AWS Whitepaper

Strategies for Migrating Oracle Databases to AWS

Strategies for Migrating Oracle Databases to AWS: AWS Whitepaper

Table of Contents

Strategies for Migrating Oracle Databases to AWS

Publication date: **January 27, 2022** (*Document history*)

Amazon Web Services (AWS) provides a comprehensive set of services and tools for deploying enterprise-grade solutions in a rapid, reliable, and cost-effective manner.

Oracle Database is a widely used relational database management system that is deployed in enterprises of all sizes. It manages various forms of data in many phases of business transactions. This whitepaper describes the preferred methods for migrating an Oracle Database to AWS, and helps you choose the method that is best for your business.

Introduction

This whitepaper presents best practices and methods for migrating Oracle Database from servers that are on-premises or in your data center to AWS. Data, unlike application binaries, cannot be recreated or reinstalled, so you should carefully plan your data migration and base it on proven best practices.

AWS offers its customers the flexibility of running Oracle Database on Amazon Relational Database Service (Amazon RDS), the managed database service in the cloud, as well as Amazon Elastic Compute Cloud (Amazon EC2):

- **Amazon RDS** makes it simple to set up, operate, and scale a relational database in the cloud. It provides cost-efficient, resizable capacity for an open standard relational database, and manages common database administration tasks.
- **Amazon EC2** provides scalable computing capacity in the cloud. Using Amazon EC2 removes the need to invest in hardware up front, so you can develop and deploy applications faster. You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage.

Running the database on Amazon EC2 is very similar to running the database on your own servers. Depending on whether you choose to run your Oracle Database on Amazon EC2 or Amazon RDS, the process for data migration can differ. For example, users don't have OS-level access in Amazon RDS instances. It's important to understand the different possible strategies, so you can choose the one that best fits your needs.

Data migration strategies

The migration strategy you choose depends on several factors:

- The size of the database
- Network connectivity between the source server and AWS
- The version and edition of your Oracle Database software
- The database options, tools, and utilities that are available
- The amount of time that is available for migration
- Whether the migration and switchover to AWS will be done in one step or a sequence of steps over time

The following sections describe some common migration strategies.

One-step migration

One-step migration is a good option for small databases that can be shut down for 24 to 72 hours. During the shut down period, all the data from the source database is extracted, and the extracted data is migrated to the destination database in AWS. The destination database in AWS is tested and validated for data consistency with the source. Once all validations have passed, the database is switched over to AWS.

Two-step migration

Two-step migration is a commonly used method because it requires only minimal downtime and can be used for databases of any size:

1. The data is extracted from the source database at a point in time (preferably during non-peak usage) and migrated while the database is still up and running. Because there is no downtime at this point, the migration window can be sufficiently large. After you complete the data migration, you can validate the data in the destination database for consistency with the source and test the destination database on AWS for performance, connectivity to the applications, and any other criteria as needed.
2. Data changed in the source database after the initial data migration is propagated to the destination before switchover. This step synchronizes the source and destination databases. This

should be scheduled for a time when the database can be shut down (usually over a few hours late at night on a weekend). During this process, there won't be any more changes to the source database because it will be unavailable to the applications.

Normally, the amount of data that is changed after the first step is small compared to the total size of the database, so this step will be quick and requires only minimal downtime. After all the changed data is migrated, you can validate the data in the destination database, perform necessary tests, and, if all tests are passed, switch over to the database in AWS.

Minimal downtime migration

Some business situations require database migration with little to no downtime. This requires detailed planning and the necessary data replication tools for proper completion.

These migration methodologies typically involve two components: an initial bulk extract/load, followed by the application of any changes that occurred during the time the bulk step took to run. After the changes have applied, you should validate the migrated data and conduct any necessary testing.

The replication process synchronizes the destination database with the source database, and continues to replicate all data changes at the source to the destination.

Synchronous replication can have an effect on the performance of the source database, so if a few minutes of downtime for the database is acceptable, then you should set up asynchronous replication instead. You can switch over to the database in AWS at any time, because the source and destination databases will always be in sync.

There are a number of tools available to help with minimal downtime migration. The AWS Database Migration Service (AWS DMS) supports a range of database engines, including Oracle running on-premises, in EC2, or on RDS. Oracle GoldenGate is another option for real-time data replication. There are also third-party tools available to do the replication.

Nearly continuous data replication

You can use nearly continuous data replication if the destination database in AWS is used as a clone for reporting and business intelligence (BI), or for disaster recovery (DR) purposes. In this case, the process is exactly the same as minimal downtime migration, except that there is no switchover and the replication never stops.

Tools used for Oracle Database migration

A number of tools and technologies are available for data migration. You can use some of these tools interchangeably, or you can use other third-party tools or open-source tools available in the market.

- **AWS DMS** helps you move databases to and from AWS easily and securely. It supports most commercial and open-source databases, and facilitates both homogeneous and heterogeneous migrations. AWS DMS offers change data capture technology to keep databases in sync and minimize downtime during a migration. It is a managed service with no client install required.
- **Oracle Recovery Manager (RMAN)** is a tool available from Oracle for performing and managing Oracle Database backups and restorations. RMAN allows full hot or cold backups, plus incremental backups. RMAN maintains a catalogue of the backups, making the restoration process simple and dependable. RMAN can also duplicate, or clone, a database from a backup or from an active database.
- **Oracle Data Pump Export** is a versatile utility for exporting and importing data and metadata from or to Oracle databases. You can perform Data Pump export/import on an entire database, selective schemas, table spaces, or database objects. Data Pump export/import also has powerful data-filtering capabilities for selective export or import of data.
- **Oracle GoldenGate** is a tool for replicating data between a source and one or more destination databases. You can use it to build high-availability architectures. You can also use it to perform real-time data integration, transactional change data capture, and replication in heterogeneous IT environments.
- **Oracle SQL Developer** is a no-cost GUI tool available from Oracle for data manipulation, development, and management. This Java-based tool is available for Microsoft Windows, Linux, or iOS X.
- **Oracle SQL*Loader** is a bulk data-load utility available from Oracle for loading data from external files into a database. SQL*Loader is included as part of the full database client installation.

Creating a database on Amazon RDS, Amazon EC2, or VMware Cloud on AWS

To migrate your data to AWS, you need a source database (either on-premises or in a data center) and a destination database in AWS. Based on your business needs, you can choose between using Amazon RDS for Oracle, or installing and managing the database on your own in Amazon EC2 instance. To help you choose the service that's best for your business, see the following sections.

Amazon RDS

Many customers prefer Amazon RDS for Oracle because it frees them to focus on application development. Amazon RDS automates time-consuming database administration tasks, including provisioning, backups, software patching, monitoring, and hardware scaling. Amazon RDS simplifies the task of running a database by eliminating the need to plan and provision the infrastructure, as well as install, configure, and maintain the database software.

Amazon RDS for Oracle makes it easy to use replication to enhance availability and reliability for production workloads. By using the Multi-Availability Zone (AZ) deployment option, you can run mission-critical workloads with high availability and built-in automated failover from your primary database to a synchronously replicated secondary database. As with all AWS services, no upfront investments are required, and you pay only for the resources you use. For more information, see Amazon RDS for Oracle.

To use Amazon RDS, log in to your AWS account and start an Amazon RDS Oracle instance from the AWS Management Console. A good strategy is to treat this as an interim migration database from which the final database will be created. Do not enable the Multi-AZ feature until the data migration is completely done, because replication for Multi-AZ will hinder data migration performance. Be sure to give the instance enough space to store the import data files. Typically, this requires you to provision twice as much capacity as the size of the database.

Amazon EC2

Alternatively, you can run an Oracle database directly on Amazon EC2, which gives you full control over setup of the entire infrastructure and database environment. This option provides a familiar approach, but also requires you to set up, configure, manage, and tune all the components, such as Amazon EC2 instances, networking, storage volumes, scalability, and security, as needed (based on

AWS architecture best practices). For more information, see the Advanced Architectures for Oracle Database on Amazon EC2 whitepaper for guidance about the appropriate architecture to choose, and for installation and configuration instructions.

VMware Cloud on AWS

VMware Cloud on AWS is the preferred service for AWS for all vSphere-based workloads. VMware Cloud on AWS brings the VMware software designed data center (SDDC) software to the AWS Cloud with optimized access to native AWS services. If your Oracle workload runs on VMware on-premises, you can easily migrate the Oracle workloads to the AWS Cloud using VMware Cloud on AWS.

VMware Cloud on AWS has the capability to run Oracle Real Application Clusters (RAC) workloads. It allows multi-cast protocols, and provides shared storage capability across VMs running in VMware Cloud on AWS SDDC. VMware provides native migration capabilities such as VMware VMotion and VMware HCX to move virtual machines (VMs) from on-premises to the VMware Cloud on AWS. Depending on Oracle workload performance patterns, service-level agreement (SLA), and the bandwidth availability, you can choose to migrate the VM either live or using cold migration methods.

Data migration methods

The remainder of this whitepaper provides details about each method for migrating data from Oracle Database to AWS. Before you get to the details, you can scan the following table for a quick summary of each method.

Each method depends upon business recovery point objective (RPO), recovery time objective (RTO), and overall availability SLA. Migration administrators must evaluate and map these business agreements with the appropriate methods. Choose the method depending upon your application SLA, RTO, RPO, tool, and license availability.

Table 1 – Migration methods and tools

Data migration method	Database size	Works for:	Recommended for:
AWS Database Migration Service	Any size	Amazon RDS Amazon EC2	Minimal downtime migration Database size limited by internet bandwidth
Oracle SQL Developer Database copy	Up to 200 MB	Amazon RDS Amazon EC2	Small databases with any number of objects
Oracle Materialized Views	Up to 500 MB	Amazon RDS Amazon EC2	Small databases with limited number of objects
Oracle SQL*Loader	Up to 10 GB	Amazon RDS Amazon EC2	Small to medium size databases with limited number of objects

Data migration method	Database size	Works for:	Recommended for:
Oracle Export and Import Oracle Utilities	Up to 10 GB	Amazon RDS Amazon EC2	Small to medium size databases with large number of objects
Oracle Data Pump	Up to 5 TB	Amazon RDS Amazon EC2 VMware Cloud on AWS	Preferred method for any database from 10 GB to 5 TB
External tables	Up to 1 TB	Amazon RDS Amazon EC2 VMware Cloud on AWS	Scenarios where this is the standard method in use
Oracle RMAN	Any size	Amazon EC2 VMware Cloud on AWS	Databases over 5 TB, or if database backup is already in Amazon Simple Storage Service (Amazon S3)
Oracle GoldenGate	Any size	Amazon RDS Amazon EC2 VMware Cloud on AWS	Minimal downtime migration

Migrating data for small Oracle databases

You should base your strategy for data migration on the database size, reliability, and bandwidth of your network connection to AWS, and the amount of time available for migration. Many Oracle databases tend to be medium to large in size, ranging anywhere from 10 GB to 5 TB, with some as large as 20 TB or more. However, you also might need to migrate smaller databases. This is especially true for phased migrations where the databases are broken up by schema, making each migration effort small in size.

If the source database is under 10 GB, and if you have a reliable high-speed internet connection, you can use one of the following methods for your data migration. All the methods discussed in this section work with Amazon RDS Oracle or Oracle Database running on Amazon EC2.

> **Note**
>
> The 10 GB size is just a guideline; you can use the same methods for larger databases as well. The migration time varies based on the data size and the network throughput. However, if your database size exceeds 50 GB, you should use one of the methods listed in the Migrating data for large Oracle databases section in this whitepaper.

Topics

Oracle SQL Developer database copy

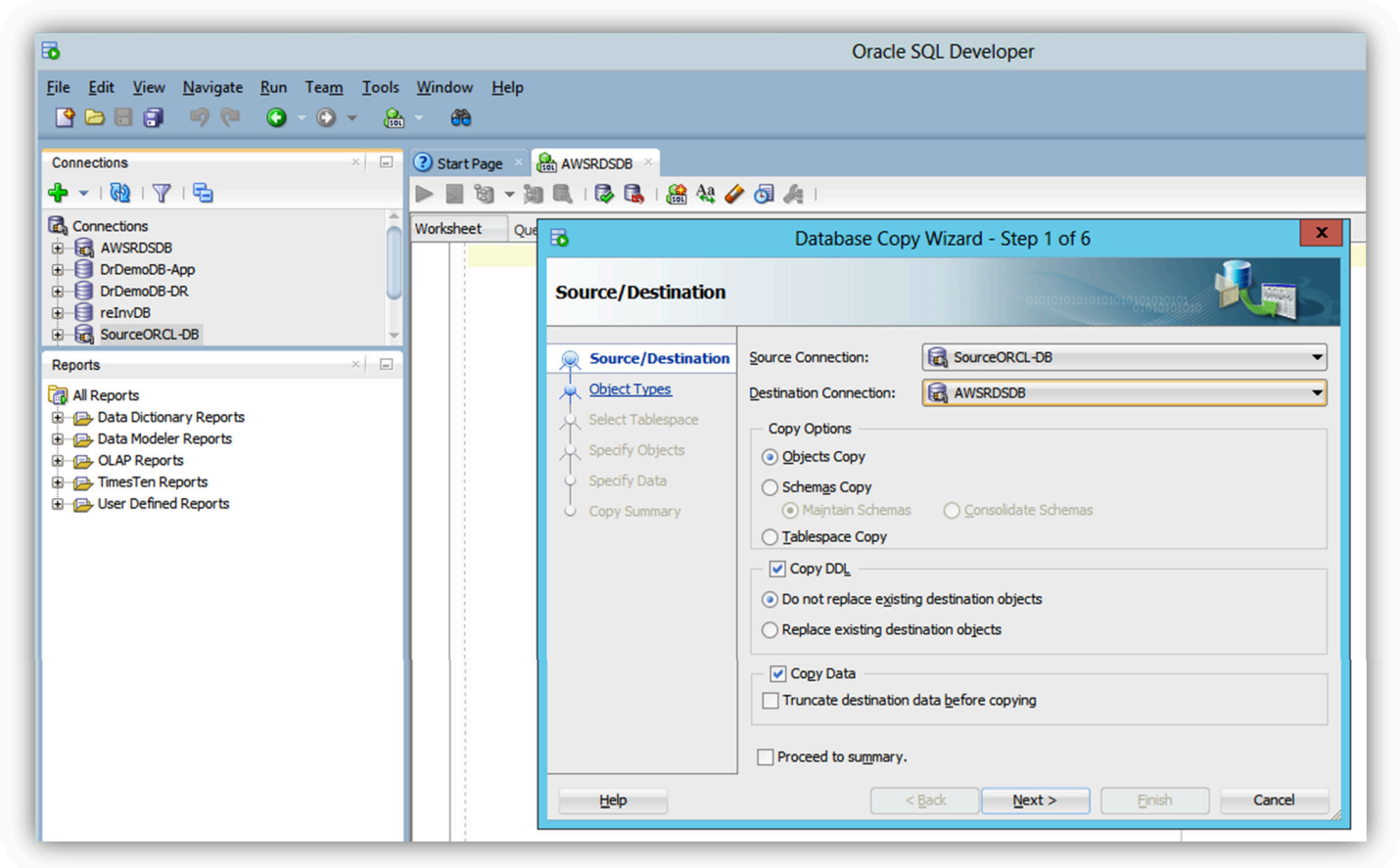
If the total size of the data you are migrating is under 200 MB, the simplest solution is to use the Oracle SQL Developer **Database Copy** function. Oracle SQL Developer is a no- cost GUI tool available from Oracle for data manipulation, development, and management. This easy-to-use, Java-based tool is available for Microsoft Windows, Linux, or Mac OS X. With this method, data transfer from a source database to a destination database is done directly, without any intermediary steps.

Because SQL Developer can handle a large number of objects, it can comfortably migrate small databases, even if the database contains numerous objects. You will need a reliable network connection between the source database and the destination database to use this method. Keep in mind that this method does **not** encrypt data during transfer.

To migrate a database using the Oracle SQL Developer **Database Copy** function, perform the following steps:

1. Install Oracle SQL Developer.
2. Connect to your source and destination databases.
3. From the **Tools** menu of Oracle SQL Developer, choose the **Database Copy** command to copy your data to your Amazon RDS or Amazon EC2 instance.
4. Follow the steps in the **Database Copy Wizard**. You can choose the objects you want to migrate and use filters to limit the data.

The following screenshot shows the **Database Copy Wizard**.



The Database Copy Wizard in the Oracle SQL Developer guides you through your data transfer

Oracle materialized views

You can use Oracle Database materialized views to migrate data to Oracle databases on AWS, for either Amazon RDS or Amazon EC2. This method is well suited for databases under 500 MB.

Because materialized views are available only in Oracle Database Enterprise Edition, this method works only if Oracle Database Enterprise Edition is used for both the source database and the destination database. With materialized view replication, you can do a one-time migration of data to AWS while keeping the destination tables continuously in sync with the source. The result is a minimal downtime cut over. Replication occurs over a database link between the source and destination databases. For the initial load, you must do a full refresh so that all the data in the source tables gets moved to the destination tables.

> **Important**
>
> Because the data is transferred over a database link, the source and destination databases must be able to connect to each other over SQL*Net. If your network security design doesn't allow such a connection, then you cannot use this method.

Unlike the preceding method (the Oracle SQL Developer **Database Copy** function) in which you copy an entire database, for this method you must create a materialized view for each table that you want to migrate. This gives you the flexibility of selectively moving tables to the database in AWS. However, it also makes the process more cumbersome if you need to migrate a large number of tables. For this reason, this method is better suited for migrating a limited number of tables.

For best results with this method, complete the following steps. Assume the source database user ID is `SourceUser` with password `PASS:`

1. Create a new user in the Amazon RDS or Amazon EC2 database with sufficient privileges.

```
Create user MV_DBLink_AWSUser identified by password
```

2. Create a database link to the source database.

```
CREATE DATABASE LINK SourceDB_lnk CONNECT TO SourceUser
```

```
IDENTIFIED BY PASS USING
'(description=(address=(protocol=tcp) (host= crmdb.acmecorp.com)
(port=1521)) (connect_data=(sid=ORCLCRM)))'
```

3. Test the database link to make sure you can access the tables in the source database from the database in AWS through the database link.

```
Select * from tab@ SourceDB_lnk
```

4. Log in to the source database and create a materialized view log for each table that you want to migrate.

```
CREATE MATERIALIZED VIEW LOG ON customers
```

5. In the destination database in AWS, create materialized views for each table for which you set up a materialized view log in the source database.

```
CREATE MATERIALIZED VIEW customer BUILD IMMEDIATE REFRESH
FAST AS SELECT * FROM customer@ SourceDB_lnk
```

Oracle SQL*Loader

Oracle SQL*Loader is well suited for small to moderate databases under 10 GB that contain a limited number of objects. Because the process involved in exporting from a source database and loading to a destination database is specific to a schema, you should use this process for one schema at a time. If the database contains multiple schemas, you need to repeat the process for each schema. This method can be a good choice even if the total database size is large, because you can do the import in multiple phases (one schema at a time).

You can use this method for Oracle Database on either Amazon RDS or Amazon EC2, and you can choose between the following two options:

Option 1

1. Extract data from the source database, such as into flat files with column and row delimiters.
2. Create tables in the destination database exactly like the source (use a generated script).
3. Using SQL*Loader, connect to the destination database from the source machine and import the data.

Option 2

1. Extract data from the source database, such as into flat files with column and row delimiters.
2. Compress and encrypt the files.
3. Launch an Amazon EC2 instance, and install the full Oracle client on it (for SQL*Loader). For the database on Amazon EC2, this can be the same instance where the destination database is located. For Amazon RDS, this is a temporary instance.
4. Transport the files to the Amazon EC2 instance.
5. Decompress and unencrypt files in the Amazon EC2 instance.
6. Create tables in the destination database exactly like the source (use a generated script).
7. Using SQL*Loader, connect to the destination database from the temporary Amazon EC2 instance and import the data.

Use the first option if your database size is small, if you have direct SQL*Net access to the destination database in AWS, and if data security is not a concern. Otherwise, use the second option, because you can use encryption and compression during the transportation phase. Compression substantially reduces the size of the files, making data transportation much faster.

You can use either SQL*Plus or SQL Developer to perform data extraction, which is the first step in both options. For SQL*Plus, use a query in a SQL script file and send the output directly to a text file, as shown in the following example:

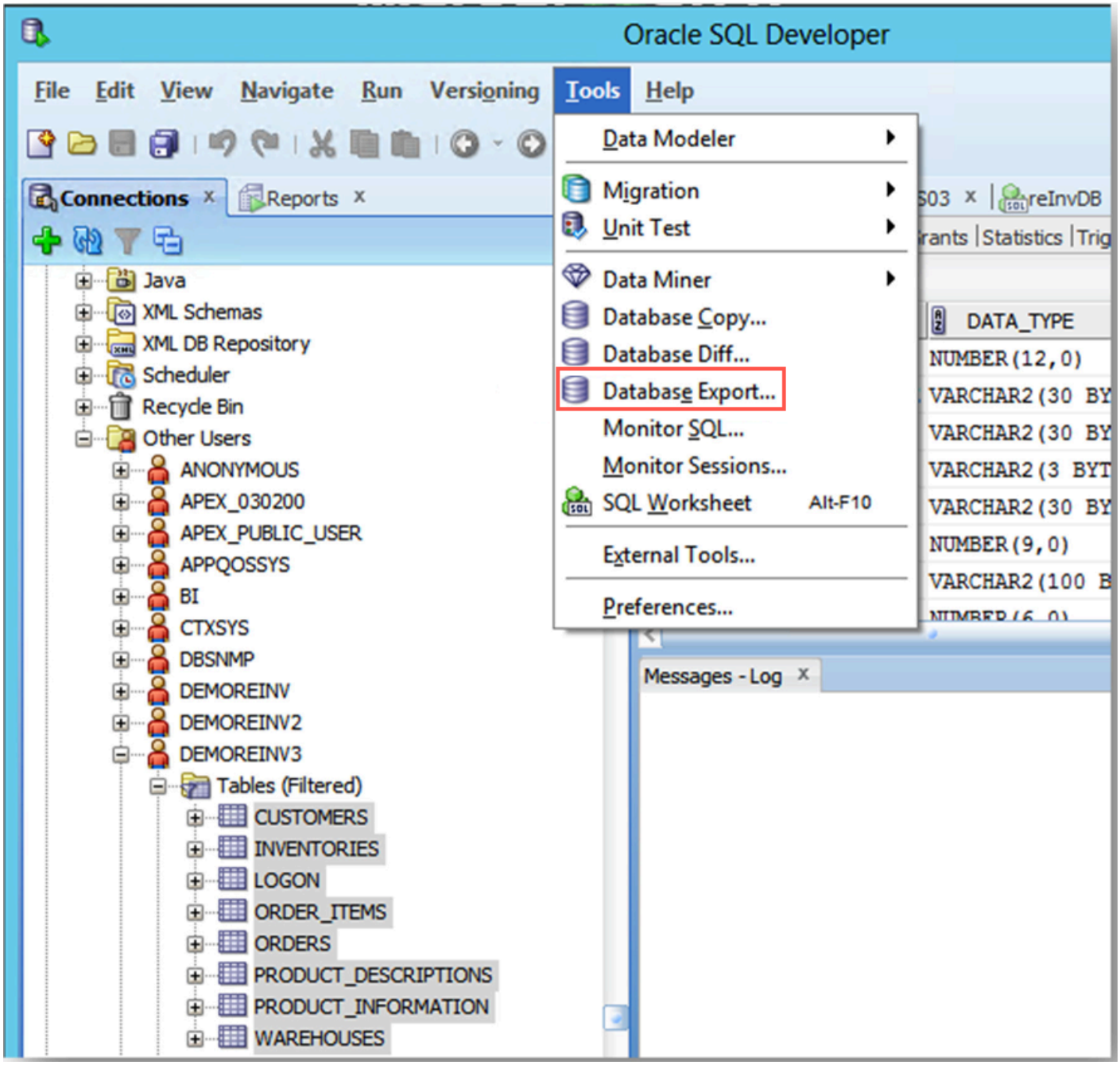
```
set pagesize 0
set head off
set feed off
set line 200

SELECT col1|| '|' ||col2|| '|' ||col3|| '|' ||col4|| '|'
||col5 from SCHEMA.TABLE;
exit;
```

To create encrypted and compressed output in the second option (see step 2 of the preceding Option 2 procedure), you can directly pipe the output to a zip utility.

You can also extract data by using Oracle SQL Developer:

1. In the **Connections** pane, select the tables you want to extract data from.

2. From the **Tools** menu, choose the **Database Export** command, as shown in the following screenshot.

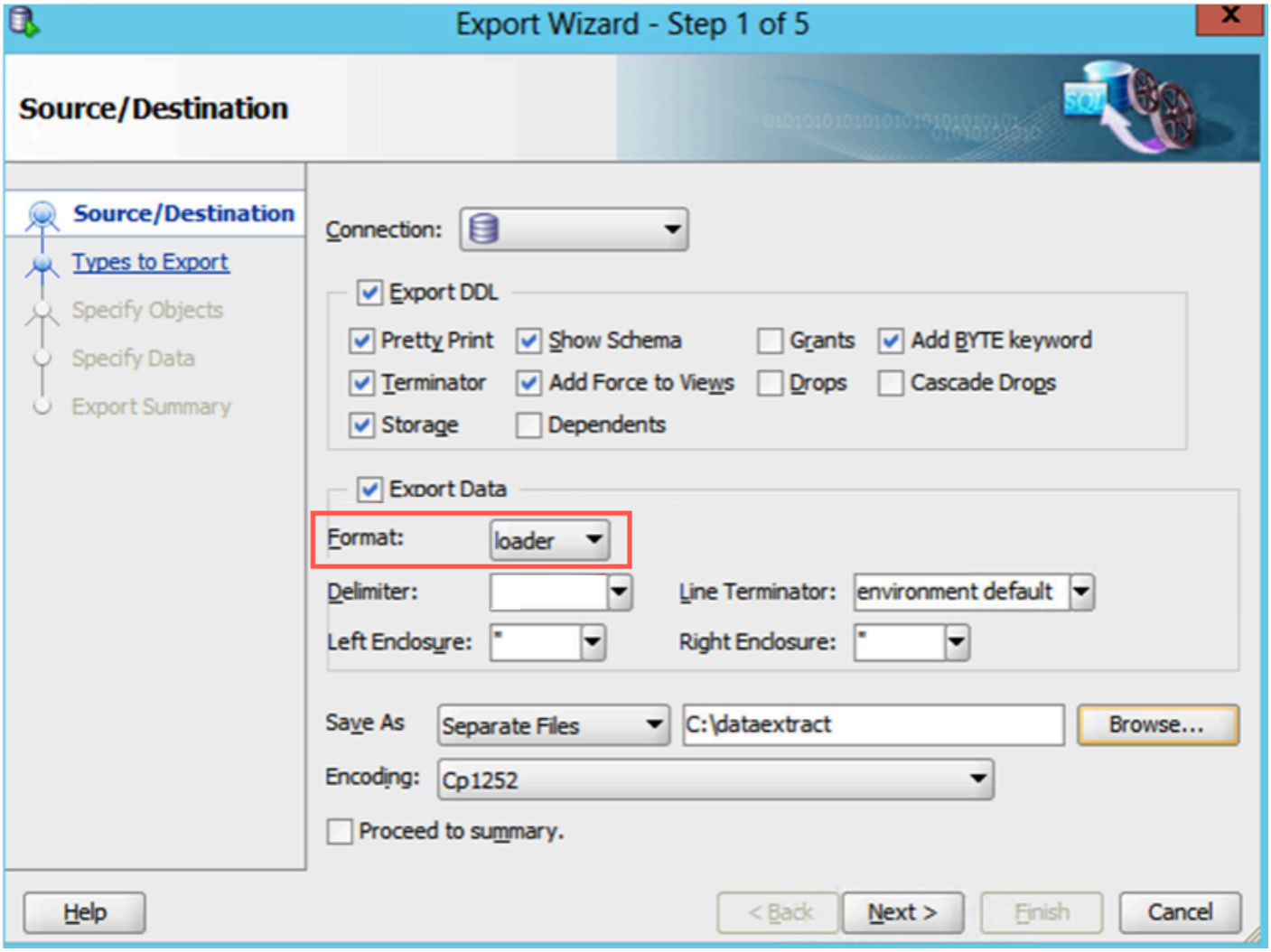


Database export command

3. On the **Source/Destination** page of the **Export Wizard** (see the next screenshot), select the **Export DDL** option to generate the script for creating the table, which will simplify the entire process.

4. In the **Format** drop-down on the same page, choose **loader**.

5. In the **Save As** box on the same page, choose **Separate Files**.



Export Wizard options on the Source/Destination page

Continue to follow the Export Wizard steps to complete the export. The Export Wizard helps you create the data file, control file, and table creation script in one step for multiple tables in a schema, making it easier than using Oracle SQL*Plus to do the same tasks.

If you use Option 1 as specified, you can run Oracle SQL*Loader from the source environment using the extracted data and control files to import data into the destination database. To do this, use the following command:

```
sqlldr userid=userID/password@$service control=control.ctl
        log=load.log bad=load.bad discard=load.dsc data=load.dat direct=y
        skip_index_maintenance=true errors=0
```

If you use Option 2, then you need an Amazon EC2 instance with the full Oracle client installed. Additionally, you need to upload the data files to that Amazon EC2 instance. For the database on Amazon EC2, this could be the same Amazon EC2 instance where the destination database is located. For Amazon RDS, this will be a temporary Amazon EC2 instance.

Before you do the upload, we recommend that you compress and encrypt your files. To do this, you can use a combination of TAR and ZIP/GZIP in Linux or a third-party utility such as WinZip or 7-Zip. After the Amazon EC2 instance is up and running and the files are compressed and encrypted, upload the files to the Amazon EC2 instance using Secure File Transfer Protocol (SFTP).

From the Amazon EC2 instance, connect to the destination database using Oracle SQL*Plus to ensure you can establish the connection. Run the `sqlldr` command shown in the preceding example for each control file that you have from the extract. You can also create a shell/bat script that will run `sqlldr` for all control files, one after the other.

> **Note**
>
> Enabling `skip_index_maintenance=true` significantly increases data-load performance. However, table indexes are not updated, so you will need to rebuild all indexes after the data load is complete.

Oracle Export and Import utilities

Despite being replaced by Oracle Data Pump, the original Oracle Export and Import utilities are useful for migrations of databases with sizes less than 10 GB where the data lacks binary float and double data types. The import process creates the schema objects, so you do not need to run a script to create them beforehand. This makes the process well suited for databases with a large number of small tables. You can use this method for Amazon RDS for Oracle and Oracle Database on Amazon EC2.

The first step is to export the tables from the source database by using the following command. Substitute the credentials as appropriate:

```
exp userID/password@$service FILE=exp_file.dmp LOG=exp_file.log
```

The export process creates a binary dump file that contains both the schema and data for the specified tables. You can import the schema and data into a destination database.

Choose one of the following two options for the next steps:

Option 1

1. Export data from the source database into a binary dump file using `exp`.
2. Import the data into the destination database by running `imp` directly from the source server.

Option 2

1. Export data from the source database into a binary dump file using `exp`.
2. Compress and encrypt the files.
3. Launch an Amazon EC2 instance and install the full Oracle client on it (for the `emp/imp` utility). For the database on Amazon EC2, this could be the same instance where the destination database is located. For Amazon RDS, this will be a temporary instance.
4. Transport the files to the Amazon EC2 instance.
5. Decompress and unencrypt the files in the Amazon EC2 instance.
6. Import the data into the destination database by running `imp`.

If your database size is larger than a gigabyte, use Option 2, because it includes compression and encryption. This method will also have better import performance.

For both Option 1 and Option 2, use the following command to import into the destination database:

```
imp userID/password@$service FROMUSER=cust_schema
TOUSER=cust_schema FILE=exp_file.dmp LOG=imp_file.log
```

There are many optional arguments that can be passed to the exp and imp commands based on your needs. For details, see the Oracle documentation.

Migrating data for large Oracle databases

For larger databases, use one of the methods described in this section rather than one of the methods described in Migrating Data for small Oracle Databases. For the purpose of this whitepaper, define a large database as any database 10 GB or more.

This section describes three methods for migrating large databases:

- **Data migration using Oracle Data Pump** – Oracle Data Pump is an excellent tool for migrating large amounts of data, and it can be used with databases on either Amazon RDS or Amazon EC2.
- **Data migration using Oracle external tables** – The process involved in data migration using Oracle external tables is very similar to that of Oracle Data Pump. Use this method if you already have processes built around it; otherwise, it is better to use the Oracle Data Pump method.
- **Data migration using Oracle RMAN** – Migration using RMAN can be useful if you are already backing up the database to AWS, or using the AWS Import/Export service to bring the data to AWS. Oracle RMAN can be used only for databases on Amazon EC2, not Amazon RDS.

Data migration using Oracle Data Pump

When the size of the data to be migrated exceeds 10 GB, Oracle Data Pump is probably the best tool to use for migrating data to AWS. This method allows flexible data- extraction options, a high degree of parallelism, and scalable operations, which enables high-speed movement of data and metadata from one database to another. Oracle Data Pump is introduced with Oracle 10*g* as a replacement for the original Import/Export tools. It is available only on Oracle Database 10*g* Release 1 or later.

You can use the Oracle Data Pump method for both Amazon RDS for Oracle, and Oracle Database running on Amazon EC2. The process involved is similar for both, although Amazon RDS for Oracle requires a few additional steps.

Unlike the original Import/Export utilities, the Oracle Data Pump import requires the data files to be available in the database-server instance to import them into the database.

You cannot access the file system in the Amazon RDS instance directly, so you need to use one or more Amazon EC2 instances (bridge instances) to transfer files from the source to the Amazon RDS instance, and then import that into the Amazon RDS database. You need these temporary

Amazon EC2 bridge instances only for the duration of the import; you can end the instances soon after the import is done. Use Amazon Linux-based instances for this purpose. You do not need an Oracle Database installation for an Amazon EC2 bridge instance; you only need to install the Oracle Instance Client.

> **Note**
>
> To use this method, your Amazon RDS database must be version 11.2.0.3 or later.

The following is the overall process for data migration using Oracle Data Pump for Oracle Database on Oracle for Amazon EC2 and Amazon RDS.

Migrating data to a database in Amazon EC2

1. Use Oracle Data Pump to export data from the source database as multiple compressed and encrypted files.
2. Use Tsunami UDP to move the files to an Amazon EC2 instance running the destination Oracle database in AWS.
3. Import that data into the destination database using the Oracle Data Pump import feature.

Migrating data to a database in Amazon RDS

1. Use Oracle Data Pump to export data from the source database as multiple files.
2. Use Tsunami UDP to move the files to Amazon EC2 bridge instances in AWS.
3. Using the provided Perl script that makes use of the `UTL_FILE` package, move the data files to the Amazon RDS instance.
4. Import the data into the Amazon RDS database using a PL/SQL script that utilizes the `DBMS_DATAPUMP` package (an example is provided at the end of this section).

Using Oracle Data Pump to export data on the source instance

When you export data from a large database, you should run multiple threads in parallel and specify a size for each file. This speeds up the export, and also makes files available quickly for the next step of the process. There is no need to wait for the entire database to be exported before moving to the next step.

As each file completes, it can be moved to the next step. You can enable compression by using the parameter `COMPRESSION=ALL`, which substantially reduces the size of the extract files. You can encrypt files by providing a password, or by using an Oracle wallet and specifying the parameter `ENCRYPTION=all`. To learn more about the compression and encryption options, see the Oracle Data Pump documentation.

The following example shows the export of a 500 GB database, running eight threads in parallel, with each output file up to a maximum of 20 GB. This creates 22 files totaling 175 GB. The total file size is significantly smaller than the actual source database size because of the compression option of Oracle Data Pump:

```
expdp demoreinv/demo full=y dumpfile=data_pump_exp1:reinvexp1%U.dmp,
 data_pump_exp2:reinvexp2%U.dmp,
data_pump_exp3:reinvexp3%U.dmp filesize=20G parallel=8
 logfile=data_pump_exp1:reinvexpdp.log compression=all
ENCRYPTION= all ENCRYPTION_PASSWORD=encryption_password job_name=reInvExp
```

```
$ ./exp.sh
Thu Nov  7 15:18:51 EST 2013

Export: Release 11.2.0.1.0 - Production on Thu Nov 7 15:18:51 2013

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Connected to: Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
Starting "DEMOREINV"."REINVEXP":  DEMOREINV/******** FULL=y DUMPFILE=DATA_PUMP_EXP1:reinvexp1%U.dmp, DATA_PUMP_EXP2:reinvexp2%U.dmp
, DATA_PUMP_EXP3:reinvexp3%U.dmp FILESIZE=20G PARALLEL=8 logfile=DATA_PUMP_EXP1:reinvexpdp.log compression=all JOB_NAME=reInvExp
Estimate in progress using BLOCKS method...
Processing object type DATABASE_EXPORT/SCHEMA/TABLE/TABLE_DATA
Total estimation using BLOCKS method: 481.3 GB
Processing object type DATABASE_EXPORT/TABLESPACE
Processing object type DATABASE_EXPORT/PROFILE
Processing object type DATABASE_EXPORT/SYS_USER/USER
Processing object type DATABASE_EXPORT/SCHEMA/USER
Processing object type DATABASE_EXPORT/ROLE
Processing object type DATABASE_EXPORT/GRANT/SYSTEM_GRANT/PROC_SYSTEM_GRANT
Processing object type DATABASE_EXPORT/SCHEMA/GRANT/SYSTEM_GRANT
Processing object type DATABASE_EXPORT/SCHEMA/ROLE_GRANT
Processing object type DATABASE_EXPORT/SCHEMA/DEFAULT_ROLE
Processing object type DATABASE_EXPORT/SCHEMA/TABLESPACE_QUOTA
Processing object type DATABASE_EXPORT/RESOURCE_COST
```

Using Oracle Data Pump to export data from the source database instance

Spreading the output files across different disks enhances input/output (I/O) performance. In the following examples, three different disks are used to avoid I/O contention.

```
Master table "DEMOREINV"."REINVEXP" successfully loaded/unloaded
******************************************************************************
Dump file set for DEMOREINV.REINVEXP is:
  /mnt/expdisk1/reinvexp101.dmp
  /mnt/expdisk2/reinvexp201.dmp
  /mnt/expdisk3/reinvexp301.dmp
  /mnt/expdisk1/reinvexp102.dmp
  /mnt/expdisk2/reinvexp202.dmp
  /mnt/expdisk3/reinvexp302.dmp
  /mnt/expdisk1/reinvexp103.dmp
  /mnt/expdisk2/reinvexp203.dmp
  /mnt/expdisk3/reinvexp303.dmp
  /mnt/expdisk1/reinvexp104.dmp
  /mnt/expdisk2/reinvexp204.dmp
  /mnt/expdisk3/reinvexp304.dmp
  /mnt/expdisk1/reinvexp105.dmp
  /mnt/expdisk2/reinvexp205.dmp
  /mnt/expdisk3/reinvexp305.dmp
  /mnt/expdisk1/reinvexp106.dmp
  /mnt/expdisk2/reinvexp206.dmp
  /mnt/expdisk3/reinvexp306.dmp
Job "DEMOREINV"."REINVEXP" successfully completed at 17:36:18
```

Parallel run in multiple threads writing to three different disks

```
$ ls -lhr exp*
expdisk3:
total 57G
-rw-r----- 1 oracle oinstall 3.7M Nov  7 17:36 reinvexp306.dmp
-rw-r----- 1 oracle oinstall 4.5M Nov  7 17:36 reinvexp305.dmp
-rw-r----- 1 oracle oinstall  11G Nov  7 17:35 reinvexp304.dmp
-rw-r----- 1 oracle oinstall  11G Nov  7 17:36 reinvexp303.dmp
-rw-r----- 1 oracle oinstall  16G Nov  7 17:36 reinvexp302.dmp
-rw-r----- 1 oracle oinstall  20G Nov  7 16:12 reinvexp301.dmp
drwx------ 2 oracle oinstall  16K Nov  7 09:41 lost+found

expdisk2:
total 62G
-rw-r----- 1 oracle oinstall 7.3M Nov  7 17:36 reinvexp206.dmp
-rw-r----- 1 oracle oinstall 5.6M Nov  7 17:36 reinvexp205.dmp
-rw-r----- 1 oracle oinstall  11G Nov  7 17:35 reinvexp204.dmp
-rw-r----- 1 oracle oinstall  12G Nov  7 17:36 reinvexp203.dmp
-rw-r----- 1 oracle oinstall  20G Nov  7 16:33 reinvexp202.dmp
-rw-r----- 1 oracle oinstall  20G Nov  7 16:33 reinvexp201.dmp
drwx------ 2 oracle oinstall  16K Nov  7 02:49 lost+found

expdisk1:
total 56G
-rw-r--r-- 1 oracle oinstall 116K Nov  7 17:36 reinvexpdp.log
-rw-r----- 1 oracle oinstall 4.4M Nov  7 17:36 reinvexp106.dmp
-rw-r----- 1 oracle oinstall 9.0G Nov  7 17:36 reinvexp105.dmp
-rw-r----- 1 oracle oinstall  13G Nov  7 17:36 reinvexp104.dmp
-rw-r----- 1 oracle oinstall  14G Nov  7 17:36 reinvexp103.dmp
-rw-r----- 1 oracle oinstall  20G Nov  7 16:12 reinvexp102.dmp
-rw-r----- 1 oracle oinstall 144M Nov  7 17:36 reinvexp101.dmp
drwx------ 2 oracle oinstall  16K Nov  7 02:33 lost+found
```

Dump files generated in each disk

The most time-consuming part of this entire process is the file transportation to AWS, so optimizing the file transport significantly reduces the time required for the data migration. The following steps show how to optimize the file transport:

1. Compress the dump files during the export.
2. Serialize the file transport in parallel. Serialization here means sending the files one after the other; you don't need to wait for the export to finish before uploading the files to AWS. Uploading many of these files in parallel (if enough bandwidth is available) further improves the performance. We recommend that you parallel upload as many files as there are disks being used, and use the same number of Amazon EC2 bridge instances to receive those files in AWS.
3. Use Tsunami UDP or a commercial wide area network (WAN) accelerator to upload the data files to the Amazon EC2 instances.

Using Tsunami to upload files to Amazon EC2

The following example shows how to install Tsunami on both the source database server and the Amazon EC2 instance:

```
yum -y install make

yum -y install automake
 yum -y install gcc

yum -y install autoconf
yum -y install cvs

 wget http://sourceforge.net/projects/tsunami-udp/files/latest/download

 tar -xzf tsunami*gz

 cd tsunami-udp*

./recompile.sh
make install
```

After you've installed Tsunami, open port 46224 to enable Tsunami communication. On the source database server, start a Tsunami server, as shown in the following example. If you do parallel upload, then you need to start multiple Tsunami servers:

```
cd/mnt/expdisk1
 tsunamid *
```

On the destination Amazon EC2 instances, start a Tsunami server, as shown in the following example.

If you do multiple parallel file uploads, then you need to start a Tsunami server on each Amazon EC2 bridge instance.

If you do not use parallel file uploads, and if the migration is to an Oracle database on Amazon EC2 (not Amazon RDS), then you can avoid the Amazon EC2 bridge instance. Instead, you can upload the files directly to the Amazon EC2 instance where the database is running. If the destination database is Amazon RDS for Oracle, then the bridge instances are necessary because a Tsunami server cannot be run on the Amazon RDS server:

```
cd /mnt/data_files
tsunami
tsunami> connect source.db.server
 tsunami> get *
```

From this point forward, the process differs for a database on Amazon EC2 versus a database on Amazon RDS. The following sections show the processes for each service.

Next steps for a database on an Amazon EC2 instance

If you used one or more Amazon EC2 bridge instances in the preceding steps, then bring all the dump files from the Amazon EC2 bridge instances into the Amazon EC2 database instance. The easiest way to do this is to detach the Amazon Elastic Block Store (Amazon EBS) volumes that contain the files from the Amazon EC2 bridge instances, and connect them to the Amazon EC2 database instance.

Once all the dump files are available in the Amazon EC2 database instance, use the Oracle Data Pump import feature to get the data into the destination Oracle database on Amazon EC2, as shown in the following example:

```
impdp demoreinv/demo full=y DIRECTORY=DPUMP_DIR
```

```
dumpfile= reinvexp1%U.dmp,reinvexp2%U.dmp, reinvexp3%U.dmp
parallel=8 logfile=DPimp.log ENCRYPTION_PASSWORD=encryption_password
job_name=DPImp
```

This imports all data into the database. Check the log file to make sure everything went well, and validate the data to confirm that all the data was migrated as expected.

Next steps for a database on Amazon RDS

Because Amazon RDS is a managed service, the Amazon RDS instance does not provide access to the file system. However, an Oracle RDS instance has an externally accessible Oracle directory object named DATA_PUMP_DIR. You can copy Oracle Data Pump dump files to this directory by using an Oracle UTL_FILE package.

Amazon. RDS supports Amazon S3 integration as well. You could transfer files between the S3 bucket and Amazon RDS instance through S3 integration of RDS. The S3 integration option is recommended when you want to transfer moderately large files to the RDS instance dba_directories. Alternatively, you can use a Perl script to move the files from the bridge instances to the DATA_PUMP_DIR of the Amazon RDS instance.

Preparing a bridge Instance

To prepare a bridge instance, make sure that Oracle Instant Client, Perl DBI, and Oracle DBD modules are installed so that Perl can connect to the database. You can use the following commands to verify if the modules are installed:

```
$perl -e 'use DBI; print $DBI::VERSION,"\n";'

$perl -e 'use DBD::Oracle; print
$DBD::Oracle::VERSION,"\n";'
```

If the modules are not already installed, use the following process below to install them before proceeding further:

1. Download Oracle Database Instant Client from the Oracle website and unzip it into ORACLE_HOME.
2. Set up the environment variable, as shown in the following example:

```
$ export ORACLE_BASE=$HOME/oracle
```

```
$ export ORACLE_HOME=$ORACLE_BASE/instantclient_11_2

$ export PATH=$ORACLE_HOME:$PATH

$ export TNS_ADMIN=$HOME/etc

$ export LD_LIBRARY_PATH=$ORACLE_HOME:$LD_LIBRARY_PATH
```

3. Download and unzip `DBD::Oracle`, as shown in the following example:

```
$ wget http://www.cpan.org/authors/id/P/PY/PYTHIAN/DBD-Oracle-1.74.tar.gz

$ tar xzf DBD-Oracle-1.74.tar.gz

$ cd DBD-Oracle-1.74
```

4. Install `DBD::Oracle`, as shown in the following example:

```
$ mkdir $ORACLE_HOME/log

$ perl Makefile.PL

$ make

$ make install
```

Transferring files to an Amazon RDS instance

To transfer files to an Amazon RDS instance, you need an Amazon RDS instance with at least twice as much storage as the actual database, because it needs to have space for the database and the Oracle Data Pump dump files. After the import is successfully completed, you can delete the dump files so that space can be utilized.

It might be a better approach to use an Amazon RDS instance solely for data migration. Once the data is fully imported, take a snapshot of RDS DB. Create a new Amazon RDS instance using the snapshot and then decommission the data migration instance. Use a single Availability Zone instance for data migration.

The following example shows a basic Perl script to transfer files to an Amazon RDS instance. Make changes as necessary. Because this script runs in a single thread, it uses only a small portion of

the network bandwidth. You can run multiple instances of the script in parallel for a quicker file transfer to the Amazon RDS instance, but make sure to load only one file per process so that there won't be any overwriting and data corruption. If you have used multiple bridge instances, you can run this script from all of the bridge instances in parallel, thereby expediting file transfer into the Amazon RDS instance:

```
# RDS instance info

my $RDS_PORT=4080;
my $RDS_HOST="myrdshost.xxx.us-east-1-devo.rds-
dev.amazonaws.com";

my $RDS_LOGIN="orauser/orapwd";
my $RDS_SID="myoradb";

my $dirname = "DATA_PUMP_DIR";
my $fname= $ARGV[0];

my $data = ‘‘dummy’’;
my $chunk = 8192;

my $sql_open   = "BEGIN perl_global.fh :=
utl_file.fopen(:dirname, :fname, 'wb', :chunk); END;";
my $sql_write = "BEGIN utl_file.put_raw(perl_global.fh,

:data, true); END;";

my $sql_close = "BEGIN utl_file.fclose(perl_global.fh);
END;";

my $sql_global = "create or replace package perl_global as
fh utl_file.file_type; end;";

my $conn =
DBI-

>connect('dbi:Oracle:host='.$RDS_HOST.';sid='.$RDS_SID.';por t='.$RDS_PORT,$RDS_LOGIN,
 '') || die ( $DBI::errstr . "\n")

;

my $updated=$conn->do($sql_global);
```

```
my $stmt = $conn->prepare ($sql_open);

$stmt->bind_param_inout(":dirname", \$dirname, 12);

$stmt->bind_param_inout(":fname", \$fname, 12);

$stmt->bind_param_inout(":chunk", \$chunk, 4);

$stmt->execute() || die ( $DBI::errstr . "\n");

open (INF, $fname) || die "\nCan't open $fname for reading:

$!\n";
binmode(INF);

$stmt = $conn->prepare ($sql_write);
my %attrib = ('ora_type’,’24’);

my $val=1;

while ($val > 0) {

   $val = read (INF, $data, $chunk);

   $stmt->bind_param(":data", $data , \%attrib);

   $stmt->execute() || die ( $DBI::errstr . "\n"); };
die "Problem copying: $!\n" if $!;

close INF || die "Can't close $fname: $!\n";

$stmt = $conn->prepare ($sql_close);

$stmt->execute() || die ( $DBI::errstr . "\n");
```

You can check the list of files in the DBMS_DATAPUMP directory using the following query:

```
SELECT * from
table(RDSADMIN.RDS_FILE_UTIL.LISTDIR('DATA_PUMP_DIR'));
```

Once all files are successfully transferred to the Amazon RDS instance, connect to the Amazon RDS database as a database administrator (DBA) user and submit a job by using a PL/SQL script that

uses DBMS_DATAPUMP to import the files into the database, as shown in the following PL/SQL script. Make any changes as necessary:

```
Declare

   h1   NUMBER;

begin

h1 := dbms_datapump.open (operation => 'IMPORT', job_mode

=> 'FULL', job_name => 'REINVIMP', version => 'COMPATIBLE');

   dbms_datapump.set_parallel(handle => h1, degree => 8);
 dbms_datapump.add_file(handle => h1, filename =>

'IMPORT.LOG', directory => 'DATA_PUMP_DIR', filetype => 3);
 dbms_datapump.set_parameter(handle => h1, name =>

'KEEP_MASTER', value => 0);

   dbms_datapump.add_file(handle => h1, filename =>
'reinvexp1%U.dmp', directory => 'DATA_PUMP_DIR', filetype =>
1);

   dbms_datapump.add_file(handle => h1, filename =>
'reinvexp2%U.dmp', directory => 'DATA_PUMP_DIR', filetype =>
1);

   dbms_datapump.add_file(handle => h1, filename =>
'reinvexp3%U.dmp', directory => 'DATA_PUMP_DIR', filetype =>
1);

   dbms_datapump.set_parameter(handle => h1, name => 'INCLUDE_METADATA', value => 1);

   dbms_datapump.set_parameter(handle => h1, name => 'DATA_ACCESS_METHOD', value =>
 'AUTOMATIC');

   dbms_datapump.set_parameter(handle => h1, name => 'REUSE_DATAFILES', value => 0);

   dbms_datapump.set_parameter(handle => h1, name => 'SKIP_UNUSABLE_INDEXES', value =>
 0);
```

```
   dbms_datapump.start_job(handle => h1, skip_current => 0,
abort_step => 0);

   dbms_datapump.detach(handle => h1);
end;
/
```

Once the job is complete, check the Amazon RDS database to make sure all the data has been successfully imported. At this point, you can delete all the dump files using `UTL_FILE.FREMOVE` to reclaim disk space.

Data migration using Oracle external tables

Oracle external tables are a feature of Oracle Database that allows you to query data in a flat file as if the file were an Oracle table. The process for using Oracle external tables for data migration to AWS is almost exactly the same as the one used for Oracle Data Pump. The Oracle Data Pump-based method is better for large database migrations.

The external tables method is useful if your current process uses this method and you don't want to switch to the Oracle Data Pump-based method. Following are the main steps:

1. Move the external table files to `RDS DATA_PUMP_DIR`.
2. Create external tables using the files loaded.
3. Import data from the external tables to the database tables.

Depending on the size of the data file, you can choose to either write the file directly to `RDS DATA_PUMP_DIR` from an on-premises server, or use an Amazon EC2 bridge instance, as in the case of the Data Pump-based method. If the file size is large and you choose to use a bridge instance, use compression and encryption on the files as well as Tsunami UDP or a WAN accelerator, exactly as described for the Data Pump-based migration.

To learn more about Oracle external tables, see External Tables Concepts in the Oracle documentation.

Data migration using Oracle RMAN

If you are planning to migrate the entire database and your destination database is self-managed on Amazon EC2, you can use Oracle RMAN to migrate data. Data migration by using Oracle Data

Pump is faster and more flexible than data migration using Oracle RMAN; however, Oracle RMAN is a better option for the following cases:

- You already have an RMAN backup available in Amazon S3 that can be used. If you are looking for options to migrate RMAN backups to Amazon S3, consider AWS Storage Gateway or AWS DataSync services.
- The database is very large (greater than 5 TB), and you are planning to use AWS Import/Export.
- You need to make numerous incremental data changes before switching over to the database on AWS.

> **Note**
>
> This method is for Amazon EC2 and VMware Cloud on AWS. You cannot use this method if your destination database is Amazon RDS.

To migrate data using Oracle RMAN:

1. Create a full backup of the source database using RMAN.
2. Encrypt and compress the files.
3. Transport files to AWS using the most optimal method.
4. Restore the RMAN backup to the destination database.
5. Capture incremental backups from the source, and apply them to the destination database until switchover can be performed.

Creating a full backup of the source database Using RMAN

Create a backup of the source database using RMAN:

```
$ rman target=/

RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON; RMAN> BACKUP DATABASE PLUS ARCHIVELOG
```

If you have a license for the compression and encryption option, then you already have the RMAN backups created as encrypted and compressed files. Otherwise, after the backup files are created,

encrypt and compress them using tools such as ZIP, 7-Zip, or GZIP. All subsequent actions occur on the server running the destination database.

Transporting files to AWS

Depending on the size of the database and the time available for migration, you can choose the most optimal method for file transportation to AWS. For small files, consider AWS DataSync. For moderate to large databases between 100 GB to 5 TB, Tsunami UDP is an option, as described in Using Tsunami to upload files to EC2. You can achieve the same results using commercial third-party WAN acceleration tools. For very large databases over 5 TB, consider using AWS Storage Gateway or AWS Snow Family devices for offline file transfer.

Migrating data to Oracle Database on AWS

There are two ways to migrate data to a destination database. You can create a new database and restore from the RMAN backup, or you can create a duplicate database from the RMAN backup. Creating a duplicate database is easier to perform.

To create a duplicate database, move the transported files to a location accessible to the Oracle Database instance on Amazon EC2. Start the target instance in `NOMOUNT` mode. Now use RMAN to connect to the destination database. For this example, we are not connecting to the source database or the RMAN catalog, so use the following command:

```
$ rman AUXILIARY /

DUPLICATE TARGET DATABASE TO DBONEC2
   SPFILE
   NOFILENAMECHECK;
```

The duration of this process varies based on the size of the database and the type of Amazon EC2 instance. For better performance, use Amazon Elastic Block Store (Amazon EBS) General Purpose (SSD) volumes for the RMAN backup files. For more information about SSD volume types, see Introducing the Amazon EBS General Purpose (SSD) volume type.

Once the process is finished, RMAN produces a completion message, and you now have your duplicate instance. After verification, you can delete the Amazon EBS volumes containing the RMAN backup files. We recommend that you take a snapshot of the volumes for later use before deleting them if needed.

Data replication using AWS Database Migration Service

AWS Database Migration Service (AWS DMS) can support a number of migration and replication strategies including a bulk upload at a point in time, a minimal downtime migration leveraging Change Data Capture (CDC), or migration of only a subset of the data. AWS DMS supports sources and targets in EC2, RDS, and on-premises. Because no client install is required, the following steps are the same for any combination of the above. AWS DMS also offers the ability to migrate data between databases as easily as from Oracle to Oracle.

The following steps show how to migrate data between Oracle databases using AWS DMS and with minimal downtime:

1. Ensure supplemental logging is enabled on the source database.
2. Create the target database and ensure database backups and Multi-AZ are turned off if the target is on RDS.
3. Perform a no-data export of the schema using Oracle SQL Developer or the tool of your choice, then apply the schema to the target database.
4. Disable triggers, foreign keys, and secondary indexes (optional) on the target.
5. Create a DMS replication instance.
6. Specify the source and target endpoints.
7. Create a “Migrate existing data and replicate ongoing changes” task, mapping your source tables to your target tables. (The default task includes all tables.)
8. Start the task.
9. After the full load portion of the tasks is complete and the transactions reach a steady state, enable triggers, foreign keys, and secondary indexes.
10. Turn on backups and Multi-AZ.
11. Turn off any applications using the original source database.
12. Let the final transactions flow through.
13. Point any applications at the new database in AWS and start.

An alternative method is to use Oracle Data Pump for the initial load and DMS to replicate changes from the Oracle System Change Number (SCN) point where data dump stopped. More details on using AWS DMS can be found in the documentation. To improve the performance of DMS

replication, the schemas and tables can be grouped into multiple DMS tasks. DMS tasks support wildcard entries for the names of the schemas and tables.

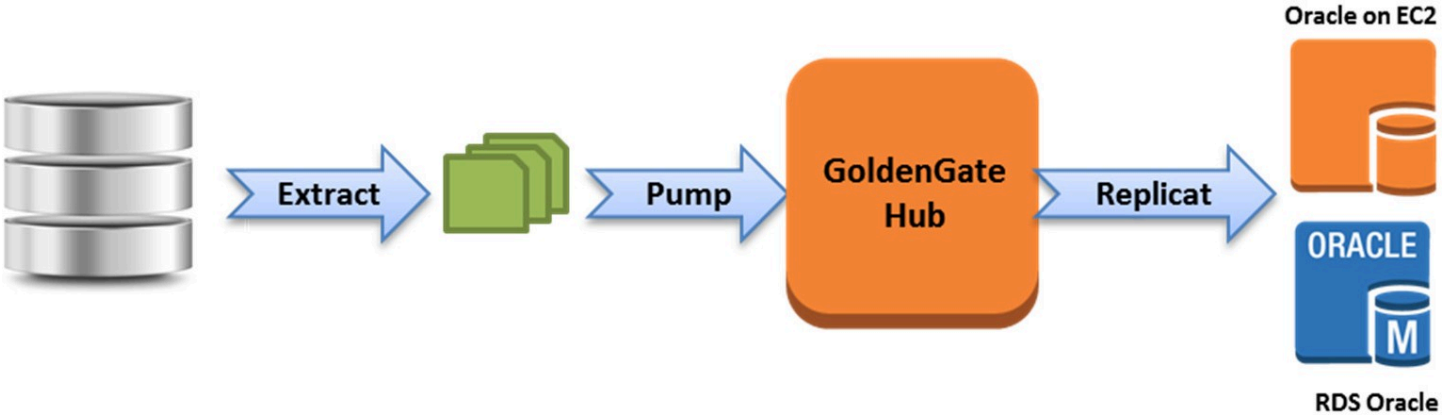
Data replication using Oracle GoldenGate

Oracle GoldenGate is a tool for real-time change data capture and replication. Oracle GoldenGate creates trail files that contain the most recently changed data from the source database, then pushes these files to the destination database. You can use Oracle GoldenGate to perform minimal downtime data migration. Oracle GoldenGate is a licensed software from Oracle. You can also use it for nearly continuous data replication. You can use Oracle GoldenGate with both Amazon RDS for Oracle, and Oracle Database running on Amazon EC2.

The following steps show how to migrate data using Oracle GoldenGate:

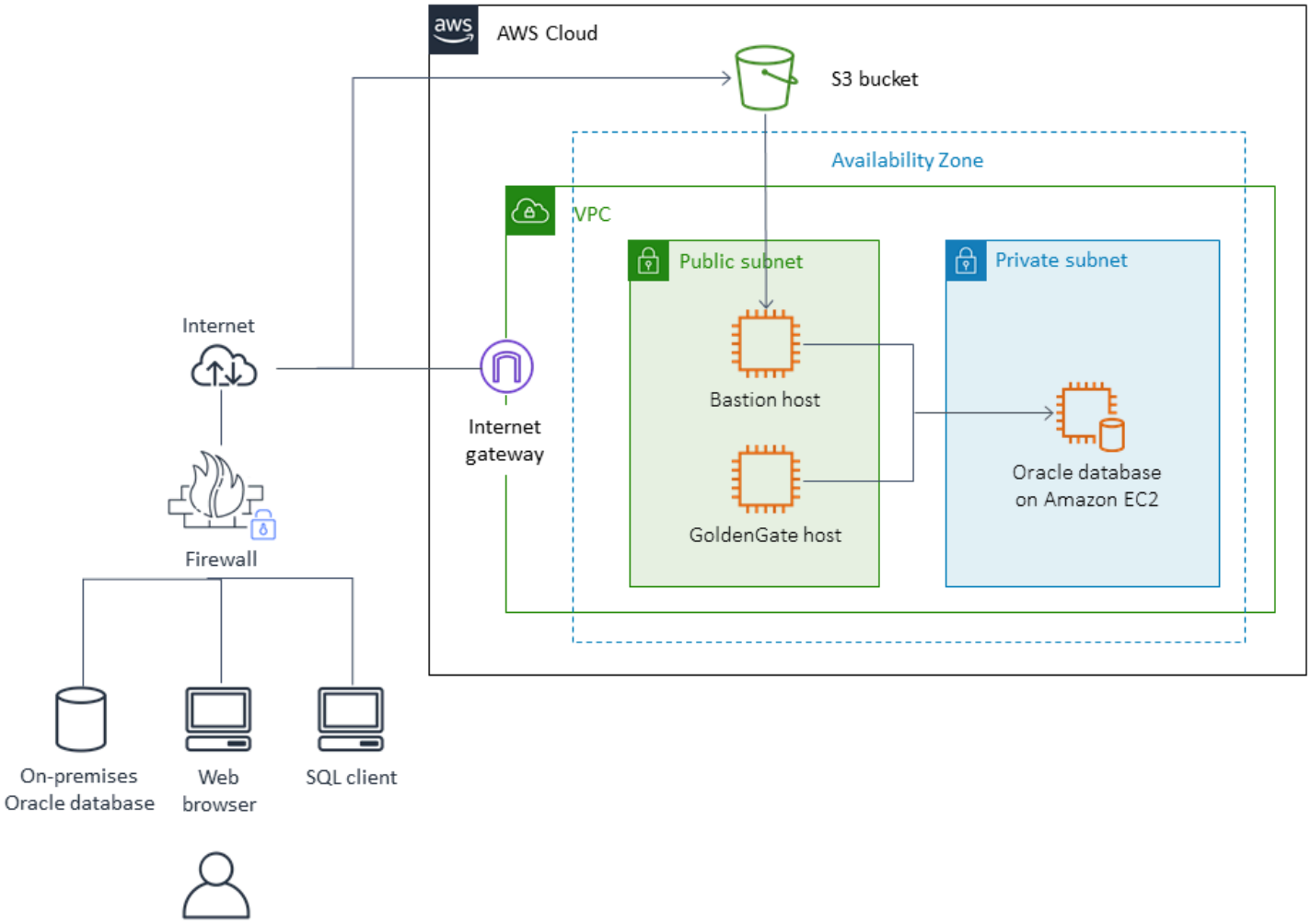
1. The Oracle GoldenGate Extract process extracts all the existing data for the first load. Extract, Pump and Replicat process refers to the GoldenGate Integrated capture mode.
2. The Oracle GoldenGate Pump process transports the extracted data to the Replicat process running in Amazon EC2.
3. The Replicat process applies the data to the destination database.
4. After the first load, the process runs continually to capture changed data and applies it to the destination database.

GoldenGate Replicat is a key part of the entire system. You can run it from a server in the source environment, but AWS recommends that you run the Replicat process in an Amazon EC2 instance within AWS for better performance. This Amazon EC2 instance is referred to as a *GoldenGate Hub*. You can have multiple GoldenGate Hubs, especially if you are migrating data from one source to multiple destinations.



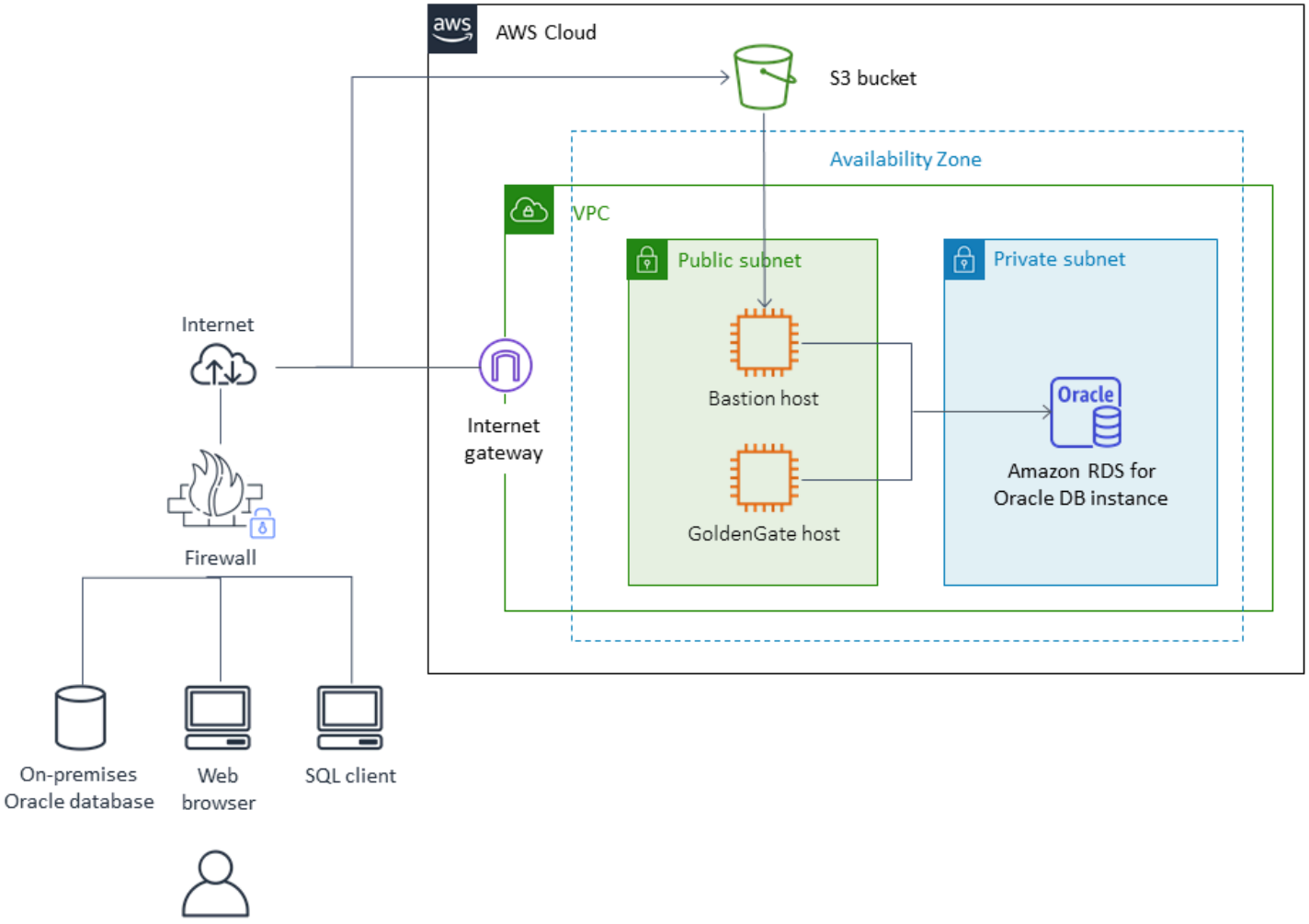
Oracle GoldenGate replication data flow process

Reference architecture for Amazon EC2:



Oracle GoldenGate replication from on-premises to Oracle Database on Amazon EC2

Reference architecture for RDS:



Oracle GoldenGate replication from on-premises to RDS Oracle Database on AWS

Setting up Oracle GoldenGate Hub on Amazon EC2

To create an Oracle GoldenGate Hub on Amazon EC2, create an Amazon EC2 instance with a full client installation of Oracle DBMS 12c version 12.2.0.3 and Oracle GoldenGate 12.3.1.4. Additionally, apply Oracle patch 13328193. For more information about installing GoldenGate, see the Oracle GoldenGate documentation.

This GoldenGate Hub stores and processes all the data from your source database, so make sure that there is enough storage available in this instance to store the trail files. It is a good practice to choose the largest instance type that your GoldenGate license allows. Use appropriate Amazon EBS storage volume types, depending on the database change rate and replication performance.

The following process sets up a GoldenGate Hub on an Amazon EC2 instance.

1. Add the following entry to the `tnsname.ora` file to create an alias. For more information about the `tnsname.ora` file, see the Oracle GoldenGate documentation.

```
$ cat /example/config/tnsnames.ora
TEST=

(DESCRIPTION=
  (ENABLE=BROKEN)
  (ADDRESS_LIST=

    (ADDRESS=(PROTOCOL=TCP)(HOST=ec2-dns)(PORT=8200))

) (
CONNECT_DATA=

    (SID=ORCL)

  )

)
```

2. Next, create subdirectories in the GoldenGate directory by using the Amazon EC2 command line shell and `ggsci`, the GoldenGate command interpreter. The subdirectories are created under the gg directory and include directories for parameter, report, and checkpoint files:

```
prompt$ cd /gg
prompt$ ./ggsci

  GGSCI> CREATE SUBDIRS
```

3. Create a GLOBALS parameter file using the Amazon EC2 command line shell. Parameters that affect all GoldenGate processes are defined in the GLOBALS parameter file. The following example creates the necessary file:

```
prompt$ cd $GGHOME
prompt$ vi GLOBALS

CheckpointTable oggadm1.oggchkpt
```

4. Configure the manager. Add the following lines to the GLOBALS file, and then start the manager by using `ggsci`:

```
PORT 8199
PurgeOldExtracts ./dirdat/*, UseCheckpoints, MINKEEPDAYS
```

Setting up the source database for use with Oracle GoldenGate

To replicate data to the destination database in AWS, you need to set up a source database for GoldenGate. Use the following procedure to set up the source database. This process is the same for both Amazon RDS and Oracle Database on Amazon EC2.

1. Set the compatible parameter to the same as your destination database (for Amazon RDS as the destination).
2. Enable supplemental logging and force logging.
3. Verify the database is in `archivelog` mode.
4. Set `ENABLE_GOLDENGATE_REPLICATION` parameter to `TRUE`.
5. Set the retention period for archived redo logs for the GoldenGate source database.
6. Create a GoldenGate user account on the source database.

Setting up the destination database for use with Oracle GoldenGate

The following steps must be performed on the target database for GoldenGate replication to work. These steps are the same for both Amazon RDS and Oracle Database on Amazon EC2.

1. Create a GoldenGate user account on the destination database.
2. Grant the necessary privileges that are listed in the following example to the GoldenGate user:

```
CREATE SESSION
ALTER SESSION
CREATE CLUSTER
CREATE INDEXTYPE
CREATE OPERATOR
CREATE PROCEDURE
CREATE SEQUENCE
CREATE TABLE
CREATE TRIGGER
CREATE TYPE
SELECT ANY DICTIONARY
CREATE ANY TABLE
ALTER ANY TABLE
LOCK ANY TABLE
SELECT ANY TABLE
INSERT ANY TABLE
UPDATE ANY TABLE
DELETE ANY TABLE
```

Working with the Extract and Replicat utilities of Oracle GoldenGate

The Oracle GoldenGate Extract and Replicat utilities work together to keep the source and destination databases synchronized by means of incremental transaction replication using trail files. All changes that occur on the source database are automatically detected by Extract, and then formatted and transferred to trail files on the GoldenGate Hub on-premises or on the Amazon EC2 instance. After the initial load is completed, the Replicat process reads the data from these files and replicates the data to the destination database nearly continuously.

Running the Extract process of Oracle GoldenGate

The Extract process of Oracle GoldenGate retrieves, converts, and outputs data from the source database to trail files. Extract queues transaction details to memory or to temporary disk storage. When the transaction is committed to the source database, Extract flushes all of the transaction details to a trail file for routing to the GoldenGate Hub on-premises or on the Amazon EC2 instance, and then to the destination database.

The following process enables and starts the Extract process.

1. First, configure the Extract parameter file on the GoldenGate Hub. The following example shows an Extract parameter file:

```
EXTRACT EABC
SETENV (ORACLE_SID=ORCL)
SETENV (NLSLANG=AL32UTF8)

USERID oggadm1@TEST, PASSWORD XXXXXX
EXTTRAIL /path/to/goldengate/dirdat/ab IGNOREREPLICATES
GETAPPLOPS
TRANLOGOPTIONS EXCLUDEUSER OGGADM1 TABLE EXAMPLE.TABLE;
```

2. On the GoldenGate Hub, launch the GoldenGate command line interface (`ggsci`). Log in to the source database. The following example shows the format for logging in:

```
dblogin userid <user>@<db tnsname>
```

3. Next, add a checkpoint table for the database:

```
add checkpointtable

Add transdata to turn on supplemental logging for the database table:

add trandata <user>.<table>
```

Alternatively, you can add `transdata` to turn on supplemental logging for all tables in the database:

```
add trandata <user>.*
```

4. Using the `ggsci` command line, use the following commands to enable the Extract process:

```
add extract <extract name> tranlog, INTEGRATED tranlog,
begin now

add exttrail <path-to-trail-from-the param-file> extract

<extractname-from-paramfile>, MEGABYTES Xm
```

5. Register the Extract process with the database so that the archive logs are not deleted. This lets you recover old, uncommitted transactions if necessary. To register the Extract process with the database, use the following command:

```
register EXTRACT <extract process name>, DATABASE
```

6. To start the Extract process, use the following command:

```
start <extract process name>
```

Running the Replicat process of Oracle GoldenGate

The Replicat process of Oracle GoldenGate is used to push transaction information in the trail files to the destination database.

The following process enables and starts the Replicat process.

1. First, configure the Replicat parameter file on the GoldenGate Hub (on-premises or on an Amazon EC2 instance). The following listing shows an example Replicat parameter file:

```
REPLICAT RABC

SETENV (ORACLE_SID=ORCL)
SETENV (NLSLANG=AL32UTF8)

USERID oggadm1@TARGET, password XXXXXX
ASSUMETARGETDEFS

MAP EXAMPLE.TABLE, TARGET EXAMPLE.TABLE;
```

2. Launch the Oracle GoldenGate command line interface (`ggsci`). Log in to the destination database. The following example shows the format for logging in:

```
dblogin userid <user>@<db tnsname>
```

3. Using the `ggsci` command line, add a checkpoint table. Note that *user* indicates the Oracle GoldenGate user account, not the owner of the destination table schema. The following example creates a checkpoint table named `gg_checkpoint`:

```
add checkpointtable <user>.gg_checkpoint
```

4. To enable the Replicat process, use the following command:

```
add replicat <replicat name> EXTTRAIL <extract trail file>
CHECKPOINTTABLE <user>.gg_checkpoint
```

5. To start the Replicat process, use the following command:

```
start <replicat name>
```

Transferring files to AWS

Migrating databases to AWS requires the transfer of files to AWS. There are multiple methods of transferring files to AWS. This section describes the methods you can adopt during the migration process.

AWS DataSync

AWS DataSync is an online data transfer service that can accelerate moving data between an on-premises storage system and AWS storage services such as Amazon S3, Amazon EFS, or FSx for Windows File Server. AWS DataSync agent connects to the on-premises storage and copies data and metadata securely to AWS. AWS DataSync is the recommended option when you have large volume of small files 100 MB or less.

AWS Storage Gateway

AWS Storage Gateway is a service connecting an on-premises software appliance with cloud-based storage to provide seamless and secure integration between an organization's on-premises IT environment and the AWS storage infrastructure. The service allows you to securely store data in the AWS Cloud for scalable and cost-effective storage. AWS Storage Gateway supports open standard storage protocols that work with your existing applications. It provides low-latency performance by maintaining frequently accessed data on-premises while securely storing all of your data encrypted in Amazon S3 or Amazon S3 Glacier. AWS Storage Gateway works with moderate or large file sizes.

AWS Storage Gateway S3 File Gateway interface provides a Network File System/Server Message Block (NFS/SMB) file share in your on-premises environment. They run a local VM in your on-premises data center. Files can be copied at the on-premises location to this local file-share. These files are copied to the designated S3 bucket in AWS. If your workload uses Windows OS, you can use Amazon FSx File Gateway to copy files from on-premises via SMB clients to the Amazon FSx for Windows File Server.

Amazon RDS integration with S3

You can use S3 integration to transfer files between an Amazon S3 bucket and an Amazon RDS instance. The Amazon RDS instance accesses S3 bucket via a defined IAM role, so you can have

granular bucket or object level policies for the Amazon RDS instance. Amazon S3 integration is useful when you have to use Oracle utilities like `utl_file` or `datapump`. Amazon RDS Oracle `rdsadmin` package supports both upload and download from S3 buckets.

Tsunami UDP

Tsunami UDP is an open-source, file transfer protocol that uses TCP control and UDP data for transfer over long-distance networks at a very fast rate. When you use UDP for transfer, you gain more throughput than is possible with TCP over the same networks.

You can download Tsunami UDP from the Tsunami UDP Protocol page at SourceForge.net. For moderate to large databases between 100 GB to 5 TB, Tsunami UDP is an option, as described in Using Tsunami to Upload Files to EC2. You can achieve the same results using commercial third-party WAN acceleration tools. For very large databases over 5 TB, using AWS Snow Family devices might be a better option. For smaller databases, you can also use the Amazon S3 multipart upload capability to keep it simple and efficient.

AWS Snow Family

AWS Snow Family offers a number of physical devices and capacity points transport up to exabytes of data into and out of AWS. Snow Family devices are owned and managed by AWS and integrate with AWS security, monitoring, storage management, and computing capabilities. For example, AWS Snowball Edge has 80 TB of usable capacity and can be mounted as an NFS mount point in the on-premises location. For smaller capacity, AWS Snowcone offers 8 TB of storage and has the capability to run the AWS DataSync agent.

Conclusion

This whitepaper described the preferred methods for migrating Oracle Database to AWS, for both Amazon EC2 and Amazon RDS. Depending on your business needs and your migration strategy, you will probably use a combination of methods to migrate your database. For best performance during migration, it is critical to choose the appropriate level of resources on AWS, especially for Amazon EC2 instances and Amazon EBS General Purpose (SSD) volume types.

Contributors

Contributors to this document include:

- Jayaraman Vellore Sampathkumar, AWS Solution Architect – Database, Amazon Web Services
- Praveen Katari, AWS Partner Solution Architect, Amazon Web Services

Further reading

For additional information on data migration with AWS services, consult the following resources:

Oracle Database on AWS:

- Advanced Architectures for Oracle Database on Amazon EC2
- Choosing the Operating System for Oracle Workloads on Amazon EC2
- Determining the IOPS Needs for Oracle Database on AWS
- Best Practices for Running Oracle Database on AWS
- AWS Case Study: Amazon.com Oracle DB Backup to Amazon S3

Oracle on AWS

- Oracle and Amazon Web Services
- Amazon RDS for Oracle

AWS Database Migration Service (AWS DMS)

- AWS Database Migration Service

Oracle licensing on AWS

- Licensing Oracle Software in the Cloud Computing Environment

AWS service details

- AWS Architecture Center
- Cloud Products
- AWS Documentation
- AWS Whitepapers & Guides

AWS pricing information

- AWS Pricing

- AWS Pricing Calculator

VMware Cloud on AWS

- VMware Cloud on AWS

Document history

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Whitepaper updated	Updated install code for Tsunami and Oracle Instant Client.	July 29, 2022
Whitepaper updated	Update to text on page 30 for clarity.	January 27, 2022
Whitepaper updated	Revised to add information on Amazon Machine Learning, AWS Lambda, Amazon OpenSearch Service; general update.	October 8, 2021
Whitepaper updated	General updates.	January 1, 2018
Initial publication	Whitepaper first published.	December 1, 2014

Note

To subscribe to RSS updates, you must have an RSS plug-in enabled for the browser that you are using.

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.