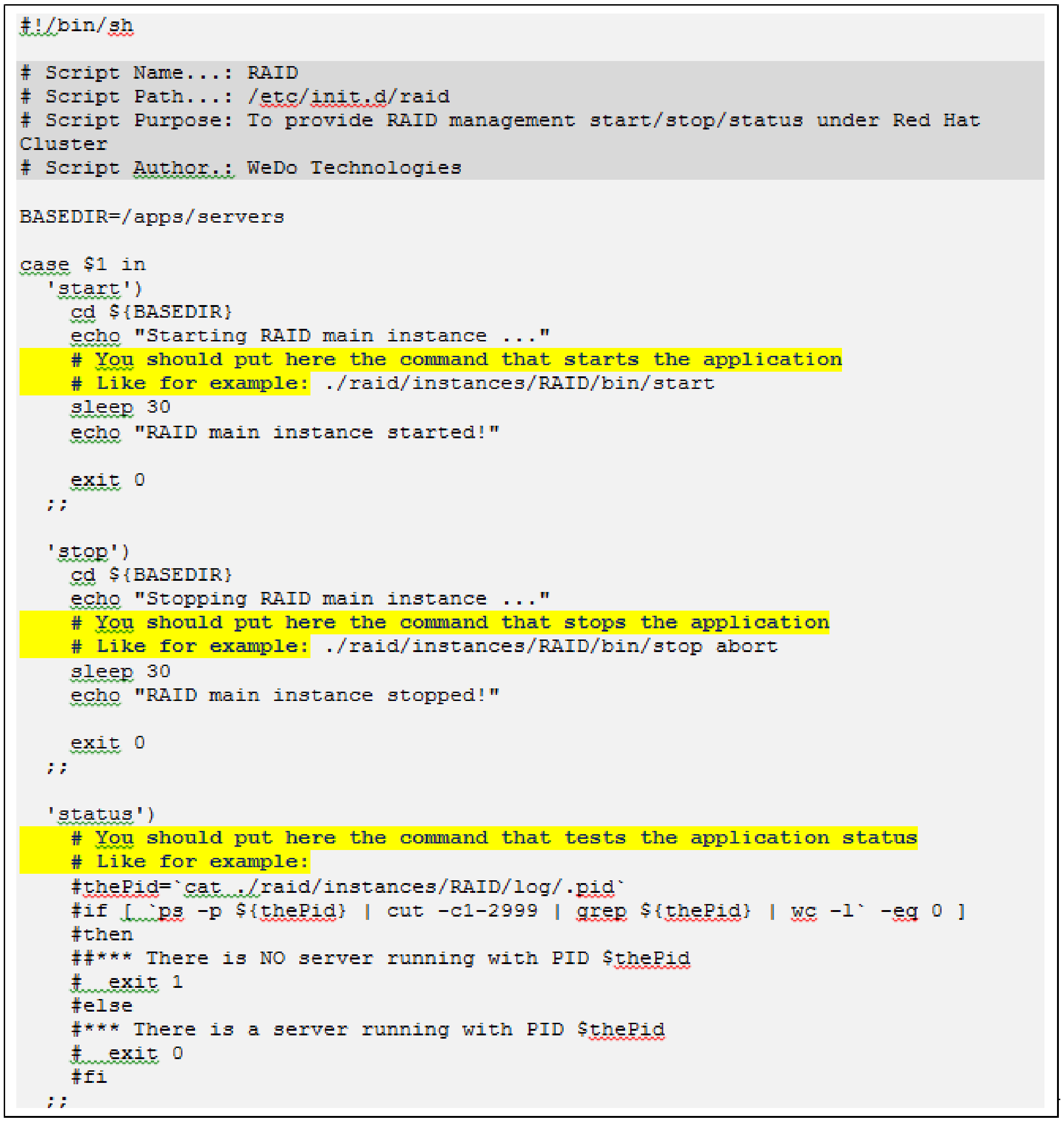
RAID Services Integration with cluster-aware software

Cluster-aware software is an application designed on top of a High-availability cluster and that calls cluster-aware APIs that are able to:

1. Determine the status of an application service.
2. Start an application service.
3. Stop an application service.

Typically cluster software calls a shell script that implements the above functions.
In order to integrate RAID in Cluster-aware software, a shell script template is already available and is located in the RAID documentation.

Example of the custom fail-over script template for a RAID central server (raid.sh) :



```
#!/bin/sh

# Script Name...: RAID
# Script Path...: /etc/init.d/raid
# Script Purpose: To provide RAID management start/stop/status under Red Hat
Cluster
# Script Author.: WeDo Technologies

BASEDIR=/apps/servers

case $1 in
  'start')
    cd ${BASEDIR}
    echo "Starting RAID main instance ..."
    # You should put here the command that starts the application
    # Like for example: ./raid/instances/RAID/bin/start
    sleep 30
    echo "RAID main instance started!"

    exit 0
  ;;

  'stop')
    cd ${BASEDIR}
    echo "Stopping RAID main instance ..."
    # You should put here the command that stops the application
    # Like for example: ./raid/instances/RAID/bin/stop abort
    sleep 30
    echo "RAID main instance stopped!"

    exit 0
  ;;

  'status')
    # You should put here the command that tests the application status
    # Like for example:
    #thePid=`cat ./raid/instances/RAID/log/.pid`
    #if [ `ps -p ${thePid} | cut -c1-2999 | grep ${thePid} | wc -l` -eq 0 ]
    #then
    ##*** There is NO server running with PID $thePid
    #  exit 1
    #else
    #*** There is a server running with PID $thePid
    #  exit 0
    #fi
  ;;
```

Regarding the Oracle Database cluster-aware integration, some vendors have already predefined script templates; this is only valid for a standard database used within one HA Cluster architecture, because an Oracle RAC database implements already everything by its own.

2.2 Database Cluster (RAC)

For the RAID database, Oracle RAC can be used in order to split workload and ensure high availability at the database level. An Oracle RAC database is a clustered database and, as seen previously, a cluster can be described as a pool of independent servers that cooperate as a single system and like that the Oracle RAC database provides:

- High Availability: In the event of a system failure, clustering ensures high availability to users. Redundant hardware components, such as additional servers, network connections, and disks, allow the cluster to provide high availability. Such redundant hardware architectures avoid a single point-of-failure and provide exceptional fault resilience.

- Performance:A clustered database is a single database that can be accessed by multiple instances. Each instance runs on a separate server in the server pool. When additional resources are required, additional servers and instances can easily be added to the server pool with no downtime. Once the new instance is started, applications using services can immediately take advantage of it with no changes to the application or application server.

- Scalability: Server Pools provide modular incremental system growth.

RAID Oracle RAC database architecture presented in the next figure:

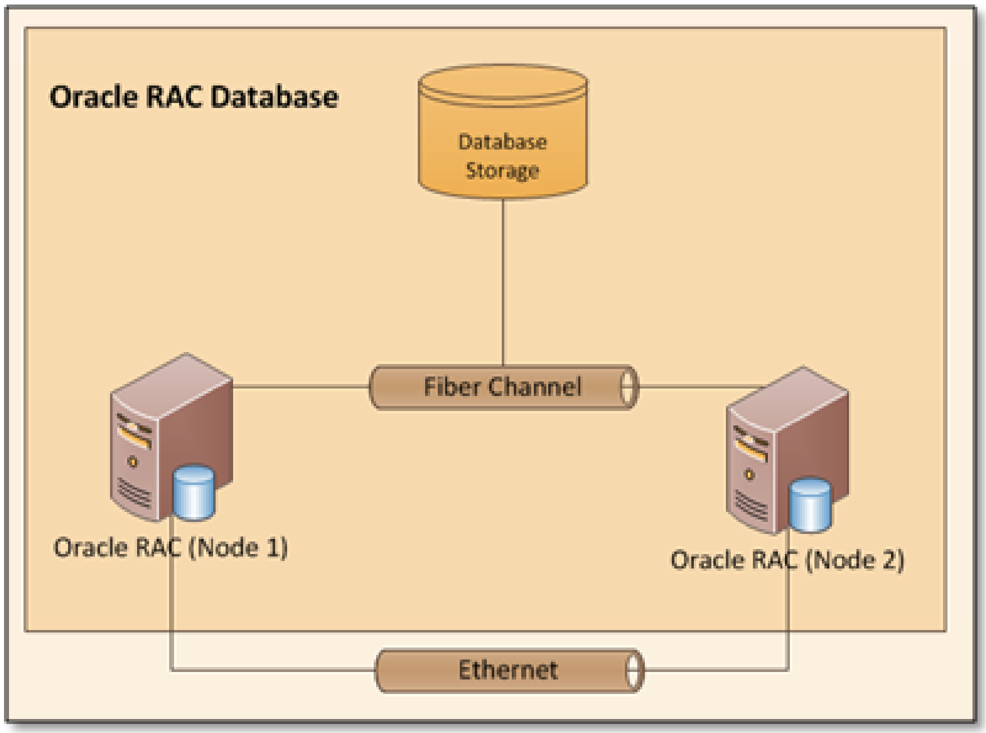


Figure 4 - RAID Oracle RAC Database

Application RAC connectivity

- **Single Client Access Name (SCAN)**

When the application is installed and connected to an Oracle Real Application Clusters (RAC) database, the connections can be define to take advance of the Single Client Access Name (SCAN).
SCAN is a new Oracle RAC 11g Release 2 feature that provides a single name for clients to access Oracle Databases running in a cluster. The benefit is that the client's connect information does not need to change if you add or remove nodes in the cluster. Having a single name to access

the cluster allows clients to use the simple JDBC thin URL to access any database running in the cluster, independently of which server(s) in the cluster the database is active.
SCAN provides load balancing and failover for client connections to the database and works as a cluster alias (**scan-name**) for database instances in the cluster.

- **SCAN - How It Works**

Connection Load Balancing using SCAN:

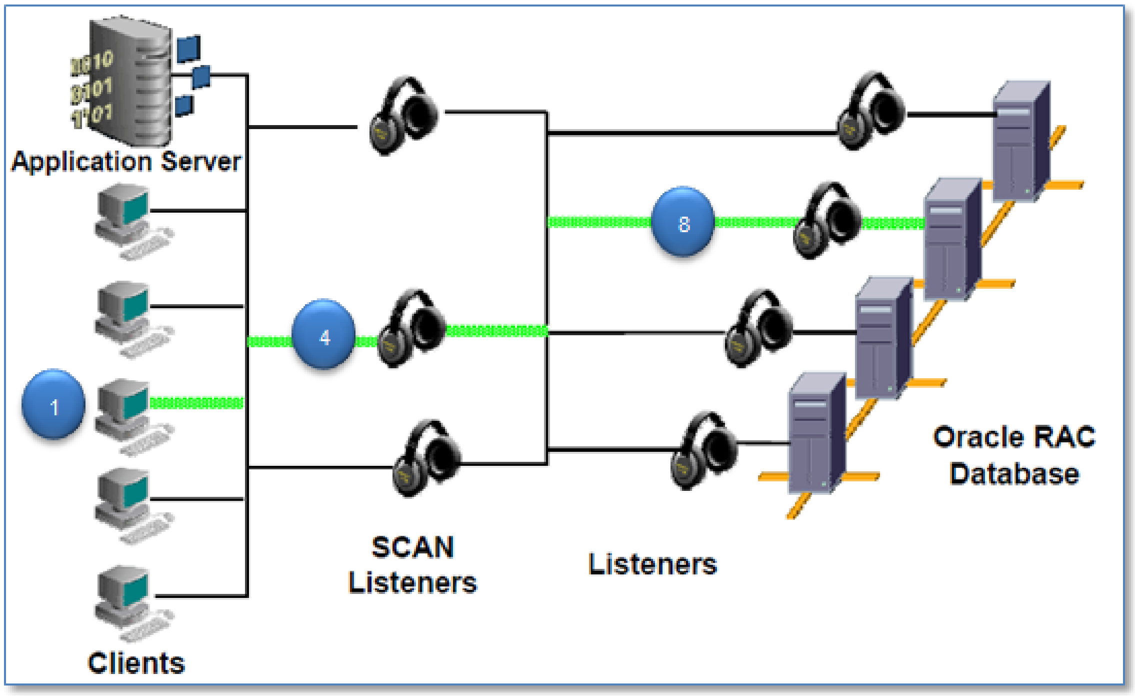


Figure 5 - SCAN figure

1. A client requests a DNS network resolution of the SCAN hostname (**scan-hostname**).
2. DNS responds with circulating list of 3 IPs.
3. Client chooses first, or random, IP in the list.
4. Client connects to SCAN Listener at chosen IP and requests to access a database service.
5. SCAN Listener is receiving service registrations and load balance advisories from all the instances in the cluster.
6. SCAN Listener chooses the least loaded instance offering the client requested service.
7. SCAN Listener re-directs the connection request to the Local Listener of that least loaded instance.
8. The Local Listener accepts the request and establishes the client session.

Note: All of these actions take place transparently to the client without any explicit configuration required in the client.

Example of a RAID Thin JDBC connection string using the SCAN connection:

```
jdbc:oracle:thin:@//scan-name:1521/raidprd
```

Oracle RAC Database Services

Database services are logical abstractions for managing workloads in Oracle Databases.
With Oracle RAC database, a database service can be spanned over one or more instances and an instance can support multiple services. The

number of instances offering a service is managed dynamically independently of the application. When outages occur, services are automatically restored to surviving instances. When instances are restored, any services that are not running are restored automatically.
Services provide the following benefits:

- Represent a single entity for managing several subsets of application's operations that are competing for the same resources.
- Allow each subsets of application's operations workload to be managed as a unit.
- Hide the complexity of the cluster from the client.

It is possible to define services that are assigned to particular subsets of application's operations and like that use services to manage the workload for different types of work.
For example, we can create a service called "**LOAD**" and assign it for all data loading operations and create another a service called "**VAL**" for all data validation processes. With an Oracle RAC database with 3 nodes having respectively the instances RAC01, RAC02 and RAC03, it is possible to allocate the service "**LOAD**" to work always on the instance RAC01 having instance RAC02 as its failover and allocate the service "**VA L**" to work always on RAC01 and RAC02 instances knowing that the Cluster will balance the service load between these 2 instances.

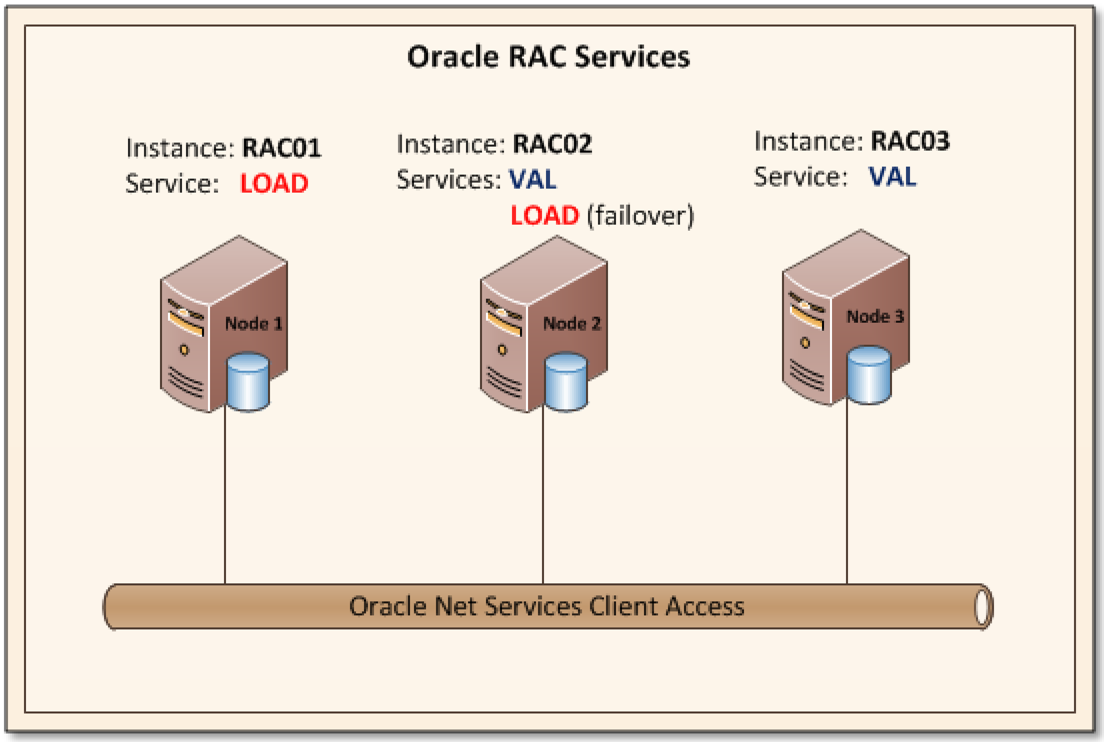


Figure 6 - Oracle RAC Services

Another configuration for instance with an Oracle RAC 2 nodes is having 2 service applications called respectively "**APP1**" and "**APP2**" that are allocated on a specific node making that the resources are not shared between nodes and are dedicated for each service:

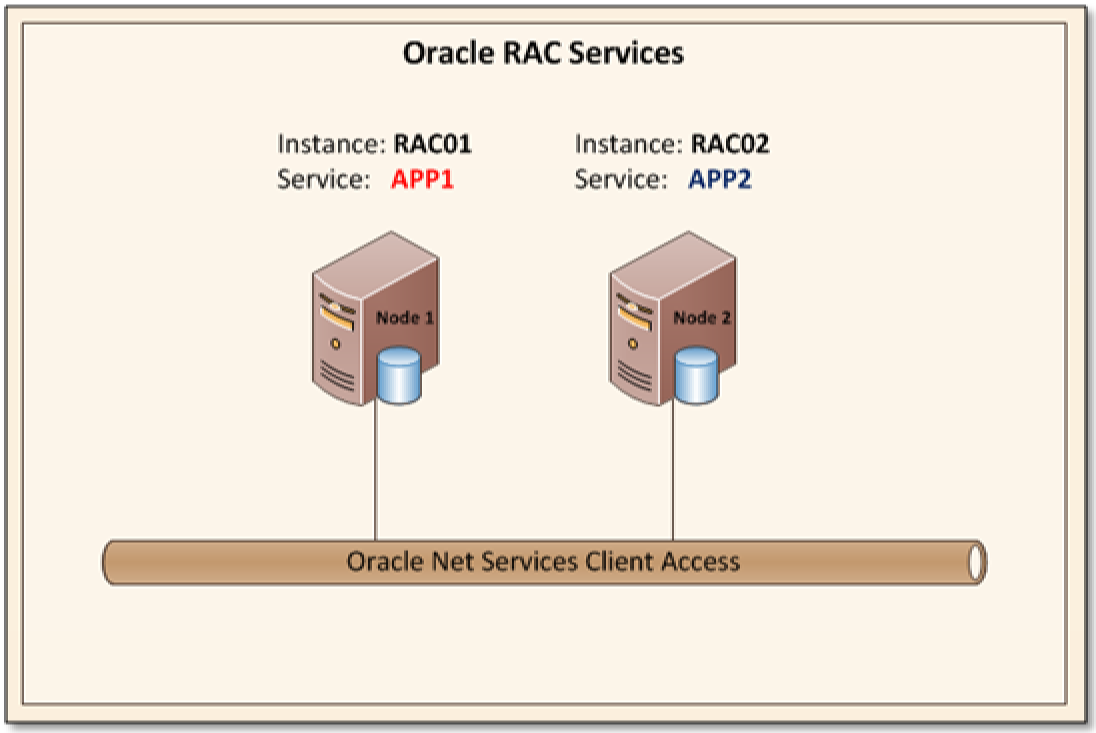
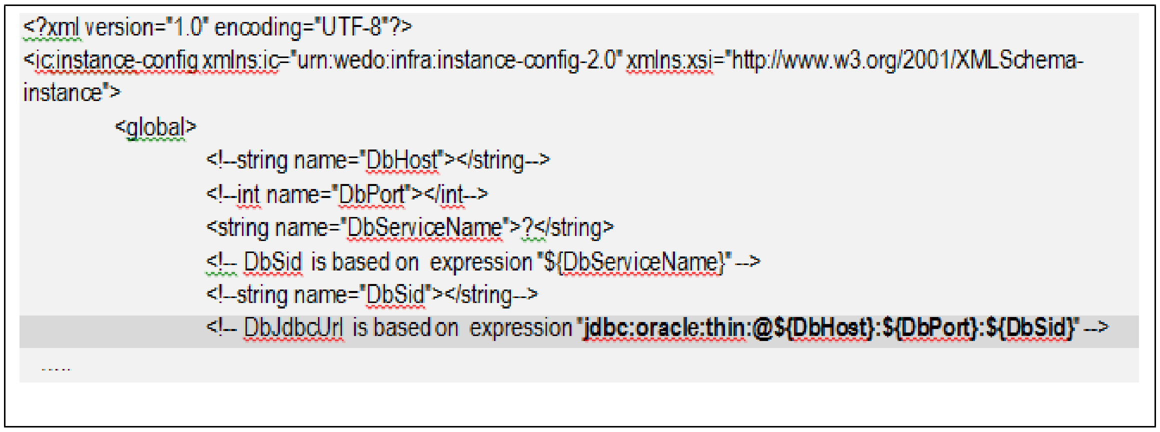


Figure 7 - Oracle RAC Application Services

RAID Installation Config File

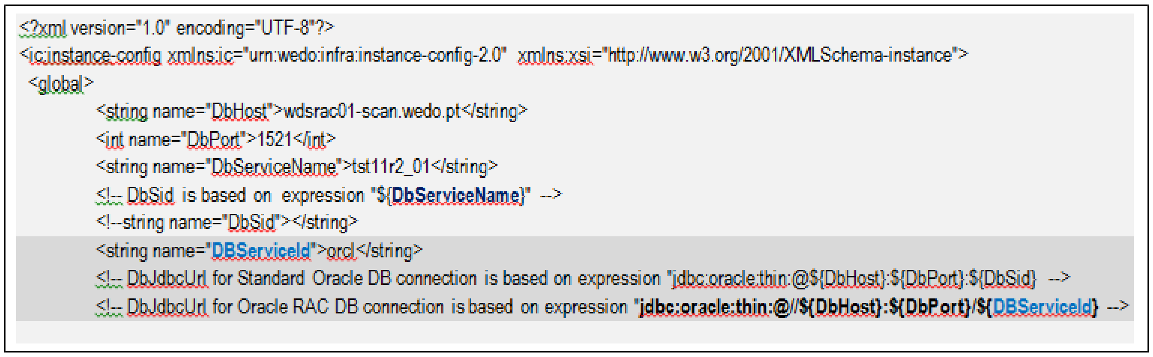
The RAID config file template, the one that is used as reference during the product installation process, has in its content the predefined JDBC URL's connection string for installing RAID towards an Oracle standard database, but the objective is to include also the new JDBC URL's connection string that will be used by the installation to connect to an Oracle RAC database.

For example, the actual template that is generated is like this:



```
<?xml version="1.0" encoding="UTF-8"?>
<ic:instance-config xmlns:ic="urn:wedo:infra:instance-config-2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
        <global>
                <!--string name="DbHost"></string-->
                <!--int name="DbPort"></int-->
                <string name="DbServiceName">?</string>
                <!-- DbSid is based on expression "${DbServiceName}" -->
                <!--string name="DbSid"></string-->
                <!-- DbJdbcUrl is based on expression "jdbc:oracle:thin:@${DbHost}:${DbPort}:${DbSid}" -->
  .....
```

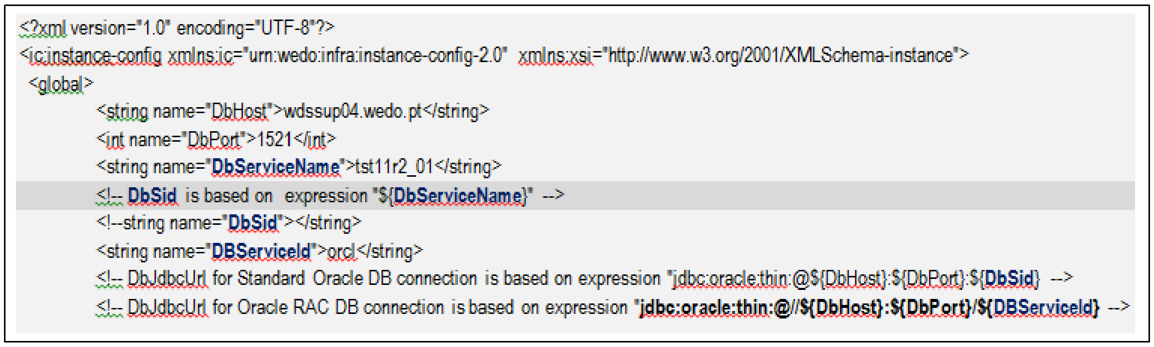
The RAID config file template will include the new RAC JDBC URL connection string, like it is shown here in the next table as an example (The new lines are in darker grey):



```
<?xml version="1.0" encoding="UTF-8"?>
<ic:instance-config xmlns:ic="urn:wedo:infra:instance-config-2.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <global>
          <string name="DbHost">wdsrac01-scan.wedo.pt</string>
          <int name="DbPort">1521</int>
          <string name="DbServiceName">tst11r2_01</string>
          <!-- DbSid is based on expression "${DbServiceName}" -->
          <!--string name="DbSid"></string>
          <string name="DBServiceId">orcl</string>
          <!-- DbJdbcUrl for Standard Oracle DB connection is based on expression "jdbc:oracle:thin:@${DbHost}:${DbPort}:${DbSid} -->
          <!-- DbJdbcUrl for Oracle RAC DB connection is based on expression "jdbc:oracle:thin:@//${DbHost}:${DbPort}/${DBServiceId} -->
```

- **Oracle connectivity variables clarifications**

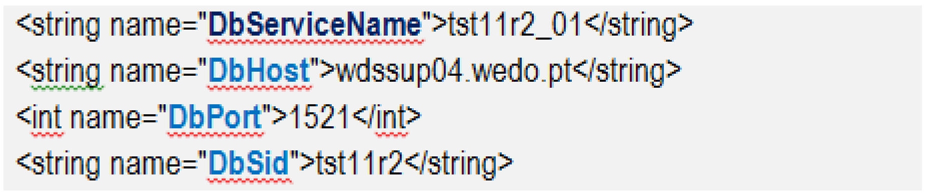
Most of the times some variables used for the Oracle connectivity are misunderstood when the actual config file is filled. These variables are in blue highlighted here in the following template (**DbServiceName, DbSid**):



```
<?xml version="1.0" encoding="UTF-8"?>
<ic:instance-config xmlns:ic="urn:wedo:infra:instance-config-2.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <global>
          <string name="DbHost">wdssup04.wedo.pt</string>
          <int name="DbPort">1521</int>
          <string name="DbServiceName">tst11r2_01</string>
          <!-- DbSid is based on expression "${DbServiceName}" -->
          <!--string name="DbSid"></string>
          <string name="DBServiceId">orcl</string>
          <!-- DbJdbcUrl for Standard Oracle DB connection is based on expression "jdbc:oracle:thin:@${DbHost}:${DbPort}:${DbSid} -->
          <!-- DbJdbcUrl for Oracle RAC DB connection is based on expression "jdbc:oracle:thin:@//${DbHost}:${DbPort}/${DBServiceId} -->
```

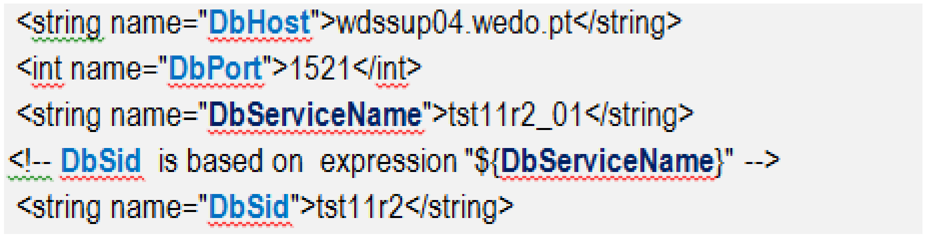
Considerations:

- The comment code "<!-- **DbSid** is based on expression "${**DbServiceName**}" -->" that is located in the above text brought lots of misunderstanding because it assumes already a default value for the "**DbSid**" variable coming from the "**DbServiceName**" variable. This default assignment is not correct because it mixes 2 concepts completely different and most of the time with a specific and different values, and due to this automatic assignment the installation is always stopped on error. Moreover, the "**DbServiceName**" declaration variable is in the middle of the variables used by the JDBC URL connection string ("**DbHost**", "**DbPort**", "**DbSid**"), increasing the misunderstanding of the concepts. It is suggested to take out this automatic assignment in the template in order to always fill the the " **DbSid**" and "**DbServiceName**" variables and it is also suggested to reorganize in the text code the variables declaration order, separating like that the "**DbServiceName**" declaration variable from the variables used by the JDBC URL connection string ("**DbHost**", "**D bPort**", "**DbSid**") like something like this:



```
<string name="DbServiceName">tst11r2_01</string>
<string name="DbHost">wdssup04.wedo.pt</string>
<int name="DbPort">1521</int>
<string name="DbSid">tst11r2</string>
```

instead of this:



```
<string name="DbHost">wdssup04.wedo.pt</string>
<int name="DbPort">1521</int>
<string name="DbServiceName">tst11r2_01</string>
<!-- DbSid is based on expression "${DbServiceName}" -->
<string name="DbSid">tst11r2</string>
```

- Another observation regarding the "**DbServiceName**" variable is that its name suggests that it represents the name of a Database service like the RAC database service, but in fact this variable is used during RAID process installation like a TNS Name that is resolved by an Oracle Resolution Name method. Instead, the RAC database service (New variable "**DBServiceId**") is different from that concept and is managed entirely by the RAC system itself. In order to not misunderstand the meaning of these 2 variables, it is suggested to rename the "**DbServiceName**" variable name into "**DbTnsName**".

3. RAID Disaster Recover Architectures

RAID HA architecture can be complemented with a DR architecture, and like that having a complex but strong HA infrastructure.
This is an example of a RAID High Availability combined with Disaster Recovery architecture:

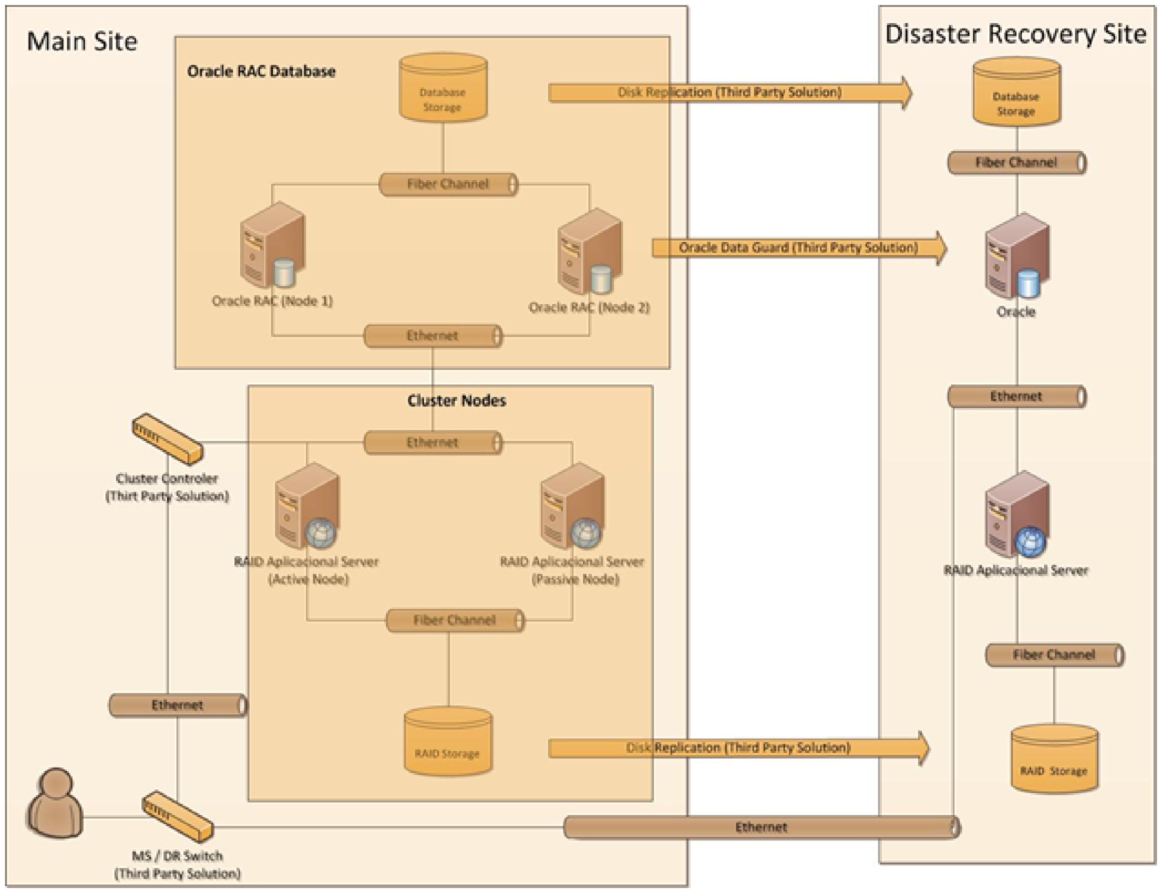


Figure 8 - RAID Disaster Recovery Infrastructure