Maximum Availability Architecture: Oracle E-Business Suite on OCI

July, 2025, Version 1.0

Purpose statement

This document provides a detailed roadmap for implementing Oracle's Maximum Availability Architecture (MAA) for an installation of Oracle E-Business Suite (EBS) 12.2 on Oracle Database 19c in Oracle Cloud Infrastructure (OCI).

It describes an alternative solution to the E-Business Suite My Oracle Support documents describing Business Continuity for Oracle E-Business Suite in Oracle Cloud Infrastructure. This document is appropriate for special situations, such as using logical host names for faster switchover and failover, detailed use of the Terraform tooling for network topology definition, and scripting that handles switchover and failover for the database and application as well as the file system.

Disclaimer

Table of contents

List of figures and tables

1 Purpose

As a corporation's core application and database environment, Oracle E-Business Suite (EBS) is critical to the functioning of a business. If a long-term outage affects the production data center, the fastest way to resume EBS operations is to have a disaster recovery (DR) replica at a remote site, hosting redundant hardware with a copy of the EBS code, tech stack, report and logging output, and database, all kept in sync with production. This secondary or standby environment can also be used to provide service during planned maintenance operations that significantly affect the platform or infrastructure of the primary site, and to provide a last-step sanity check for significant maintenance of EBS itself.

Oracle Data Guard manages replication of data stored in the database from the primary database to the standby. Certain file systems also need to be replicated – the application and middle tier software, and the file-based output generated by the application programs. The files holding application output need to be as close to in sync with the data inside the database as possible, so that if a site failover is required it possible to resolve differences between the state of data in the database and the report output and interface files generated for other systems. The application and middle tier software also needs to be in sync, although it changes less frequently.

This document provides instructions for setting up a disaster recovery (DR) infrastructure for Oracle E-Business Suite Release 12.2 in Oracle Cloud Infrastructure (OCI). It gives a basic set of instructions, configuration hints, and sample scripts for:

- Creating the full standby environment for EBS,
- Keeping it in sync with daily operations and when the primary application and middleware are patched,
- Failing over to it if a disaster occurs,
- Switching to it for planned maintenance of the underlying infrastructure at the primary,
- Opening it in Snapshot Standby mode for application suite sanity testing when making major updates to EBS, and
- Automating the process using Full Stack Disaster Recovery (Full Stack DR)

2 Core Requirements and Assumptions

We have several requirements for this set of instructions to succeed in building a highly available E- Business Suite environment in Oracle Cloud Infrastructure (OCI):

- You have an EBS R12.2 environment already running in OCI, at EBS release 12.2.3 or higher.
- The EBS environment is on AD/TXK C.Delta 14 or higher, including all required patches, as described in My Oracle Support Document 1617461.1: *Applying the Latest AD and TXK Release Update Packs to Oracle E-Business Suite Release 12.2.*
- The database for this environment is version 19c. It is deployed on ExaDB-D, with two or more RAC nodes. See My Oracle Support Document 2368508.1: *Installing Oracle Database 19c Patch Updates for Oracle E-Business Suite with Oracle Exadata Database Service on Dedicated Infrastructure or Cloud@Customer*.
- Users access their EBS sessions via an OCI Load Balancer as a Service (LBaaS), with SSL certificates already configured.
- The application tier is in the same Availability Domain (AD) as the database. There are two or more compute instances in the application tier.
- The tenancy is subscribed to a second OCI region and can provision and manage resources in this second region to create the secondary/standby site.
- The standby site will be provisioned with the same footprint (number and type of each server) as at the primary OCI environment.
- You are on the latest ExaDB-D DBaaS tooling at both primary and standby sites. Note that the tooling in place when we tested this process will be different from the tooling in place when you implement these concepts, resulting in minor differences in the details of a given step.
- You have moved your oraInventory to a location within the EBS application file system. See section 4.5 of My Oracle Support Document 2033780.1: *Oracle E-Business Suite Applications DBA and Technology Stack Release Notes for R12.AD.C.DeltA7 and R12.Txk.C.DELTA7.*

We made a small number of assumptions, simplifying the implementation:

- All network configuration is held in a specific Network compartment. We used the tool Terraform to “discover” the configuration of the network compartment, to simplify replication of the network configuration at the standby site, and to more easily manage creation of network policies.
- The application tier is using a shared file system. You will need to adjust the process documented here if you are not sharing your application file system. File Storage Service (FSS) is not a requirement but is recommended.
- The SCAN name for the database at the primary site is normally not resolvable at the secondary site, and vice-versa. If you decide to configure your network to make them resolvable at each site, you must use either the sqlnet.ora parameter `SQLNET.INVITED_NODES` or `SQLNET.EXCLUDED_NODES` on the database nodes to appropriately limit access. See section 8.2, Configure RUN File System on Secondary Application Servers, for more information.
- You are using SSL. Thus, at your primary, you:
  - Allow SSL termination at the load balancer by uncommenting (removing the # sign from) the context variable `s_enable_sslterminator` on all application tier compute instances.
  - Use PEM format for your SSL certificate files. For further information on certificate prerequisites, please see SSL Certificates for Load Balancers.

- Configure the EBS web entry point to redirect URL and URI requests to the load balancer front-end, using SSL port 443.

There are two requirements that you will need to implement during a downtime window if they are not already in place in your production environment:

- Your database must be in archivelog mode. If it is not, follow these steps to enable it.
  - Shut down all Oracle RAC instances but one.
  - Start a SQL*Plus session as sysdba in the remaining Oracle RAC instance and issue these commands:
    ```
    shutdown immediate;
    startup mount
    alter database archivelog;
    alter database open;
    ```
  - Restart all other Oracle RAC instances. On each of the other database servers:
    ```
    $ srvctl start database -db <DB unique name>
    ```
- This document assumes you have implemented EBS *logical hostnames* on both the database and application tiers of your production OCI installation. If you have not yet done this:
  - Follow the steps in sections 8.2.2 through 8.2.5 in the document Oracle E-Business Suite Release 12.2 Maximum Availability Architecture to implement logical hostnames. *
  - After completing the work in 8.2.5 of Oracle E-Business Suite Release 12.2 Maximum Availability Architecture, edit `/etc/oci-hostname.conf` and set `PRESERVE_HOSTINFO=2` on all application tier OCI compute instances
  - Do a brief sanity check of the application from an end-user perspective and by running `adop -validate`

* Note: If you use a domain name different from the one assigned to the application tier compute instance subnet, you may encounter bug 22153846 – RAPIDCLONE ON THE DB TIER DOES NOT PROMPT FOR DB DOMAIN NAME when running `adcfgclone.pl` as described in step 8.2.5 of Oracle E-Business Suite Release 12.2 Maximum Availability Architecture. If so, you will need execute the steps outlined in Section 15: Appendix.

2.1 Concepts and Terms

Table 2-1. Conventions used in this document.

Convention	Meaning
Application tier	Machines running Forms, Web, Concurrent Processing, and other servers. Sometimes referred to as the middle tier.
Database tier	Machines running the EBS database on Oracle Exadata Database Service on Dedicated Infrastructure.
Primary/Source	Primary EBS database on Oracle Exadata Database Service on Dedicated Infrastructure or primary application tier. This primary site is considered the source of truth, is the source of database and file system replication to the secondary / standby site.
Secondary/Standby/Target	Standby Oracle EBS database on Oracle Exadata Database Service on Dedicated Infrastructure (physical standby) or standby application tier. This site serves as an active

	backup of the primary for both file system and database, and is the target for replication. We use “standby” and “secondary” interchangeably, encouraging the practice of switching back and forth between the environments on a regular basis to assist with maintenance, to ensure the viability of both sites, and to validate disaster recovery readiness.
oracle	User account that owns the database file system (database ORACLE_HOME and associated files).
applmgr	User account that owns the application file system In some implementations, this is implemented as the user oracle.
CONTEXT_NAME	The CONTEXT_NAME variable specifies the name of the applications context that is used by AutoConfig. The default is <PDB name>_<hostname>.
CONTEXT_FILE	Environment variable holding the full path to the applications context file on the application tier or database tier. The default locations are: • Application tier context file: <INST_TOP>/appl/admin/CONTEXT_NAME.xml • Database tier context file: <ORACLE_HOME>/appsutil/<CONTEXT_NAME>.xml
`Monospace text`	Represents command line text. Type such a command exactly as shown, excluding prompts.
`< >`	Text enclosed in angle brackets represents a variable. Substitute a value for the variable text. Do not type the angle brackets.
`\`	On UNIX, the backslash character can be entered at the end of a command line to indicate continuation of the command on the next line. It can also “escape” the next character in a shell command.
TNS	Transparent Network Substrate, a proprietary Oracle computer-networking technology, supports homogeneous peer-to-peer connectivity on top of other networking technologies such as TCP/IP, SDP and named pipes. TNS operates mainly for connection to Oracle databases.
CDB	Container database
PDB	Pluggable database
Oracle RAC	Oracle Real Application Cluster (Oracle RAC) allows multiple servers to run Oracle’s RDBMS software while accessing the same physical database on shared storage.
VIP	Virtual IP address – the IP address used by the Oracle Grid Infrastructure. For example, if the host name is host1, the VIP name will be host1-vip.
SCAN	Single Client Access Name – single name for clients to access any Oracle database instance running in a RAC cluster.
OCI	Oracle Cloud Infrastructure
AD	Availability Domain

VCN	Virtual Cloud Network
CIDR	Classless Inter-Domain Routing

Table 2-2. Top-Level Directories.

Directory	Purpose
ORACLE_HOME	Oracle 19c Database ORACLE_HOME
APPL_TOP	EBS product directories and files.
COMMON_TOP	Directories and files used by many different EBS products, possibly including 3rd party products
OracleAS 10.1.2 ORACLE_HOME	Oracle tools technology stack.
FMW_HOME	FMW_HOME installed by Oracle EBS on the application tier.
INST_TOP	Directory that contains application instance directories and files.
SCRIPT_DIR	Directory holding your custom scripts

3 Provision the Disaster Recovery Environment

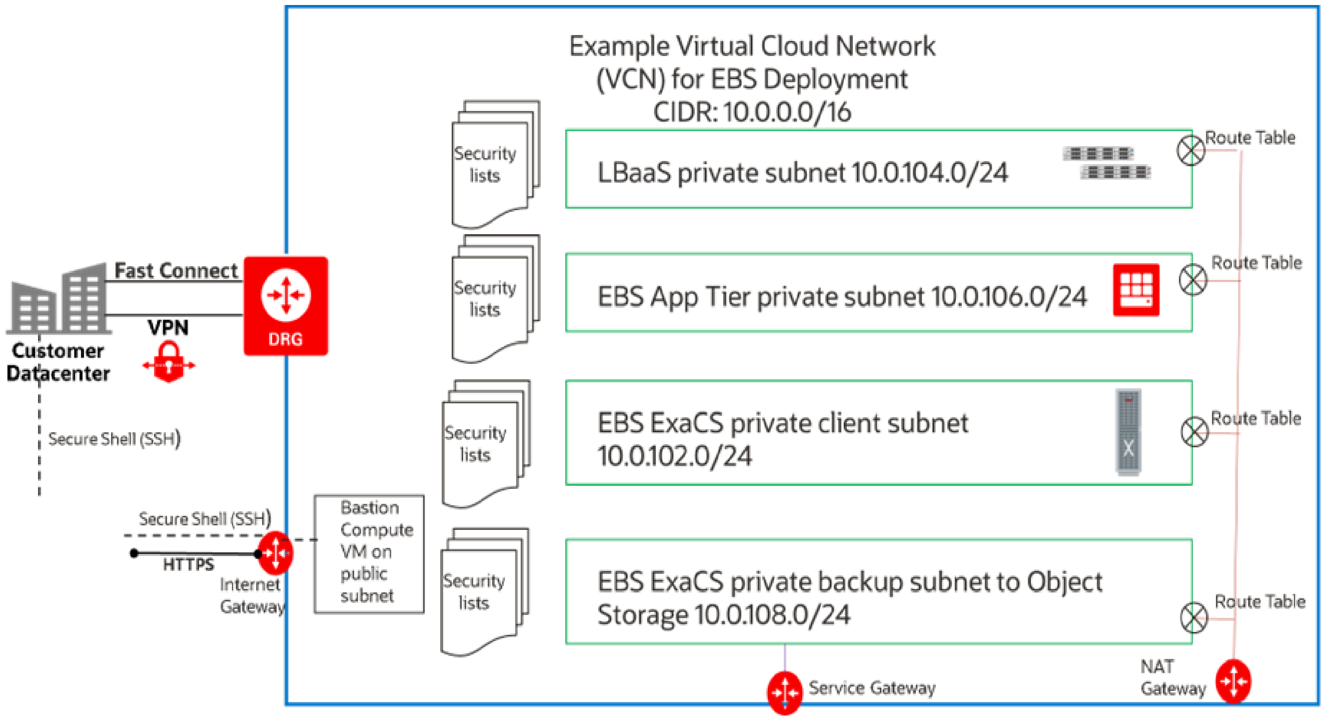
In this section, we will provision the required resources at the secondary site and do the basic configuration. We will first build the network infrastructure, then provision the database tier, the application tier, the application tier shared file system, a pair of servers to manage application tier file system replication, and finally configuring the ports and firewall rules to secure the secondary site. For quicker switchover and failover operations, we will rely on logical hostnames to simplify reconfiguration, which require the basic footprint at the secondary site to be the same as at the primary site.

3.1 Secondary Site Network

The network structure for the core implementation will be the same at the secondary site as at the primary, for both the application and the database tiers. We will be adding a pair of servers to manage application tier file system replication, one at each site, and will adjust the network configuration to handle this pair after the existing network definition is replicated.

The primary site network topology used in this example is shown here:

Figure 3.1. Example VCN for EBS


Example Virtual Cloud Network (VCN) for EBS Deployment CIDR: 10.0.0.0/16
Security lists
LBaaS private subnet 10.0.104.0/24
Route Table
Security lists
EBS App Tier private subnet 10.0.106.0/24
Route Table
Security lists
EBS ExaCS private client subnet 10.0.102.0/24
Route Table
Security lists
EBS ExaCS private backup subnet to Object Storage 10.0.108.0/24
Route Table
Fast Connect
VPN
DRG
Customer Datacenter
Secure Shell (SSH)
Secure Shell (SSH)
HTTPS
Internet Gateway
Bastion Compute VM on public subnet
Service Gateway
NAT Gateway

The network topology image depicts the following:

- A virtual cloud network (VCN) with a CIDR of 10.0.0.0/16
- These subnets within the VCN:
  - LBaaS (load balancer as a service) private subnet with a CIDR block of 10.0.104.0/24
  - EBS App Tier private subnet with a CIDR block of 10.0.106.0/24
  - EBS ExaCS private subnet for Grid and database client network with a CIDR block of 10.0.102.0/24
  - EBS ExaCS private subnet for backups to OCI object storage with a CIDR block of 10.0.108.0/24
  - Bastian public subnet for access via Internet

- Each of the above subnets has one or more associated security lists.
- Each of the above subnets has an associated route table.
- There are several gateways:
  - Internet gateway for traffic to and from the public Internet
  - Network Address Translation (NAT) gateway to access private subnets and for translation between public and private subnets
  - Service Gateway provides access to OCI region local services such as object storage, the local YUM repository for OS images and other services
  - Dynamic Routing Gateway (DRG) to enable traffic routing to other entities outside the VCN such as FastConnect, IPSeC VPNs and other VCNs in other OCI regions.

This network needs to be faithfully replicated at the secondary site while establishing different addresses and display names. To do this, you will discover the primary network using the tool Terraform, creating a Terraform *stack* – a collection of files defining infrastructure resources that you configure, provision, and manage as a unit – for the network. You will make simple edits to the addresses and display names, then use the edited stack to provision the network at the secondary site. The basic method is described in this section. Please see Section 13, Working with Terraform, for a more detailed example of discovering a network configuration on one environment and recreating it on another.

To use Terraform to duplicate your primary site network definition to your secondary site:

- Execute a Terraform “discovery” to discover network resources at the primary region within the tenancy.
- Edit the Terraform files (.tf), creating a new stack.
- Use Terraform “plan” to validate your new stack against the secondary site region and resolve any errors.
- Use Terraform “apply” with your validated stack, to provision the network resources at the secondary region.

Once this is complete, you will adjust the network definition for the replication compute instances.

3.1.1 Discover the Primary Network Configuration

Use the OCI console to execute a Terraform discovery within the network compartment at the primary. This will create a stack holding the structure of your primary network compartment. It will be in the form of a zip file containing several .tf files in JSON format that hold the definitions of all OCI resources within the compartment.

3.1.2 Edit the Terraform Files

You will need to adjust the details of the network configuration held in the stack to completely define the network at your secondary site in a way that does not conflict with your primary site. You will adjust the display names and network addresses to accomplish this, picking new patterns for the secondary site – e.g.:

- If your primary VCN CIDR block is addressed 10.0.0.0/16, you could use 10.10.0.0/16.
- If your primary display names include the airport code for the data center, change the secondary site display names to use the airport code for the secondary data center.

Download the stack to your laptop or workstation and unzip it. Make the following changes to the .tf files:

- Change the VCN CIDR block to your new address. Our test environment had a primary CIDR block address of 10.0.0.0/16, which we changed to 10.10.0.0/16 for the secondary site.

- Change all private subnet addresses to match the new CIDR block just assigned to your secondary VCN. For example, our primary ExaCS private database client subnet is 10.0.102.0/24. We will change this to 10.10.102.0/24 for the secondary.
- Change the display names to reflect resources provisioned at the secondary site. We found it convenient to use each data center's airport code as part of the name, as indicated above.
- Modify the availability_domain.tf file to specify the secondary site availability domain.

When all the required .tf file changes are completed, create a new zip file holding your new stack definition. Include the files `core.tf`, `AD.tf`, and `vars.tf`. Give the zip file a meaningful name – e.g., `<target region>_network_tf_stack.zip`. Use the OCI console to upload this new stack so that it can be deployed at the secondary site.

3.1.3 Validate the Stack, Create a Terraform Plan

Use the Terraform interface in the OCI console to create a "plan" for the uploaded stack.

If the creation of the plan fails, use the console to identify the errors, then edit your original .tf files to make the required corrections. Delete the stack with the failed plan, then once again create your zip file, upload it, create a new stack, and create a plan to validate the stack – repeating this process until the validation is successful.

3.1.4 Provision the Secondary Network

Once there is a valid Terraform plan – one with no errors – use Terraform's "apply" function to provision the network resources defined in the stack.

3.1.5 Configure Network Path for File System Replication

Adjust the network configuration by adding access for the servers that will manage file system replication across the two sites.

- Network access via port 22 will have been configured during provisioning of these compute instances. This is the only port that should be configured for these servers.
- Adjust the network security list in each region to allow the opposite server pair access on port 22.

Once remote peering has been established (see Section 4, Configure Remove VCN Peering), ingress rules can be added to the appropriate security list. Refer to section 4.3 for example security list ingress rules.

Note that there is no need to run any additional firewall-`cmd` commands as these servers will not run any E-Business software.

3.2 Provision the Secondary Site Database Tier Infrastructure

With the secondary site network in place, you can provision the database infrastructure - the hardware and the VM cluster that will ultimately host the database. This is done in three steps:

- Provision your target ExaDB-D infrastructure.
- Provision the VM cluster.
- Configure logical hostnames.

These tasks are presented briefly below. Refer to [Creating an Exadata Cloud Infrastructure Instance](#)for current steps for these tasks.

3.2.1 Provision Secondary ExaDB-D Infrastructure

Using the OCI console, select the Exadata model and shape, then specify the availability domain that will host your infrastructure. Complete and submit the provisioning request and wait until the infrastructure provisioning has completed.

Once the ExaDB-D is provisioned, use the OCI console to review your target environment. Verify your Exadata generation, number of database compute nodes, number of storage cells, etc.

3.2.2 Provision Secondary Exadata VM Cluster

Once the Exadata infrastructure is provisioned, use the OCI console to provision the VM cluster onto the infrastructure.

Select the Grid Infrastructure version, starter database version, OCPU count for the cluster, and ASM disk group storage properties.

You will be prompted for where your backups should be stored. There are three choices:

- Local – the flash recovery area of ASM (not recommended),
- Region-local – the object storage in the secondary region, and
- Autonomous Recovery Service – Zero Data Loss Autonomous Recovery Service (ZRCV).

In general, we expect you to choose the same solution as you have defined in your primary site. Considerations for storing your backups:

- Do not store your backups in local storage / the flash recovery area. This is not a best practice for your EBS database.
- To provision region-local object storage for your backups, first de-select local storage. When local storage is de-selected, the ExaDB-D dialog will present additional fields for specifying the backup subnet and the compartment hosting that subnet.

Complete and submit the provisioning request and wait until the VM cluster is ready. This activity should only take a few hours.

When your VMs are up and running, use the OCI console to verify the environment was created as you specified:

- The correct number of domU compute VM nodes
- Clusterware / Grid Infrastructure
- SCAN name with three IP addresses on the client subnet
- SCAN and grid VIPs with their respective listener
- High redundancy ASM disk groups

Note: You will notice various small additional disk groups which were created automatically to support ACFS. They do not need to be adjusted.

3.2.3 Configure Logical Hostnames for the Secondary Database Tier

This document assumes you have already configured logical hostnames on your primary site. If you have not, schedule a downtime for this action and follow the steps in section 8.2.2 Set Up Logical Hostnames of Database Servers, in the document Oracle E-Business Suite Release 12.2 Maximum Availability Architecture.

You need to make three changes to the `/etc/hosts` files on each of your new database servers, to set them up for logical host names. On each secondary database server, log in as the root user, and open `/etc/hosts` for editing. Then:

1. Inside the "Generated by Exadata" block, add aliases that translate the public IP addresses to the new logical host names. The pattern for the public IP addresses is:

   ```
   <IP Address> <Physical Name with domain> <Physical Name without Domain> <Logical Host Name
   with domain> <Logical Host Name without domain>
   ```

2. Inside the "Generated by Exadata" block: add aliases that translate the client VIP addresses to the new logical hostnames. The pattern for the client VIP addresses is:

   ```
   <IP Address> <DNS-registered VIP Name with domain> <DNS-registered VIP Name without Domain>
   <Logical VIP Name with domain> <Logical VIP Name without domain>
   ```

   Note: To find or verify the DNS-registered VIP's IP addresses, issue this command:

   ```
   srvctl config vip –n <assigned node name>
   ```

   For example:

   ```
   srvctl config vip –n exaadb03
   ```

3. Below the "Added by Configuration Utility" block, add entries that translate the cluster SCAN addresses to a set of logical SCAN names. The pattern for the cluster SCAN addresses is:

   ```
   <IP Address> <Logical SCAN name1>
   <IP Address> <Logical SCAN name2>
   <IP Address> <Logical SCAN name3>
   ```

An example database server `/etc/hosts` file from our test environment:

```
cat /etc/hosts
#### BEGIN Generated by Exadata. ####
127.0.0.1 localhost.localdomain localhost
10.23.27.130 exaadb01.example.com exaadb01 ebsdb1.example.com ebsdb1
10.23.27.131 exaabd02.example.com exaabd02 ebsdb2.example.com ebsdb2
10.23.52.48 exaa01-vip.example.com exaa01-vip ebsdb1-vip.example.com ebsdb1-vip
10.23.52.49 exaa02-vip.example.com exaa02-vip ebsdb2-vip.example.com ebsdb2-vip
#### END Generated by Exadata ####
…
#### BEGIN Added by Configuration Utility ####

…

#### END Added by Configuration Utility ####

### SCAN IP addresses for EBS 12.2
10.23.52.53 ebsdb-scan1
```

```
10.23.52.54 ebsdb-scan2
10.23.52.55 ebsdb-scan3
```

When you have completed your edits, make sure you can ping all the new host aliases from each of the secondary site database nodes. For example, in our environment we ran these ping commands on each Oracle RAC node:

```
ping ebsdb1
ping ebsdb2
ping ebsdb1-vip
ping ebsdb2-vip
ping ebsdb-scan1
ping ebsdb-scan2
ping ebsdb-scan3
```

3.3 Provision the Secondary Site Application Tier Infrastructure

There are several components to provision at the secondary site for the application tier:

1. Application tier compute instances
2. RPMs, OS groups and users, and logical hostnames
3. File System Service
4. File system on each application compute instance
5. Authorized access across application compute instances

We will also be provisioning and configuring a new pair of compute instances in this tier, to manage synchronization of the application file system from the primary site to the secondary site. To accomplish this, we need a pair of OCI compute instances (one in each region) that:

- Can access and mount the application tier RUN file systems in that region,
- Do not run any EBS software, or any Oracle client software beyond SQL*Plus, which is used by the file system replication script. This allows file system replication to continue when the standby is in snapshot standby mode for configuration or testing,
- Have the applmgr (or oracle) user created with the exact same uid and gid as the EBS application servers,
- Have OS user equivalency for the applmgr (or oracle) user configured across sites, and
- Have the OCI command-line interface (CLI) installed and configured.

Each OCI compute instance of this pair need only mount the file system used by EBS and source the RUN file system. The steps to create and configure these compute instances are included below.

3.3.1 Provision Compute Instances

You will provision compute instances at the secondary site for the application tier, and a pair of compute instances - one at each site - for file system replication.

1. Use the OCI console to review the configuration of the existing application tier compute instances at the primary site, documenting:
   - Number of compute instances
   - Physical memory size of each compute instance

- Number of OCPUs for each compute instance
- Boot block volume storage size for each compute instance

2. Use the OCI console to replicate this configuration at the secondary site. Provision the same number of compute instances, each with the same configuration. Ensure these compute instances are provisioned in the EBS application tier private subnet at the secondary site.
3. In addition to the compute instances serving your application tier, we recommend you provision a new pair of compute instances to handle file system replication across the two sites, one at each site. The default configuration of 2 OCPUs and 16GB memory should serve. If you need to consistently run several rsync processes in parallel, you may need to adjust the size. Monitor your activity and use OCI's elastic configuration to adjust as needed.

3.3.2 Add Application Tier RPMs, OS Groups and Users, and Logical Hostnames to Application Compute Instances

NOTE: Only perform these steps on the application tier compute instances. Do not do these steps on either the file system replication compute instances or the ExaDB-D cloud service compute nodes.

On each application tier compute instance at the *secondary* site:

1. Follow steps 1 – 4 in section "Oracle E-Business Suite Preinstallation RPM (available for Oracle Linux 6, 7 and 8)" in My Oracle Document 1330701.1: *Oracle E-Business Suite Installation and Upgrade Notes Release 12 (12.2) for Linux x86-64*.

   These steps will install the RPMs required by the EBS application tier and will create the OS users oracle and applmgr with appropriate user limits. Going forward, use the same OS user that owns the file system structures at the primary application tier. In our environment we use applmgr, which is reflected in the examples in this document.
2. Set each compute instance up for logical hostnames, matching the names node for node:
   a. Log in as the root user

      Add aliases that translate the public IP addresses to the new logical host names. The pattern for the public IP addresses is:
      ```
      <IP Address> <Physical Name with domain> <Physical Name without Domain> <Logical
      Host Name with domain> <Logical Host Name without domain>
      ```

      Add aliases for the client VIP addresses of the database nodes to the new logical hostnames. The pattern for the client VIP addresses is:

      ```
      <VIP IP Address> <Logical VIP Name with domain> <Logical VIP Name without domain>
      ```

      Note: To find or verify the DNS-registered VIP's IP addresses, issue this command on the database nodes:
      ```
      srvctl config vip –n <assigned node name>
      ```

3. Add entries that translate the cluster SCAN addresses to a set of logical SCAN names. The pattern for the cluster SCAN addresses is:
   ```
   <IP Address> <Logical SCAN name1>
   <IP Address> <Logical SCAN name2>
   <IP Address> <Logical SCAN name3>
   ```

4. To preserve these changes across reboots, edit the `/etc/oci-hostname.conf` file and set the parameter `PRESERVE_HOSTINFO=2`

An example `/etc/hosts` file, for a server called phxvisprd-app01:

```
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
## Logical hostnames
10.25.55.14 phxvisprd-app01.example.com. phxvisprd-app01 ebsapp1.example.com ebsapp1
10.25.55.16 phxvisprd-app02.example.com phxvisprd-app02 ebsapp2.example.com ebsapp2
# DB Tier
10.23.27.130 exaadb01.example.com exaadb01 ebsdb1.example.com ebsdb1
10.23.27.131 exaabd02.example.com exaabd02 ebsdb2.example.com ebsdb2
10.23.52.48 exaa01-vip.example.com exaa01-vip ebsdb1-vip.example.com ebsdb1-vip
10.23.52.49 exaa02-vip.example.com exaa02-vip ebsdb2-vip.example.com ebsdb2-vip
### SCAN IP addresses for EBS 12.2
10.23.52.53 ebsdb-scan1
10.23.52.54 ebsdb-scan2
10.23.52.55 ebsdb-scan3
```

The `/etc/hosts` file for the second application tier node in our test environment (phxvisprd-app02) is similar to the one above.

3.3.3 Add OS Users to Replication Compute Instances

On the new file system replication compute instances at *each* site:

- Create the OS users oracle and applmgr with the same configuration as the application tier compute instances.

These compute instances do not need the rest of the configuration described in section 3.3.2.

3.3.4 Provision Secondary Site Application Tier File System (FSS)

You will provision a duplicate of your application tier file system at the secondary site. Use the OCI console to verify and document the configuration of your primary site's application tier file system. Look for:

- Which compartment hosts the file system,
- The root directory structure for the R12.2 application file system,
- How that is exported, and
- How the export is configured.

In our examples, we assume the use of OCI's File System Service (FSS), and that FSS is contained in the EBS application tier compartment.

To provision the secondary site file system, log in to the OCI console, switch to the region of the secondary site, and follow these steps:

1. From the main menu, select **Storage**, then **File Systems** under **File Storage**
2. Change the compartment to where you want the file system to be placed: e.g., ebs-app-compartment
3. Click **Create File System**
4. Select **File System for NFS**

5. Under **File System Information**, click **Edit details**
    a. Change the default name to a name of your choosing, e.g., PHX_EBS_APP_FS
    b. Make sure the availability domain is the one your secondary compute instances were provisioned, e.g., US-PHOENIX-AD1
    c. Select the compartment where the file system should be placed: e.g., ebs-app-compartment
    d. Select which encryption option you desire. We used the default: Oracle Managed Keys.
6. Under **Export Information**, click **Edit Details**
    a. Provide an export directory path
    b. If required, check the checkbox for secure exports. See the information icon next to this option for more details.
7. Under **Mount Target Information**, click **Edit details**
    a. Verify or create your mount target. Select the *Click Here to Enable Compartment selections* link to expand the list of compartments, and move to the compartment that will contain your mount target:
        i. If you have already set up your mount targets in this region, select the compartment that contains the mount target for this file system.
        ii. If you need to set up the desired mount target in this region, click on the compartment you want to define it in. Configure the new mount target as required.
8. Click **Create**

Your secondary site application tier file system will be provisioned.

3.3.5 Configure File System on Each Middle Tier Server

OCI will provide the information you need to configure your application servers to use your new file system. To gather this data, after the file system is provisioned:

1. Log in to the OCI console and switch to the secondary region.
2. From the main menu, select **Storage**, then **File Systems**.
3. Select the compartment that contains the file system.
4. Select the name of the file system you provisioned.
5. In the **Exports** table, click on the **Export path** link to see the export target for your file system.
6. Click on **Mount Commands**. A new window will display, providing the ingress and egress rules and the commands used to mount the file system. Use the **Copy** button to copy the mount commands. Paste them into a text document.

    Note: If you used Terraform to duplicate the network definition from the primary to the secondary site, the ingress and egress rules will already be in place. If they are not, highlight and copy the ingress and egress rules for use in the next step. Paste them into the text document.

    The mount commands to copy:
    a. Install the NFS client – the `sudo yum install` command.
    b. Create the mount directory – the `sudo mkdir` command,

c. Mount the file system.

7. If necessary (Terraform not used, need to implement the ingress / egress rules), edit the security list associated with the subnet that will be used to mount FSS, adding the ingress and egress rules collected in the prior step.

To use the information: On each application server at the standby site, and on the replication servers at both the primary and standby sites:

1. Log in to each compute instance and become root
   ```
   $ sudo su – root
   ```
2. Edit /etc/fstab and add an entry for the new FSS file system:
   ```
   <IP Address provided from step 6 above>:/<export path from step 6 above> /u02 nfs
   rw,rsize=131072,wsize=131072,bg,hard,timeo=600,nfsvers=3 0 0
   ```
3. Save the fstab file
4. Create the mount directory:
   ```
   # mkdir /u02
   ```
5. Mount the file system:
   ```
   # mount /u02
   ```
6. Create the base directory exactly the same as on the primary. On our system:
   ```
   # mkdir -p /u02/app/ebs/visprd
   # chmod applmgr:oinstall -R /u02/apps/ebs
   ```

3.3.6 Establish Secure Access Across Application and Replication Compute Instances

Once the users are created and the file system is configured, you need to generate a public/private key pair for each application tier compute instance, including the replication compute instances. You will add each compute instance's public key to every compute instance's authorized_keys file in the middle tier. Follow these steps:

1. On each application tier compute instance, as applmgr (or oracle), generate SSH key pairs, accepting the default file and not specifying a passphrase:
   ```
   $ /usr/bin/ssh-keygen -t rsa -b 2048
   Generating public/private rsa key pair.
   Enter file in which to save the key (<your home directory>/.ssh/id_rsa): <-- Accept the
   default
   Enter passphrase (empty for no passphrase): <-- Press Enter
   Enter same passphrase again: <-- Press Enter
   Your identification has been saved in <your home directory>/.ssh/id_rsa.
   Your public key has been saved in /home/applmgr/.ssh/id_rsa.pub.
   The key fingerprint is:
   SHA256:/YUBjJ3tl3wQuf3F0rP5kg1C6rswvX4WDAkQX5ED5z0 applmgr@phxvisprd-app01.example.com
   The key's randomart image is:
   +---[RSA 2048]----+
   | oo.=== .o |
   ```

```
| .o+*o. o |
| ...+E. B |
| .o o+*.*|
| S .=...o*|
| ...+..o.|
| o.. .o +.|
| o..o o o|
| .== . |
+----[SHA256]-----+
```

The key pair files are written to the user's .ssh directory `/home/applmgr/.ssh`. The private key file is `id_rsa` and the public key file is `id_rsa.pub`.

On each application tier compute instance, as applmgr (or oracle), add the public key generated in the prior step to a temporary file on your shared file system. Create the file when adding in the first public key entry:

```
$ cat id_rsa.pub > /u02/app/ebs/scratch/auth_key_entries
```

Then append the public key entries for each of the other application tier compute instances:

```
$ cat id_rsa.pub >> /u02/app/ebs/scratch/auth_key_entries
```

2. On each application tier compute instance, as applmgr (or oracle), append this temporary work file to the `authorized_keys` file:

   ```
   $ cat /u02/app/ebs/scratch/auth_key_entries >> authorized_keys
   ```

3. Remove your temporary work file holding each instance's public key:

   ```
   $ rm /u02/app/ebs/scratch/auth_key_entries
   ```

When this is completed, you should be able to ssh from any application tier node to any other without being prompted for a password. Verify this by issuing a command like this from each application tier node to each other, substituting the name of one of your other application tier nodes for each execution:

```
$ ssh ebsappN “date”
```

3.4 Configure Ports and Firewall Rules for Secondary Site Compute Instances

The firewall configuration of the application tier servers at the primary and secondary sites must match – e.g.,

- The port numbers must be the same,
- The firewall rules for each application tier compute instance must allow traffic from all other application tier compute instances, and
- The firewall rules must allow traffic to and from the WLS Admin server, node manager, and the load balancer.

3.4.1 Discover the Primary Site Firewall Rules

On each primary site application server, log in and become root, then list the firewall rules:

```
$ sudo su - root
# firewall-cmd --list-rich-rules
```

Note that on the WLS admin server there will be extra ports, as it will need one for the RUN file system, one for the PATCH file system, and one for managing the WLS managed servers.

This is an example listing from the Weblogic admin server in our test environment:

```
rule family="ipv4" source address="10.0.103.0/24" port port="7001" protocol="tcp" accept
rule family="ipv4" source address="10.0.103.0/24" port port="7051" protocol="tcp" accept
rule family="ipv4" source address="10.0.103.0/24" port port="8000" protocol="tcp" accept
rule family="ipv4" source address="10.0.103.0/24" port port="5556" protocol="tcp" accept
rule family="ipv4" source address="10.0.104.0/24" port port="8000" protocol="tcp" accept
```

This is an example from a Weblogic managed server in our primary environment:

```
rule family="ipv4" source address="10.0.103.0/24" port port="8000" protocol="tcp" accept
rule family="ipv4" source address="10.0.104.0/24" port port="8000" protocol="tcp" accept
```

In the above examples:

- Ports 7001 and 7051 are the WLS admin ports for the RUN and PATCH file systems, as defined in the context file variable `s_wls_adminport`.
- Port 8000 is the front-end http listener port for accessing the application, as defined in the context file variable `s_webport`.
- Port 5556 is the node manager port used to manage the WLS managed servers, as defined in the context file variable `s_nmport`.
- Subnet CIDR block 10.0.103.0/24 is the private application tier subnet
- Subnet CIDR block 10.0.104.0/24 is the private load balancer subnet.

3.4.2 Script the Secondary Site Firewall Rules

We recommend you write, then execute, small scripts to configure your firewall commands. In the examples here, we used the IP address ranges and port numbers from the data gathered above.

The firewall-command has the following syntax:

```
firewall-cmd --zone=public --add-rich-rule 'rule family=ipv4 source address=<network IP address
| CIDR block> port port=<port | port range> protocol=<network protocol> accept' –permanent
```

For the WLS Admin server, your script will add the rules for the run and patch file systems (here using ports 7001 and 7051), the managed server node manager (here using port 5556), and the normal application communications, then will reload the firewall to have the configuration changes take effect. On our system, we used these commands:

```
# firewall-cmd --zone=public --add-rich-rule 'rule family=ipv4 source address=10.10.103.0/24
port port=7001 protocol=tcp accept' –permanent
# firewall-cmd --zone=public --add-rich-rule 'rule family=ipv4 source address=10.10.103.0/24
port port=7051 protocol=tcp accept' –permanent
# firewall-cmd --zone=public --add-rich-rule 'rule family=ipv4 source address=10.10.103.0/24
port port=5556 protocol=tcp accept' –permanent
```

```
# firewall-cmd --zone=public --add-rich-rule 'rule family=ipv4 source address=10.10.103.0/24
port port=8000 protocol=tcp accept' –permanent
# firewall-cmd --zone=public --add-rich-rule 'rule family=ipv4 source address=10.10.104.0/24
port port=8000 protocol=tcp accept' –permanent
# firewall-cmd –reload
```

At each application tier managed node, you will just need the two rules added for normal application communications. The scripts for these nodes will need these three lines, with your IP configuration and port number:

```
# firewall-cmd --zone=public --add-rich-rule 'rule family=ipv4 source address=10.10.103.0/24
port port=8000 protocol=tcp accept' –permanent
# firewall-cmd --zone=public --add-rich-rule 'rule family=ipv4 source address=10.10.104.0/24
port port=8000 protocol=tcp accept' --permanent
# firewall-cmd –reload
```

Notes: In our environment:

- Subnet CIDR block 10.10.103.0/24 is the private application tier subnet
- Subnet CIDR block 10.10.104.0/24 is the private load balancer subnet

3.4.3 Update the WLS Admin Console Access Filter

You will also need to update the WLS Admin console access filter for the above rules. See Only Allow Access to Oracle WebLogic Server Administration Ports from Trusted Hosts, and modify the context variable `s_wls_admin_console_access_nodes` according to the steps in that document.

4 Configure Remote VCN Peering

Remote VCN *peering* is the process of connecting two VCNs in one tenancy that are in different regions. The peering allows the VCNs' resources to communicate securely using private IP addresses, without routing the traffic over the internet or through an on-premises network.

4.1 Peer the Primary and Secondary Networks

In section 3.1, you configured the virtual cloud network (VCN) at the secondary site to be the same as at the primary. In this step, you will follow the documentation Peering VCNs in different regions through a DRG to peer the two networks. This will:

- Provision a Dynamic Routing Gateway (DRG) at each region.
- Attach a Remote Peering Connection (RPC) to each DRG, to define the peering between the VCNs at the two regions.
- Implement an explicit agreement as an IAM policy for each VCN, agreeing to the peering relationship.
- Add route table rules in subnets at each VCN to route traffic for resources that need to communicate across regions. The DRG at each VCN has a route table specific for remote VCN peering.
- Add security list ingress and egress rules to the subnets that will be allowed to have traffic between regions.

4.2 Allow Traffic to Traverse the Regions

Once you have peered the two networks, you need to update route tables at both regions to allow traffic to traverse. The following tables provide examples from our case study. The rows containing the target type of "Dynamic Routing Gateway" represent the rules that route traffic through that region's local DRG to the DRG at the other region, via the remote peering connection (RPC).

Note: we capitalized the letters DRG in the names of our dynamic routing gateways to make it easy to see that these are gateways across regions.

Updated route tables in the Ashburn region, our primary site:

Table 4-1. Primary site db-private-RT.

DESTINATION	TARGET TYPE	TARGET
0.0.0.0/0	NAT Gateway	maa-ngw
10.10.102.0/24	Dynamic Routing Gateway	iad-cloudmaa-vcn-DRG
All IAD Services in Oracle Service Network	Service Gateway	maa-lad-sgw

Table 4-2. Primary site app-private_RT

DESTINATION	TARGET TYPE	TARGET
0.0.0.0/0	NAT Gateway	maa-ngw
10.10.106.0/24	Dynamic Routing Gateway	iad-cloudmaa-vcn-DRG

Updated route tables in the Phoenix region, our secondary site:

Table 4-3. Secondary site db-private_RT.

DESTINATION	TARGET TYPE	TARGET
0.0.0.0/0	NAT Gateway	maa-ngw
10.0.101.0/24	Dynamic Routing Gateway	phx-maacloud2-vcn-DRG
All PHX Services in Oracle Service Network	Service Gateway	maa-phx-sgw

Table 4-4. Secondary site app-private-RT.

DESTINATION	TARGET TYPE	TARGET
0.0.0.0/0	NAT Gateway	maa-ngw
10.0.103.0/24	Dynamic Routing Gateway	phx-maacloud2-vcn-DRG

5 Provision the Database Software Image at the Secondary Site

You will provision an Oracle database software image at the secondary site that is at the same database version and patch level as the primary database, then use that image to create the database homes at the secondary site. The database homes will be customized for EBS in a later step.

Note: There is a new feature in OCI that allows you to copy an Oracle database software image to an environment in another region. If you have a custom database software image of your primary database available, skip the first two sections here and source that image for your secondary site database home in section 5.3.

5.1 List the Patches Applied to the Primary Database Home

If you need to create an image to deploy at your secondary site, you will start by listing all the patches applied to the primary database Oracle home.

Log in to the OCI console. Select the region of your primary EBS environment. Open the main menu, then:

1. Select **Oracle Exadata Database Service on Dedicate Infrastructure**.
2. In the left-hand panel, under **Resources**, select **Database software images**.
3. Select the custom database software image that is used by the primary database.
4. Select the **Copy All** link next to **One-Off Patches.** This will copy the comma-delimited list of all patches applied to the selected image to your clipboard.
5. Open a new document in a text editor. Paste in the list of patches. Save your document for use in the next step, where you create the standby database software image.

5.2 Create a Database Software Image at the Secondary Site

If you are creating a new image for your secondary site, follow these steps to create a software image that matches the version, release, and patch level of the database at the primary site.

Log in to the OCI console. Select the region hosting your standby environment. Open the main menu, then:

1. Select **Oracle Databases**.
2. Select **Oracle Exadata Database Services on Dedicated Infrastructure**.
3. In the left-hand panel, under **Resources**, select **Database software images**.
4. Select **Create Database Software Image**.
    a. Provide the information needed to create the correct software image:
    b. Give the image a name similar to the name at the primary.
    c. Select the compartment that will host the image (e.g., exa-db-compartment)
    d. Choose the service: Oracle Exadata Database on Dedicated Infrastructure
    e. From the combo box dropdown, select the database version
    f. From the combo box dropdown, select the database release update (RU)
    g. In the box labeled **Enter one-off patch numbers (optional)** , paste the list of patches copied from the primary in step 5 in section 5.1
5. Click **Create**

The custom image will be created, including both the database and Oracle Clusterware (OCW) patches.

NOTE: If a listed patch has been superseded by another, the newer patch will be applied to the image created at the standby.

5.3 Provision the Oracle Home at the Secondary Site

To provision the oracle home to each Exadata database node at the secondary site:

1. Log in to the OCI console and switch to the region that will host the standby.
2. Open the main menu, and select **Oracle Databases**
3. Select **Oracle Exadata Database Service on Dedicated Infrastructure** on the main page.
4. Select **Exadata VM Clusters**.
5. On the main page in the table under **Exadata VM clusters in <compartment name>…**, select the link for your Exadata VM Cluster (e.g., PHX-Exa-VMCluster-1). The page for the VM cluster is displayed.
6. Under **Resources** (left side) click on **Oracle Homes**. The page changes to display a table under **Database homes**.
7. Click the **Create database home** button to display the **Create database home** page. Enter the following:
    a. For the Database Home display name, enter a name of your choosing.
    b. For the database image, click the **Change database image** button. A new page is displayed.
    c. For **Image Type**, there are two options – **Oracle Database Software Images** and **Custom Database Software Images**. Select **Custom Database Software Images.**
    d. In the compartment dropdown combo box, select the compartment that the custom image was created in.
    e. In the region dropdown combo box, select the region that the custom image was created in.
    f. For the **Select a Database version**, select the correct database version – e.g., 19c.
    g. There will be a table listing all the custom database software images to choose from. Check the checkbox next to the correct image.
    h. Press the **Select** button outside of the table. You will be taken back to the previous page, which will now show the custom software image you chose in the previous step.
8. Navigate to the **Create** button and select it. This will start the work request to begin creating the oracle home on each RAC node at the secondary site, using the custom software image.

5.4 Configure the Oracle Homes

Once the database home provisioning is complete, configure each secondary site RAC node for EBS:

1. Create the 9idata NLS directories on each database node at the secondary site. On each database node, log in as the user oracle, then execute the following command:

```
$ perl $ORACLE_HOME/nls/data/old/cr9idata.pl
```

2. Log in to each database node at the primary site and:
    a. Copy each of these two directories with their contents to the same location at the corresponding database node at the secondary site:

```
$ORACLE_HOME/network/admin/<PDB_NAME>_<NODE_NAME>
$ORACLE_HOME/appsutil
```

b. Copy these two environment files to the same location at the corresponding database node at the secondary site:

```
$ORACLE_HOME/CDB_<node_name>.env file
$ORACLE_HOME/PDB_<node_name>.env file
```

6 Copy: Physical Standby Database, EBS File System

Use Oracle Data Guard Association to set up the physical standby database at the secondary region, then add role-based services for EBS. While the initial copy of the database is underway, copy the EBS application tier file system to the secondary site.

6.1 Copy and Configure the Standby Database

OCI will collect the information needed to enable Oracle Data Guard Association to establish your standby database, then will manage the process for you.

1. Log in to the OCI console and switch to the region that hosts the primary EBS database.
2. From the main menu, select **Oracle Databases** then **Oracle Exadata Database on Dedicated Infrastructure**
3. Change compartments (in left side region of the page) to the compartment that holds the primary Exadata infrastructure and VM cluster.
4. In the table on the main page, click on the **VM cluster** from the list where the EBS primary database is located. The page for the VM cluster will be displayed.
5. Below the VM cluster section of the page is a table listing databases. Click on the EBS primary database. The database page for the selected database will be displayed.
6. On the left side under **Resource**, click on the link **Data Guard Associations**. The page changes to show Data Guard Associations and a table below the Database Information section. There should be no rows in the table.
7. Click the **Enable Data Guard** button above the table. The Enable Data Guard dialog page will be displayed.
8. Under **Select VM Cluster**, make the appropriate selections from each combo box and, if required, change to the correct compartment with the **Change Compartment** link:
    a. Select the peer region where the standby has been configured.
    b. Select the availability domain for where the standby Exadata infrastructure was provisioned.
    c. Select the Exadata infrastructure hosting the standby VM cluster.
    d. Select the Exadata VM Cluster that will host the standby database.
9. Under **Data Guard Details**, select either **Active Data Guard** or **Data Guard** (default)
10. For **Choose Database Home**, select the radio button: **Select an existing database home**
11. Select the oracle database home that was provisioned in section 5.3 above.
12. Under **Configure Standby Database**, do the following:
    a. Optional: Provide a database unique name. If you leave this field blank, the system will generate one for you.
    b. Enter the administrator's (SYS) password. It must be the SYS password of the primary database.
13. Click the **Show Advanced Options link**.
14. Under **Management**, there is a field to enter the Oracle SID prefix. Enter the same Oracle SID prefix as that of the primary database. For example, if the primary database has two instances with SIDs of EBSCDB1 and EBSCDB2, then the SID prefix would be EBSCDB without any of the numbers. IMPORTANT: DO NOT enter

the PDB name for the SID prefix. Failure to follow this step will result in having to reconfigure EBS for each future switchover or failover.

15. Click the **Enable Data Guard** button to start the prechecks and the instantiation of the physical standby database.

Monitor progress, check for any errors reported to the console, and address as needed.

6.2 Copy the EBS Application Tier File System

The entire EBS installation must be copied from the primary application tier file system to the secondary site, to exactly the same directory structure. You can start this process after completing the steps in section 5, while the database is being copied.

We used rsync to copy the file system from the primary to the secondary, so that rsync could then be used to keep the two sets of file systems in sync.

Log in to the primary application tier compute instance as applmgr (or oracle) and run rsync:

```
$ time rsync -avzh –progress/<Base directory>/<EBS system name>/ username@< IP Address of
compute instance at DR site>:/<base directory>
```

For example, in our case study the base directory is /u02/app/ebs and visprd is the name of the EBS system which is also the name of the EBS PDB. We issued this command:

```
$ time rsync -avzh –progress /u02/app/ebs/visprd/ applmgr@<IP Address of compute instance at DR
site>:/u02/app/ebs/visprd/
```

In the above example, the base directory is `/u02/app/ebs` and the EBS system name is visprd. The EBS system name is based on the EBS PDB name in the database.

The rsync command will copy the entire EBS R12.2 installation. This will take some time to complete.

6.3 Add Role-Based Database Services for EBS

Once the standby database is established (step 5.1), the role-based database services configured at the primary can be configured at the standby. Note: the roles configured at your standby must match those at your primary.

The pattern for the command to add a service is:

```
$ srvctl add service -db <db_unique_name> -pdb <pluggable_database> -service <service_name> -
preferred "<database instances>" -notification TRUE -role "PRIMARY,SNAPSHOT_STANDBY" -
failovermethod BASIC -failovertype SELECT -failoverretry 10 -failoverdelay 3
```

To create your services, log in to one of the standby database nodes as the user oracle, source the CDB environment, and create the services needed by the user processes in your environment.

We used these commands to create standby database services in our environment:

```
$ . ./EBSCDB.env
$ srvctl add service -db EBSCDB_phx3xd -pdb VISPRD -service VISPRD_PCP_BATCH -preferred
"EBSCDB1,EBSCDB2" -notification TRUE -role "PRIMARY,SNAPSHOT_STANDBY" -failovermethod BASIC -
failovertype SELECT -failoverretry 10 -failoverdelay 3

$ srvctl add service -db EBSCDB_phx3xd -pdb VISPRD -service VISPRD_OACORE_ONLINE -preferred
```

```
"EBSCDB1,EBSCDB2" -notification TRUE -role "PRIMARY,SNAPSHOT_STANDBY" -failovermethod BASIC -
failovertype SELECT -failoverretry 10 -failoverdelay 3

$ srvctl add service -db EBSCDB_phx3xd -pdb VISPRD -service VISPRD_FORMS_ONLINE -preferred
"EBSCDB1,EBSCDB2" -notification TRUE -role "PRIMARY,SNAPSHOT_STANDBY" -failovermethod BASIC -
failovertype SELECT -failoverretry 10 -failoverdelay 3
```

7 Complete Secondary Database Configuration

When the standby database instantiation is complete and EBS database services are available, you can finish configuring the database home. Use Oracle Data Guard's Snapshot Standby feature to temporarily open the standby database, to complete this configuration.

7.1 Place the Standby in SNAPSHOT_STANDBY Mode

On one of the standby database nodes, as the user oracle, place the database into SNAPSHOT STANDBY mode:

```
$ dbaascli dataguard convertStandby --dbname <standby DB name> --standbyName <standby DB unique
name> --standbyType snapshot
```

7.2 Start Database Services at the Standby

When the database is available in snapshot standby mode, as the user oracle, source the CDB environment then start the role-based services just created.

In our environment, we issued these commands from the user oracle's home directory:

```
$ . ./EBSCDB.env
$ srvctl start service -db EBSCDB_phx3xd -service VISPRD_OACORE_ONLINE
$ srvctl start service -db EBSCDB_phx3xd -service VISPRD_FORMS_ONLINE
$ srvctl start service -db EBSCDB_phx3xd -service VISPRD_PCP_BATCH
```

7.3 Update Configuration Files

On each ExaDB-D compute node at the standby, run the EBS-supplied perl script `txkSyncDBConfig.pl` to generate updated configuration files at the standby.

The script requires parameters that can be specific to each node. You need to edit the script on each node to customize it for your environment. Tips:

- To obtain the SCAN name at the standby site, issue this command from any one of the standby database nodes as the oracle user:

  ```
  $ . ./<CDB_NAME>.env
  $ srvctl config scan
  ```
- The `txkSyncDBConfig.pl` example is specifying logical hostnames for both the VIP and hostnames. As noted earlier, both the primary and the standby sites must have:
  - Matching logical hostnames for each database node
  - The same network domain portion of both the `virtualhostname` and the `logicalhostname`
  - Entries for `virtualhostname` and `logicalhostname` present in `/etc/hosts` at both sites, as described in Sections 3.2.3 and 3.3.2 above.

Here is the format of the txkSyncDBConfig.pl script, with more hints given in bolded text:

```
$ source $ORACLE_HOME/<PDB_hostname>.env
$ perl $ORACLE_HOME/appsutil/bin/txkSyncDBConfig.pl \
-dboraclehome=$ORACLE_HOME \
-outdir=$ORACLE_HOME/appsutil/out/<PDB_hostname> \
-cdbsid=<CDB SID> \ <-- The Oracle SID for the node this script is running on.
-pdbname=<PDB Name> \ <-- The name of the EBS PDB.
-dbuniquename=<Database Unique Name> \ <-- The DB_UNIQUE_NAME on this cluster.
```

```
-israc=YES \ <-- Set to YES for RAC instances
-virtualhostname=< Logical hostname for the VIP on THIS host > \ <-- The logical hostname configured for the VIP on this host and network domain.
-logicalhostname=<logical hostname.domain> \ <-- The logical hostname for this host and domain.
-scanhostname=<SCAN name.domain at the standby> \ <-- The SCAN name defined on this cluster.
-scanport=1521 \ <-- The SCAN listener port, typically 1521
-dbport=1521 \ <-- The listener port, typically the grid listener on port 1521
-appsuser=APPS <-- The APPS schema name
```

When prompted, enter the APPS password.

Note: The script `txkSyncDBConfig.pl` runs autoconfig. Autoconfig will likely fail, because the `UTL_FILE` directories are not yet configured at the standby. If, after reviewing the autoconfig log file, you see the failure is caused by inability to access `UTIL_FILE_DIR`, proceed to the next step. Autoconfig will run again in that step.

7.4 Conditional: Create UTL_FILE_DIR Directory Structures

Follow the instructions in section 3.1.2 of Document 2525754.1: *Using UTL_FILE_DIR or Database Directories for PL/SQL File I/O in Oracle E-Business Suite Releases 12.1 and 12.2* to create the required directories at the secondary site.

This will create the file `$ORACLE_HOME/dbs/<PDB_Name>_utlfiledir.txt`. Copy this file to the `$ORACLE_HOME/dbs` directory of all RAC database nodes.

Note: the `UTL_FILE` directories only need to be created once. However, if your database home paths are different on the standby environment, the value of `UTL_FILE_DIR` must be updated after each switchover or failover. We provide a script for automating this process later in this document.

7.5 Run autoconfig on the First RAC Node

Autoconfig was run on each database node by the script executed in step 7.3. It needs to be run once more on the first RAC node to complete the configuration: As the user oracle, on the first RAC node:

```
$ cd <ORACLE_HOME>
$ . ./<PDBName_hostname>.env
$ cd appsutil/scripts/<PDBname_hostname>
$ ./adautocfg.sh
```

Address errors if any, and re-run if needed. At this point, autoconfig should complete successfully.

8 Set Up Application Tier at Secondary Site

Note: these actions are completed while the standby database is still in snapshot standby mode.

8.1 Synchronize the EBS File System

Before we configure the application tier at the secondary site, we will use `rsync` once again to copy over any files that have changed since we first copied the file system from the primary site.

Log in to the primary application tier compute instance as applmgr (or oracle) and run the same `rsync` command you issued in Section 6.2 above.

Note that once we configure the middle tiers, we will no longer be able to do this simple one-line copy, but will need to exclude all directories containing configuration information.

8.2 Configure RUN File System on Secondary Application Servers

In this section, we will configure each application server at the secondary site with multi-address TNS connect strings so that, when switching or failing over between environments, it is not necessary to do a full application reconfiguration. We will reconfigure the primary site in this way when the secondary site is fully viable.

Before configuring the application servers at the secondary site, first log in to each application compute instance at the **primary** site, as the application owner. Source the RUN environment and gather the following information:

- The location of the context file on each primary application node:

  ```
  $ echo $CONTEXT_FILE
  ```

- The location of adautocfg.sh on each primary application node:

  ```
  $ which adautocfg.sh
  ```

Now, at the **secondary** site, log in to each application server as the application owner. Do not source the environment, as the environment files are not yet configured.

Change directories to the directory holding the context file on this server, which will be the same as $CONTEXT_FILE at the corresponding application server at the primary. In our environment:

```
$ cd /u02/app/ebs/visprd/fs1/inst/apps/VISPRD_ebsapp1/appl/admin
```

Edit the context file to provide access to the services created in section 6.3, Add Role-Based Database Services for EBS:

1. Make sure the context variable `s_hostname` is set to the logical hostname of this application server. For example, a primary application tier node with a logical hostname of ebsapp1 will have the context variable `s_hostname` set as follows:
   `<host oa_var="s_hostname">ebsapp1</host>`
2. Edit the `apps_jdbc_connect_descriptor` and the `apps_jdbc_patch_connect_descriptor` entries, configuring each DESCRIPTION to have its own set of addresses specifying the SCAN name at each site, so that failover is simple.

   Note: Because, to make failover simpler the connect strings have multiple addresses, as stated in our original assumptions, the SCAN names at each site should not be resolvable at the other site. As stated earlier, if the

SCAN names do resolve at both regions, you must use either the sqlnet.ora parameter `SQLNET.INVITED_NODES` or `SQLNET.EXCLUDED_NODES` on the database nodes to appropriately limit access.

Note that as EBS application patch activities are limited to one RAC node, the `s_apps_jdbc_patch_connect_descriptor` context variable specifies INSTANCE_NAME in its TNS connect string, while the `s_apps_jdbc_connect_descriptor` does not.

```
s_apps_jdbc_connect_descriptor
<jdbc_url
oa_var="s_apps_jdbc_connect_descriptor">jdbc:oracle:thin:@(DESCRIPTION_LIST=(LOAD_BALANCE
=off)(FAILOVER=on)(DESCRIPTION=(CONNECT_TIMEOUT=5)(TRANSPORT_CONNECT_TIMEOUT=3)(RETRY_COU
NT=3)(ADDRESS_LIST=(LOAD_BALANCE=on)(ADDRESS=(PROTOCOL=TCP)(HOST=<primary SCAN
name>)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=<OACORE RUN service
name>)))(DESCRIPTION=(CONNECT_TIMEOUT=5)(TRANSPORT_CONNECT_TIMEOUT=3)(RETRY_COUNT=3)(ADDR
ESS_LIST=(LOAD_BALANCE=on)(ADDRESS=(PROTOCOL=TCP)(HOST=<secondary SCAN
name>)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME= OACORE RUN service name))))</jdbc_url>

s_apps_jdbc_patch_connect_descriptor
<patch_jdbc_url
oa_var="s_apps_jdbc_patch_connect_descriptor">jdbc:oracle:thin:@(DESCRIPTION_LIST=(LOAD_B
ALANCE=off)(FAILOVER=on)(DESCRIPTION=(CONNECT_TIMEOUT=5)(TRANSPORT_CONNECT_TIMEOUT=3)(RET
RY_COUNT=3)(ADDRESS_LIST=(LOAD_BALANCE=on)(ADDRESS=(PROTOCOL=TCP)(HOST=<primary SCAN
name>)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=<OACORE PATCH service
name>)(INSTANCE_NAME=<CDB instance
name>)))(DESCRIPTION=(CONNECT_TIMEOUT=5)(TRANSPORT_CONNECT_TIMEOUT=3)(RETRY_COUNT=3)(ADDR
ESS_LIST=(LOAD_BALANCE=on)(ADDRESS=(PROTOCOL=TCP)(HOST=<secondary SCAN
name>)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=<OACORE PATCH service
name>)(INSTANCE_NAME=<CDB instance name>))))</patch_jdbc_url>
```

An example from our test system:

```
s_apps_jdbc_connect_descriptor
<jdbc_url
oa_var="s_apps_jdbc_connect_descriptor">jdbc:oracle:thin:@(DESCRIPTION_LIST=(LOAD_BALANCE
=off)(FAILOVER=on)(DESCRIPTION=(CONNECT_TIMEOUT=5)(TRANSPORT_CONNECT_TIMEOUT=3)(RETRY_COU
NT=3)(ADDRESS_LIST=(LOAD_BALANCE=on)(ADDRESS=(PROTOCOL=TCP)(HOST=phxexadb-krppw-
scan.example.com)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=VISPRD_OACORE_ONLINE)))(DESCRIP
TION=(CONNECT_TIMEOUT=5)(TRANSPORT_CONNECT_TIMEOUT=3)(RETRY_COUNT=3)(ADDRESS_LIST=(LOAD_B
ALANCE=on)(ADDRESS=(PROTOCOL=TCP)(HOST=iadexadb-bw5wn-
scan.example.com)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=VISPRD_OACORE_ONLINE))))</jdbc_
url>

s_apps_jdbc_patch_connect_descriptor
<patch_jdbc_url
oa_var="s_apps_jdbc_patch_connect_descriptor">jdbc:oracle:thin:@(DESCRIPTION_LIST=(LOAD_B
ALANCE=off)(FAILOVER=on)(DESCRIPTION=(CONNECT_TIMEOUT=5)(TRANSPORT_CONNECT_TIMEOUT=3)(RET
RY_COUNT=3)(ADDRESS_LIST=(LOAD_BALANCE=on)(ADDRESS=(PROTOCOL=TCP)(HOST=phxexadb-krppw-
scan.example.com)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=VISPRD_ebs_patch)(INSTANCE_NAME
=EBSCDB1)))(DESCRIPTION=(CONNECT_TIMEOUT=5)(TRANSPORT_CONNECT_TIMEOUT=3)(RETRY_COUNT=3)(A
DDRESS_LIST=(LOAD_BALANCE=on)(ADDRESS=(PROTOCOL=TCP)(HOST=iadexadb-bw5wn-
scan.example.com)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=VISPRD_ebs_patch)(INSTANCE_NAME
=EBSCDB1))))</patch_jdbc_url>
```

3. While still editing the context file, verify that the context variable `s_enable_sslterminator` does not have the "#" set. For example, the context variable in the context file should be as follows:

   ```
   <sslterminator oa_var="s_enable_sslterminator"/>
   ```

4. When you have finished editing and have saved the context file, run autoconfig:

   ```
   $ cd /u02/app/ebs/visprd/fs1/inst/apps/VISPRD_ebsapp1/admin/scripts
   $ ./adautocfg.sh
   ```

   Address any errors autoconfig may encounter until autoconfig runs successfully.

Repeat these steps for each remaining application compute instance.

8.3 Configure PATCH File System on Secondary Application Servers

Do the work in section 8.2 on each application server at the standby site, for the PATCH environment.

Note that when running autoconfig on the PATCH file system, if there is not an open patch edition you will get an error. This error can be ignored.

8.4 Configure RUN and PATCH File Systems on Primary Application Servers

Repeat the work in sections 8.2 and 8.3 on each application server at the primary site. This allows switchover to be performed in either direction without having to make any adjustments to the connect strings.

8.5 Set Up Regular File System Synchronization

The application tier file systems at the primary will continue to be modified, whether via application / middleware patching exercises or as output is generated and purged from concurrent manager program execution. This file system data needs to be kept current at the standby site in a way that avoids replicating the application tier configuration data that is unique to each server.

The basic requirements:

- Regularly synchronize the file systems holding the relatively unchanging application and middle tier software to the standby
- Aggressively synchronize the file systems holding runtime output, keeping them as up to date as possible at the standby
- Allow file system replication to continue while using the standby database for snapshot standby testing
- Avoid replicating instance-specific configuration data
- Change replication direction based on the role of that region's database

If you have implemented your application tier file system on a service such as ZFS, you can use the synchronization feature of that product to keep your secondary site file system current with changes at the primary. If you are using FSS, you can use the FSS file system synchronization, but be aware that it will only refresh the standby site once per hour. For our tests, we built scripts based on rsync, so that we could keep the secondary site's file system in sync more frequently. See the scripts in Section 14.5 File System Replication (rsync) Scripts for an example. Please note that these sample scripts are examples only, not certified. You need to craft a solution for your environment.

9 Provision, Configure, and Test OCI Load Balancer (LBaaS) at Secondary Site

Note: these actions are completed while the standby database is still in snapshot standby mode.

As stated in the assumptions at the beginning of this document, we are assuming logical host names are configured, and that the OCI LBaaS is configured to provide load balancing services for the environment.

9.1 Prerequisites

These three prerequisites for EBS to be configured with the OCI LBaaS were listed in Section 2:

- On all application tier compute instances, the context variable `s_enable_sslterminator` must have the "#" removed. This configures the EBS HTTP server to allow SSL termination at the load balancer.
- You need to prepare an SSL bundle with certificate files in PEM format. For further information on certificate prerequisites, please see SSL Certificates for Load Balancers.
- The EBS web entry point must be configured to redirect URL and URI requests to the load balancer front-end, using SSL port 443.

There may be other requirements based on security patches that may require changes to the load balancer. As of the October 2023 CPU, there are two mandates:

- The hostname in the login URL for the EBS login page must match the hostname used for the front-end load balancer.
- The load balancer health check URL path must be set to `/robots.txt`.

9.2 Provision and Configure Load Balancer

This section does *not* walk you through the provisioning process of the OCI load balancer because as OCI is continually enhanced, the actual steps change. It does describe the basic information you need to gather and the basic goals of the task. See the OCI documentation on load balancing for the current steps needed to configure the OCI load balancer and the listeners.

You will configure user access to the standby application tier for both (1) normal production use when the secondary site serves as primary and (2) application testing when the secondary site's database is in snapshot standby mode. This section describes configuring the load balancer for operation as standby. We will address setting up the standby environment for testing in section 9.4.

Points 1-3 below are for the load balancer itself. Points 4-6 are for each listener – one for normal traffic and one for use for testing at the standby site when the database is in snapshot standby mode. You will configure the second listener in section 9.4.

Note the replication servers are independent of this exercise – they are not accessed by the application users, not accessed via the load balancer.

These are the configuration choices for our LBaaS load balancer:

1. **LBaaS shape** – choose the shape and characteristics required for your environment.
2. **LBaaS display name** – PHX_EBS_LBaaS_PROD
3. **Backend set**:
    a. **Traffic distribution policy**: "Weighted round robin"
    b. **Session persistence**: "Enable Application Cookie persistence".

c. **Cookie Name**: * (asterisk)

d. **Use SSL**: unchecked
If you choose to have the backend servers SSL enabled, check the checkbox for **Use SSL** and fill in the appropriate information. We did not check this in our test environment.

e. **Health check**: see table below
Health check must be defined for the backend set. It is applied to all available backend servers to determine their health according to your configuration. The load balancer will not route traffic to an unhealthy backend server.
These are the attributes we specified for our health check:

Table 9-1. Load balancer health check.

ATTRIBUTE	VALUE
Protocol	HTTP Since SSL is terminated at the load balancer, HTTP is selected.
Port	8000 HTTP port for all EBS web servers
Interval in milliseconds	15000 Number of milliseconds between each check. 15000ms = 15 seconds
Timeout milliseconds	3000 Number of milliseconds that the check will wait before timing out. 3000 = 3 seconds.
Number of retries	3 Number of attempts to get a response from a given backend server.
Status code	200 The expected successful status code for HTTP GET calls.
URL path (URI)	/robots.txt URL path, now required to be /robots.txt
Response Body RegEx	* Regular expression that allows any response returned from the HTML page to be acceptable.

f. Back ends: Add each application server to this backend set. Do not add the replication servers.

4. **Host Names** – OCI LBaaS provides a facility to create one or more virtual host names that can be associated with one or more LBaaS listeners.
5. **Certificate Name** – Configure SSL and terminate on the load balancer listener. This requires an SSL bundle to be created and uploaded to the load balancer. You will give this SSL bundle a certificate name when you upload it.

   You must specify the host name above when creating this certificate.

The LBaaS listener can accept the following types of SSL certificates:

a. Vendor-signed CA certificates such as Verisign or GoDaddy

b. Self-signed certificates using open-source tools such as OpenSSL to generate the x509 private key, the CA request, and the self-signed certificate. These should only be used for testing purposes. They are not trusted by web browsers, and will require the user to accept the certificate when initiating their session.

   For further details on SSL certificates for LBaaS, please refer to SSL Certificate Management in the Oracle Cloud Infrastructure documentation.

6. **Load Balancer Listener**:
   This table holds the typical information you will need to create the load balancer listener:

Table 9-2. Load balancer listener settings.

LBaaS LISTENER FIELD	VALUE
Display name	A user-chosen listener name that will show up in the OCI console.
Protocol	HTTPS Typically HTTPS for SSL based connections. HTTP is also allowed.
Port	443 The default port for HTTPS is 443
Use SSL checkbox	Checked Next to the port value is a checkbox labelled “Use SSL”. If port is set to 443, this checkbox will have a check mark.
Certificate Resource	Load Balancer Managed Certificates There are two options: “Load Balancer Managed Certificate” and “Certificate Service Managed Certificate”. Our project used “Load Balancer Managed Certificates”, as we had a certificate to upload.
Certificate Name	This is the name you provided when the certificate bundle was uploaded.
Hostnames	If you created any virtual hostnames within the load balancer hostname section, add one or more if they are to be associated with this listener.
Backend Set	This is a drop-down combo field providing a list of one or more backend sets. Select the backend set defined earlier.
Idle Timeout in Seconds	60 This is the default for HTTPS.

When you have finished entering the information, press the Create button to create the load balancer and its listener.

9.3 Create an FSS Snapshot and Clone the File System

We do not want to disrupt file system replay when testing at the standby site, but do need test users to be able to write to an EBS file system. We accommodate this by creating a read-write snapshot of the standby file system, then cloning the standby file system to the snapshot.

9.3.1 Create FSS Snapshot

To create the snapshot and make the file system accessible to users:

1. Log in to the OCI console
2. Change to the region hosting the secondary site (in our case Phoenix)
3. From the main menu, click on **Storage**.
4. Under **File Storage**, click on **File systems**.
5. Change the compartment if needed.
6. In the table, click on the file system used for EBS, in our case: PHX_EBS_APP_FS. The page for the selected file system will be displayed.
7. Under **Resources**, select **Snapshots**.
8. A dialog page is displayed. Enter the following fields.
    a. **Name**: Provide a name for the snapshot e.g., PHX_EBS_APP_FS_SNAPSHOT
    b. (Optional) **Expiration time**: Enter the expiration time if you wish to have the snapshot automatically deleted after a specified period of time.
9. Click **Create**. Once the new snapshot is created, a page showing details of the new snapshot is displayed.
10. In this new page, click **Clone** to create a new file system using the snapshot.
11. Click **Edit** and enter the following fields:
    a. **Name**: Provide a name for the new file system e.g., PHX_EBS_APP_SNAPSHOT_FS.
    b. **Create in Compartment**: From the dropdown, select the compartment to create the file system in. In our case, ebs-app-compartment.
    c. **Encryption**: From the radio button, select the type of encryption type. We accepted the default: Encrypt using Oracle-managed keys.
    d. **Attach snapshot policy**: If you have a snapshot policy you wish to attach, check this checkbox then select the appropriate snapshot policy.
    e. If you wish to have this cloned file system detached from its parent, then check the checkbox for "**Detach this File System from the parent File System**".
12. Click **Create**. The cloned file system will be created. Once the new file system is created, the detail page for the file system will be displayed.

9.3.2 Create Export Path for Cloned File System

After the file system is created, create an export path to configure the application tier servers to access. From the cloned file system detailed page:

1. Click **Create Export**. A dialog page is displayed.

2. In the **Export Information** section, if you wish to change the default export path, click the **Edit Details** link. Update the following fields:
    a. **Export Path**: Change or edit this field to an export path you wish to use.
    b. Optional: If you wish to **Use secure export options**, check this checkbox.
    c. Optional: If you need to **Use LDAP group list**, check this checkbox.
3. In the **Mount Target information** section of the page, click Edit details link, fields will be displayed. Enter the following fields:
    a. Verify or create your mount target. Select the **Click Here to Enable Compartment Selections** link to expand the list of compartments, and move to the compartment that will contain your mount target:
        i. If you have already set up your mount targets in this region, select the compartment that contains the mount target for this file system.
        ii. If you need to set up the desired mount target in this region, click on the compartment you want to define it in. Configure the new mount target as required.
4. Click **Create**. The export path is created, and its detail page is displayed.

9.3.3 Mount the Cloned File System

Note: These steps are only to be done on the application compute instances. They are **not** to be done on the pair of replication compute instances.

On each application compute instance:

1. Log in to the application server as root.
2. Unmount the shared file system where EBS is installed. In our case, /u02:
```
# umount /u02
```
3. Edit `/etc/fstab:` comment out the current mount point line for your production shared file system (for us, /u02) and add a new line for the cloned snapshot file system:
```
# Mount of EBS R12.2 PROD shared FSS
#<IP address of mount target>:/PHXEBSAPPFS  /u02           nfs
rw,rsize=131072,wsize=131072,bg,hard,timeo=600,nfsvers=3 0 0
# Mount the snapshot clone FSS
<IP address of mount target>:/PHXEBSAPPSNAPSHOTFS  /u02           nfs
rw,rsize=131072,wsize=131072,bg,hard,timeo=600,nfsvers=3 0 0
```
4. Save the file.
5. Mount the cloned snapshot file system:

    # mount /u02

When this has been completed on all application compute instances, your will have new web entry points to give to your test users for the secondary site.

9.4 Configure and Start Test Application Services at the Secondary Site

With the secondary site set up for snapshot standby operation against both the file system and the database, you can give new web entry points to users for safe testing. You will do this by adding to the configuration of the load balancer created in step 9.2 above.

First, create your test URL for accessing the snapshot environment by repeating steps 4-6 from section 9.2, Provision and Configure Load Balancer, to create your test listener for the snapshot environment.

With your snapshot listener configured, adjust the configuration of each application server to access the snapshot environment. Log in to each application compute instance and become applmgr (or oracle), source the RUN environment, and complete the following steps:

1. Reset the EBS web entry point: Mimic the steps in the `ConfigContext` routine of the `startEBS.sh` script provided in Section 14.4.2, using an environment file you have set up similar to those in `web_entry_test.env` from section 14.4.1.
2. Run autoconfig. Once autoconfig completes, the application will be ready to use in "snapshot" mode:
   ```
   $ adautocfg.sh
   ```
3. Start application services:
   ```
   $ adstrtal.sh
   ```

Monitor the startup and ensure all services start.

Check the LBaaS backend servers to ensure they each come up with a status of "OK". It may take a few minutes for the status to change from a red diamond "Critical" to a yellow triangle "Warning", then to a green "OK".

At this point, users should be able to log in to the standby EBS application through the LBaaS, using the test URL created in step 1 above:

```
https://<load-balancer-alias-test-host-name.doman>/OA_HTML/AppsLogin
```

For this project, we reset the URL to:

```
https://apps.mycompany-test.example.com/OA_HTML/AppsLogin
```

9.5 Convert Snapshot Standby Back to Physical Standby

Once application configuration and user testing is complete at the secondary site, shut down the snapshot standby EBS application services and the database and revert both back to standard recovery status.

9.5.1 Revert the File System

Do these steps on each secondary application server:

1. Log in to the application compute instance as applmgr (or oracle).
2. Shut down all application services:
   ```
   $ adstpall.sh
   ```
3. Log out and log back in to the application compute instance as root.
4. Unmount the cloned snapshot file system:
   ```
   # umount /u02
   ```

5. Edit `/etc/fstab` and comment out the mount for the clone snapshot file system and uncomment the mount for the production file system:

```
# Mount of EBS R12.2 PROD shared FSS
<IP address of mount target>:/PHXEBSAPPFS  /u02           nfs
rw,rsize=131072,wsize=131072,bg,hard,timeo=600,nfsvers=3 0 0

# Mount the snapshot clone FSS
# <IP address of mount target>:/PHXEBSAPPSNAPSHOTFS  /u02           nfs
rw,rsize=131072,wsize=131072,bg,hard,timeo=600,nfsvers=3 0 0
```

6. Mount the production fie system:

```
# mount /u02
```

9.5.2 Delete and Drop the Snapshot Clone File System

Once all application services are shut down, delete then drop the snapshot clone.

First, we delete the export of the file system:

1. Log in to the OCI console.
2. Change the region to the secondary site (in our case Phoenix)
3. From the main menu, click on **Storage**.
4. Under File Storage, click on **File systems**.
5. Change the compartment to the compartment if needed.
6. In the table listing file systems, click on the clone file system (in our case: PHX_EBS_APP_SNAPSHOT_FS). The page for the selected file system is displayed.
7. In the main region, below **Exports**, there is a table that will list the export paths defined for this file system. Delete the entry in the table that corresponds to the export path you edited in `/etc/fstab` in step 5 of section 9.5.1 above by activating the Action column (3 vertical dots) (note: screen readers should say "Action") and selecting **Delete**.
8. In the main region of the page, Click on **Delete**. A confirmation dialog window will be display.
9. Confirm that you want to delete the export path by clicking on **Delete**.
10. In the same dialog window, the export path will show as deleted with a check mark and the word "Done". Click **OK** to close the dialog window. You will be taken back to the file system detail page for the clone file system.
11. In the main region of the file system detail page, click **Delete**. A dialog window will be displayed.
12. Confirm that you want to delete this file system by clicking on **Delete** in the dialog window.
13. The file system will then be deleted.

Now drop the snapshot

1. On the left region, under Resource, click on **Snapshots**. A table of one or more snapshots will be displayed in the main region under **Snapshots**.
2. In this table, on the row for the snapshot to be deleted, activate the Action column (3 vertical dots) (note: screen readers should say "Action") then select **Delete**. A confirmation dialog window will be displayed.

3. Confirm that you want to delete the snapshot by clicking **Delete**.
4. In the same dialog window, the snapshot will show as deleted with a check mark and the word "Done". Click **OK** to close the dialog window. You will be taken back to the file system detail page for the clone file system.

9.5.3 Return the Standby to Physical Standby

Once all application services are shut down, return the secondary database back to a physical standby by logging in to one of the ExaDB-D compute nodes and becoming oracle, then doing the following:

```
$ . ./EBSCDB.sh
$ dbaascli dataguard convertStandby –dbname <CDB Name> --standbyName <CDB Unique Name> --
standbyType physical
```

For example:

```
$ dbaascli dataguard convertStandby --dbname EBSCDB --standbyName EBSCDB_phx3xd --standbyType
physical
```

10 Switch Over to Secondary Site

A switchover is a planned no-data-loss event. In this section, we will manually perform a full-stack switchover. In a later section, we will set up the Full Stack Disaster Recovery (Full Stack DR) cloud service for a fully orchestrated switchover.

Note: A switchover does require a small amount of downtime.

To perform a switchover:

1. Shut application services down on each primary application compute instance.
    a. Log in to the application compute instance as applmgr (or oracle).
    b. Source the RUN environment, then stop application services:

    ```
    $ . /<Base directory>/<EBS System Name>/EBSapps.env RUN
    $ adstpall.sh
    ```

2. At the primary site, perform a final replication of the file systems from the primary to the standby. Fully replicate `fs_ne`, which contains your batch processing logs and output files: You may also need to do a final replication of your RUN and PATCH file systems. Remember that these file systems contain a mixture of configuration and common data – only the common data is to be kept in sync.

    See section 14.5: File System Replication (rsync) Scripts for more details.

    Your file synchronization needs to complete before bringing application services up at the other site.

3. With application services shut down, perform an Oracle Data Guard switchover using Data Guard Association on the OCI console:
    a. Log in to the OCI console.
    b. Change the region to primary site.
    c. Activate the menu then select **Oracle Exadata Database Service on Dedicated Infrastructure**.
    d. Select the ExaDB-D VM cluster hosting the primary database from within the **Exadata VM Clusters** table. The **VM Cluster** page is displayed.
    e. From the **Databases** table, select the primary EBS database. The Database detail page is displayed.
    f. Under **Resources** (left side), select **Data Guard Association** link. The Data Guard Association detail page is displayed.
    g. In the main region, under **Data Guard Association**, there is a table listing information about the standby database. In the row of the peer standby database, click on the action menu (three vertical dots) – screen readers: last column labeled "action".
    h. From the action menu, select **Switchover**.
    i. Enter the database Admin (SYS) password.
    j. Click **OK** to start the switchover process.

4. Once the switchover process starts, click on **Work Resource** under **Resources** to monitor the progress of the switchover. When the switchover is completed, its status should be "Succeeded". The database switchover is complete.

5. Conditional: Configure autoconfig on database nodes

   If the Oracle database home path on the new primary is different from that of the old primary (new standby), you need to follow these steps to reconfigure autoconfig on the database nodes. If the database home paths are the same, proceed to step 6.

   Note: this process uses the script `txkSyncDBConfig.pl`. It will fail trying to access files configured in `UTL_FILE_DIR` in the old primary. This can be ignored – it will be addressed in the next step.

   On each RAC node on the new primary database:

   a. Log in as oracle to the database VM node.
   b. Source the `<CDB_Name>.env` file
      `$ . ./<CDB_Name>.env`
   c. Move to the <ORACLE_HOME> directory.
      `$ cd $ORACLE_HOME`
   d. Source the `<PDBName_hostname>.env` file. This will set up the environment for running the `txkSyncDBConfig.pl` script in the next step.
      `$ . ./<PDBName_hostname.env>`
   e. Run the `txkSyncDBConfig.pl` script to configure the oracle home path as follows:

```
 perl $ORACLE_HOME/appsutil/bin/txkSyncDBConfig.pl \
-dboraclehome=$ORACLE_HOME \
-outdir=$ORACLE_HOME/appsutil/out/<PDBName_hostname \
-cdbsid=<ORACLE_SID> \
-pdbname=<PDBName> \
-dbuniquename=<DB_UNIQUE_NAME> \
-israc=YES \
-virtualhostname=<logical_hostname-vip.domain> \
-logicalhostname=<logical_hostname.domain> \
-scanhostname=<SCAN-name-on-this-cluster.domain> \
-scanport=<SCAN listener port number> \
-dbport=<Grid listener port number> \
-appsuser=APPS
```

      Example:

```
$ perl $ORACLE_HOME/appsutil/bin/txkSyncDBConfig.pl \
-dboraclehome=$ORACLE_HOME \
-outdir=$ORACLE_HOME/appsutil/out/VISPRD_ebsdb1 \
-cdbsid=EBSCDB1 \
-pdbname=VISPRD \
-dbuniquename=EBSCDB_phx3xd \
-israc=YES \
-virtualhostname=ebsdb1-vip. example.com \
-logicalhostname=ebsdb1. example.com \
-scanhostname=phxexadb-krppw-scan.example.com \
-scanport=1521 \
-dbport=1521 \
-appsuser=APPS
```

Enter the APPS password when prompted.

For each RAC node, replace `ebsdb1` above with the logical hostname of the database node being maintained.

f. Configure UTL_FILE_DIR by running `txkCfgUtlfileDir.pl` script twice – once with “set” mode and once with “sync” mode.

   Source the `<ORACLE_HOME>/<PDBName_hostname>.env` file if not already done.

   ```
   $ cd $ORACLE_HOME
   $ . ./<PDBName_hostname>.env
   ```

   Run the perl script `txkCfgUtlfileDir.pl` with `mode= setUtlFileDir`:

   ```
   $ perl $ORACLE_HOME/appsutil/bin/txkCfgUtlfileDir.pl \
   -contextfile=$CONTEXT_FILE \
   -oraclehome=$ORACLE_HOME \
   -outdir=$ORACLE_HOME/appsutil/log \
   -mode=setUtlFileDir \
   -servicetype=opc
   ```

   Enter the EBS_SYSTEM (or SYS) and APPS passwords when prompted.

   Now run the perl script `txkCfgUtlfileDir.pl` with `mode =syncUtlFileDir`:

   ```
   $ perl $ORACLE_HOME/appsutil/bin/txkCfgUtlfileDir.pl \
   -contextfile=$CONTEXT_FILE \
   -oraclehome=$ORACLE_HOME \
   -outdir=$ORACLE_HOME/appsutil/log \
   -mode=syncUtlFileDir \
   -servicetype=opc
   ```

   Enter the APPS password when prompted.

   Note: using `mode=syncUtlFileDir` will cause autoconfig to run. It should complete successfully. Correct all errors.

   EBS is now configured on the new primary RAC database.

6. Configure application nodes and start application services.

   Because logical hostnames have been implemented on the application tier compute VMs, there are only a few small tasks that need to be performed before starting application services.

   On each application compute VM:

   a. Log in to the application compute instance as applmgr (or oracle). Start with the compute instance that runs the WLS Admin server.

   b. Source the RUN environment.

      ```
      $ . ./<Base directory>/<EBS system name>/EBSApps.env RUN
      ```

NOTE: You will see an error similar to "`ERROR: This env should be sourced from <logical hostname>!`" This error can be ignored – the environment will still source.

Conditional: If different URLs are used to access EBS are on the primary and secondary sites, you must reset the web entry points. Mimic the steps in the `ConfigContext` routine of the `startEBS.sh` script provided in Section 14.4.2, using an environment file you have set up similar to those in `web_entry_prod.env` from section 14.4.1.

c. Run autoconfig:

```
$ adautocfg.sh
```

Enter the APPS schema password when prompted. Autoconfig should complete successfully – if there are errors, correct them.

d. Start application services.

```
$ adstrtal.sh
```

Repeat the above steps for each application tier compute instance.

Check the OCI load balancer back-end sets as described earlier. You should see a green "OK" once all services are up.

11 Fail Over to Secondary Site

If the EBS environment deployed at the primary site cannot be accessed – an unplanned outage – it will be necessary to perform a failover to your secondary site. This is an unplanned event and may result in data loss.

A failover event will be a noticeable disruption to the operation. Depending on the cause of the disruption, it might be of benefit for the root fault to be corrected and operations resumed at the primary, without failing over to the standby. For this reason, most customers keep the decision to fail over a human one, not an automated one – meaning that, while the effort should be scripted, the decision to execute a failover is usually made by a human.

We will assume in the following procedure that the database is unavailable on any RAC node – a complete database down and/or a full cluster outage.

To perform a full-stack failover:

1. If possible: Shut down application services.
   With the database inaccessible, the application tier services will begin failing and throwing errors. To ensure the application is completely down, if the application tier compute instances and file system are still accessible, source the RUN environment and stop application services on each application compute instance:

   ```
   $ . /u02/app/ebs/visprd/EBSapps.env RUN
   $ adstpall.sh
   ```

2. If possible: Perform a final rsync.
   If there is still network connectivity between the sites, execute a final rsync of all EBS file systems:

   ```
   $ time rsync -avzh –progress /<Base directory>/<EBS system name>/fs_ne/inst/username@<IP
   Address of compute instance at DR site>:/<base directory>/fs_ne/inst/
   ```

   You may also need to do a final replication of your RUN and PATCH file systems. These file systems contain a mixture of configuration and common data; only the common data is to be kept in sync.

   See Section 14.5: File System Replication (rsync) Scripts for more details.

3. Perform a database failover:

   a. Log in to the OCI console.

   b. Change the region to the standby site.

   c. Activate the menu, then select **Oracle Exadata Database Service on Dedicated Infrastructure**.

   d. Select the ExaDB-D VM cluster hosting the standby database from within the **Exadata VM Clusters** table. The VM Cluster page is displayed.

   e. From the **Databases** table, select the standby EBS database. The Database detail page is displayed.

   f. Under **Resources** (left side), select the **Data Guard Association** link. The Data Guard Association detail page is displayed.

   g. In the main region, under **Data Guard Association**, there is a table listing information about the standby database. In the row of the peer primary database, click on the action menu (three vertical dots) (screen readers: this is the last column, labeled “action”).

   h. From the action menu, select **Failover**.

   i. Enter the database administrator (SYS) password.

j. Click **OK** to start the failover process.

k. Once the failover process starts, click on **Work Resource** under **Resources** to monitor progress. Once the failover is completed, its status should be "**Succeeded**". The database failover is complete.

4. Complete configuration and start application services.
   With the secondary database running as primary, complete the role change transition described in Section 10: Switch Over to Secondary Site, starting with step 5.

5. Clean Up State Files (Conditional)

   If you find that the OPMN and HTTP servers fail to start when starting the application services at your new primary, you will need to clean up the state files in the RUN file system. Each application tier node has a subdirectory with a name ending with an integer, where the integer corresponds to the application tier node. In our environment, EBS_web_VISPRD_OHS1 is for the master application tier node, EBS_web_VISPRD_OHS2 for the second application tier node, and so on.

   To do this work, log in to each application tier server as applmgr (or oracle), source the RUN environment, and do the cleanup task on that server. In our environment:

```
$ cd
/u02/app/ebs/visprd/fs1/FMW_Home/webtier/instances/EBS_web_VISPRD_OHS1/config/OPMN/opmn
$ mv states states_backup
```

6. Attempt to start the OPMN and HTTP services:

```
$ adopmnctl.sh start
$ adapcctl.sh start
```

   For further details on the internal error and the state cache file, see My Oracle Support Document 2323987.1 *12.2 E-Business Suite Applications Technology Stack Administrator Is Unable To Start OPMN Via 'adopmnctl.sh start' Due to 'ERROR Timed out Interrupted Exception' After Previous System Shutdown.*

12 Integrating Full Stack Disaster Recovery Cloud Service with EBS

This section describes how to implement the OCI Full-Stack DR cloud service to automate switchover and failover for EBS. It covers the process of setting up Full Stack DR for an EBS installation then testing the result, using our test environment as an example.

Automating and managing switchover and failover of an Oracle database is easily implemented using Oracle Data Guard and Full Stack DR. We present this here, then go into more detail for managing the application tier. We have provided sample scripts for this activity, including:

- Scripts automating the process for the EBS application tier, and
- Scripts using rsync to synchronize the file system.

This section covers:

- Assumptions
- Setting up Full Stack DR infrastructure
- Implementing Full Stack DR
- Performing a planned switchover

Note that while this document does not cover the topic, you will likely need to integrate other applications into your overall switchover / failover planning and scripting.

12.1 Assumptions

These items need to be in place to implement Full Stack DR cloud service for EBS:

- OCI groups that have privileges to:
  - Create OCI vaults
  - Create Disaster Recovery Protection Plans
  - Access all database and compute resources that will be protected by Full Stack DR
- Fully configured primary and standby environments, with the standby in a different availability domain, preferably in a separate region.
  - Oracle Data Guard Association should already be set up for the database.
  - You have scripted synchronizing your middle tier file systems. We used rsync for this task.
- VCN remote peering
- Tested manual switchover / failover scripts

Note: To ensure the new environment is available to users as quickly as possible, the implementation presented here does not use Full Stack DR's VM Migration. With this paradigm, all EBS compute instances are pre-provisioned and pre-configured at the secondary site and the file systems are kept in sync, as described earlier in this paper.

12.2 Set Up Full Stack DR Infrastructure

We will first do the groundwork of setting up the infrastructure:

1. Create compartments for DR protection groups and OCI Vaults
2. Create an IAM dynamic group

3. Create OCI policies for the group
4. Add cloud agent run-command files
5. Create region-local OCI vaults and add secrets
6. Install OCI CLI
7. Create object storage buckets for Full Stack DR

12.2.1 Create Compartments (Optional but Recommended)

We recommend you create two new compartments:

1. One compartment for the DR protection groups. We named ours `apps-MAA-protection-group-compartment`
2. One compartment for OCI vaults. We named ours `apps-vault-compartment`

This creates a finer level of security and management control within OCI.

Use the OCI console to create your compartments.

12.2.2 Create IAM Dynamic Group

Dynamic groups allow you to designate a set of OCI compute instances as "principal actors", similar to user groups. With dynamic groups, you can create policies to give the compute instances permission to make API calls against the OCI infrastructure services.

Use the OCI console to create your dynamic groups.

In our environment, we created one dynamic group, with these attributes:

- Dynamic Group Name: `DRPG_runcommand_EBS_Apps`
- Matches ALL rules
- Matching Rule 1 (the only rule in this case):
  All {instance.compartment.id = '<OCID of the compartment that the EBS app tiers belong to>'}

12.2.3 Create a Policy for the Dynamic Group

Using the OCI console, create a policy to allow the runcommand agents to operate in the application tier compartment.

In our environment:

- **Policy Name**: EBS-DR-Protection-Runcommand-Policy
- **Description**: Allow agents to execute runcommands on EBS application tier compute instances
- **Statement 1:**
  Allow dynamic-group DRPG_runcommand_EBS_Apps to manage instance-agent-command-execution-family in compartment ebs-app-compartment where request.instance.id=target.instance.id
- **Statement 2:**
  Allow dynamic-group DRPG_runcommand_EBS_Apps to read objects in compartment ebs-app-compartment
- **Statement 3:**
  Allow dynamic-group DRPG_runcommand_EBS_Apps to manage objects in compartment ebs-app-compartment

12.2.4 Add Cloud Agent Run-Command Files

As root, on each application tier compute instance, create a cloud agent run command file under `/etc/sudoers.d`.

Name the file, using this name: `101-oracle-cloud-agent-run-command`

The contents of this file should be:

```
ocarun ALL=(ALL) NOPASSWD:ALL
```

12.2.5 Create Region-Local OCI Vaults

You will need OCI vaults in each region to hold credentials required for accessing the database and application tiers and the application file system. These credentials will be "secrets" in the vaults.

For details on OCI vault concepts, creating vaults, key management, etc., please see Oracle Cloud Infrastructure Documentation: Vault.

Use the OCI console to create an OCI vault in each region with the following attributes:

- Place the vault into the compartment for vaults if one was created (step 12.2.1 above)
- Use a symmetric key.
- Key protection mode can be software.
- Key length 256 bits or longer.

Once the vaults are created, add your credentials as secrets to each vault:

- SYS password for CDB$ROOT
- EBS_SYSTEM schema in the EBS PDB
- apps schema password in the EBS PDB
- WebLogic application tier admin password

12.2.6 Install OCI CLI

The OCI CLI utility must be available on each application tier compute instance at both sites.

Follow the instructions in Oracle Cloud Infrastructure Documentation: Quickstart: Installing the CLI, paying close attention to the configuration file setup.

Once installed and configured, log in as applmgr (or oracle) and query your tenancy name to ensure that you can access your tenancy and verify that OCI CLI is running properly:

```
$ oci os ns get
```

The output will be in JSON format and will resemble this:

```
{
"data": "<your OCI tenancy name>"
}
```

If the command fails, address the error before proceeding. Likely causes of an error at this point:

- The OCI python script cannot open the configuration file. This is typically located in $HOME/.oci
- The fingerprint specified in the OCI configuration file does not match that in the private key PEM file and the public key file stored in OCI.

- The PEM file with the private key cannot be accessed by OCI.
- The OCI user's OCID may be incorrect.

12.2.7 Create Object Storage Bucket for Full Stack DR

Use the OCI console to create an object storage bucket to be used by the DR Protection Group logs. We recommend this bucket be in the same compartment as the DR Protection Group. This bucket will contain the Create Object Storage Bucket for Full Stack DR logs.

12.3 Implement Full Stack DR

With the above prerequisites in place, we can fully script switchover and failover activities, then integrate our scripts into the Full Stack DR cloud service.

12.3.1 Script Switchover and Failover

Switchover and failover need to be automated so the steps are easily and accurately repeatable. To accomplish this:

1. Write scripts that run on each compute instance to stop, start, and configure all required services, and to synchronize the file system. These scripts must:
   a. Have the same OS user as owner – not root – on all compute instances for each type of server,
   b. Run without prompting for credentials or other input,
   c. Not hard-code user credentials or expose them on a command line,
   d. Be identical at each region for each type of server, and
   e. Run the same way regardless of which region and compute instance they run in.
2. Test each script locally to ensure they run as expected.
3. Use the scripts to perform a manual switchover and switch back, to ensure they work as expected.

Once you have tested the scripts by completing a full switchover in both directions, you can integrate them into the Full Stack DR cloud service.

Notes:

- Where passwords are required, use OCI CLI commands to obtain the credentials from region-local OCI vaults so the scripts and the passwords can be maintained securely.
- Pay careful attention to file system synchronization. Be sure that:
  - Replication from the primary to the secondary is complete before switching to a new primary during a switchover, and
  - Changes in replication direction are managed correctly based on database role.

See Section 14: Automation Scripts Switchover, Failover, File Sync for the scripts we used in our environment, including these called via the Full Stack DR cloud service:

- startEBS.sh
- stopEBS.sh

12.3.2 Create Full Stack DR Protection Groups

You need to create two Full Stack DR protection groups - one for the primary and one for the standby region. To do this, log in to the OCI console, go to Migration and Disaster Recovery, then DR Protection Group. This will bring up the

Disaster Recovery page. In the left menu area, select the DR protection group compartment created earlier. For each region:

1. Select a region in the top banner.
2. Click on **Create DR Protection Group**
3. Configure the DR protection group:
    a. Give the DR Protection Group a name that indicates its region and purpose. We used IAD-EBS or PHX-EBS, indicating EBS in Ashburn (IAD) or Phoenix (PHX)
    b. Specify the compartment to place the DR Protection Group into. See section 12.2.1: Create Compartments (Optional but Recommended)
    c. Specify the compartment holding the object storage bucket. See section 12.2.7: Create Object Storage Bucket for Full Stack DR
    d. For the first protection group you create, specify **Not Configured** for Role. See 12.3.3 Associate Protection Groups for more information.
    e. For the database - click on **Add Members**
        i. **Resource type**: Database -> VM Cluster Database (Exadata).
        ii. Change the compartment to the one hosting the Exadata VM database cluster, then select the cluster.
        iii. Select the name of the database to be added as a member (e.g., EBSCDB).
        iv. For **Database Password**, change the compartment to where the region local OCI vault resides.
        v. Select the secret name to be used. See section 12.2.5: Create Region-Local OCI Vaults.
        vi. Click **Add**. Note: This adds all RAC nodes of the database in one action.
    f. For each compute instance - click on **Add Members**
        i. **Resource type**: Compute
        ii. Change to the compartment hosting the application compute instances.
        iii. Select an application compute instance that is not yet configured. Note: Do not select any compute instance used only for rsync replication.
        iv. Check **Non-Moving Instance on switchover or failover.**
        v. Click **Add**. Note: This adds one application tier compute instance.

        Repeat these steps until all application / middle tier compute instances have been added. Do not add any compute instances whose only purpose is for rsync replication.
4. Click **Create** to create the DR Protection Group

IMPORTANT: As of the publication of this document, adding or removing members after creating DR plans will automatically update plan groups with new or removed steps, and will cause the plans to be set to a “needs refresh” state. Please see Understanding the DR Plan Refresh Process for more information about this feature of Full Stack DR.

12.3.3 Associate Protection Groups

Associate the protection groups with each other.

In 12.3.2, step 3.d for creating DR Protection Groups tells you to select **Not Configured** for the first protection group. When creating the second DR Protection Group, you can select a role (Primary or Standby) for that region. If you selected a role, two more combo box fields would be presented:

- **Peer Region**
- **Peer DR Protection Group**

When these fields are specified and you click **Create**, the two DR protection groups are then associated.

Alternatively, if you specified **Not Configured** when creating your second DR Protection Group, once both have been created you can click on the **Associate** button on the main DR Protection Group page for one of the DR Protection Group. When doing so, select using the combo boxes presented:

- **Peer Region**
- **Peer DR Protection Group**

12.3.4 Create Plans and Plan Groups at the Secondary Site

Next, plans and plan groups need to be created for each DR protection group. A plan has a plan type (switchover or failover) and consists of one or more plan groups within the plan. Each plan group can have one or more steps.

Important: Note that plans and plan groups can only be added to a DR protection group when the hosting database is in the STANDBY role. Thus, you will need to set your plans up at your standby site, then perform a switchover so that you can set up your plans at what was the primary, in order to fully configure Full Stack DR.

12.3.4.1 Create a Switchover Plan

A switchover plan will perform the following tasks, operating from the standby:

- Shut all EBS application services down at the primary, then perform a final rsync from the primary to the standby file system.
- Reverse direction of the rsync scripts.
- Perform a Data Guard switchover to the standby database, bringing the standby database up as primary.
- Run configuration scripts on the database servers if required, run autoconfig on the application servers, and start the application services.

NOTE: To shut application services down as quickly and cleanly as possible, concurrent manager jobs will need to be stopped or suspended prior to the time the switchover is executed. This would be done outside the Full Stack DR framework.

To add a switchover plan, locate the link labeled **Plans** under **Resources** in the left-side menu in OCI, then:

1. Click on the **Plans** link unless you are already in Plans page.
2. Click **Create Plan**. A dialog box for the plan is displayed.
3. Give the plan a name. Since this will be a switchover plan, we used the name “Switch <primary>-to-<standby>”, substituting the remote site code for primary and the local site code for standby.
4. For the **Plan Type**, select **Switchover (Planned)** from the combo box.
5. Click the **Create** button.

The new plan will have two default plan groups: **Built-in Prechecks** and **Switchover Database**. These two groups are created and managed automatically by Full Stack DR.

We will add two plan groups – one to shut EBS application services down at the primary and one to start them at the standby. These plan groups include scripts that manage reversal of application tier file system synchronization.

12.3.4.1.1 Application Services Shutdown Plan Group

The plan group to shut application services down at the old primary will have one step for each application server. To define this step, using IAD as the primary region and PHX as the standby:

1. Click the **Add Group** button. A new dialog box is displayed.
2. Give the group a name that indicates its purpose. We used "IAD-Shut-Down-EBS-App-Services".
3. For your first application server, click on **Add Step** to point to the script that stops all services on an application server:
    a. Provide a name for the step, e.g. Stop-EBS-iad-app01.
    b. Check the **Enable Step** checkbox.
    c. Select the primary region from the combo box.
    d. Check the radio button **Run Local Script**.
    e. Select the target compute instance where you want to stop EBS application services. Change the compartment and region if required. In our case, we needed to stop the services on iad-app01, which is one of the compute instance members we added to the IAD-EBS protection group.
    f. For **Script Parameters**, enter the fully qualified path and name of your script that wraps all the steps required to shut services down on an application tier. Note the 's' argument as this specifies a switchover to the script. We used:

       /u02/app/ebs/custom_admin_scripts/<PDB_NAME>/stopEBS.sh s
    g. For **Run as user**, enter the OS user that owns your application install (applmgr or oracle).
    h. For **Error mode**, select the handling mode: we chose Stop on error from the combo box.
    i. For **Timeout in seconds**, the default is 3600 (1 hour). This means that if the script does not complete within this time, a timeout will occur, causing the switchover plan to fail.

       Note: We left this value at the default, to give file system synchronization plenty of time to complete. Adjust the value according to your requirements.
    j. Click **Add Step**. You should see a table under the group with your new step.

   Repeat these tasks for each of the rest of your application servers.
4. When you have added a step for each of your application servers, click **Add** to add the group to the Switchover plan.

12.3.4.1.2 Application Services Startup Plan Group

The plan group to start application services at the new primary post database switchover will have one step for each application server:

1. Click the **Add Group** button. A new dialog box is displayed.
2. Give the group a name that indicates its purpose. We used "PHX-Start-EBS-App-Services".
3. For your first application server, click on **Add Step** to point to the script that starts services on an application server:
    a. Provide a name for the step, e.g., Start-EBS-phx-app01.
    b. Select the region from the combo box, e.g. US West Phoenix.
    c. Check the radio button **Run Local Script**.

d. Select the target compute instance where you want to start EBS application services. Change the compartment if required. In our case, we needed to start the services on phxvisprd-app01, which is one of the compute instance members we added to the PHX-EBS protection group.

e. For **Script Parameters**, enter the fully qualified path and name of your script that wraps all the steps required to start services on an application tier. We used:

```
/u02/app/ebs/custom_admin_scripts/startEBS.sh s
```

f. For **Run as user**, enter the OS user that owns your application install (applmgr or oracle).

g. For **Error mode**, select the handling mode: we chose Stop on error from the combo box.

h. For **Timeout in seconds**, the default is 3600 (1 hour). If the script does not complete within this time a timeout will occur, causing the switchover plan to fail.

   Note: It is typical to reduce this value to 1200 or 1800. Test your overall startup timing requirements and adjust this value if needed.

i. Check the **Enable step** checkbox.

j. Click **Add Step**. You should see a table under the group with your new step.

Repeat step 3 for each of the rest of your application servers.

When you have added a step for each of your application servers, click **Add** to add the group to the Switchover plan.

12.3.4.1.3 Check Your Work

You should now have four plan groups similar to this: Built-in Prechecks, <Old Primary>-Shut-Down-EBS-App-Services, <New Primary>-Start-EBS-App-Services, and Switchover Database (Standby). They need to be executed in a particular order:

1. Built-in Prechecks
2. <Old Primary>-Shut-Down-EBS-App-Services
3. Switchover Databases (Standby)
4. <New Primary>-Start-EBS-App-Services

If the groups are not ordered correctly, click on the **Actions** drop-down button, select **Reorder Groups**, and reorder the groups as needed.

Once all the plan groups with their steps have been added to the DR protection group, you should be able to execute a precheck to ensure that the switchover plan can:

- Validate that both primary and standby regions are available
- Validate the configuration status of the primary and standby databases
- Validate that all scripts at both primary and standby are present

Prechecks can only be run on a DR protection group that is in the STANDBY role. To run the precheck, do the following:

1. On the OCI console, ensure that you are on the page for the DR protection group in the STANDBY role. In our case: PHX-EBS.
2. Click on **Run Prechecks** button. A dialog window is displayed.
3. Select the DR plan. In our case: Switchover-IAD-to-PHX.

4. Provide a name for the precheck run.

5. Click the **Run Prechecks** button.

When the prechecks start, a new status window will appear, showing all plan groups. This can be expanded to show all steps within each plan group. As the prechecks progress, they will report the time duration whether the precheck succeeded or failed. Make sure all prechecks succeed before executing an actual switchover.

12.3.4.1.4 Conditional: Repeat at Other Site

The initial creation of and any later updates to the plan need to be completed at both sites.

Remember that this creates your plan at what is currently the secondary or standby site. You will need to schedule, or be prepared to execute, a switchover so that this initial setup work can be repeated at what is currently the primary site when that site is in secondary / standby mode.

12.3.4.2 Create a Failover Plan

A failover is an unplanned event caused by the complete, abrupt disruption of services at or access to the primary environment or by a serious corruption of the primary such as a lost write detected by the standby database. It does take time to bring services up at the standby site, and there is a potential for data loss as it is possible that not all redo has been replayed to the standby database. A decision needs to be made as to whether to recover the primary or perform a failover to the standby, keeping the following in mind:

- Determine if the primary site can be recovered within an acceptable amount of time, factoring in business-critical transactions and service level agreements.
- Consider integration interfaces, external and third-party components, and the effort to manage these as part of the failover process.

While the decision to perform a failover is likely best made by business and IT personnel, the process of executing a failover should be automated. Full Stack DR can perform virtually all the failover tasks via a plan that you create in much the same way you created the switchover plan above. A Full Stack DR failover plan will not access the old primary environment at all, but will simply bring services up at the standby, making it the new primary.

Some things to consider when deciding to execute a failover plan:

- In a cross-region DR configuration within OCI, there will be some data loss, since Data Guard Association cloud service only supports Maximum Performance with ASYNC redo shipping.
- It is a best practice to enable Flashback Database on the primary database, to allow the database to be reinstated as a standby should a failover plan be executed.

Note: Full Stack DR assumes the primary site is not available when executing a failover. There is one task that is highly desirable if it is possible, that cannot be scripted in Full Stack DR – to get the standby application tier file system fully in synch with the old primary. See Section 14.5 File System Replication (rsync) Scripts for ideas on writing scripts for this activity.

12.3.4.2.1 Create Full Stack DR Failover Plan

Just as you needed to define the steps for switching from one site to another, you will need to define the steps for failing over to the standby site, making it production.

To create a Full Stack DR failover plan, in the OCI console:

- Click on the **Plans** link unless you are already in the Plans page.
- Click on **Create Plan**. A dialog box for the plan is displayed.
- Give the plan a name. Since this will be a failover plan, we used the name “Fail Over IAD-to-PHX”.

- For the **Plan Type**, select **Failover (Unplanned)** from the combo box.
- Click the **Create** button.

12.3.4.2.2 Add Full Stack DR Failover Plan Groups/Steps for Application Tier

Once the failover plan is created, it will contain two plan groups: Built-in Prechecks and Failover Database.

At this point, you will add the plan groups and steps to start the EBS application services to your Failover plan, as described in the prior sections. Remember that Full Stack DR has the assumption that the old primary site is not available. Thus, we will not add the steps described in Section 12.3.4.1.1: Application Services Shutdown Plan Group. We will only add the steps described to start application services at the new primary:

- Section 12.3.4.1.2: Application Services Startup Plan Group

12.3.4.2.3 Check Your Work

Once your plan group with its steps have been added, perform prechecks to validate the failover plan.

12.3.4.2.4 Conditional: Repeat at Other Site

The initial creation of and any later updates to the plan need to be completed at both sites.

Remember that this creates your plan at what is currently the secondary or standby site. You will need to schedule, or be prepared to execute, a switchover so that this initial setup work can be repeated at what is currently the primary site when that site is in secondary / standby mode.

12.4 Perform a Switchover

With Full Stack DR integrated with EBS, you can do a switchover from the OCI console:

1. Log in to the OCI console.
2. Switch to the region hosting the DR / standby environment.
3. Activate the main menu, then select **Migration and Disaster Recovery**
4. Select **DR Protection Groups**, change compartment if required. The DR Protection Groups page is displayed.
5. From the list of DR protection groups, click on the DR protection group for this environment. The page for the DR protection group is displayed.
6. We recommend you run prechecks before executing the actual switchover. To do so, click the **Run Prechecks** button. A dialog page is displayed.
    a. Select the DR plan from the combo dropdown list - e.g., **Switchover**.
    b. Enter a name (optional). This name will flow into the log file names in the object storage buckets created by Full Stack DR.
    c. Click the **Run Prechecks** button. The prechecks start and the page changes to show the plan groups, the steps, and the status of the prechecks as they are run. Click **Expand All** to see all plan groups and steps within each plan group.
    d. Monitor the page showing the status of each step within the plan. Once all prechecks have been completed, each step within each plan group should have a status of Succeeded.
    e. If a precheck failed, go to the **Actions** button (three vertical dots) and click **View logs** to review the logs to determine what failed.
7. If all prechecks succeeded, then return to the DR protection group details page by clicking the link **DR protection group details**, found towards the top of the page.

8. Click the **Execute DR plan** button. A dialog page is displayed.
   a. Select the DR plan from the combo dropdown list - e.g., **Switchover**.
   b. Enter a name (optional)
   c. Uncheck the **Run Prechecks** button if you do not want to re-execute the prechecks.
   d. Click the **Execute DR Plan** button. The switchover plan starts with the prechecks if you have not deselected them, and the page changes to show the plan groups, the steps, and the status of the execution as it is run. Click **Expand All** to see all plan groups and steps within each plan group.
   e. Monitor the page showing the status of each step within the plan. Once all steps have been completed, each step within each plan group should have a status of **Succeeded**.
   f. If a step failed, then go to the Actions button (three vertical dots) and click **View logs** to review the logs to determine what failed.

Note that logs can be downloaded from the **View logs** action menu.

If all steps have succeeded, the switchover is complete.

12.4.1 Conditional: Configure New Standby for Switchover / Failover

When you are first establishing your disaster recovery setup in Full Stack DR, you will need to repeat the Full Stack DR configuration work described in section 12.3.4, Create Plans and Plan Groups, at the original production site when it is the standby. The first time you perform a switchover to your standby is an excellent time to do this configuration.

12.5 Perform a Failover

If your primary database becomes inaccessible and you decide you need to fail over to your secondary site, follow the steps in this section.

12.5.1 Conditional: Finalize File System Synchronization

It is possible that your primary database has become unusable but your primary application tier file system can still be accessed. If the shared file system at the failed primary site is still intact and accessible, before performing a failover and switching services fully to the secondary, it is best to make sure all application tier file system changes have been replicated to the secondary site.

Your method for doing this will depend on your application tier file system features. With our scripts, we first halted any currently-running rsync operations then did a final sweep of the file systems, looking for changes to replicate. The process was to disable the rsync processes then force a final rsync with these commands for each file system being replicated:

```
./disable_ebs_rsync.sh <file system>
./syncEBS.sh <file system> -f
```

Example:

```
./disable_ebs_rsync.sh fs1
./disable_ebs_rsync.sh fs2
./disable_ebs_rsync.sh fs_APPLCSF
./syncEBS.sh fs1 -f
./syncEBS.sh fs2 -f
./syncEBS.sh fs_APPLCSF -f
```

An example that matches the sample scripts provided in section 14:

```
./disable_ebs_rsync.sh slowFiles
./disable_ebs_rsync.sh fastFiles
./syncEBS.sh slowFiles -f
./syncEBS.sh fastFiles -f
```

12.5.2 Perform A Failover

When you are satisfied that your file system is ready on the new primary site, follow the pattern described above for performing a switchover, but choose the Failover DR plan.

Note: If OPMN and HTTP services do not start, clean up their state files as described in Section 11 – Fail Over to Secondary Site.

13 Working with Terraform

The OCI resource discovery feature can be used to create Terraform files for existing resources in a compartment. One use for these files is to duplicate a portion of your infrastructure in a new region.

This project has several subnets, each with one or more security lists – some complex and containing dozens of ingress rules. For this case study, we chose to use Terraform to discover, then replicate, the resources in the network compartment:

- Virtual Cloud Network (VCN)
- Gateways (Internet, NAT, and Service gateways)
- Route tables
- Security lists
- Subnets

To execute Terraform resource discovery:

1. Log in to the OCI console.
2. Change the region to the primary region.
3. Click on **Main Menu**.
4. Click **Development Services**.
5. Under **Resource Manager**, click **Stacks**.
6. Click on the **Create Stack** button.
7. Select the **Existing Compartment** radio option with the text that says: **Create a stack that captures resources from the selected compartment (resource discovery)**.
8. For the **Compartment for Resource Discovery** drop down combo box, select the compartment to discover resources from. You will need to expand the root to get the full list of compartments. We discovered the network compartment.
9. For **Region for Resource Discovery**, select the OCI primary region.
10. For **Terraform Provider Services** radio button options, select **All**.
11. Provide a name for the ZIP file that will be created, in the Name text box.
12. Provide a description if desired (optional).
13. Choose the compartment that the stack should be created in.
14. Click **Next** twice to get to the **Review** page.
15. If the information on the **Review** page is correct, click **Create**.

Once the stack creation job completes, the stack will appear in the compartment selected in step 13. Click on the link for the stack to get the stack details page. On the stack details page, click on the **Download** link on the Terraform Configuration, to download the stack ZIP file to your local computer. Save the ZIP file to a directory of your choosing, then unzip it.

13.1 Editing Terraform Files

When you unzip the stack ZIP file, you will find several Terraform files in JSON format, ending with .tf. The files will contain descriptions / definitions of the resources in the compartment where the Terraform discovery was performed.

You will need to modify the files to duplicate those resources at your secondary site. For example:

- New display name
- Different DNS label
- Different VNC reference
- Non-overlapping IP addresses in the CIDR block
- The text "export_" removed from all resource definitions

Most of the resource definitions will be found in the core.tf JSON file.

The following table shows examples of Terraform definitions from the primary region in our test environment, and the changes needed for the secondary region.

Note: Before making changes to the .tf files, we recommend you back them up.

Table 13-1. Terraform resource definitions and modifications.

RESOURCE TYPE	PRIMARY REGION DEFINITION	AS MODIFIED FOR SECONDARY REGION

Virtual Cloud Network	resource oci_core_vcn export_iad-cloudmaa-vcn {
	#cidr_block = <<Optional value not found in discovery>>
	cidr_blocks = [
	"10.0.0.0/16",
	]
	compartment_id = var.compartment_ocid
	defined_tags = {
	}
	display_name = "iad-cloudmaa-vcn"
	dns_label = "iadcloudmaavcn"
	freeform_tags = {
	}
	#is_ipv6enabled = <<Optional value not found in discovery>>
	}
		Required modifications include removing "export_", assigning a different non-overlapping CIDR, display name and changing the DNS label:
		resource oci_core_vcn phx-cloudmaa-vcn {
		#cidr_block = <<Optional value not found in discovery>>
		cidr_blocks = [
		"10.10.0.0/16",
		]
		compartment_id = var.compartment_ocid
		defined_tags = {
		}
		display_name = "phx-cloudmaa-vcn"
		dns_label = "phxcloudmaavcn"
		freeform_tags = {
		}
		#is_ipv6enabled = <<Optional value not found in discovery>>
		}

NAT Gateway	resource oci_core_nat_gateway export_iadmaa-ngwy {
	block_traffic = "false"
	compartment_id = var.compartment_ocid
	defined_tags = {
	}
		Modifications include removing "export_", changing the display name and VCN reference.
		resource oci_core_nat_gateway phxmaa-ngwy {
		block_traffic = "false"
		compartment_id = var.compartment_ocid
		defined_tags = {

	display_name = “iadmaa-ngwy”
	freeform_tags = {
	}
	public_ip_id = “<public_ip_id ocid>”
	vcn_id = oci_core_vcn.export_iad-cloudmaa-vcn.id
	}
		}
		display_name = "phxmaa-ngwy"
		freeform_tags = {
		}
		public_ip_id = "< public_ip_id ocid >"
		vcn_id = oci_core_vcn.phx-cloudmaa-vcn.id
		}

Route Table	resource oci_core_route_table export_iad-db-private-RT {
	compartment_id = var.compartment_ocid
	defined_tags = {
	}
	display_name = "iad-db-private-RT"
	freeform_tags = {
	}
	route_rules {
	#description = <<Optional value not found in discovery>>
	destination = "0.0.0.0/0"
	destination_type = "CIDR_BLOCK"
	network_entity_id = oci_core_nat_gateway.export_iadmaa-ngwy.id
	}
	vcn_id = oci_core_vcn.export_iad-cloudmaa-vcn.id
	}
		Modifications include removing “export_”, changing the name of the route table, display name, and VCN reference.
		
		resource oci_core_route_table phx-db-private-RT {
		compartment_id = var.compartment_ocid
		defined_tags = {
		}
		display_name = "phx-db-private-RT"
		freeform_tags = {
		}
		route_rules {
		#description = <<Optional value not found in discovery>>
		destination = "0.0.0.0/0"
		destination_type = "CIDR_BLOCK"
		network_entity_id = oci_core_nat_gateway.phxmaa-ngwy.id
		}
		vcn_id = oci_core_vcn.phx-cloudmaa-vcn.id
		}

Security list	resource oci_core_security_list export_iad-db-private-seclist {
	compartment_id = var.compartment_ocid
	defined_tags = {
	}
	display_name = "iad-db-private-seclist"
	egress_security_rules {
	#description = <<Optional value not found in discovery>>
	destination = "0.0.0.0/0"
	destination_type = "CIDR_BLOCK"
	#icmp_options = <<Optional value not found in discovery>>
	protocol = "6"
	stateless = "false"
	#tcp_options = <<Optional value not found in discovery>>
	#udp_options = <<Optional value not found in discovery>>
		Modifications include removing “export_”, changing name of the security list and its display name, changing the CIDR blocks in each ingress rule that have 10.0.x.y to 10.10.x.y, and changing the VCN reference. Leave the 0.0.0.0/0 unchanged.
		
		resource oci_core_security_list phx-db-private-seclist {
		compartment_id = var.compartment_ocid
		defined_tags = {
		}
		display_name = "phx-db-private-seclist"
		egress_security_rules {
		#description = <<Optional value not found in discovery>>
		destination = "0.0.0.0/0"
		destination_type = "CIDR_BLOCK"
		#icmp_options = <<Optional value not found in discovery>>
		protocol = "6"
		stateless = "false"

	}
	egress_security_rules {
	#description = <<Optional value not found in discovery>>
	destination = "0.0.0.0/0"
	destination_type = "CIDR_BLOCK"
	#icmp_options = <<Optional value not found in discovery>>
	protocol = "1"
	stateless = "false"
	#tcp_options = <<Optional value not found in discovery>>
	#udp_options = <<Optional value not found in discovery>>
	}
	freeform_tags = {
	}
	ingress_security_rules {
	#description = <<Optional value not found in discovery>>
	#icmp_options = <<Optional value not found in discovery>>
	protocol = "6"
	source = "10.0.102.0/24"
	source_type = "CIDR_BLOCK"
	stateless = "false"
	#tcp_options = <<Optional value not found in discovery>>
	#udp_options = <<Optional value not found in discovery>>
	}
	ingress_security_rules {
	#description = <<Optional value not found in discovery>>
	#icmp_options = <<Optional value not found in discovery>>
	protocol = "1"
	source = "10.0.102.0/24"
	source_type = "CIDR_BLOCK"
	stateless = "false"
	#tcp_options = <<Optional value not found in discovery>>
	#udp_options = <<Optional value not found in discovery>>
	}
	ingress_security_rules {
	#description = <<Optional value not found in discovery>>
		#tcp_options = <<Optional value not found in discovery>>
		#udp_options = <<Optional value not found in discovery>>
		}
		egress_security_rules {
		#description = <<Optional value not found in discovery>>
		destination = "0.0.0.0/0"
		destination_type = "CIDR_BLOCK"
		#icmp_options = <<Optional value not found in discovery>>
		protocol = "1"
		stateless = "false"
		#tcp_options = <<Optional value not found in discovery>>
		#udp_options = <<Optional value not found in discovery>>
		}
		freeform_tags = {
		}
		ingress_security_rules {
		#description = <<Optional value not found in discovery>>
		#icmp_options = <<Optional value not found in discovery>>
		protocol = "6"
		source = "10.10.102.0/24"
		source_type = "CIDR_BLOCK"
		stateless = "false"
		#tcp_options = <<Optional value not found in discovery>>
		#udp_options = <<Optional value not found in discovery>>
		}
		ingress_security_rules {
		#description = <<Optional value not found in discovery>>
		#icmp_options = <<Optional value not found in discovery>>
		protocol = "1"
		source = "10.10.102.0/24"
		source_type = "CIDR_BLOCK"
		stateless = "false"
		#tcp_options = <<Optional value not found in discovery>>
		#udp_options = <<Optional value not found in discovery>>

	#icmp_options = <<Optional value not found in discovery>>
	protocol = "6"
	source = "10.0.103.0/24"
	source_type = "CIDR_BLOCK"
	stateless = "false"
	tcp_options {
	max = "22"
	min = "22"
	#source_port_range = <<Optional value not found in discovery>>
	}
	#udp_options = <<Optional value not found in discovery>>
	}
	ingress_security_rules {
	#description = <<Optional value not found in discovery>>
	#icmp_options = <<Optional value not found in discovery>>
	protocol = "6"
	source = "10.0.103.0/24"
	source_type = "CIDR_BLOCK"
	stateless = "false"
	tcp_options {
	max = "1530"
	min = "1521"
	#source_port_range = <<Optional value not found in discovery>>
	}
	#udp_options = <<Optional value not found in discovery>>
	}
	vcn_id = oci_core_vcn.export_iad-cloudmaa-vcn.id
	}
		}
		ingress_security_rules {
		#description = <<Optional value not found in discovery>>
		#icmp_options = <<Optional value not found in discovery>>
		protocol = "6"
		source = "10.10.103.0/24"
		source_type = "CIDR_BLOCK"
		stateless = "false"
		tcp_options {
		max = "22"
		min = "22"
		#source_port_range = <<Optional value not found in discovery>>
		}
		#udp_options = <<Optional value not found in discovery>>
		}
		ingress_security_rules {
		#description = <<Optional value not found in discovery>>
		#icmp_options = <<Optional value not found in discovery>>
		protocol = "6"
		source = "10.10.103.0/24"
		source_type = "CIDR_BLOCK"
		stateless = "false"
		tcp_options {
		max = "1530"
		min = "1521"
		#source_port_range = <<Optional value not found in discovery>>
		}
		#udp_options = <<Optional value not found in discovery>>
		}
		vcn_id = oci_core_vcn.phx-cloudmaa-vcn.id
		}

Subnet	resource oci_core_subnet export_exadb-private-subnet-ad2 {
	availability_domain = "LoSv:US-ASHBURN-AD-2"
	cidr_block = "10.0.101.0/24"
	compartment_id = var.compartment_ocid
	defined_tags = {
	"Oracle-Tags.CreatedBy" = "<Oracle-
		Modifications include removing "export_" where it appears, changing CIDR to a subnet within the VCN for the Phoenix region, changing the availability domain, changing the route table and VCN references.
		
		resource oci_core_subnet exadb-private-subnet-ad1 {
		availability_domain = "LoSv:US-PHOENIX-AD-1"
		cidr_block = "10.10.101.0/24"
		compartment_id = var.compartment_ocid

	Tags.CreatedBy ocid>/<oci user name>"
	"Oracle-Tags.CreatedOn" = "2020-03-13T18:50:55.371Z"
	}
	dhcp_options_id = oci_core_vcn.export_iad-cloudmaa-vcn.default_dhcp_options_id
	display_name = "exadb-private-subnet-ad2"
	dns_label = "exadbprivate"
	freeform_tags = {
	}
	#ipv6cidr_block = <<Optional value not found in discovery>>
	prohibit_internet_ingress = "true"
	prohibit_public_ip_on_vnic = "true"
	route_table_id = oci_core_route_table.export_iad-db-private-RT.id
	security_list_ids = [
	oci_core_security_list.export_siteguard-seclist.id,
	oci_core_security_list.export_bastion-private-seclist.id,
	oci_core_security_list.export_iad-db-private-seclist.id,
	]
	vcn_id = oci_core_vcn.export_iad-cloudmaa-vcn.id
	}
		defined_tags = {
		"Oracle-Tags.CreatedBy" = "<Oracle-Tags.CreatedBy ocid>/<oci user name>"
		"Oracle-Tags.CreatedOn" = "2020-03-13T18:50:55.371Z"
		}
		dhcp_options_id = oci_core_vcn.phx-cloudmaa-vcn.default_dhcp_options_id
		display_name = "exadb-private-subnet-ad1"
		dns_label = "exadbprivate"
		freeform_tags = {
		}
		#ipv6cidr_block = <<Optional value not found in discovery>>
		prohibit_internet_ingress = "true"
		prohibit_public_ip_on_vnic = "true"
		route_table_id = oci_core_route_table.phx-db-private-RT.id
		security_list_ids = [
		oci_core_security_list.siteguard-seclist.id,
		oci_core_security_list.bastion-private-seclist.id,
		oci_core_security_list.phx-db-private-seclist.id,
		]
		vcn_id = oci_core_vcn.phx-cloudmaa-vcn.id
		}

As there are patterns to the items that must be changed, using editing tools such as sed can aid in automating the necessary changes.

If you provisioned some components using Terraform and other components using the OCI console or other means, then you must adjust the Terraform resource definitions you plan to use. For example, if you provisioned the VCN and a NAT gateway using the OCI console, then any resource that references the VCN and the NAT gateway within the .tf files will need the following change:

1. In the vars.tf file, add and set the value of the two variables vcn_ocid and nat_gateway_ocid with these patterns:
```
variable vcn_ocid { default = "<OCID of VCN>" }
variable nat_gateway_ocid { default = "<OCID of NAT gateway>" }
```
2. Search all .tf files that have resources with definitions that have references to the VCN and/or the NAT gateway. For example, search for the pattern “vcn_id” and “network_entity_id”. For each occurrence, set the variable to the new value, as shown below:
```
vcn_id = "${var.vcn_ocid}"
network_entity_id = "${var.nat_gateway_ocid}"
```

3. Modify the availability_domain.tf file to include all availability domains in the target region. Find the list of availability domains in OCI by clicking Compute, then Instance. The availability domains will be shown on the left side of your screen.

   Using Phoenix as an example:

   ```
   ## This configuration was generated by terraform-provider-oci
   ## then modified to include all ADs at the target
   data oci_identity_availability_domain LoSv-US-PHOENIX-AD-1 {
   compartment_id = var.compartment_ocid
   ad_number = "1"
   }
   data oci_identity_availability_domain LoSv-US-PHOENIX-AD-2 {
   compartment_id = var.compartment_ocid
   ad_number = "2"
   }
   data oci_identity_availability_domain LoSv-US-PHOENIX-AD-3 {
   compartment_id = var.compartment_ocid
   ad_number = "3"
   }
   ```

Note: Get the OCID from the OCI console by clicking the **Show** or **Copy** link of the OCID for the desired resource.

Here is an example of changes required to the core.tf file containing the definition of the route table resource that uses the variables defined above:

```
resource oci_core_route_table phx-db-private-RT {
compartment_id = var.compartment_ocid
defined_tags = {
}
display_name = "phx-db-private-RT"
freeform_tags = {
}
route_rules {
#description = <<Optional value not found in discovery>>
destination = "0.0.0.0/0"
destination_type = "CIDR_BLOCK"
#network_entity_id = oci_core_nat_gateway.maa-phx-ngw.id
network_entity_id = "${var.nat_gateway_ocid}"
}
#vcn_id = oci_core_vcn.ebs-maacloud2-vcn.id
vcn_id = "${var.vcn_ocid}"
}
```

13.2 Deploying Resources with Terraform

Once you have finished editing the Terraform files, you will collect and deploy the .tf files describing these resources. You must have the following files:

- vars.tf – This file contains all Terraform variables required to execute Terraform.

- availability_domain.tf – This file contains the definitions of all availability domains for the secondary region.
- One or more .tf files that contain the resource definitions for deploying the chosen resources.

You do not need to include all the .tf files that were generated by the Terraform discovery process at the primary site. Only the files mentioned above are required.

Follow these steps to use the OCI console to deploy the resources:

1. Zip up the required .tf files into a single ZIP file. This will be used to create your Terraform stack.
2. Log in to the OCI console and navigate to **Development Services**, then **Stacks** under **Resource Manager**.
3. Use the compartment combo-dropdown to specify the compartment you want the stack ZIP file to be placed.
4. Click the **Create Stack** button.
5. Select the **My Configuration** option.
6. Under **Terraform Source**, choose **Zip file** then browse and select the ZIP file you created in step 1 above.
7. Optional: Provide a name for your stack.
8. Optional: Provide a description of your stack.
9. Select the compartment in which the stack is to be created.
10. Best practice: Select the latest version of Terraform.
11. Optional: Add any tags.
12. Click **Next**.
13. A page holding a list of variables read from the vars.tf file will be displayed. Verify that the variables have the correct values; change them if they are incorrect.
14. Click **Next**.
15. The **Review** page is shown. As you are only creating a Terraform stack – which is simply a definition of all the resources to be deployed – **do NOT** check the checkbox for "Run Apply".
16. Click **Create**.

Once the Terraform stack is created, the **Stack Details** page is shown with several action buttons, one of which is **Plan** Click the **Plan** button to create the plan.

Terraform will validate the stack while it is creating the plan. If creation of the plan fails, the OCI console will indicate that the job failed and will display the log, which will show which .tf files and which resource definitions had an error. Correct the errors in the .tf files, recreate the Terraform stack, and try to create the Plan again.

Once all errors have been resolved and the plan job runs successfully, click on **Apply** to start a job that will create all the resources defined in the Terraform stack. The amount of time the job will run depends on the type and number of resources being deployed. Deploying the network structure should be quick. Creating compute instances or a database service (VM DB or ExaDB-D) will take a measurable amount of time.

14 Automation Scripts for Switchover, Failover, File Sync

These scripts are provided as a potential starting point for implementing full stack automation for EBS Release 12.2. They are examples. They are not part of any Oracle product nor are they under any warranty. They are intended for customers to learn from and perhaps modify for their deployment.

The scripts for this project fall into the following categories:

- Scripts to set up database structures needed for the process.
- Scripts to configure the database nodes during switchover if necessary, executed via a database role change trigger.
- Scripts that replicate middle tier file system contents from one site to another.
- Wrapper scripts that call the EBS startup and shutdown scripts as well as managing file system replication and application configuration, within the context of simple startup/shutdown, standby testing, and site role transition.
- A set of standard functions that make the scripted tasks easier to implement and maintain.

The wrapper scripts can be used by OCI Full Stack DR to automate switchover and failover.

Always thoroughly test your scripts in a test environment prior to promoting to production.

14.1 Prerequisites

These requirements must be in place to implement the scripts described in this section.

- Logical hostnames – This simplified, reduced-outage method of switching between sites relies on logical hostnames for both the application and database tiers at both the primary and secondary sites.
- OCI Command-line Interface (OCI CLI) – The OCI CLI must be installed and configured on all servers hosting or accessing the database or application tier file system, at each region.
- Region-local OCI vaults - To keep passwords secret, all required passwords must be stored in region-local OCI vaults that can be accessed by the OCI CLI. This includes the SYS password for the CDB, the EBS_SYSTEM schema password, the EBS APPS password, and the WebLogic password.
- Cross-site user equivalency – A pair of application tier servers (one from each region) need to have user equivalency configured, to allow passwordless rsync access. Note that while user equivalency can be met with a pair of compute instance VMs that are also serving as EBS middle tier servers, we strongly recommend using a pair of servers dedicated to the task of keeping the file systems in sync. This simplifies snapshot standby testing.

We also set an environment variable SCRIPT_DIR in the administrative user's unix profile, pointing to the custom directory holding these scripts.

14.2 KSH Standard Functions

This script holds a set of standard functions – generic routines that are useful when scripting Linux, database, and EBS administrative tasks – to include in calling scripts on both the database and the application tier. Using these common functions simplifies each working script, making the code easier to write and maintain.

Korn shell (KSH) was used for these scripts, as it provides the ability to use a *coroutine* – a process that is forked by the main shell script but not spawned to be fully separate. The KSH coroutine remains an attached "child" of the outer KSH script, able to receive requests from the outer script and return information to the outer script. In this case, the outer script forks a SQL*Plus session as a coroutine. The SQL*Plus session remains in place for the duration of the execution life of the shell script. This removes the overhead of repeatedly spawning SQL*Plus for every single database request.

As this file is simply a collection of common routines to include in a functionally specific script, it does not start with a shebang. The driving script needs its first line to be `#!ksh` for these routines to work properly.

```
###############################################################################
# stdfuncs.sh
# Collection of standard functions.
#
# Rev:
# 05/27/25  Added check for error messages on login, adjusted CkRunDB
# 10/01/24  added CkRunDB
# 9/27/24   Updated for today's ksh, cloud tooling
# 7/19/99   added GetLogon
# 1/17/96   added MultiThread and CkRun
# 8/23/95   created
###############################################################################
# GetLogon:
# Get the password for the user secret name passed in.  Set the variable
# LOGON. In the calling routine, set the intended password variable.
#
###############################################################################
GetLogon()
{
if [ $# = 0 ]; then
   LogMsg "No user specified"
   exit 1
fi

uname=$1

SECRET_OCID=$(oci vault secret list -c $COMPARTMENT_OCID --raw-output --query "data[?\"secret-
name\" == '$uname'].id | [0]")

if [ ${#SECRET_OCID} = 0 ]; then
   LogMsg "User ${uname} not set up"
   exit 1
fi

LOGON=$(oci secrets secret-bundle get --raw-output --secret-id $SECRET_OCID --query
"data.\"secret-bundle-content\".content" | base64 -d )

if [ ${#LOGON} = 0 ]; then
   LogMsg "No password found for uesr ${uname}"
   exit 1
fi

}
```

```
############################################################################
# EnvSetup:
# Set up the unix and oracle environment as needed...
#
# Do we need this?  How should it be adjusted for modern era?
# Removed export of ORAENV_ASK=NO
#
############################################################################
EnvSetup()
{
# set TS to the timestamp.  Used to create the output directory for the run.
TS=`date +"%Y%m%d_%H%M%S"`
myPID=`echo $$`
HostName=$(hostname)

}

############################################################################
# LaunchCoroutine:
# Start a sqlplus session to pipe commands to from this shell session.
#
# Input:  DB password secret needs to be set prior to calling
# Output: None, but the coroutine stays up and running
# Return: exits if error is detected.
############################################################################
LaunchCoroutine()
{
uname=$1
pwd=$2
cxHost=$3

cxString=$uname/$pwd@$cxHost

if [[ ! "$uname" || ! "$pwd" || ! "$cxHost" ]]; then
  LogMsg "Usage: LaunchCoroutine user_name password host"
  LogMsg "User name: $uname"
  LogMsg "Password: $pwd"
  LogMsg "Host: $cxHost"
  LogMsg "Connection string: $cxString"
  exit 1
fi

# Verify the Oracle connection
sqlplus -silent <<! >$TMP_FILE 2>&1
${cxString}
!

# Check for success:
```

```
if [ $? -ne 0 ]; then
    LogMsg "LaunchCoroutine: Unable to connect to Oracle."
    LogMsg "LaunchCoroutine: Verify password, state of database"
    cat $TMP_FILE | LogMsg
    exit 1
fi

# sqlplus might have exited with a return code of 0, but there still may be
# a login error such as ORA-12514.  These will be in the TMP_FILE.
rc=`grep -i "ERROR|ORA-|SP2-" $TMP_FILE | wc -l`
if [ $rc != 0 ]; then
    LogMsg "LaunchCoroutine: Received error connecting to Oracle."
    LogMsg "Errors can be found in ${TMP_FILE}:"
    cat $TMP_FILE | LogMsg
#     tmpstr=`cat "${TMP_FILE}" `
#     LogMsg "${tmpstr}"
    exit 1
fi

# Launch the coroutine itself.
sqlplus -s |&
echo -e "${cxString}" >&p
echo -e "set scan off\n" >&p
echo -e "set heading off\n" >&p
echo -e "set feedback off\n" >&p
echo -e "set pagesize 0\n" >&p
echo -e "set linesize 2000\n" >&p
echo -e "set long 65535\n" >&p
echo -e "select '<-DONE->' from dual;\n" >&p

tmp="$IFS"
IFS=""

# read from the coroutine's output 'til you see <-DONE->
while [ "1" ]; do
    read answer <&p
    if [ "$answer" = "<-DONE->" ]; then
      break
    fi
done

IFS="$tmp"
rm $TMP_FILE

}

##############################################################################
# ExecSql
# Send a SQL statement to the sqlplus coroutine; return the result to the
# calling routine.
#
#  Input: sql statement
# Output: result of sql statement
# Return: none
###############################################################################
ExecSql ()
{
# send the statement to the coroutine with the "-p"
  echo -e "$1\n" >&p
  echo -e "select '<-DONE->' from dual;\n" >&p

  tmp="$IFS"
  IFS=""

# read from the coroutine's output 'til you see <-DONE->
  while [ "1" ]; do
    read answer <&p

    if [ "$answer" = "<-DONE->" ]; then
      break
    else
      echo -e "$answer"
    fi
  done

  IFS="$tmp"
}

###############################################################################
# LogMsg: Echo a message to stdout and the run's log file.
#  Usage: LogMsg "message text"
#
#  Input: Message text
#         Note: If message text = "-p", then input is read from stdin
#         (so you can pipe to this).
# Output: Writes the timestamp, host name, database name, and message to
#         stdout and to LOG_OUT.  Requires LOG_OUT be already defined
# Return: None
###############################################################################
LogMsg ()
{
  LogMsgTimestamp=`date +"%m%d %T"`

# If they are piping the message in to print from a command
# (eg, cat file | LogMsg), need to reset the interfield separator.
# After that, we read the message piece by piece from stdin.
  if [ "$1" = "-p" ]; then
    tmp="$IFS"
    IFS=""
    while read message
    do
      printf "${LogMsgTimestamp}|${HostName}|${DbName}|${myPID}| $message\n"
      if [ ! -z "$LOG_OUT" ]; then
         printf "${LogMsgTimestamp}|${HostName}|${DbName}|${myPID}| $message\n" >> $LOG_OUT
      fi
    done
    IFS="$tmp"

# Not piping anything to the message - just text passed straight in
  else
    printf "${LogMsgTimestamp}|${HostName}|${DbName}|${myPID} $1\n"
    if [ ! -z "$LOG_OUT" ]; then
       printf "${LogMsgTimestamp}|${HostName}|${DbName}|${myPID} $1\n" >> $LOG_OUT
    fi
  fi
}

#############################################################################
# GetDbName
# Get the database name for this oracle sid
#
# Input:  None
# Output: Sets DbName variable
# Return: None
#############################################################################
GetDbName()
{
  sql="select name from v\$database;"
  DbName=`ExecSql "${sql}"`
}

##############################################################################
# CkRunDB
# Monitor running DATABASE processes to see if any are still running.
# To use this, set up a construct inside the database that you can check to
# determine that the required work is complete.  A simple example is to insert
# a row in a table for each chunk of work to be done, then to remove the row
# as part of the committed chunk of work.
#
# This process will execute your sql statement built to check for your
# completion criteria.  It will return control when your sql statement reports
# the criteria is met.
#
# REQUIRED: set the variable "sql" to the sql statement that will retrieve the
#           completion criteria.  This query must return null / empty or the
#           value 0 when the task is complete.
#
```

```
#  Input: number of seconds to sleep between checks (required)
#         HIDDEN: set the variable $sql to your sql statement as described
#         above
# Output: Items matching what you're checking for, sleep time, a dividing line
# Return: Returns control when query returns zero matches
##############################################################################
CkRunDB()
{
thislong=$1
LogMsg "CkRunDB: Monitoring for $sql"

if [ -z "$thislong" -o -z "$sql" ]; then
   LogMsg "CkRunDB: ERROR: Insufficient arguments."
   LogMsg "CkRunDB: thislong: $thislong"
   LogMsg "CkRunDB: sql: $sql"
   exit 1
fi

while true
do
   matches=`ExecSql "${sql}"`
   LogMsg "CkRunDB: Matches: $matches"
   if [ -z "${matches}" ] || [ "${matches}" == "0" ]; then
      LogMsg "CkRunDB: ==================================================================="
      LogMsg "Break from CkRunDB"
      break;
   fi
   LogMsg "Sleeping $thislong"
   LogMsg "CkRunDB: ==================================================================="
   sleep $thislong
done
LogMsg "CkRunDB: Returning control to calling routine."
}

##############################################################################
# CkRunOS
# Monitor running Unix processes to see if any are still running.
#
#  Input: 1: Unique thing to check on a ps line (required)
#         2: number of seconds to sleep between checks (required)
#         3: S if the run is to be relatively Silent
# Output: None.
# Return: Returns control when all monitored processes are complete.
##############################################################################
CkRunOS()
{
ckit=$1
thislong=$2
silent=$3
LogMsg "CkRunOS: Monitoring for $ckit"

if [ "$ckit" = "" ] || [ "$thislong" = "" ]; then
   LogMsg: "CkRunOS: ERROR: Insufficient arguments."
   exit 1
fi

while true
do
   if [ "$silent" != "S" ]; then
      ps -ef | grep $ckit | grep -v grep | grep -v ckrun
   fi
   matches=`ps -ef | grep $ckit | grep $$ | grep -v grep | wc -l`
   LogMsg "CkRunOS: Matches: $matches"
   LogMsg "CkRunOS: ====================================================================="
   if [ $matches = 0 ]; then
      LogMsg "Break from CkRunOS"
      break;
   fi
   LogMsg "CkRunOS: Sleeping $thislong"
   sleep $thislong
done

LogMsg "CkRunOS: Returning control to calling routine."
}

################################################################################
# MultiThread
# Submit multiple tasks in the background.  "Mark time" until they are all
# complete, then return control to the calling routine.
#
# You provide a file which has command lines for the tasks, one task per line.
# These lines are "nohup'd" and run in the background.  Control is returned
# to your calling routine when all the tasks are complete.
#
# You limit the number of tasks which can be run at once, and you provide a
# unique piece of info to "grep" for on a ps -ef line, to use to count the
# number of running tasks and determine whether or not there are tasks still
# running.
#
# Possible "gotcha": if you are vi'ing or otherwise accessing something
# which puts your "unique" grep field on the ps command line, this routine
# WILL NOT return control to the calling routine!
#
#  Input: 1: name of a file which holds commands to run.  one line per command.
#         2: max # processes to run at once
#         3: unique part of the command line to "grep" for when determining
#            how many of these processes are running
#         4: number of seconds to sleep between checks for task completion
#         5: verbose mode (yes/no) -- not used
# Output: Your tasks are complete.
# Return: exit 1 if incomplete info is provided.
##############################################################################
MultiThread()
{
tasks=$1
max_processes=$2
differentiator=$3
sleep_time=$4

LogMsg "MultiThread: Starting multi-threaded execution of $tasks."
LogMsg "MultiThread: No more than $max_processes will run at once."
LogMsg "MultiThread: Using \"$differentiator\" for unique ps search."

# read the tasks from the file.  for each one:
cat $tasks | while read i
do
   LogMsg "MultiThread: Starting $i"
   nohup ${i} >/dev/null 2>&1 &
   # count the number of jobs running already.
   matches=`ps -ef | grep $differentiator | grep -v grep | grep -v ckrun | wc -l`
   LogMsg "matches: $matches"
   # if the number of jobs running is >= max_processes, fall into this loop.
   # otherwise go through the outer loop again to kick off another job.
   until [ $matches -lt $max_processes ]
   do
      LogMsg "==========================================================================="
      sleep $sleep_time
      matches=`ps -ef | grep $differentiator | grep -v grep | grep -v ckrun | wc -l`
      LogMsg "MultiThread: $matches tasks running..."
   done
done

# Run the monitor.  It finishes when the tasks are done.  Need this,
# even though the jobs are kicked off when needed, as when the above loop
# is done there are still tasks running.  We can't return control to the
# calling routine until they are complete.
# CkRunOS $differentiator $sleep_time S
CkRunOS $differentiator $sleep_time

LogMsg "MultiThread: Tasks complete.  Returning control to calling routine."
}

##############################################################################
# initialize common variables.  this is NOT inside a function so that it gets
# executed right away.  It sets up a scratch pad for the run.
##############################################################################
TMP=/tmp
```

```
TMP_FILE=$TMP/TMP_FILE.$$
```

14.3 Database Scripts and Role Change Trigger

In OCI, when the database software home is created for the standby database, there is no guarantee that the oracle home path will match that of the primary. If, after a switchover or failover, the oracle home path is different from that at the primary, EBS tooling and run-time operations will fail. We've provided a mechanism to check for this condition, and, if needed, to reset the oracle database home path and reconfigure `UTL_FILE_DIR` on the new primary database nodes, on switchover or failover:

- A database role change trigger that is fired when the database transitions from standby to primary. The trigger uses `DBMS_SCHEDULER` to dispatch a job that runs the script `EBS_DB_RoleChange.sh`. This job is configured to execute immediately.
- The script `EBS_DB_RoleChange.sh`, which tests to see if the database home path is the same here on the standby (new primary) as it was on the (old) primary. If it is the same, it exits – no more work is needed. If it is different, it inserts one row for each database instance into a custom controlling table (`APPS.XXX_EBS_ROLE_CHANGE`), then executes the EBS code to reconfigure the database home paths, once on each RAC node:

If reconfiguration is needed, as configuration work is completed on each RAC database node, that database node's row in the controlling table is deleted. The custom EBS startup script we provide for the application tier checks to be sure that table is empty when executed using "Switchover" mode, signifying the database configuration is complete and it is safe to start application services.

The first RAC instance that completes the role change transition to the PRIMARY role will fire the database role change trigger. As this cannot be determined in advance, the directory structure must be the same on all RAC nodes and these scripts must be deployed to each RAC node.

14.3.1 DB Node Environment Files

The scripts provided here source specific environment files as part of their execution. One environment file is `EBSCFG.env`, which must be modified for your environment. Here is an example from our project:

```
###############################################################################
# EBSCFG.env
# This environment file is called by scripts on the DB nodes that execute EBS
# TXK commands to configure EBS metadata post switchover or failover.
# The scripts assume logical host names have been implemented for EBS R12.2.
#
# Environment variables in this env file:
# - SCRIPT_DIR: The directory where these scripts reside.
# - LOG_DIR: These scripts will write their execution logs in this directory
# - THIS_HOSTNAME: The logical host name for THIS host, without the .domain.
# - VIRTUAL_HOSTNAME: The logicalhostname.domain alias for the VIP on THIS.
#              node. This would be the VIP for the Grid listener or the VIP
#              of a dedicated EBS listener.  Use the command
#              "srvctl config vip -n $HOSTNAME" to get this setting.
# - LOGICAL_HOSTNAME: The logical hostname.domain for THIS server. The EBS
#              script txkSyncDBConfig.pl will use this variable to configure
#              EBS on THIS node. If using physical hostnames, then
#              set LOGICAL_HOSTNAME=""
# - SCAN_NAME: The SCAN name for THIS cluster. Use the command: "srvctl
              config scan" to obtain the scan name.
# - SCAN_LISTENER_PORT: The listener port number for the SCAN listeners.
#             Use the command "srvctl config scan_listener" to get this
#             value.
# - DB_LISTENER_PORT: The listener port number for the database. This can
#             be the Grid listener port or the listener port of a dedicated
#             EBS listener if one is configured and running.
# - DB_HOSTS: Space-delimited list of network assigned / physical hostnames
#             for the EBS RAC database, enclosed in double quotes.
#             Do not use logical host names for this list.
#             Example: "physicalhost1 physicalhost2 physicalhostN..."
# - CDB_NAME: The name of the database CDB as set by the database parameter
#             DB_NAME.
# - DB_UNIQUE_NAME: The database unique name set by the database parameter
#             DB_UNIQUE_NAME.
# - PDB_NAME: The name of the PDB that the EBS database resides in.
# - PDB_TNS_CONNECT_STRING: The TNS connect string for the PDB
# - CONTEXT_ENV_FILE: The file holding DB configurations required by EBS
# - DB_SECRET_NAME: The name of the secret in the OCI vault holding the
#             database EBS_SYSTEM password/credential.
# - APPS_SECRET_NAME: The name of the secret in the OCI vault holding the
#             APPS schema password/credential.
# - APPS_USERNAME: The name of the APPS schema, typically "APPS".
# - COMPARTMENT_OCID: The OCID of the OCI compartment that contains the
#             OCI vault.
# - TNS_ALIAS: <PDB_NAME>
# - EBS_DEFAULT_SERVICE_NAME: If relying on EBS service definitions within
#             the PDB directory structure, services may not start when the
#             PDB is opened as part of a role transition.  See the
#             StartDBService.sh script for more info.
###############################################################################

SCRIPT_DIR=/home/oracle/ebscdb/custom_admin_scripts/VISPRD
LOG_DIR=${SCRIPT_DIR}/log

THIS_HOSTNAME=ebsdb1
VIRTUAL_HOSTNAME=ebsdb1-vip.example.com
LOGICAL_HOSTNAME=ebsdb1.example.com

SCAN_NAME=iadexadb-bw5wn-scan.example.com
SCAN_LISTENER_PORT=1521
DB_LISTENER_PORT=1521

DB_HOSTS="iadexadb-bw5wn1 iadexadb-bw5wn2"
CDB_NAME=EBSCDB
DB_UNIQUE_NAME=EBSCDB_9cv_iad

PDB_NAME=VISPRD
```

```
PDB_TNS_CONNECT_STRING=visprd

CONTEXT_ENV_FILE=${ORACLE_HOME}/${PDB_NAME}_${THIS_HOSTNAME}.env

DB_SECRET_NAME=ebs-db-secret
APPS_SECRET_NAME=ebs-apps-secret
APPS_USERNAME=APPS

# put your OCID here..
COMPARTMENT_OCID="<OCID of compartment that contains the OCI vault>"

TNS_ALIAS=visprd

# This is only required if relying on TNS definitions in the PDB directory
# EBS_DEFAULT_SERVICE_NAME=ebs_VISPRD
```

14.3.2 Create the Database Role Change Trigger

The script that creates the database role change trigger is run one time, on the primary database, when setting up the automation of the switchover / failover processes. This script creates the custom table `apps.xxx_EBS_role_change` and the database role change trigger. As these structures are in the database, this work is replicated to the standby database via standard standby database recovery replay.

```
REM
REM Name: crt_db_role_change_trigger.sql
REM
REM Purpose: Create a role change trigger that will submit an external job to
REM the database scheduler.  The role change trigger will run on the database server
REM hosting the first instance that comes up when changing roles to primary.
REM
REM The job that is scheduled is the script EBS_DB_RoleChange.sh, which will change
REM oracle home paths stored in the database if there’s a difference between primary
REM and secondary
REM
REM The job that runs on each RAC node will insert a row into the
REM apps.XXX_EBS_ROLE_CHANGE table.  Application tier configuration will
REM only start when that table is empty.
REM NOTE: This table is not managed by ADOP and does not undergo any
REM edition-based redefinition changes.
set echo on
alter session set container = <PDB name>;


REM Ignore error if table does not exist
drop table apps.xxx_EBS_role_change;


create table apps.xxx_EBS_role_change (
host_name varchar2(128),
rolechange_date date
);

REM Role change database trigger must be defined at the CDB level.
alter session set container = CDB$ROOT;

CREATE OR REPLACE TRIGGER Configure_EBS_AfterRoleChange AFTER db_role_change ON DATABASE
DECLARE
v_role VARCHAR(30);
BEGIN
SELECT DATABASE_ROLE INTO v_role FROM V$DATABASE;
IF v_role = 'PRIMARY' THEN
   -- Submit the job that will change the paths if needed.
   -- The job will run immediately.
   DBMS_SCHEDULER.CREATE_JOB (
   Job_name=> 'EBS_DB_RoleChange',
   Job_type=> 'EXECUTABLE',
   Job_action => '/home/oracle/ebscdb/custom_admin_scripts/EBS_DB_RoleChange.sh',
   Enabled => TRUE
   );
   DBMS_SCHEDULER.ENABLE('EBS_DB_RoleChange');
END IF;
END;
/
alter trigger Configure_EBS_AfterRoleChange enable;
```

14.3.3 Database Role Change Script

The `EBS_DB_RoleChange.sh` script is called by the database role change trigger created above. When the physical standby database transitions to the primary role, the RAC instance that performs the role transition will fire the role change trigger. This script must be placed on all RAC database nodes hosting the EBS database at both primary and standby sites so that any database instance at either site is able to perform a role change to primary from standby.

Note – if the paths are different, the EBS configuration scripts will run autoconfig twice. Autoconfig will fail during the first run if `UTL_FILE_DIR` is not yet configured for this node. It will be configured during the second run, so the configuration will be complete.

```
#!/bin/ksh
############################################################################
# EBS_DB_RoleChange.sh
# Spawn the scripts to reconfigure EBS on the database RAC nodes if required.
#
# Note: This script blocks when running the txk configuration scripts, so
#       errors are caught here instead of floating around in the ether.
#
# Note: If the homes need to change: We're assuming all database instances
#       are visible in gv$instance when this is executed.  If an instance
#       is down, THAT HOME WILL NOT BE RECONFIGURED.
#
```

```
# Requires user equivalency across all RAC nodes.
#
# Things to ignore or be aware of:
# stty: standard input: Inappropriate ioctl for device.  Irritating but
#       benign.
# The random creation of +DATAC1 in your script directory (bug#: .38086306)
# Creation of logs in your script directory named mmddhhmm.log
#
# No parameters passed in
#
# Rev:
# 06/18/2025 Using info in .profile to build environment, adjusted comments
# 04/28/25   Added check for logical hostnames.
# 8/28/24    Simplified, added ksh coroutine for efficiency
# 1/15/24    Created
###########################################################################

# Set up the environment variables
. $HOME/EBSCDB.env
. ${SCRIPT_DIR}/EBSCFG.env
. ${CONTEXT_ENV_FILE}


HostName=$(hostname)
MYHOST=$(hostname)

# Call the standard functions routine
. $SCRIPT_DIR/stdfuncs.sh

# Include the common code for reconfiguring the path.  This will be
# executed both from this script and from the spawned shell scripts on
# other RAC DB nodes
. $SCRIPT_DIR/ChangeHomePath.sh

EnvSetup
LOG_OUT=${LOG_DIR}/${HostName}_EBS_DB_RoleChange_${TS}.log
LogMsg "EBS_DB_RoleChange.sh: Started"

# OPTIONAL, and commented out here:
# Start any required database services that do not get started automatically
# by CRS, or that are defined only in the PDB itself.
# IF THIS IS REQUIRED IN YOUR ENVIRONMENT, uncomment the code for
# running $SCRIPT_DIR/startDBServices.sh.
# More complete explanation in startDBServices.sh.

# This work is done with the default DB login – can’t use the EBS database
# login data for this.

# First, check if the service might already be up.
```

```
#if [ s=$(lsnrctl status | grep -i ebs_${PDB_NAME} | wc -l ) -eq 0 ]; then
#   $SCRIPT_DIR/StartDBServices.sh
#   if [ $? -ne 0 ]; then
#      LogMsg "Attempt to start database services failed."
#      LogMsg "Refer to log files in ${LOG_DIR}."
#      exit 1
#   fi
#   else
#      LogMsg "Service ebs_${PDB_NAME} is already up."
#fi

# Continue with setup
GetLogon $APPS_SECRET_NAME
APPS_SECRET=$LOGON

# Start sqlplus in coroutine, logged in as apps user
LaunchCoroutine APPS $APPS_SECRET $PDB_TNS_CONNECT_STRING

GetDbName

# Main flow:
# What does EBS think my oracle home path is?
sql="select XMLTYPE(t.text).EXTRACT('//ORACLE_HOME/text()').getStringVal() \
from apps.fnd_oam_context_files t  \
  where name = '` echo ${PDB_NAME}`_`echo ${THIS_HOSTNAME}`'  \
  and node_name = '`echo ${THIS_HOSTNAME}`'  \
  order by last_update_date desc  \
  fetch first row only;"
EBS_DB_OH=`ExecSql "${sql}"`

LogMsg "ORACLE_HOME: $ORACLE_HOME"
LogMsg "EBS_DB_OH: $EBS_DB_OH"

# Only spawn the jobs if the oracle home path is different
# If the path is the same, the bulk of this work is skipped
if [ "${ORACLE_HOME}" != "${EBS_DB_OH}" ]
then
  LogMsg "Oracle home paths are different.  Reconfigure hosts"

  # Insert a row for each RAC node, then select those rows to drive the fix
  # scripts.  Need the rows in the table so the middle tiers can see
  # configuration is not done yet across all instances.

  # We have both the list of hosts in DB_HOSTS and the list of hosts
  # that have been restarted so far.  This code is a bit simplistic and assumes
  # assumes they are the same.  For a  larger environment, enhance this to be
  # sure the count of the two is the same.
  sql="INSERT INTO apps.xxx_EBS_role_change (host_name,rolechange_date) \
  SELECT host_name, sysdate \
  FROM gv\$instance;
  commit;"
  ExecSql "${sql}"

  LogMsg "All DB_HOSTS: $DB_HOSTS"
  LogMsg "This database server: MYHOST: $MYHOST"

  # DB_HOSTS is a field in your .env file that holds the names of
  # of every database host in your RAC setup
  for i in ${DB_HOSTS}
  do
    if [ "${i}" == "${MYHOST}" ];
    then
      LogMsg "Configuring database homes for EBS on local host ${i}"
      ReConfig
    else
      LogMsg "Configuring database homes for EBS on remote host: ${i}"
      LogMsg "ssh -t oracle@${i} cd ${SCRIPT_DIR}; ${SCRIPT_DIR}/callReConfig.sh"
      ssh -t oracle@${i} "cd ${SCRIPT_DIR}; ${SCRIPT_DIR}/callReConfig.sh"
    fi

    # all done - remove the rows driving the work
    sql="DELETE FROM apps.xxx_EBS_role_change where host_name='${i}';
         commit;"
    ExecSql "${sql}"
  done
fi

LogMsg "Completed: EBS_DB_RoleChange.sh."
```

14.3.4 Conditional: Start Database Services Script

In this paper, we assume you will be configuring Oracle RAC, thus we recommend that you place role-based database service definitions under the Clusterware / the Grid home directory.

As a standard that serves all customers – including those who do not use Oracle Clusterware – E-Business Suite creates its network services under the PDB home directory. These services are created:

- `ebs_<PDB_NAME>`
- `<PDB_NAME>_ebs_patch`

Services defined under the PDB home directory may not be started when first opening the database in the primary role as part of switchover or failover, including the EBS-defined service `ebs_<PDB_NAME>`. Refer to My Oracle Support document Best Practices for Pluggable Database End User and Application Connection and Open on Database Startup (Doc ID 2833029.1) for a more complete explanation of when this scenario might occur.

This script will ensure the services you are relying on in the PDB home directory are started on switchover from your primary to your standby even if you have not followed the instructions in this paper to define user services under the Grid home.

This script can be used to start services that:

- Are not defined and managed by Oracle Clusterware.
- Are only defined in the PDB.
- May not start when the PDB is opened.

Important: If you need to use this script, uncomment the call to startDBServices.sh in the EBS_DB_RoleChange.sh script by removing the hash tags.

startDBServices.sh:

```
#!/bin/ksh
#############################################################################
# startDBServices.sh
# Start any services that do not start when the PDB is opened, either because
# they are not managed by CRS or they are defined within a pluggable database
#
# This script should only be used to start services that are:
# - not defined in CRS
# - defined only in a PDB
# - may not start when the database and PDB it is defined in opens.
#
# This script connects to the database using the PDB's default service, one
# that is always started when the PDB is opened.  This login is never used by
# end users – it is only used to start the services that end users use.
#
# This work is only needed on switchover if your environment uses the TNS
# configuration the PDB home instead of in the grid home.
#
# No parameters passed in.
#
# Rev:
# 06/18/2025  Removed hard-coding of call to environment specific code
# 6/3/2025    Created.
#############################################################################
#
. $HOME/EBSCDB.env
. ${SCRIPT_DIR}/EBSCFG.env
HostName=$(hostname)
MYHOST=$(hostname)

# Call the standard functions routine
. $SCRIPT_DIR/stdfuncs.sh

EnvSetup
LOG_OUT=${LOG_DIR}/${HostName}_startDBServices_${TS}.log

LogMsg "startDBServices.sh: Started"
LogMsg "Starting database services ${EBS_DEFAULT_SERVICE_NAME} from node: ${MYHOST}"

GetLogon $APPS_SECRET_NAME
```

```
APPS_SECRET=$LOGON

# Connect to the PDB using sqlplus, building the connect string to
# access the default PDB service
output_str=`sqlplus -s /nolog <<EOF!
connect ${APPS_USERNAME}/${APPS_SECRET}@${SCAN_NAME}:${SCAN_LISTENER_PORT}/${PDB_NAME}
exec dbms_service.start_service('${EBS_DEFAULT_SERVICE_NAME}',DBMS_SERVICE.ALL_INSTANCES);
exit
EOF!
`
# Were you able to connect?
tmp_str=$( echo "${output_str}" | grep -i "ERROR|ORA-|SP2-" )
if [ ${#tmp_str} -ne 0 ]; then
   LogMsg "An error occurred while attemping to start database services
${EBS_DEFAULT_SERVICE_NAME}:"
   LogMsg "${output_str}"
   exit 1
fi

LogMsg "Result of sqlplus session: ${output_str}"
LogMsg "startDBServices.sh: Completed."

exit 0
```

14.3.5 Change DB Home Path Routine

This is a function included in local and remote `ksh` scripts to change the database home path on switchover. It provides functionality needed in both the `callReConfig.sh` and the `EBS_DB_RoleChange.sh` scripts.

```
############################################################################
# ChangeHomePath.sh
# An include script holding the common routine that changes the oracle
# database home path if needed when switching a standby database to primary.
#
# This is an INCLUDE file.  It is code to be included in an outer ksh
# script.  It just holds the routine ReConfig, which calls the EBS code.
# It does not need a shebang, as it is simply a part of a larger program.
#
# Note: assumes all the database instances are present in gv_instance when
# the check is done in the main script.
#
# Requires user equivalency across all RAC nodes.
#
# No parameters passed in
#
# Rev:
# 8/23/24  Re-formed as a standard routine to execute from different scripts
# 1/15/24  Created
############################################################################
```

```
ReConfig()
{
LogMsg "Started ReConfig - Database Home Reconfiguration"

LogMsg "DbName: $DbName"

# Calling script has the apps password.  Need to get the database password
GetLogon $DB_SECRET_NAME
DB_SECRET=$LOGON

PERLBIN=`dirname $ORACLE_HOME/perl/bin/perl`
LogMsg "PERLBIN = $PERLBIN"

PATH=${PERLBIN}:${PATH}

PERL5LIB=$ORACLE_HOME/perl/lib/5.32.0:$ORACLE_HOME/perl/lib/site_perl/5.32.0:$ORACLE_HOME/appsut
il/perl

# If logical hostname is passed in, call txkSyncDBConfig.pl with the
# -logicalhostname parameter, else call it without this parameter.
if [ "${LOGICAL_HOSTNAME}" == "" ];
then
   # Call the EBS script txkSyncDBConfig.pl
   LogMsg "Running txkSyncDBConfig.pl to configure with physical hostname."
   { echo "$APPS_SECRET"; } | perl $ORACLE_HOME/appsutil/bin/txkSyncDBConfig.pl \
    -dboraclehome=$ORACLE_HOME \
    -outdir=$ORACLE_HOME/appsutil/log \
    -cdbsid=${ORACLE_SID} \
    -pdbname=${PDB_NAME} \
    -dbuniquename=${DB_UNIQUE_NAME} \
    -israc=YES \
    -virtualhostname=${VIRTUAL_HOSTNAME} \
    -scanhostname=${SCAN_NAME} \
    -scanport=${SCAN_LISTENER_PORT} \
    -dbport=${DB_LISTENER_PORT} \
    -appsuser=${APPS_USERNAME} | tee -a ${LOG_OUT}
else
   LogMsg "Running txkSyncDBConfig.pl to configure with logical hostname."
   { echo "$APPS_SECRET"; } | perl $ORACLE_HOME/appsutil/bin/txkSyncDBConfig.pl \
    -dboraclehome=$ORACLE_HOME \
    -outdir=$ORACLE_HOME/appsutil/log \
    -cdbsid=${ORACLE_SID} \
    -pdbname=${PDB_NAME} \
    -dbuniquename=${DB_UNIQUE_NAME} \
    -israc=YES \
    -virtualhostname=${VIRTUAL_HOSTNAME} \
    -logicalhostname=${LOGICAL_HOSTNAME} \
    -scanhostname=${SCAN_NAME} \
    -scanport=${SCAN_LISTENER_PORT} \
    -dbport=${DB_LISTENER_PORT} \
    -appsuser=${APPS_USERNAME} | tee -a ${LOG_OUT}
fi

if [ $? -ne 0 ]; then
   LogMsg "txkSynDBConfig.pl returned an error"
   exit 1
fi
LogMsg "Running txkCfgUtlfileDir.pl with mode=setUtlFileDir"

{ echo "$APPS_SECRET"; echo "$DB_SECRET"; } | perl $ORACLE_HOME/appsutil/bin/txkCfgUtlfileDir.pl
 -contextfile=$CONTEXT_FILE \
 -oraclehome=$ORACLE_HOME \
 -outdir=$ORACLE_HOME/appsutil/log \
 -mode=setUtlFileDir \
 -servicetype=opc | tee -a ${LOG_OUT}
if [ $? -ne 0 ]; then
   LogMsg "txkCfgUtlfileDir.pl mode setUtlFileDir returned an error"
   exit 1
fi

LogMsg "Running txkCfgUtlfileDir.pl with mode=syncUtlFileDir"

{ echo "$APPS_SECRET"; } | perl $ORACLE_HOME/appsutil/bin/txkCfgUtlfileDir.pl \
 -contextfile=$CONTEXT_FILE \
 -oraclehome=$ORACLE_HOME \
 -outdir=$ORACLE_HOME/appsutil/log \
 -mode=syncUtlFileDir \
 -servicetype=opc | tee -a ${LOG_OUT}
if [ $? -ne 0 ]; then
   LogMsg "txkCfgUtlfileDir.pl mode syncUtlFileDir returned an error"
   exit 1
fi

LogMsg "Completed ReConfig - Database Home Reconfiguration"
}
```

14.3.6 Remote DB Home Path Update

Reconfigure the database home paths on remote database nodes.

```
#!/bin/ksh
###############################################################################
# callReConfig.sh
# Call the scripts to reconfigure EBS on the database RAC nodes on remote
# hosts.
#
# Requires user equivalency across all RAC nodes.
#
```

```
# This script establishes the environment, gets the apps password.
# then executes ReConfig on remote hosts.  It's called remotely by
# EBS_DB_RoleChange.sh
#
# Rev:
# 06/18/2025  No hard-coding to find env files
# 08/29/2024  Created
##############################################################################

mypath=`pwd`

. $HOME/EBSCDB.env
. ${SCRIPT_DIR}/EBSCFG.env
HostName=$(hostname)

. ${CONTEXT_ENV_FILE}

# Include the standard functions routines
. ${SCRIPT_DIR}/stdfuncs.sh

# Include the common code for reconfiguring the path.  This holds the
# code that is executed in this script for the DB nodes remote to the main
# driving script.
. ${SCRIPT_DIR}/ChangeHomePath.sh

EnvSetup
LOG_OUT=${LOG_DIR}/${HostName}_callReConfig_${TS}.log

LogMsg "callReConfig.sh: Started"

# Get the apps password - needs to be set before executing ReConfig
GetLogon $APPS_SECRET_NAME
APPS_SECRET=$LOGON

# Note: we are not executing sqlplus for anything in this script, so are
# not launching the coroutine.  Side effect: for this script, we need
# to substitute HostName for DbName, which is used in LogMsg
# GetDbName
DbName=$HostName

# Finally – do the work:
ReConfig

LogMsg "callReConfig.sh: Completed."
```

14.4 Application Tier Scripts

The automation scripts for the application tier start and stop EBS application services and manage replication of the application file systems for different scenarios – normal operations at the primary, switching the standby to primary, and using the standby for snapshot standby testing.

The scripts should be placed in a common location at each site, one that each application tier compute instance at that site can access. As on the database servers, this directory is referenced as `$SCRIPT_DIR`. We recommend (and assume) adding the script directory to the administrator account's PATH.

As with the database scripts, these scripts are written in Korn shell (`ksh`), use common routines / standard functions, and spawn SQL*Plus in a coroutine if needed, so that only one database connection is started.

14.4.1 Environment Files

There are three environment files holding settings for managing startup and shutdown of services on the middle tier:

- One holding configuration details for the middle tier scripts, so that the scripts themselves can be kept generic across all nodes.
- Two for setting web entry points for snapshot standby testing or, if needed, when switching the standby environment to production.

ebsAppTier.env:

```
# ebsAppTier.env
# This environment file is included by scripts on the app tier nodes
#
# - SCRIPT_DIR: The directory where these scripts reside.
# - PDB_TNS_CONNECT_STRING: The TNS connect string for the PDB
# - DB_SECRET_NAME: The name of the secret in the OCI vault holding the
#              database EBS_SYSTEM password/credential.
# - APPS_SECRET_NAME: The name of the secret in the OCI vault holding the APPS
#              schema password/credential.
# - APP_OWNER: The unix user that owns the EBS install, typically "applmgr".
# - WLS_SECRET_NAME: The name of the secret in the OCI vault holding the
#              WebLogic schema password/credential.
# - COMPARTMENT_OCID: The OCID of the OCI compartment that contains the OCI
#              vault.
# - LOG_DIR: These scripts will write their execution logs in this directory
#
# The EBS EBSApps.env is sourced to the RUN file system.

#


SCRIPT_DIR=/u02/app/ebs/custom_admin_scripts/VISPRD
PDB_TNS_CONNECT_STRING=visprd

# system secret name
DB_SECRET_NAME=ebs-db-secret

# apps owner secret name
```

```
APPS_SECRET_NAME=ebs-apps-secret
# used for ps checks of unix processes:
APP_OWNER="applmgr"

# WLS owner secret name
WLS_SECRET_NAME="ebs-wls-secret"

# Modify this for your environment
COMPARTMENT_OCID="<OCID of compartment that contains the OCI vault>"

LOG_DIR=${SCRIPT_DIR}/log

# Source the EBS app tier RUN file system.
if [ ! -f ${EBS_APP_ENV_DIR}/EBSapps.env ]; then
   echo "Cannot find the EBS environment file: ${EBS_APP_ENV_DIR}/EBSaps.env."
   exit 1
fi

. ${EBS_APP_ENV_DIR}/EBSapps.env run
```

There are also potentially two environment files holding the configuration settings that need to be adjusted when configuring the standby environment for snapshot standby testing or reconfiguring it to be ready for production switchover or failover.

These examples are built assuming the users will connect to <webentryhost>.<webentrydomain>.

The environment file for using the standby database to test, via snapshot standby:

```
# web_entry_test.env
# Snapshot standby test parameters for web entry points.
# Used by startEBS.sh
#
# Date        What
# 09/05/2024  Created
#
# This example is built assuming ebsapps-test.example.com is the hostname to be
# configured in the OCI load balancer for snapshot standby testing.
webentryhost=ebsapps-test
webentrydomain="example.com"

login_page="https://ebsapps-test.example-test.com/OA_HTML/AppsLogin"
external_URL="https://ebsapps-test.example-test.com"
webport=8000
active_webport=443
```

The environment file to use when opening the standby database as production.

```
# web_entry_prod.env
# Production parameters for web entry points.  Need if they differ between sites.
# Used by startEBS.sh
#
# Date        What
# 09/05/2024  Created
#
# This example is built assuming ebsapps.exmple.com is the hostname to be
# configured for normal standby mode.
webentryhost=ebsapps
webentrydomain=example.com
login_page="https://ebsapps.example.com/OA_HTML/AppsLogin"
external_URL="https://ebsapps.example.com"
webport=8000
active_webport=443
```

14.4.2 startEBS.sh

This script combines the tasks needed to start EBS for three scenarios – normal operations, at the standby site for testing purposes, and as the final part of a switchover from the primary to the standby site for production operations.

```
#!/bin/ksh
################################################################################
# Name: startEBS.sh
#
# Purpose: Start EBS application tier in various ways, depending on need.
#          The basic tasks:
#          a. Configure the EBS CONTEXT_FILE *
#          b. Run autoconfig
#          c. Start EBS application services
#          d. Make sure replication is enabled
#          e. Make sure replication is complete
#          f. Block until database configuration is complete
#
# * Note: Customize this for:
#          - your configuration of the EBS CONTEXT_FILE in ConfigContext routine
#          - your background rsync scripts in the EnableReplication routine
#
#          The calling choices:
#          n: Normal: does c and d above
#          t: Test (standby): does a, b, and c above
#          s: Switchover: does a, e, b, c, and d above
#
# Usage: startEBS.sh [arguments]
#        n) Normal:
#           - Start EBS application services
#           - Make sure replication is enabled
#        t) Test (standby):
#           - Make sure the DB is in SNAPSHOT STANDBY mode
#           - Configure CONTEXT_FILE for snapshot standby
#           - Run autoconfig
#           - Start EBS application services
#        s) Switch this environment to production;
#           - Block until any required database configuration is complete
#           - Make sure there are no rsync processes running
#           - Configure CONTEXT_FILE for production
#           - Run autoconfig
#           - Start EBS application services
#           - Make sure file system replication to new standby is enabled
#
# ASSUMPTIONS: User has privileges to run EBS admin scripts.
#              User's environment is already set for EBS
#             ($NE_BASE/../EBSapps.env run has been executed)
#
# Errors: A non-zero value is returned for bad arguments, inability to
#         connect, ...
#
# Revisions:
# Date        What
# 09/10/2024  Consolidated earlier scripts into one
##########################################################################
# ParseArgs:
# Parse the command line arguments
#
#  Input: command line arguments
# Output: run proffile set, determining which routines are executed
# Return: exit 1 if arguments are no good
##########################################################################
ParseArgs()
{
#
# Make sure at least 1 argument is present
if [ $# -lt 1 ]
then
   echo "$0: ERROR: You have entered insufficient arguments"
   Usage
   exit 1
fi
# make sure the parameter passed was correct
if [[ $1 = "n" || $1 = "t" || $1 = "s" ]]
then
   break
else
   echo "$0: ERROR: You have entered an incorrect argument"
   Usage
   exit 1
fi

#
# They sent in an n, t, or s, which will drive behavior
```

```
#
runPlan=$1

}

################################################################################
# Usage:
# Standard usage clause
################################################################################
Usage()
{
echo "Usage: startEBS.sh [mode]"
echo "Mode can be:"
echo "   n = Normal startup.  Start EBS and make sure replication is enabled."
echo "   t = Test: Start the environment as snapshot"
echo "   s = Switch: Do EBS-side switchover to make this environment production."
echo ""
}

################################################################################
# SetEnv
# Get environment variables, standard include routines
#
# Input:  None
# Output: Environments set for the run
# Return: Exit 1 if can't find environment files, etc.
################################################################################
SetEnv()
{
# Include the basic "where am I" environment settings
if [ ! -f ${SCRIPT_DIR}/ebsAppTier.env ]; then
   echo "Cannot find the file ${SCRIPT_DIR}/ebsAppTier.env."
   exit 1
fi

. ${SCRIPT_DIR}/ebsAppTier.env


# Include the standard functions routines
if [ ! -f ${SCRIPT_DIR}/stdfuncs.sh ]; then
   echo "Cannot find the standard functions script (stdfuncs.sh)"
   exit 1
fi

. $SCRIPT_DIR/stdfuncs.sh

EnvSetup
LOG_OUT=${LOG_DIR}/${HostName}_startEBS_${TS}.log
```

```
# Make sure the the apps env. is already set
if [ ! -f $NE_BASE/../EBSapps.env ]; then
   LogMsg "EBS environment is not set.  Cannot find the file EBSapps.env."
   LogMsg "NE_BASE: $NE_BASE"
   exit 1
fi
}

################################################################################
# GetCreds
# Get EBS and FMW credentisls needed in this run
#
# Input:  None
# Output: The creds we need, set
# Return: 1 if can't find one of the users referenced (GetLogon breaks out)
################################################################################
GetCreds()
{
LogMsg "Getting EBS credentials"
GetLogon $APPS_SECRET_NAME
APPS_SECRET=$LOGON

GetLogon $WLS_SECRET_NAME
WLS_SECRET=$LOGON
}

################################################################################
# ConnectDB
# Launch the coroutine.  Need for t and s only
#
# Input:  None
# Output: SQL*Plus child process initiated
# Return: 1 if can't start sqlplus in the background
################################################################################
ConnectDB()
{
LogMsg "ConnectDB: LaunchCoroutine"

# Start sqlplus in coroutine, logged in as apps user
LaunchCoroutine APPS $APPS_SECRET $PDB_TNS_CONNECT_STRING

LogMsg "SQLPlus coroutine started"
}

################################################################################
# WaitDBConfig
# Part of switchover: wait until all database homes have been configured.
#
# The switchover process on the database side puts a row into a custom table for
# each DB node.  Then as each node fixes its config files, it removes its entry.
# If the work is not needed, the table is not populated.  If it is needed,
# as the work is done on each EBS instance its row is removed, until the table
# is empty.  When the table is empty work can proceed on the middle tiers.
#
# Potential source of a hang, since it's theoretically possible for rows to
# be inserted and not deleted, but that would be an issue that needs to be
# resolved.
#
# Input:  None
# Output: Number of DB instances needing reconfig, sleep time, dividing line
# Return: 1 if can't start sqlplus in the background
#           if no sleep time
#           if $sql is empty
################################################################################
WaitDBConfig()
{
LogMsg "Wait for database home reconfiguration to complete"

sql="select ltrim('host: ' || host_name) from apps.xxx_EBS_role_change;"
CkRunDB 5

LogMsg "Database tier configuration complete - ok to proceed."
}

################################################################################
# StandbyCheck
# Make sure the database is in snapshot standby mode
#
# Input:  None
# Output: None
# Return: 1 if database is not in SNAPSHOT STANDBY mode
################################################################################
StandbyCheck()
{
LogMsg "Making sure the database is in snapshot standby mode"

sql="select rtrim(database_role) from v\$database;"
dbMode=`ExecSql "$sql"`

if [ ${dbMode} != "SNAPSHOT STANDBY" ]; then
   LogMsg "Database is not in SNAPSHOT STANDBY mode"
   LogMsg "Cannot do snapshot standby testing"
   exit 1
fi

LogMsg "StandbyCheck: Succeeded"
}
```

```
################################################################################
# ConfigContext
# Configure the context file for snapshot standby testing or for production
# when switching this site to prod
#
# NOTE: Add more commands if you need to change the value of more settings
#       (e.g., port numbers)
#
# Input:  Env file to use to switch configuration values in EBS's CONTEXT_FILE
# Output: CONTEXT_FILE reconfigured as requested
# Return: 1 if env file can't be found
################################################################################
ConfigContext()
{
envFile=$1

LogMsg "envFile: ${SCRIPT_DIR}/${envFile}"

if [ ! -f ${SCRIPT_DIR}/${envFile} ]; then
   LogMsg "ConfigContext: No env file specified or file not found"
   exit 1
else
   LogMsg "ConfigContext: Setting environment"
fi

. $SCRIPT_DIR/$envFile

# Back up the context file
LogMsg "Backing up CONTEXT_FILE ${CONTEXT_FILE}"
cp $CONTEXT_FILE $CONTEXT_FILE.'date +"%Y%m%d"'

# These commands are each one line (this wraps in your display)
# hostname used in the SSL CA certificate.
LogMsg "Set s_webentryhost ${webentryhost}"
$RUN_BASE/EBSapps/comn/util/jdk32/jre/bin/java -classpath
$RUN_BASE/EBSapps/comn/java/classes:$RUN_BASE/FMW_Home/Oracle_EBS-app1/shared-libs/ebs-
appsborg/WEB-INF/lib/ebsAppsborgManifest.jar oracle.apps.ad.context.UpdateContext $CONTEXT_FILE
s_webentryhost ${webentryhost}
if [ $? -ne 0 ]; then
   LogMsg "UpdateContext returned an error for s_webentryhost"
   exit 1
fi

LogMsg "Set s_webentrydomain ${webentrydomain}"

$RUN_BASE/EBSapps/comn/util/jdk32/jre/bin/java -classpath
$RUN_BASE/EBSapps/comn/java/classes:$RUN_BASE/FMW_Home/Oracle_EBS-app1/shared-libs/ebs-
appsborg/WEB-INF/lib/ebsAppsborgManifest.jar oracle.apps.ad.context.UpdateContext $CONTEXT_FILE
s_webentrydomain ${webentrydomain}
```

```
if [ $? -ne 0 ]; then
   LogMsg "UpdateContext returned an error for s_webentrydomain"
   exit 1
fi


# s_webport is the http port for the application server on the application tier.
LogMsg "Set s_webport ${webport}"
$RUN_BASE/EBSapps/comn/util/jdk32/jre/bin/java -classpath
$RUN_BASE/EBSapps/comn/java/classes:$RUN_BASE/FMW_Home/Oracle_EBS-app1/shared-libs/ebs-
appsborg/WEB-INF/lib/ebsAppsborgManifest.jar oracle.apps.ad.context.UpdateContext $CONTEXT_FILE
s_webport ${webport}
if [ $? -ne 0 ]; then
   LogMsg "UpdateContext returned an error for s_webport"
   exit 1
fi


# s_active_webport is used for the front-end load balancer.

LogMsg "Set s_active_webport ${active_webport}"

$RUN_BASE/EBSapps/comn/util/jdk32/jre/bin/java -classpath
$RUN_BASE/EBSapps/comn/java/classes:$RUN_BASE/FMW_Home/Oracle_EBS-app1/shared-libs/ebs-
appsborg/WEB-INF/lib/ebsAppsborgManifest.jar oracle.apps.ad.context.UpdateContext $CONTEXT_FILE
s_active_webport ${active_webport}

if [ $? -ne 0 ]; then
   LogMsg "UpdateContext returned an error for s_active_webport"
   exit 1
fi


# EBS login page.
LogMsg "Set s_login_page ${login_page}"
$RUN_BASE/EBSapps/comn/util/jdk32/jre/bin/java -classpath
$RUN_BASE/EBSapps/comn/java/classes:$RUN_BASE/FMW_Home/Oracle_EBS-app1/shared-libs/ebs-
appsborg/WEB-INF/lib/ebsAppsborgManifest.jar oracle.apps.ad.context.UpdateContext $CONTEXT_FILE
s_login_page ${login_page}
if [ $? -ne 0 ]; then
   LogMsg "UpdateContext returned an error for s_login_page"
   exit 1
fi

# The base URL that EBS will use for redirects back through the load balancer.
LogMsg "Set s_external_URL ${external_URL}"
$RUN_BASE/EBSapps/comn/util/jdk32/jre/bin/java -classpath
$RUN_BASE/EBSapps/comn/java/classes:$RUN_BASE/FMW_Home/Oracle_EBS-app1/shared-libs/ebs-
appsborg/WEB-INF/lib/ebsAppsborgManifest.jar oracle.apps.ad.context.UpdateContext $CONTEXT_FILE
s_external_URL ${external_URL}
if [ $? -ne 0 ]; then
   LogMsg "UpdateContext returned an error for s_external_URL"
   exit 1
fi

LogMsg "CONTEXT_FILE reconfigured"
}

################################################################################
# RunAutoConfig
# Run autoconfig on all application servers
#
# Input:  None
# Output: EBS configuration updated
# Return: 1 if can't start sqlplus in the background
################################################################################
RunAutoConfig()
{
LogMsg "Running autoconfig on all application servers"

{ echo $APPS_SECRET; }| perl $AD_TOP/bin/adconfig.pl -contextfile=$CONTEXT_FILE -parallel
promptmsg=hide | tee -a ${LOG_OUT}

# Check for success
if [ $? -ne 0 ]; then
  LogMsg "$AD_TOP/bin/adconfig.pl reported failure."
  exit 1
fi

LogMsg "Completed: RunAutoConfig."
}

################################################################################
# StartServices
# Run autoconfig on all application servers
#
# Input:  None
# Output: EBS application and middle tiers started on all app servers
# Return: 1 if can't start sqlplus in the background
################################################################################
StartServices()
{
LogMsg "Starting EBS on all application servers"

{ echo "APPS"; echo $APPS_SECRET; echo $WLS_SECRET; } | $ADMIN_SCRIPTS_HOME/adstrtal.sh -msimode
| tee -a ${LOG_OUT}

# Did adstrtal report failure?
if [ $? -ne 0 ]; then
  LogMsg "$ADMIN_SCRIPTS_NAME/adstrtal.sh reported failure."
  exit 1
```

```
fi

LogMsg "Completed: StartServices."
}

################################################################################
# CkRsync
# DB shutdown on switchover is supposed to finish and stop rsync background
# processes.  This checks to see if rsync is still running.  At this point in
# the flow, it should not be running, so exit with error if it is.
#
# Input:  None
# Output: None
# Return: 1 if rsync processes are running somewhere.
################################################################################
CkRsync()
{
LogMsg "Make sure replication is not running anywhere."

if [ -f "${SCRIPT_DIR}/ebsrsync.lck" ]; then
   LogMsg "CkRsync: There's an rsync process running on `cat $SCRIPT_DIR/ebsrsync.lck`"
   LogMsg "CkRsync: Wait until database side is completely ready before switching"
   LogMsg "         to this site."
   exit 1
fi

LogMsg "Completed: CkRsync."
}

################################################################################
# EnableReplication
# Make sure replication is enabled with this site as primary
#
# Input:  None
# Output: None
# Return: 1 if can't start sqlplus in the background
################################################################################
EnableReplication()
{
LogMsg "Make sure replication is enabled with this site as source"

# Customize this for your replication scripts
${SCRIPT_DIR}/enable_ebs_rsync.sh ${SCRIPT_DIR}/slowFiles.env
${SCRIPT_DIR}/enable_ebs_rsync.sh ${SCRIPT_DIR}/fastFiles.env

LogMsg "Completed: EnableReplication."
}

################################################################################
```

```
# Execution starts here.
################################################################################

ParseArgs $*
# Leave ParseArgs with runPlan set to n, t, or s

SetEnv

LogMsg "runPlan: $runPlan"

LogMsg "startEBS.sh: Started"

GetCreds

case $runPlan in
   n) LogMsg "Normal startup"
      StartServices
      EnableReplication
      ;;
   t) LogMsg "Start for snapshot testing"
      ConnectDB
      StandbyCheck
      # Note: hard-coded env file reference
      ConfigContext web_entry_test.env
      RunAutoConfig
      StartServices
      ;;
   s) LogMsg "Switch to this environment"
      ConnectDB
      WaitDBConfig
      CkRsync
      # Note: hard-coded env file reference
      ConfigContext web_entry_prod.env
      RunAutoConfig
      StartServices
      EnableReplication
      ;;
   *) Usage
      exit 1
      ;;
esac

LogMsg "Completed: startEBS.sh"
```

14.4.3 stopEBS.sh

Stop EBS and middle tier services, either as a normal shutdown or as part of a switchover to a standby site.

```
#!/bin/ksh
```

```
################################################################################
# Name: stopEBS.sh
#
# Purpose: Stop EBS application tier in various ways, depending on need.
#          The basic tasks:
#          a. Stop EBS application services
#          b. Kill remainng services if any running after x time
#          c. Grab a lock for managing file system synchronization
#          d. Block until all EBS connections to DB are gone
#          e. Make sure replication is complete, do one last pass of shared
#             file system
#
#          The calling choices:
#          n: Normal: does a and b above
#          s: Switchover: does c, a, b, d, and e above
#
# Usage: stopEBS.sh [arguments]
#        n) Normal:
#              - Stop EBS application services
#              - Kill remaining services if any running after x time
#        s) Switch this environment to standby:
#              - Grab a lock for managing file system synchronization
#              - Stop EBS application services
#              - Kill remaining services if any running after x time
#              - Block until all EBS connections to DB are gone
#              - Make sure replication is complete, do one last pass of shared
#                file system
#
# ASSUMPTIONS: User has privileges to run EBS admin scripts.
#              User's environment is already set for EBS
#             ($NE_BASE/../EBSapps.env run has been executed)
#
# Errors: A non-zero value is returned for bad arguments, inability to
#         connect, ...
#
# Revisions:
# Date        What
# 09/10/2024  Consolidated earlier scripts into one
################################################################################
# ParseArgs:
# Parse the command line arguments
#
#  Input: command line arguments
# Output: run profile set, determining which routines are executed
# Return: exit 1 if arguments are no good
################################################################################
ParseArgs()
{
#
```

```
# Make sure at least 1 argument is present
if [ $# -lt 1 ]
then
   echo "$0: ERROR: You have entered insufficient arguments"
   Usage
   exit 1
fi

# make sure the parameter passed was correct
# set `getopt n:s: $*`
if [[ $1 = "n" || $1 = "s" ]]
then
   break
else
   echo "$0: ERROR: You have entered an incorrect argument"
   Usage
   exit 1
fi

#
# They sent in an n or s, which will drive behavior
#
runPlan=$1

}

################################################################################
# Usage:
# Standard usage clause
################################################################################
Usage()
{
echo "Usage: stopEBS.sh [mode]"
echo "Mode can be:"
echo "   n = Normal shutdown.  Stop EBS and make sure all DB cx are gone."
echo "   s = Switchover shutdown - \"Normal\" plus finalize replication before returning
control."
echo ""
}

################################################################################
# SetEnv
# Get environment variables, standard include routines
#
# Input:  None
# Output: Environments set for the run
# Return: Exit 1 if can't find environment files, etc.
################################################################################
SetEnv()
```

```
{
# Include the basic "where am I" environment settings
if [ ! -f ${SCRIPT_DIR}/ebsAppTier.env ]; then
   echo "Cannot find the file ${SCRIPT_DIR}/ebsAppTier.env."
   exit 1
fi

. ${SCRIPT_DIR}/ebsAppTier.env

# Include the standard functions routines
if [ ! -f ${SCRIPT_DIR}/stdfuncs.sh ]; then
   echo "Cannot find the standard functions script (stdfuncs.sh)"
   exit 1
fi

. ${SCRIPT_DIR}/stdfuncs.sh

EnvSetup
LOG_OUT=${LOG_DIR}/${HostName}_stopEBS_${TS}.log

# Make sure the the apps env. is already set
if [ ! -f $NE_BASE/../EBSapps.env ]; then
   LogMsg "EBS environment is not set.  Cannot find the file EBSapps.env."
   LogMsg "NE_BASE: $NE_BASE"
   exit 1
fi

}

################################################################################
# GetCreds
# Get EBS and FMW credentials needed in this run
#
# Input:  None
# Output: The creds we need, set
# Return: 1 if can't find one of the users referenced
################################################################################
GetCreds()
{
LogMsg "Getting EBS credentials"
GetLogon $APPS_SECRET_NAME
APPS_SECRET=$LOGON

GetLogon $WLS_SECRET_NAME
WLS_SECRET=$LOGON
}

################################################################################
# GetLock
# First app server to initiate the stop-to-switch-over process is responsible
# for making sure replication is complete / all file system changes are
# captured.
#
# That app server grabs a lock to make sure no one else does
# replication work.  Then it disables the rsync process so none will start
# in the background as we are shutting down.
#
# Input: None
# Output: If first in - create the lock file, set SKIP_RSYNC to 1,
#            and disable automatic rsyncs
#         If not, show which server has the lock and set SKIP_RSYNC to 0
# Return: None
################################################################################
GetLock()
{
LogMsg "GetLock: Looking for $SCRIPT_DIR/ebsrsync.lck"

if [ -f "${SCRIPT_DIR}/ebsrsync.lck" ]; then
   SKIP_RSYNC=1
   LogMsg "GetLock: Someone else is managing rsync shutdown"
else
   echo ${HOSTNAME} >> ${SCRIPT_DIR}/ebsrsync.lck
   SKIP_RSYNC=0
   LogMsg "GetLock: This process will manage rsync shutdown"

   thing=`cat ${SCRIPT_DIR}/ebsrsync.lck`
   LogMsg "GetLock: Contents of lock file: ${thing}"

   LogMsg "GetLock: Disable rsync so we won't start a fresh session."
   ${SCRIPT_DIR}/disable_ebs_rsync.sh ${SCRIPT_DIR}/slowFiles.env
   ${SCRIPT_DIR}/disable_ebs_rsync.sh ${SCRIPT_DIR}/fastFiles.env
   # Trigger the killSync process to terminate any running rsyncs.
   # We have to do this via shared file system because stopEBS.sh will be run
   # on a server that hosts EBS, but the rsync processes should be on dedicated
   # servers.
   LogMsg "GetLock: Trigger killSync to stop any running rsync processes."
   # If an in-progress rsync processes are killed, rsync is robust enough to not
   # replace a file until it is completely copied to its target.
   # List the file system env files below and before the EOF.
   cat <<EOF >> ${SCRIPT_DIR}/.abortSync
   ${SCRIPT_DIR}/fastFiles.env
   ${SCRIPT_DIR}/slowFiles.env
EOF
fi
}

################################################################################
# ConnectDB
# Launch the coroutine.  Need for t and s only
#

# Input:  None
# Output: SQL*Plus child process initiated
# Return: 1 if can't start sqlplus in the background
################################################################################
ConnectDB()

{
LogMsg "ConnectDB: LaunchCoroutine"

# Start sqlplus in coroutine, logged in as apps user
LaunchCoroutine APPS $APPS_SECRET $PDB_TNS_CONNECT_STRING

LogMsg "SQLPlus coroutine started"
}

################################################################################
# BlockUntilGone
# Part of shutting down for switchover: wait until all EBS client sessions
# have ended.  Look for service names that are configured for any EBS user
# session.  In our system: VISPRD_OACORE_ONLINE, VISPRD_PCP_BATCH, and
# VISPRD_FORMS_ONLINE
#
# Input:  None
# Output: Number of DB instances needing reconfig, sleep time, dividing line
# Return: 1 if can't start sqlplus in the background
#           if no sleep time
#           if $sql is empty
################################################################################
BlockUntilGone()
{
LogMsg "stopEBS.sh: Checking number of remaining database sessions before performing rsync."
sql="select ltrim(count(*)) \
   from gv\$instance a, gv\$session b \
   where a.inst_id = b.inst_id \
   and service_name in ('VISPRD_OACORE_ONLINE','VISPRD_PCP_BATCH','VISPRD_FORMS_ONLINE');"
CkRunDB 5

LogMsg "All EBS user sessions ended - ok to proceed."
}

################################################################################
# StopServices
# Request a clean exit for all EBS application services on the middle tiers.
#
# Input:  None
# Output: Request for stopping EBS application \ middle tiers on all app servers
# Return: 1 if can't start sqlplus in the background
################################################################################
StopServices()
{
LogMsg "StopServices: Stopping EBS on all application servers"

{ echo "APPS"; echo "$APPS_SECRET"; echo "$WLS_SECRET"; } | $ADMIN_SCRIPTS_HOME/adstpall.sh |
tee -a ${LOG_OUT}

# Did adstpall report failure?
if [ $? -ne 0 ]; then
  LogMsg "StopServices: $ADMIN_SCRIPTS_NAME/adstpall.sh reported failure."
  exit 1
fi

LogMsg "Completed: StopServices."
}

################################################################################
# KillSessions
# The prior routine asked EBS to properly terminate all EBS sessions.  This
# routine pauses progress if anything is still running.  It checks a couple of
# times, after a bit loses patience and just kills what’s left.
#
# NOTE: PARAMETERIZE THIS or adjust things like number of seconds
#
# Input:  None
# Output: EBS application and middle tiers fully down on all app servers
# Return: None
################################################################################
KillSessions()
{
LogMsg "KillSessions: Wait for sessions to complete.  Kill after x number of tries."
LogMsg "Return control as soon as possible so we can proceed with next steps."

PROCESS_COUNT=$(ps -elf | grep "${APP_OWNER}" | grep -E "${GREP_STRING}" | grep -v grep | wc -l
)
LogMsg "KillSessions: Number of remaining processes: ${PROCESS_COUNT}"
i=1
while [ ${PROCESS_COUNT} -ne 0 ];
do
  # Sleep for 10 seconds to let some processes terminate.
  sleep 10
  PROCESS_COUNT=$(ps -elf | grep "${APP_OWNER}" | grep -E "${GREP_STRING}" | grep -v grep | wc -
l )
  # PID_LIST=$(ps -elf | grep "${APP_OWNER}" | grep -E "${GREP_STRING}" | grep -v grep | awk '{
print $4 }' )
  Running=$(ps -elf | grep "${APP_OWNER}" | grep -E "${GREP_STRING}" | grep -v grep )
  LogMsg "${PROCESS_COUNT} remaining processes: ${Running}"
  if [[ $i -gt 3 && ${PROCESS_COUNT} -ne 0 ]]; then
    # we have only so much patience
    PID_LIST=$(ps -elf | grep "${APP_OWNER}" | grep -E "${GREP_STRING}" | grep -v grep | awk '{
print $4 }' )
    LogMsg "stopEBS.sh: Killing processes: ${PID_LIST}"
    kill -9 ${PID_LIST}
    # we're assuming they're dead now.  This breaks us out of the while loop.
    PROCESS_COUNT=0
  fi
  ((i=i+1))
done
LogMsg "stopEBS.sh: All EBS services down on this server."

LogMsg "Completed: KillSessions."
}


################################################################################
# CompleteReplication
# DB sessions all finished.  Now tackle final rsync.
#
# Rsync was disabled earlier, but there still might be an rsync process
# running on the dedicated servers.  Check; if so, wait for it to complete.
#
# Finally, do a cleanup rsync of fastFiles directory structure.
#
# We need to source the appropriate file to get the SOURCE_RSYNC_DIR env
# variable set.  We are assuming it's appropriate to check only for fastFiles
# replication...
#
# Input:  None
# Output: Updates on checking for rsync script locks
# Return: None
################################################################################
CompleteReplication()
{
LogMsg "CompleteReplication: Checking for running rsync processes via their lock file."

# do we want to wait forever?  these processes can run a very long time, or
# our lock system could be broken
# Note: we are now killing the rsync processes in a separate process
# (killSync.sh), so the final sweep can be started sooner.
while true; do
   if [ -f ${SCRIPT_DIR}/.slowFiles.lck | -f ${SCRIPT_DIR}/.fastFiles.lck ]; then
      LogMsg "One of the lock files is present."
      LogMsg "Sleeping..."
      LogMsg
"=========================================================================================="
      sleep 5
   else
      # we know any already-running rsync scripts are complete.  Do one final sweep.
      # Here, we just sync output files. You may choose to also sync program
      # directories.
      LogMsg "All locks are clear.  Do one final sweep of fastFiles"
      ${SCRIPT_DIR}/syncEBS.sh ${SCRIPT_DIR}/fastFiles.env s
      LogMsg "CompleteReplication: Succeeded"
      break
   fi
done

}

################################################################################
# Execution starts here.
################################################################################

ParseArgs $*
# Leave ParseArgs with runPlan set to n or s

SetEnv

LogMsg "runPlan: $runPlan"

LogMsg "stopEBS.sh: Started"

GetCreds

# Check this for remaining EBS sessions
GREP_STRING="visprd|FNDLIB|FNDIMON|PALIBR"

# Remember: SKIP_RSYNC is set to 0 if you do NOT want to skip rsync,
# but instead WANT to do rsync, and to 1 if you DO want to skip rsync.
# Only the first server in has that flag set to 0 THUS DOES the rsync
case $runPlan in
   n) LogMsg "Normal shutdown"
      StopServices
      KillSessions
      ;;
   s) LogMsg "Switch away from this environment"
      GetLock
      StopServices
      KillSessions
      ConnectDB

      BlockUntilGone
      [ ${SKIP_RSYNC} == 0 ] && CompleteReplication
      [ ${SKIP_RSYNC} == 0 ] && rm ${SCRIPT_DIR}/ebsrsync.lck
      ;;
   *) Usage
      exit 1
      ;;
esac

LogMsg "Completed: stopEBS.sh"
```

14.5 File System Replication (rsync) Scripts

Data Guard manages replication of data stored in the database from the primary site to the standby. Certain file systems also need to be replicated – the application and middle tier software, and the file-based output generated by the application programs. The software directories do not change frequently but do need to be kept in sync across the sites when the source is patched. The file-based output holds batch processing logs, report output, and interface files that have been generated for other systems outside the EBS database. These files need to be as close to in sync with the data inside the database as feasible, to ease the task of resolving differences between the state of data in the database and these flat files on failover.

The rsync script in this section replicates specific file system directories from the primary to the standby. The core synchronization script addresses these basic requirements:

- Regularly synchronize the file systems holding the relatively unchanging application and middle tier software to the standby
- Aggressively synchronize the file systems holding runtime output, keeping them as up to date as possible at the standby
- Avoid replicating instance-specific configuration data
- Allow file system replication to continue while using the standby for snapshot standby testing
- Multi-thread the data transfers so that they complete in a reasonable amount of time

The rsync direction is managed through site role transitions. The scripts are deployed at each region/site and are designed to always be running at each site as a cron job. The script's behavior depends on whether the rsync process is enabled or disabled and whether the site's database is in the PRIMARY or STANDBY role. These scripts respond to database role transitions and automatically change the direction of the replication.

We recommend the actual rsync script be run on a dedicated pair of compute instances, one at each site, so that snapshot standby testing at the DR site can take place without disrupting replication of production changes to the standby file system. We provisioned our replication servers in section 3.3.

Note: These scripts are built with the assumption that the EBS application tiers are deployed on shared file systems in OCI. We used OCI's File System Service (FSS).

In the next sections, we describe the environment files used to control each run, the scripts to enable and disable the rsync process – which are called by startEBS.sh and stopEBS.sh, and the core rsync script – which is normally executed via cron. The last section shows setting the core rsync script up in cron for automatic execution.

14.5.1 rsync Environment Files

In our implementation, the replication script `syncEBS.sh` first accesses the file `ebsAppTier.env,` used in section 14.4 for basic middle tier environment setup. Then there is one environment file (`ebsRsync.env`) for the overall synchronization process, and finally one for each set of directories that will be replicated at a specific frequency. This last environment file is passed to the script as a parameter.

The `ebsRsync.env` file:

```
# Environment settings for rsync scripts.  They need database access and
# the ability to connect to the remote app tiers for file system replication

# OS user for app install:
USER=applmgr

# use to get the site role, as sysdba:
CDB_CONNECT_STRING=<TNS connect string alias to the CDB of the EBS database>
dbSecretName="<Name of the secret within the OCI vault for the database credential (EBS_SYSTEM
or SYS)>"

targetHostIP=<IP address of the remote server>
```

You will have environment files that hold pointers to text files describing exactly which directories will be managed in a given execution of the syncEBS.sh script. We have two – one for files to synchronize infrequently (e.g., software directories) and one for files that change more rapidly (e.g., log and out directories) and need to be synchronized more frequently:

The `slowFiles.env` file:

```
# These files don't change often unless there's an EBS patch event
# The rsync process can be run less frequently.
fsAlias=slowFiles

copyDirectories=${SCRIPT_DIR}/staticDirectories.txt
excludeFiles=${SCRIPT_DIR}/staticExcludeFiles.txt
copyFiles=${SCRIPT_DIR}/staticIncludeFiles.txt
```

The `fastFiles.env` file:

```
# These files change frequently - they include EBS log and out directories
# The rsyncn process should be scheduled to run very often.
fsAlias=fastFiles

copyDirectories=${SCRIPT_DIR}/dynamicDirectories.txt
excludeFiles=${SCRIPT_DIR}/dynamicExcludeFiles.txt
copyFiles=${SCRIPT_DIR}/dynamicIncludeFiles.txt
```

The `fsAlias` variable in each of the above env files is used to organize and manage the output generated by the scripts. The referenced text files – `copyDirectories`, `excludeFiles`, and `copyFiles`.

- The `copyDirectories` variable points to a file that holds one line for each directory to be copied. Each line has two entries – the source directory and the target directory. This list of directory pairs can be arbitrarily long – the design goal was to break the work up into many pieces so they can run concurrently thus finish more quickly. In our case, we were able to bring the no-op synchronization process for the EBS software directories from over a half hour to under five minutes by splitting the work into 377 pieces.

- The `excludeFiles` variable points to a file that holds one line for each file or subdirectory that needs to be skipped when copying the above directories. This file can be empty.
- The `copyFiles` variable points to a file holding individual files that must be kept in sync but that would be skipped given how your `copyDirectories` entries are formed. This file can be empty.

14.5.1.1 Sample Static Files

Our `slowFiles.env` file points to `staticDirectories.txt`, `staticExcludeFiles.txt`, and `staticIncludFiles.txt`. Our `staticDirectories.txt` file has 377 entries. We moved the from / to pairs that take the longest time to scan up to the top of the list so they will be working while the smaller copies are completed. Here are the first 15 entries of this file in our environment:

```
# Relatively static directories to sync to other side
# Put the fattest directories towards the top of the list
/u02/app/ebs/visprd/fs1/EBSapps/comn/java /u02/app/ebs/visCopy/fs1/EBSapps/comn/java/classes
/u02/app/ebs/visprd/fs1/EBSapps/comn/java /u02/app/ebs/visCopy/fs1/EBSapps/comn/java/lib
/u02/app/ebs/visprd/fs2/EBSapps/comn/java /u02/app/ebs/visCopy/fs2/EBSapps/comn/java/classes
/u02/app/ebs/visprd/fs2/EBSapps/comn/java /u02/app/ebs/visCopy/fs2/EBSapps/comn/java/lib
/u02/app/ebs/visprd/fs2/FMW_Home /u02/app/ebs/visCopy/fs2/FMW_Home
/u02/app/ebs/visprd/fs2/inst /u02/app/ebs/visCopy/fs2/inst
/u02/app/ebs/visprd/fs1/FMW_Home /u02/app/ebs/visCopy/fs1/FMW_Home
/u02/app/ebs/visprd/fs1/inst /u02/app/ebs/visCopy/fs1/inst
/u02/app/ebs/visprd/ETCC /u02/app/ebs/visCopy/ETCC
/u02/app/ebs/visprd/oraInventory /u02/app/ebs/visCopy/oraInventory
/u02/app/ebs/visprd/pairsfile /u02/app/ebs/visCopy/pairsfile
/u02/app/ebs/visprd/fs1/EBSapps/10.1.2 /u02/app/ebs/visCopy/fs1/EBSapps/10.1.2
/u02/app/ebs/visprd/fs1/EBSapps/comn/admin /u02/app/ebs/visCopy/fs1/EBSapps/comn/admin
```

Our `staticExcludeFiles.txt` file has these entries that must be skipped when synchronizing the directories in `staticDirectories.txt`:

```
# Files and subdirectories to exclude from directories being copied
/u02/app/ebs/visprd/fs1/inst/apps/VISPRD_ebsapp1/ora/10.1.2/network/admin
/u02/app/ebs/visprd/fs2/inst/apps/VISPRD_ebsapp2/ora/10.1.2/network/admin
/u02/app/ebs/visprd/EBSapps.env
/u02/app/ebs/visprd/fs1/EBSapps/appl/APPSVIS_ebsapp1.env
/u02/app/ebs/visprd/fs1/EBSapps/appl/APPSVIS_ebsapp2.env
/u02/app/ebs/visprd/fs1/EBSapps/appl/VIS_ebsapp1.env
/u02/app/ebs/visprd/fs1/EBSapps/appl/VIS_ebsapp2.env
/u02/app/ebs/visprd/fs2/EBSapps/appl/APPSVIS_ebsapp1.env
/u02/app/ebs/visprd/fs2/EBSapps/appl/APPSVIS_ebsapp2.env
/u02/app/ebs/visprd/fs2/EBSapps/appl/VIS_ebsapp1.env
/u02/app/ebs/visprd/fs2/EBSapps/appl/VIS_ebsapp2.env
```

The copyFiles entry points to a file that holds one line for each file that needs to be copied that would otherwise be missed, given how you break down your list of directories to be copied. This file can be empty. If you have a list of from / to files here, a single script that copies these files will be generated and executed as part of the synchronization exercise.

14.5.1.2 Sample Dynamic Files

The contents of the .txt files supporting your fastFiles.env file will reflect how you have configured your EBS log and out directories. We originally had our directory structure set for the application short names, but needed a better, more random distribution of output. To accomplish this distribution, we set the variable APPLLDM to reqidmod:100. With this setting, the Concurrent Manager processes use the last two digits of the UNIX process ID as a destination directory for the log and out files, thus splitting the work across 100 directories in a way that is well-distributed during runtime.

To change the directory structure for Concurrent Manager log and out files, shut the concurrent managers down, change the value of the APPLLDM parameter, then bring the concurrent managers back up. We used this command to change our setting to reqidmod:100:

```
$RUN_BASE/EBSapps/comn/util/jdk32/jre/bin/java -classpath
$RUN_BASE/EBSapps/comn/java/classes:$RUN_BASE/FMW_Home/Oracle_EBS-app1/shared-libs/ebs-
appsborg/WEB-INF/lib/ebsAppsborgManifest.jar oracle.apps.ad.context.UpdateContext $CONTEXT_FILE
s_applldm “reqidmod:100”
```

Note that the new Concurrent Manager output destination directories will be created as needed, over time.

Consider adjusting this script to create your list of from and to fast directories to copy if your directories are already set up:

```
for i in `ls /u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp2/logs/appl/conc/`
do
  echo "/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp1/logs/appl/conc/${i}
/u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp1/logs/appl/conc/${i}"
  echo "/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp2/logs/appl/conc/${i}
/u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp2/logs/appl/conc/${i}"
done
```

This script will create a list of from/to directories that will be generated for Concurrent Manager output when APPLLDM is set to reqidmod:100 :

```
x=0
while [ $x -lt 100 ]
do
   string=$(printf "%02d" "$x")
   echo "/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp1/logs/appl/conc/${string}
/u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp1/logs/appl/conc/${string}"
   echo "/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp2/logs/appl/conc/${string}
/u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp2/logs/appl/conc/${string}"
   let x++
done
```

Our `fastFiles.env` file points to `fastDirectories.txt`, `fastExcludeFiles.txt`, and `fastIncludFiles.txt`. Our `fastDirectories.txt` file has 580 lines due to spanning a change in configuration of our log and out directory structure. We moved the directories that do not include Concurrent Manager output to the top of the list as we expect the CM logs and output files to be the bulk of the work and to take consistent time.

Here are the first 19 entries in our fastDirectories.txt file:

```
# dynamicDirectories.txt: everything under fs_ne.  Split log and out directories across 100 sets of subdirectories
# based on APPLLDM = "reqidmod:100"
/u02/app/ebs/visprd/fs_ne/EBSapps/appl /u02/app/ebs/visCopy/fs_ne/EBSapps/appl
/u02/app/ebs/visprd/fs_ne/EBSapps/log /u02/app/ebs/visCopy/fs_ne/EBSapps/log
/u02/app/ebs/visprd/fs_ne/EBSapps/patch /u02/app/ebs/visCopy/fs_ne/EBSapps/patch
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp1/ad /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp1/ad
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp1/certs /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp1/certs
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp1/oracle_common /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp1/oracle_common
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp1/soa /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp1/soa
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp2/ad /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp2/ad
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp2/certs /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp2/certs
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp2/oracle_common /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp2/oracle_common
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp2/soa /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp2/soa
# include the default log and out location "just in case"
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp1/logs/appl/conc/log /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp1/logs/appl/conc/log
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp1/logs/appl/conc/out /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp1/logs/appl/conc/out
# log and out directories based on APPLLDM reqidmod:100
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp1/logs/appl/conc/00 /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp1/logs/appl/conc/00
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp2/logs/appl/conc/00 /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp2/logs/appl/conc/00
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp1/logs/appl/conc/01 /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp1/logs/appl/conc/01
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp2/logs/appl/conc/01 /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp2/logs/appl/conc/01
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp1/logs/appl/conc/02 /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp1/logs/appl/conc/02
/u02/app/ebs/visprd/fs_ne/inst/VISPRD_ebsapp2/logs/appl/conc/02 /u02/app/ebs/visCopy/fs_ne/inst/VISPRD_ebsapp2/logs/appl/conc/02
```

Our `fastExcludeFiles.txt` and `fastIncludFiles.txt files` are both empty, as there are no configuration files to exclude and our directory pairs did not create orphan files.

14.5.2 enable_ebs_rsync.sh

```
#!/bin/sh
############################################################################
# File name:    enable_ebs_rsync.sh
#
# Description:  Remove the *_rsync_disabled file, thus telling the rsync
#               script to be active here.
#
# Usage:        enable_ebs_rsync.sh <fully qualified name of run env file>
#               Can be called directly, but is called by the startEBS.sh
#               script.
#
# Errors:       No run environment file specified
#                Cannot find run environment file
#
# Revisions:
# Date       What
# 9/26/2024  Base updates
# 7/1/2023   Created
############################################################################

# Enable the EBS rsync job.

. ./ebsRsync.env

if [ $# -eq 0 ]
then
    echo "No run env file supplied.  Please provide a run env. file."
    echo "Usage:     enable_ebs_rsync.sh <run env file>"
    echo "Example:   enable_ebs_rsync.sh fastFiles.env"
    exit 1
fi

if  [ ! -f "$1" ]
then
    echo "File $1 does not exist."
    exit 1
fi

source $1

if [ -f "${SCRIPT_DIR}/.${fsAlias}_rsync_disabled" ]
then
    rm -f ${SCRIPT_DIR}/.${fsAlias}_rsync_disabled
    echo "EBS rsync job for ${fsAlias} enabled."
else
    echo "EBS rsync job for ${fsAlias} already enabled."
fi
```

14.5.3 disable_ebs_rsync.sh

```
#!/bin/sh
############################################################################
# File name:     disable_ebs_rsync.sh
#
# Description:  Create the *_rsync_disabled file, showing file system
#               replication is not active at this site.  This must be set
#               at the standby site.
#
# Usage:        disable_ebs_rsync.sh <fully qualified name of run env file>
#               Can be called directly, but is called by stopEBS.sh
#
```

```
# Errors:       No run environment file specified
#               Cannot find run environment file
#
# Revisions:
# Date       What
# 7/1/2023   Created
############################################################################

. ./ebsRsync.env

if [ $# -eq 0 ]
then
    echo "No run environment file supplied.  Please provide a run env file."
    echo "Usage:     disable_ebs_rsync.sh <run env file>"
    echo "Example:   disable_ebs_rsync.sh fastFiles.env"
    exit 1
fi

if  [ ! -f "$1" ]
then
    echo "File $1 does not exist."
    exit 1
fi

source $1

if [ -f "${SCRIPT_DIR}/.${fsAlias}_rsync_disabled" ]
then
    echo "EBS rsync job for ${fsAlias} already disabled."
else
    touch ${SCRIPT_DIR}/.${fsAlias}_rsync_disabled
    echo "EBS rsync job for ${fsAlias} disabled."
fi
```

14.5.4 syncEBS.sh

```
#!/bin/ksh
################################################################################
# Name: syncEBS.sh
#
# Purpose: Use rsync to synchronize the EBS file systems to the standby site
#
#          rsync needs to compare the timestamp on the source file to the
#          timestamp on the target file, to determine wthether or not to copy
#          the file.  To make that faster, here we drill down to lower
#          directory levels and multi-thread the process, making the directory
#          parse complete more quickly.
#
#          We expect the first step of this exercise to be a full rsync copy of
#          the file system to the remote site, which will take signficant time.
#          The goal here is for later runs to go more quickly, using this
#          script.
#
#          NOTE: Because we expect file synchronization actvities to run on a
#          separate set of VMs, not on the EBS middle tiers, we do not control
#          script execution directly from the stopEBS.sh or startEBS.sh scripts.
#          Instead, those scripts communicate state across servers via hidden
#          files on shared file system.
#          To that end, this script has an added role of making sure a separate
#          script killSync.sh is running in the background.  killSync.sh
#          helps during a switchover from a primary site to a secondary site,
#          making sure any in-process rsync is killed so that one last sweep
#          can be done to make sure all file system changes have been sent
#          from the primary to the secondary, for a clean switchover.
#
#          syncEBS.sh will start killSync.sh under these conditions:
#           -  Replication is enabled.
#           -  The site is in the PRIMARY role.
#           -  syncEBS.sh is not performing a forced sync (f parameter).
#           -  syncEBS.sh is not running in switchover mode (s parameter).
#
#  Usage:  syncEBS.sh <env file controlling this run> [ F|f | S|s ]
#
#  Errors: environment file does not exist, or various parameters within the
#          env file are nonsense
#
# Revisions:
# Date        What
# 6/1/2025    Start killSync.sh on the server running rsync
# 10/09/2024  Created
################################################################################
# ParseArgs:
# Parse the command line arguments
#
#  Input: command line arguments
# Output: source and target directories set
# Return: exit 1 if arguments are no good
################################################################################
ParseArgs()
{
# Make sure they passed in an env file and it's a file with content
if [ ! -f "${1}" ]; then
   echo "$0: ERROR: ${1} is not a file, does not exist, or is empty."
   Usage
   exit 1
fi

runEnv=${1}
```

```
# If there's a second argument, it could be:
# -  F or f, to force rsync even if cannot connect to the database, or
# -  S or s, for running a final sync before switchover.
x=$2
forceRsync=0
syncForSwitchover=0
go=""
# is x empty?  if not zero characters then look at the value
if [ ${#x} -ne 0 ]; then
   if [[ "${x}" == "f" || "${x}" == "F" ]]; then
      while true
      do
         # Need to read the input using the Korn shell syntax, not bash syntax.
         #read -p "Are you sure you want to force rsync from this site? [Y|y|N|n] :" go
         read -n 2 go?'Are you sure you want to force rsync from this site? [Y|y|N|n] : '
         go=$( echo ${go} | tr -d '[:cntrl:][:blank:]' )
         [[ ${go} = 'Y' || ${go} = 'y' || ${go} = 'N' || ${go} = 'n' ]] && break
         go=""
      done
      if [[ ${go} = 'N' || ${go} = 'n' ]]; then
         echo "You do not want to force rsync.  Exiting."
         Usage
         exit 1
      else
         echo "You do want to force the rsync.  Continuing."
         forceRsync=1
      fi
   elif [[ "${x}" == "s" || "${x}" == "S" ]]; then
      # This is for performing a final sync as part of switchover.
      syncForSwitchover=1
      echo "syncEBS.sh running in syncForSwitchover mode."
   else
      echo "syncEBS.sh: Your second parameter does not make sense."
      echo "Second parameter: ${x}"
      Usage
      exit 1
   fi
fi

}

################################################################################
# Usage:
# Standard usage clause
################################################################################
Usage()
{
```

```
echo "Usage: syncEBS.sh <driver file> [ F | f || S | s ] "
echo “       Required:”
echo "       <driver file> is an env file holding parameters needed for this run."
Echo “       Optional - either F or S:”
echo "       F or f will FORCE an rsync.  This should only be done if the primary"
echo "       database has CRASHED but you are able to manually synchronize the"
echo "       middle tier file system."
echo "       S or s triggers a fresh full rsync to be done.  It is called during “
echo “       a switchover to a new site."
echo "       NOTE: If you have a second parameter, it can be either F|f or S|s”
echo “       You cannot specify three parameters."
echo ""
}

################################################################################
# SetEnv
# Get environment variables, standard include routines
#
# Input:  None
# Output: Environments set for the run
# Return: Exit 1 if can't find environment files, etc.
################################################################################
SetEnv()
{

# Include the basic "where am I" environment settings
if [ ! -f ${SCRIPT_DIR}/ebsAppTier.env ]; then
   echo "Cannot find the file ${SCRIPT_DIR}/ebsAppTier.env."
   exit 1
fi

. ${SCRIPT_DIR}/ebsAppTier.env


# Include the standard functions routines
if [ ! -f ${SCRIPT_DIR}/stdfuncs.sh ]; then
   echo "Cannot find the standard functions script (stdfuncs.sh)"
   exit 1
fi

. ${SCRIPT_DIR}/stdfuncs.sh

# Include the environment settings for rsync to remote site
if [ ! -f ${SCRIPT_DIR}/ebsRsync.env ]; then
   echo "Cannot find the file ebsRsync.env."
   exit 1
fi

. ${SCRIPT_DIR}/ebsRsync.env
```

```
# Include the environment file they passed in
# this sets copyFiles and copyDirectories
if [ ! -f ${runEnv} ]; then
   echo "Cannot find the file ${runEnv}."
   Usage
   exit 1
fi

. ${runEnv}

EnvSetup
LOG_OUT=${LOG_DIR}/${HostName}_syncEBS_${fsAlias}_${TS}.log

# This process creates a ton of little rsync scripts.
# Create the script directory if it doesn't already exist
if [ ! -d syncScripts_${fsAlias} ]; then
   mkdir syncScripts_${fsAlias}
fi

# do the config files exist?
if [[ ! -f ${copyDirectories} || ! -f ${excludeFiles} ]]; then
   LogMsg "Your config files holding directories to copy and files/directories to exclude must
exist."
   LogMsg "They may be empty."
   exit 1
fi

}

################################################################################
# RunYesNo
# Is it ok to do this sync run?  This could be explicitly disabled, we could
# be at the standby, or there could be one already running.  If one of these
# is true, do not run.
# Note we force it to run during a switchover, in which case syncForSwitchover
# would be = 1.  This is managed in a separate script killSync.sh
#
#  Input:
# Output:
# Return: Exit if should not run, else proceed
################################################################################
RunYesNo()
{
LogMsg "RunYesNo: Is it appropriate to proceed?"
if [ -f ${SCRIPT_DIR}/.${fsAlias}_rsync_disabled ]; then
   if [ ${syncForSwitchover} == 0 ]; then
      LogMsg "RunYesNo: rsync is disabled at this site.  If appropriate, re-enable with
enableRsync.sh"
```

```
      exit 0
   else
      LogMsg "RunYesNo: rsync is disabled but syncEBS.sh is running in syncForSwitchover mode."
      LogMsg "RunYesNo: This will run a final rsync."
   fi
fi

# is there a thread of this flavor already running at this site?
# exit if lock file present.  create lock file if not, and continue
LogMsg "RunYesNo: lock file: ${SCRIPT_DIR}/.${fsAlias}.lck"
if [ -f ${SCRIPT_DIR}/.${fsAlias}.lck ]; then
   LogMsg "RunYesNo: rsync is currently running for ${fsAlias}"
   LogMsg "RunYesNo: Clear the lock file ${SCRIPT_DIR}/.${fsAlias}.lck if need to resume post-
crash"
   exit 0
else
   LogMsg "RunYesNo: Proceed - rsync is not running for ${fsAlias}"
   touch ${SCRIPT_DIR}/.${fsAlias}.lck
fi

GetLogon $dbSecretName
dbSecret=$LOGON

# Expensive, but need to check to be sure this database is PRIMARY
if [ ${forceRsync} = 0 ]; then
   LaunchCoroutine system $dbSecret $CDB_CONNECT_STRING

   sql="select rtrim(database_role) from v\$database;"
   role=`ExecSql "$sql"`

   if [ "${role}" != "PRIMARY" ]; then
      LogMsg "RunYesNo: This site is in ${role} role.  Only rsync from PRIMARY site."
      LogMsg "Clearing the lock for ${fsAlias}."
      rm ${SCRIPT_DIR}/.${fsAlias}.lck
      exit 1
   fi

   # We are not doing a forced sync and the site is in the PRIMARY role.
   # Start the killSync.sh process in the background if it’s not already running.
   # We are doing this here because we assume the syncEBS.sh script is
   # running on a server dedicated to the sync process, not a normal EBS
   # application server (else we would just kill a partially completed
   # sync process directly and move quickly to a final rsync)
   PROCESS_COUNT=$(ps -elf | grep "${APP_OWNER}" | grep -E "killSync.sh" | grep -v grep | wc -l
)
   if [[ ${PROCESS_COUNT} -eq 0 && ${syncForSwitchover} -eq 0 ]]; then
      # specifying 10 seconds between checks
      nohup ${SCRIPT_DIR}/killSync.sh 10 >/dev/null 2>&1 &
      LogMsg "RunYesNo: killSync.sh process started by syncEBS.sh."
   fi
else
   # User specified "force".  Make sure ok by connecting to DB
   mode=`sqlplus -s /nolog <<EOF!
   connect system/$dbSecret@${CDB_TNS_CONNECT_STRING}
   set heading off
   set feedback off
   select rtrim(database_role) from v\$database;
   exit
EOF!
`
  mode1=$( echo "${mode}" | grep "PRIMARY|PHYSICAL STANDBY|SNAPSHOT STANDBY" )
   if [ ${#mode1} -ne 0 ]; then
      if [ "${mode1}" = "PRIMARY" ]; then
         break
      else
         LogMsg "RunYesNo: Not safe to copy files from this site when role is ${mode1}"
         exit 1
      fi
    else
      LogMsg "RunYesNo: Got an error from SQL*Plus connection.  Proceeding with FORCE"
      LogMsg "mode: $mode"
   fi
fi

}

################################################################################
# SyncFiles
# Make a fresh copy of individual files from source to target.  Need this due
# to having to drill into the EBS directory structure in order to chunk up the
# directories fine enough for good multi-threading
#
#  Input: File containing source and target files to copy
# Output: Copied files
# Return: Exit 1 if source files do not exist
################################################################################
SyncFiles()
{
# Need an empty file to contain generated commands.
# Ignore error if it's not there.
# Doing this outside the loop, to clear out possible historical artifacts.
# Could leave them in place, since the execution of the generated script is
# done within this routine (leaving the old file in place would not result
# in accidentally running an old set of commands).  Let the reader decide...
LogMsg "SyncFiles: Recreating syncFiles_${fsAlias}.sh"
rm syncFiles_${fsAlias}.sh 2>/dev/null
touch syncFiles_${fsAlias}.sh
```

```
# They may not need to copy files.  Do nothing if that's the case.
LogMsg "SyncFiles: Any files to copy?"
if [ -f ${copyFiles} ]; then
   # Build the copy file commands
   cat ${copyFiles} | while read Source Destination
   do
      # skip comments
      if [ "${Source}" != "#" ]; then
         # make sure the source is a file
         if [ ! -f "${Source}" ]; then
            echo "$0: ERROR: ${Source} is not a file."
            exit 1
         fi
         LogMsg "Source: ${Source}  Destination: ${Destination}"
         # for local testing:
         # echo "cp ${Source} ${Destination}" >> syncFiles_${fsAlias}.sh
         echo "scp ${Source} ${USER}@${targetHostIP}:${Destination}" >> syncFiles_${fsAlias}.sh
      fi
   done

   chmod u+x syncFiles_${fsAlias}.sh

   LogMsg "SyncFiles: Synchronizing flat files."
   ./syncFiles_${fsAlias}.sh

else
   LogMsg "SyncFiles: No files to copy: ${copyFiles} is empty"
fi

LogMsg "SyncFiles: completed"

}

################################################################################
# SyncDirectories
# Build commands to rsync directories from source to target.
#
# This will exclude files or directories based on matching patterns
#
# NOTE: easy command to find long list of directories in a folder:
# ls -l | grep ^d | awk '{print $NF}'
#
#  Input: File containing source and target directories to rsync
# Output: Scripts to sync directories, file listing scripts
# Return: Warning if source directories do not exist
################################################################################
SyncDirectories()
{
LogMsg "SyncDirectories: Started"
```

```
# Need an empty file to contain generated commands
# ignore error if it's not there
rm ${fsAlias}.txt 2>/dev/null
touch ${fsAlias}.txt

counter=0

LogMsg "SyncDirectories: Building the driving text file for directories."
cat ${copyDirectories} | while read Source Destination
do
   # skip comments
   if [ "${Source}" != "#" ]; then
      # make sure the source is a directory
      if [ ! -d "${Source}" ]; then
         # If you just started using a new scheme for log and out directories
         # the directories may not yet exist - thus this is a warning, not
         # an error.  If you are confident your directories are in place,
         # make this an error with an exit 1.
         LogMsg "$0: Warning: Directory ${Source} does not exist."
      else
         ((counter++))
         command="find \"$Source\" -maxdepth 1 -mindepth 1 -type d -exec rsync -avPr --exclude-
from \"$excludeFiles\" --delete \"{}\" \"${USER}@${targetHostIP}:${Destination}\" \;"
         # This version is for local testing:
         # command="find \"$Source\" -maxdepth 1 -mindepth 1 -type d -exec rsync -avPr --
exclude-from \"$excludeFiles\" --delete \"{}\" \"$Destination\" \;"
         echo "${command}" > syncScripts_${fsAlias}/do_${counter}.sh
         chmod u+x syncScripts_${fsAlias}/do_${counter}.sh
         echo syncScripts_${fsAlias}/do_${counter}.sh >> ${fsAlias}.txt
      fi
   fi
done

LogMsg "SyncDirectories: ${counter} threads queued."

}

################################################################################
# Execution starts here.
################################################################################

ParseArgs $*
# Leave ParseArgs with copyDirectories set to the file holding the source
# and target directories, copyFiles (if present) set to a file holding specific
# files to copy

SetEnv

LogMsg "syncEBS.sh: Started"
```

```
# Is it appropriate to run at this time?
RunYesNo

# Did they provide a parameter pointing to a file holding a list of files
# to copy?
if [ "${copyFiles}" != "" ]; then
   SyncFiles
fi

# Build ${fsAlias}.txt, a text file containing rsync commands to sync
# lower-level directories
SyncDirectories

# Run the commands built by SyncDirectories.  MultiThread's parameters:
# name of driving file, # threads to kick off, grep phrase, # seconds to sleep
MultiThread ${fsAlias}.txt 15 syncScripts_${fsAlias} 3

# We're done.  Remove the lock file for the process.
rm ${SCRIPT_DIR}/.${fsAlias}.lck

LogMsg "Completed: syncEBS.sh"
```

14.5.5 killSync.sh

The script `stopEBS.sh` is written to block until any currently-running replication process finishes. It then starts one last complete file system replication, ensuring the application tier file system at the standby-to-be-primary is completely up to date. Blocking for a running rsync process may delay switchover for several minutes.

killSynch.sh runs in the background on the server running file synchronization processes at the primary. It is only triggered to do work on switchover. Its purpose is to kill any already-running file system synchronizations so that, after the database and all application servers are down, the final file system replication can start as soon as possible. The expectation is that this could save a few minutes during a switchover exercise.

```
#!/bin/ksh
################################################################################
# Name: killSync.sh
#
# Purpose: Kill any already-running file sync processes if we are doing a
#          switchover.
#
#          killSync.sh is started by syncEBS.sh when running at the primary
#          site.  It runs in the background.  There is no user interaction.
#          Script behavior is managed through the use of shared files.
#
# Usage:  killSync.sh [ sleep time in seconds ]
#
#  Errors:
#          - if env files do not exist.  See SetEnv.
#          - if the file system alias was not specified.
#
# Revisions:
# Date        What
# 05/23/2025  Created
################################################################################
# SetEnv
# Get environment variables, standard include routines
#
# This program is executed via cron or some other scheduling agent.  It is not
# executed from the command line by the OS user.  Make sure the variable
# SCRIPT_DIR is set in the OS user environment.  This can be done directly
# in the cron table.

# Input:  None
# Output: Environment set for the run
# Return: Exit 1 if can't find environment files, etc.
################################################################################
SetEnv()
{
# Include the basic "where am I" environment settings
if [ ! -f ${SCRIPT_DIR}/ebsAppTier.env ]; then
   echo "Cannot find the file ${SCRIPT_DIR}/ebsAppTier.env."
   exit 1
fi

. ${SCRIPT_DIR}/ebsAppTier.env

# include the standard functions routines
if [ ! -f ${SCRIPT_DIR}/stdfuncs.sh ]; then
   echo "Cannot find the standard functions script (stdfuncs.sh)"
   exit 1
fi

. ${SCRIPT_DIR}/stdfuncs.sh

EnvSetup
LOG_OUT=${LOG_DIR}/${HostName}_killSync_${TS}.log

}

################################################################################
# ChkAbort()
# Check for the presence of the .abortSync file and checks its content.
# Return a value to the calling routine.
#
# Input:  None
# Output: Environments set for the run
# Return: The content of the file if an abort is to be initiated
```

```
################################################################################
ChkAbort()
{

# Check to see if the .abortSync file is present.
if [ -f ${SCRIPT_DIR}/.abortSync ]; then
   chkrc=$( cat ${SCRIPT_DIR}/.abortSync | wc -l )

   # Check to see if any of the .env files do not exists. If there is a missing
   # env file, exit as we do not want to proceed.
   if [ ${chkrc} -ne 0 ]; then

      for i in $( cat ${SCRIPT_DIR}/.abortSync )
      do
         if [ ! -f ${i} ];
         then
            LogMsg "ChkAbort: ENV file: ${i} does not exist."
            LogMsg “Exiting.”
            exit 1
         fi
      done

   else
      LogMsg "ChkAbort: No env file specified."
      LogMsg   “Exiting.”
      exit 1
   fi
fi

}

################################################################################
# KillSync
#
# NOTE: PARAMETERIZE THIS
#
# Input:  None
# Output: Kill any and all rsync and syncEBS processes and remove the locks.
#         This allows for a full final sync to take place once all processes
#         on all app servers are down.
#
# Return: None
################################################################################
KillSync()
{

LogMsg "KillSync: Started."

# The .abortSync file will have the list of the env files that will provide the fsAlias we need.
for i in $( cat ${SCRIPT_DIR}/.abortSync )
do
   if [ ! -f ${i} ];
   then
      LogMsg "KillSync:  ENV file: ${i} does not exists."
      LogMsg "KillSync:  Continuing...."
   else
      # Source the env file.
      . ${i}
   fi

   # The fsAlias variable is defined once the env file has been source.
   GREP_STRING="${fsAlias}"

   LogMsg "KillSync: Terminating existing rsync processes for ${fsAlias}."

   PROCESS_COUNT=$(ps -elf | grep "${APP_OWNER}" | grep -E "${GREP_STRING}" | grep -v grep | wc
-l )
   LogMsg "KillSync: Number of remaining processes: ${PROCESS_COUNT}"
   while [ ${PROCESS_COUNT} -ne 0 ];
   do
     Running=$(ps -elf | grep "${APP_OWNER}" | grep -E "${GREP_STRING}" | grep -v grep )
     LogMsg "${PROCESS_COUNT} remaining processes: ${Running}"
     PID_LIST=$(ps -elf | grep "${APP_OWNER}" | grep -E "${GREP_STRING}" | grep -v grep | awk '{
print $4 }' )
     LogMsg "KillSync: Killing processes: ${PID_LIST}"
     kill -9 ${PID_LIST}
     PROCESS_COUNT=$(ps -elf | grep "${APP_OWNER}" | grep -E "${GREP_STRING}" | grep -v grep |
wc -l )
   done
   # Remove the lck files.

   if [ -f ${SCRIPT_DIR}/.${fsAlias}.lck ]; then
      rm ${SCRIPT_DIR}/.${fsAlias}.lck
   fi

done

LogMsg "KillSync: Completed."

}

################################################################################
# Execution starts here.
################################################################################

SetEnv
```

```
SLEEP_TIME_SEC=$1

if [ -z ${SLEEP_TIME_SEC} ]; then
   SLEEP_TIME_SEC=15
fi

LogMsg "killSync.sh: Started."
LogMsg "Sleep time ${SLEEP_TIME_SEC} seconds..."

while true
do
   chkrc=0
   # ChkAbort will set the chkrc variable if an abort was triggered.
   ChkAbort

   if [ ${chkrc} -ne 0 ]; then
      LogMsg "killSync.sh: Received Abort-Sync."
      KillSync
      rm ${SCRIPT_DIR}/.abortSync
      break
   fi
   sleep ${SLEEP_TIME_SEC}
done

LogMsg "killSync.sh: Shutdown completed."
```

14.5.6 Example CRON Job Entries

In this project, we are synchronizing file systems at two rates, one category once a day ("slowFiles") and one that we synchronize as frequently as feasible ("fastFiles").

- The directories that hold the EBS log and out files need to match the contents of the database as closely as possible, thus need to have a higher frequency of replication. We set ours to start every 5 minutes by configuring the minutes column (`*/5`).
- The directories holding EBS software are more static and are set up to replicate once a day, at 1:00 AM (`0 1`).

When the cron job runs, it will not source the user's .profile. Therefore we need to define the SCRIPT_DIR environment variable before running the commands, as shown here:

```
SCRIPT_DIR=/u02/app/ebs/custom_admin_scripts/VISPRD
*/5 * * * * applmgr $SCRIPT_DIR/syncEBS.sh $SCRIPT_DIR/fastFiles.env
0 1 * * * applmgr $SCRIPT_DIR/syncEBS.sh $SCRIPT_DIR/slowFiles.env
```

If you have multiple EBS environments to manage, best practice is for each environment to be owned and managed by a unique OS user and for each to have a dedicated application code home directory. To keep these separate EBS environments in sync, when you can add entries to crontab you will specify the unique user and their SCRIPT_DIR directory.

```
SCRIPT_DIR=/u02/app/ebs/custom_admin_scripts/EBSPRD
*/5 * * * * applmgr2 $SCRIPT_DIR/syncEBS.sh $SCRIPT_DIR/fastFiles.env
0 1 * * * applmgr2 $SCRIPT_DIR/syncEBS.sh $SCRIPT_DIR/slowFiles.env
```

Note that these crontab entries are configured at both sites and must be active at each site.

The `syncEBS.sh` script will only replicate from a site if:

- The synchronization process is not explicitly disabled and the database is up and in the PRIMARY role,
- The database is down, the site is in the PRIMARY role, and the script is called with the parameter “S” to force a final synchronization on switchover, or
- The database is down, the site is in the PRIMARY role, and the user specified “F” to force file synchronization.

15 Appendix

In Section 2: Core Requirements and Assumptions, you are instructed to configure logical host names in your primary environment if you have not already done so. This can be done following the steps in 8.2.2 to 8.2.5 in Oracle E-Business Suite Release 12.2 Maximum Availability Architecture.

While doing this, if you use a domain name different from the one assigned to the application tier compute instance subnet, you may encounter bug 22153846 – RAPIDCLONE ON THE DB TIER DOES NOT PROMPT FOR DB DOMAIN NAME when running `adcfgclone.pl` when following the steps in section 8.2.5. To work around the bug:

- Do the tasks described in 8.2.2 – 8.2.4, then steps 1-4 of 8.2.5 as described in Oracle E-Business Suite Release 12.2 Maximum Availability Architecture.
- Replace step 5 of section 8.2.5 with the steps below.
- Do 8.2.5 steps 6 and 7 as described in Oracle E-Business Suite Release 12.2 Maximum Availability Architecture.
- Skip 8.2.5 step 8, as the web entry points were configured with the below actions.
- Complete 8.2.5 steps 9-16.

These actions replace step 5 of 8.2.5 in Oracle E-Business Suite Release 12.2 Maximum Availability Architecture:

- For safety, first create a backup copy of your `CONTEXT_FILE` on each application tier node.
- Note the current values of these variables in the `CONTEXT_FILE`:
  - s_webentryhost
  - s_webentrydomain
  - s_login_page
  - s_external_url
  - s_webport
  - s_active_webport
  - s_http_listen_parameter
- Create a pairsfile that can specify the domain name, tagged `s_domain`. See the sample pairsfile below.
- Execute `adcfgclone.pl`, passing in your pairsfile as shown here:

  ```
  $ perl adcfgclone.pl component=appsTier pairsfile=<pairsfile name> dualfs=yes
  ```

  Note: when using the pairsfile, the prompts required to run adcfgclone.pl will differ from those described in section 8.2.5 of Oracle E-Business Suite Release 12.2 Maximum Availability Architecture.

After adcfgclone.pl completes and autoconfig runs successfully, do these tasks:

1. Source the new cloned RUN file system.
2. Edit the context file – remove the # from the context variable `s_enable_sslterminator`.
3. Reset the context variables saved above to their original values.
4. Run autoconfig.

Repeat these four steps for the PATCH file system.

Do these steps on each middle tier node as you add it back to the configuration.

This sample pairsfile from our environment has comments pointing out the settings you will need to change, to work around the configuration bug. Note the adjustments to the web service addresses, specifying `https://` instead of `http://`.

```
[DB Info]
s_dbSid=VISPRD              ← PDB name, not the database SID.
s_dbhost=ebsdb1-vip         ← The database VIP.
s_dbdomain=example.com      ← The domain of your database server.
s_dbport=1521

[Base Directory]
s_base=/u02/app/ebs/visprd  ← The base directory for where the app tier will be installed.
s_current_base=%s_base%/fs1
s_other_base=%s_base%/fs2
s_fmw_home=%s_base%/FMW_Home
s_at=%s_base%/fs1/EBSapps/appl
s_at1=%s_at%
s_at2=%s_at%
s_at3=%s_at%
s_com=%s_base%/fs1/EBSapps/comn
s_tools_oh=%s_base%/fs1/EBSapps/10.1.2
s_weboh_oh=%s_base%/fs1/FMW_Home/webtier
s_inst_base=%s_base%
s_clonestage=%s_com%/clone
s_proxyhost=
s_proxyport=
s_ne_base=%s_base%/fs_ne
s_contextname=EBSDB_apps
s_config_home=/u02/app/ebs/visprd/fs1/inst/apps/VISPRD_ebsapp1

[General]
s_appltmp=/usr/tmp
s_applptmp=/u02/app/ebs/visprd/temp/VISPRD
s_hostname=ebsapp1           ← Logical hostname for this node.
s_dbCluster=false
s_display=localhost:5.0
s_forms-c4ws_display=apps:0.0
s_options_symlinks=Options FollowSymLinks
s_file_edition=run

[Web Entry Point Configuration]
s_webentryurlprotocol=http
s_webentryhost=apps
s_webentrydomain=example.com
s_active_webport=8000
```

```
s_endUserMonitoringURL=https://apps.example.com:8000/oracle_smp_chronos/oracle_smp_chronos_sdk.g
if
s_external_url=https://apps.example.com:8000
s_login_page=https://apps.example.com:8000/OA_HTML/AppsLogin

[Services]
# Please provide values for the context variables listed below .
# Enter "enabled" without the quotes to enable the service on the new node .
# Enter "disabled" without the quotes to disable the service on the new node .
# The Root service include the Node Manager .
# The Web Application Services include the Node Manager, Admin Server,
# Managed Servers ( oacore, forms, oafm, formsc4-ws).
#
# To enable the configuration of Node Manager and the Managed Servers,
# set s_web_applications_status to enabled
#
# The Web Entry Services include the OPMN and Oracle HTTP Server .
# To enable the configuration of OPMN,OHS , set s_webentry_status and s_apcstatus to enabled.
#
# The Batch Processing Services include the FNDFS Listener, Concurrent Mgr, ICSM# and JTF
FullFillment Server
# To enable the configuration of Concurrent Manager# set s_batch_status to enabled.
# The Other Services group include the Forms Metric Server, Forms Metric Clients# Forms Server
for the socket mode configuration and Mobile Web Application
# Server (MWA).

# s_adminserverstatus is set to disabled since this service can only be enabled
# on the primary application tier.
#
# The services enabled on the primary application tier node are as shown
# below in the comment section.

[Services Enabled on the Primary Application Tier Node]
s_web_applications_status=enabled
s_web_entry_status=enabled
s_apcstatus=enabled
s_root_status=enabled
s_batch_status=enabled
s_other_service_group_status=disabled
s_adminserverstatus=enabled
s_web_admin_status=enabled

[Variables specific for patch file system]
# Please define these patch_<vars> for single go RC and add node.
# If are using the pairsfile just to clone the context file then these patch_<vars>
# are not needed to be defined

s_port_pool=0
patch_s_port_pool=50
```

```
#
#  Variables for this VM
#
s_appsuser=applmgr
s_appsgroup=oinstall
s_domainname=example.com    ← The domain name that you want to assign. This can be different
from the subnet-assigned domain name.
s_apps_pass=<APPS password>
s_weblogic_pass=<WLS Admin password>

# Additional variables can be added below as per your requirement.

s_proxyport=80
```

Connect with us

Call **+1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

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