Database Migration Guide
Step-by-Step Walkthroughs
# Table of Contents

Database Migration Guide Step-by-Step Walkthroughs ................................................................. 1

Migrating Databases to Amazon Web Services Managed Databases ............................................. 2

- Migrating a MySQL Database to Amazon RDS for MySQL or Amazon Aurora MySQL ............ 2
  - Full Load ................................................................. 3
  - Performance Comparison ............................................. 7
  - AWS DMS Ongoing Replication ..................................... 7
  - Resources ................................................................ 8
- Full Load .................................................................. 10
  - Performance Comparison ............................................. 18
  - Ongoing Replication .................................................. 20
  - Summary .................................................................. 20
  - Resources ................................................................ 20
- Migrating PostgreSQL Databases to Amazon RDS for PostgreSQL or Amazon Aurora PostgreSQL 8
  - Full Load .................................................................. 10
  - Performance Comparison ............................................. 18
  - Ongoing Replication .................................................. 20
  - Summary .................................................................. 20
  - Resources ................................................................ 20
- Migrating an Oracle Database to Amazon RDS for Oracle ......................................................... 21
  - Full Load .................................................................. 23
  - Performance Comparison ............................................. 32
  - Ongoing Replication .................................................. 33
  - Summary .................................................................. 34
  - Resources ................................................................ 34
- Migrating a SQL Server Always On Database to AWS ............................................................. 34
  - Prerequisites ............................................................. 35
  - SQL Server Always On Availability Groups ............................................................. 35

Migrating SQL Server Databases to Amazon RDS for SQL Server ............................................... 38

- Full Load .................................................................. 40
- Performance Comparison ............................................. 47
- Ongoing Replication .................................................. 49
- Summary .................................................................. 49
- Resources ................................................................ 49

Migrating from Amazon RDS for Oracle to Amazon RDS for PostgreSQL and Aurora PostgreSQL 50

- Can My Oracle Database Migrate? ....................................................................................... 50
- Migration Strategies ............................................................................................................... 51
- The 12 Step Migration Process ............................................................................................... 52
- Automation ................................................................. 54
- Future State Architecture Design ........................................................................................... 54
- Database Schema Conversion ................................................................................................. 56
- Application Conversion or Remediation ................................................................................ 57
- Script/ETL/Report Conversion ................................................................................................ 58
- Integration with Third-Party Applications ............................................................................ 59
- Data Migration Mechanism ................................................................................................... 59
- Testing and Bug Fixing ............................................................................................................ 60
- Performance Tuning ................................................................................................................ 62
- Setup, DevOps, Integration, Deployment, and Security .......................................................... 62
- Documentation and Knowledge Transfer ............................................................................. 65
- Project Management and Version Control ............................................................................. 65
- Post-Production Support ....................................................................................................... 66
- Platform Differences ............................................................................................................. 66

Migrating from SAP ASE to Amazon Aurora MySQL ................................................................. 67

- Prerequisites ............................................................. 69
- Preparation and Assessment .................................................. 69
- Database Migration ........................................................................... 71
- Best Practices ........................................................................... 71

Migrating Databases to the Amazon Web Services Cloud Using the Database Migration Service 77

Migrating an On-Premises Oracle Database to Amazon Aurora MySQL ....................................... 78

- Costs ........................................................................... 79
Validate the migration .................................................................................................................. 275
Cut over ........................................................................................................................................ 276
Migrating from MongoDB to Amazon DocumentDB ................................................................. 277
  Launch an Amazon EC2 instance ......................................................................................... 277
  Install and configure MongoDB community edition .......................................................... 278
  Create an AWS DMS replication instance .......................................................................... 279
Create source and target endpoints ....................................................................................... 280
Create and run a migration task .............................................................................................. 282
You can use AWS Database Migration Service (AWS DMS) to migrate your data to and from most widely used commercial and open-source databases such as Oracle, PostgreSQL, Microsoft SQL Server, Amazon Redshift, Amazon Aurora, MariaDB, and MySQL. The service supports homogeneous migrations such as Oracle to Oracle, and also heterogeneous migrations between different database platforms, such as Oracle to MySQL or MySQL to Amazon Aurora MySQL-Compatible Edition. The source or target database must be on an AWS service.

In this guide, you can find step-by-step walkthroughs that go through the process of migrating sample data to AWS:

- Migrating an On-Premises Oracle Database to Amazon Aurora MySQL (p. 78)
- Migrating an Amazon RDS Oracle Database to Amazon Aurora MySQL (p. 97)
- Migrating a SQL Server Database to Amazon Aurora MySQL (p. 131)
- Migrating an Amazon RDS for SQL Server Database to an Amazon S3 Data Lake (p. 148)
- Migrating an Oracle Database to PostgreSQL (p. 168)
- Migrating Oracle Databases to Aurora PostgreSQL with DMS Schema Conversion (p. 186)
- Migrating SQL Server Databases to Amazon RDS for MySQL with DMS Schema Conversion (p. 197)
- Migrating an Amazon RDS for Oracle Database to Amazon Redshift (p. 209)
- Migrating a BigQuery Project to Amazon Redshift (p. 241)
- Migrating MySQL-Compatible Databases (p. 252)
- Migrating a MySQL-Compatible Database to Amazon Aurora MySQL (p. 253)
- Migrating a MariaDB Database to Amazon RDS for MySQL or Amazon Aurora MySQL (p. 269)
- Migrating from MongoDB to Amazon DocumentDB (p. 277)
- Migrating a MySQL Database to Amazon RDS for MySQL or Amazon Aurora MySQL (p. 2)
- Migrating PostgreSQL Databases to Amazon RDS for PostgreSQL or Amazon Aurora PostgreSQL (p. 8)
- Migrating an Oracle Database to Amazon RDS for Oracle (p. 21)
- Migrating a SQL Server Always On Database (p. 34)
- Migrating SQL Server Databases to Amazon RDS for SQL Server (p. 38)
- Migrating from Amazon RDS for Oracle to Amazon RDS for PostgreSQL and Aurora PostgreSQL (p. 50)
- Migrating from SAP ASE to Amazon Aurora MySQL (p. 67)

In the DMS User Guide, you can find additional resources:

- Migrating large data stores
Migrating Databases to Amazon Web Services Managed Databases

You can move from existing, self-managed, open-source, and commercial databases to fully managed AWS databases of the same engine. The following walkthroughs show how to move your databases to Amazon Relational Database Service (Amazon RDS) and Amazon Aurora.

Topics
- Migrating a MySQL Database to Amazon RDS for MySQL or Amazon Aurora MySQL (p. 2)
- Migrating PostgreSQL Databases to Amazon RDS for PostgreSQL or Amazon Aurora PostgreSQL (p. 8)
- Migrating an Oracle Database to Amazon RDS for Oracle (p. 21)
- Migrating a SQL Server Always On Database to AWS (p. 34)
- Migrating SQL Server Databases to Amazon RDS for SQL Server (p. 38)
- Migrating from Amazon RDS for Oracle to Amazon RDS for PostgreSQL and Aurora PostgreSQL (p. 50)
- Migrating from SAP ASE to Amazon Aurora MySQL (p. 67)

Migrating a MySQL Database to Amazon RDS for MySQL or Amazon Aurora MySQL

You can use these two main approaches for migrating a self-managed MySQL database to an Amazon RDS for MySQL or Amazon Aurora MySQL database.

- Use a native or third-party database migration tool such as mysqldump to perform the full load and MySQL replication to perform ongoing replication. Typically this is the simplest option.
- Use a managed migration service such as the AWS Database Migration Service (AWS DMS). AWS DMS provides migration-specific services such as data validation that are not available in the native or third-party tools.

The following diagram displays these two approaches.
You can use a hybrid strategy that combines native or third-party tools for full load and AWS DMS for ongoing replication. The following diagram displays the hybrid migration approach.

The hybrid option delivers the simplicity of the native or third-party tools along with the additional services that AWS DMS provides. For example, in AWS DMS, you can automatically validate your migrated data, row by row and column by column, to ensure the data quality in the target database. Or, if you are only migrating a subset of the tables, it will be simpler to use AWS DMS to filter your tables than the equivalent configuration in the native or third-party tools.

Topics
- Full Load (p. 3)
- Performance Comparison (p. 7)
- AWS DMS Ongoing Replication (p. 7)
- Resources (p. 8)

Full Load

You can use one of these three tools to move data from your MySQL database to Amazon RDS for MySQL or Amazon Aurora MySQL. Follow the steps described in this document to perform the full data load.

mysqldump

This native MySQL client utility installs by default with the engine that performs logical backups, producing a set of SQL statements that you can execute to reproduce the original database object definitions and table data. mysqldump dumps one or more MySQL databases for backup or transfer to another MySQL server. For more information, see the mysqldump documentation.

mysqldump is appropriate when the following conditions are met:
- The data set is smaller than 10 GB.
- The network connection between source and target databases is fast and stable.
- Migration time is not critical, and the cost of re-trying the migration is very low.
- You don’t need to do any intermediate schema or data transformations.

You can decide not to use this tool if any of the following conditions are true:
- You migrate from an Amazon RDS for MySQL DB instance or a self-managed MySQL 5.5 or 5.6 database. In that case, you can get better performance results with Percona XtraBackup.
- It is impossible to establish a network connection from a single client instance to source and target databases due to network architecture or security considerations.
- The network connection between the source and target databases is unstable or very slow.
- The data set is larger than 10 GB.
- An intermediate dump file is required to perform schema or data manipulations before you can import the schema or data.

For details and step-by-step instructions, see Importing data to an Amazon RDS for MySQL or MariaDB DB instance with reduced downtime in the Amazon RDS User Guide.

Follow these three steps to perform full data load using mysqldump.

1. Produce a dump file containing source data.
2. Restore this dump file on the target database.
3. Retrieve the binary log position for ongoing replication.

For example, the following command creates the dump file. The --master-data=2 parameter creates a backup file, which you can use to start the replication in AWS DMS.

```
sudo mysqldump  
   --databases <database_name> 
   --master-data=2 
   --single-transaction 
   --order-by-primary 
   -r <backup_file>.sql 
   -u local_user 
   -p <local_password>
```

For example, the following command restores the dump file on the target host.

```
mysql -h host_name -P 3306 -u db_master_user -p < backup_file.sql
```

For example, the following command retrieves the binary log file name and position from the dump file. Save this information for later when you configure AWS DMS for ongoing replication.

```
head mysqldump.sql -n80 | grep "MASTER_LOG_POS"
-- Will Get output similar to
-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000125', MASTER_LOG_POS=150;
```

**Percona XtraBackup**

Amazon RDS for MySQL and Amazon Aurora MySQL support migration from Percona XtraBackup files that are stored in an Amazon S3 bucket. Percona XtraBackup produces a binary backup files which can be significantly faster than migrating from logical schema and data dumps using tools such as mysqldump. The tool can be used for small-scale to large-scale migrations.

Percona XtraBackup is appropriate when the following conditions are met:

- You have administrative, system-level access to the source database.
- You migrate database servers in a 1-to-1 fashion: one source MySQL server becomes one new Amazon RDS for MySQL or Aurora DB cluster.

You can decide not to use this tool if any of the following conditions are true:

- You can't use third-party software because of operating system limitations.
• You migrate into existing Aurora DB clusters.
• You migrate multiple source MySQL servers into a single Aurora DB cluster.
• For more information, see Limitations and recommendations for importing backup files from Amazon S3 to Amazon RDS.

For details and step-by-step instructions, see Migrating data from MySQL by using an Amazon S3 Bucket in the Amazon RDS User Guide.

Follow these three steps to perform full data load using Percona XtraBackup.

1. Produce a backup file containing source data.
2. Restore this backup file from Amazon S3 while launching a new target database.
3. Retrieve the binary log position for ongoing replication.

For example, the following command creates the backup file and streams it directly to Amazon S3.

```bash
xtrabackup --user=<myuser> --backup --parallel=4 --stream=xbstream --compress | aws s3 cp - s3://<bucket_name>/<backup_file>.xbstream
```

Use the Amazon RDS console to restore the backup files from the Amazon S3 bucket and create a new Amazon Aurora MySQL DB cluster. For more information, see Restoring an Amazon Aurora MySQL DB cluster from an Amazon S3 bucket.

For example, the following command prints the binary log (binlog) information after you finish the creation of a compressed backup.

```
MySQL binlog position: filename 'mysql-bin.000001', position '481'
```

For example, the following command retrieves the binary log file name and position from the from the xtrabackup_binlog_info file. This file is located in the main backup directory of an uncompressed backup.

```
$ cat </on-premises/backup>/xtrabackup_binlog_info
// Output
mysql-bin.000001 481
```

**mydumper**

mydumper and myloader are third-party utilities that perform a multithreaded schema and data migration without the need to manually invoke any SQL commands or design custom migration scripts. mydumper functions similarly to mysqldump, but offers many improvements such as parallel backups, consistent reads, and built-in compression. Another benefit to mydumper is that each individual table gets dumped into a separate file. The tools are highly flexible and have reasonable configuration defaults. You can adjust the default configuration to satisfy the requirements of both small-scale and large-scale migrations.

mydumper is appropriate when the following conditions are met:

• Migration time is critical.
• You can’t use Percona XtraBackup.

You can decide not to use this tool if any of the following conditions are true:
You migrate from an Amazon RDS for MySQL DB instance or a self-managed MySQL 5.5 or 5.6 database. In that case, you might get better results Percona XtraBackup.

You can’t use third-party software because of operating system limitations.

Your data transformation processes require intermediate dump files in a flat-file format and not an SQL format.

For details and step-by-step instructions, see the mydumper project.

Follow these three steps to perform full data load using mydumper.

1. Produce a dump file containing source data.
2. Restore this dump file on the target database using myloader.
3. Retrieve the binary log position for ongoing replication.

For example, the following command creates the backup of DbName1 and DbName2 databases using mydumper.

```
mydumper
--host=<db-server-address>
--user=<mydumper-username> --password=<mydumper-password>
--outputdir=/db-dump/mydumper-files/
-G -E -R --compress --build-empty-files
--threads=4 --compress-protocol
--regex '^(DbName1\.|DbName2\.)' 
-L /<mydumper-logs-dir>/mydumper-logs.txt
```

For example, the following command restores the backup to the Amazon RDS instance using myloader.

```
myloader
--host=<rds-instance-endpoint>
--user=<db-username> --password=<db-password>
--directory=<mydumper-output-dir>
--queries-per-transaction=50000 --threads=4 
--compress-protocol --verbose=3 -e 2><myload-output-logs-path>
```

For example, the following command retrieves the binary log information from the mydumper metadata file.

```
cat <mydumper-output-dir>/metadata
# It should display data similar to the following:
SHOW MASTER STATUS:SHOW MASTER STATUS:
  Log: mysql-bin.000129
  Pos: 150
  GTID:
```

**Note**

1. To ensure a valid dump file of logical backups in mysqlldump and mydumper, don’t run data definition language (DDL) statements while the dump process is running. It is recommended to schedule a maintenance window for these operations. For details, see the single-transaction documentation.

2. While exporting the data with logical backups, it is recommended to exclude MySQL default schemas (mysql, performance_schema, and information_schema), functions, stored procedures, and triggers.

3. Remove definers from schema files before uploading extracted data to Amazon RDS. For more information, see How can I resolve definer errors.
4. Any backup operation acquires a global read lock on all tables (using FLUSH TABLES WITH READ LOCK). As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. For more information, see Establishing a Backup Policy. For logical backups this step done at the beginning of the logical dump, however for physical backup (Percona XtraBackup) this step done at the end of backup.

**Performance Comparison**

We tested these three full load options using a Mysql 5.7 database on EC2 as the source and Aurora MySQL 5.7 as the target. The source database contained the AWS DMS sample database with a total of 9 GB of data. The following image shows the performance results.

![Performance Chart](image)

Percona XtraBackup performed 4x faster than mysqldump and 2x faster than mydumper backups. We tested larger datasets, for example with a total of 400 GB of data, and found that the performance scaled proportionally to the dataset size.

Percona XtraBackup creates a physical backup of the database files whereas the other tools create logical backups. Percona XtraBackup is the best option for full load if your use case conforms to the restrictions listed in the Percona XtraBackup section above. If Percona XtraBackup isn’t compatible with your use case, mydumper is the next best option. For more information about physical and logical backups, see Backup and Recovery Types.

**AWS DMS Ongoing Replication**

To configure the ongoing replication in AWS DMS, enter the native start point for MySQL, which you have retrieved at the end of the full load process as described for each tool. The native start point will be similar to mysql-bin-changelog.000024:373.

In the **Create database migration task** page, follow these three steps to create the migration task.

1. For **Migration type**, choose **Replicate ongoing changes**.
2. Under **CDC start mode for source transactions**, choose **Enable custom CDC start mode**.
3. Under **Custom CDC start point**, paste the native start point you saved earlier.

For more information, see Creating tasks for ongoing replication and Migrate from MySQL to Amazon RDS.

**Note**

The AWS DMS CDC replication uses plain SQL statements from the binary log to apply data changes in the target database. Therefore, it is slower and more resource-intensive than the native Primary/Replica binary log replication in MySQL. For more information, see Replication with a MySQL or MariaDB instance running external to Amazon RDS.

You should always remove triggers from the target during the AWS DMS CDC replication. For example, the following command generates the script to remove triggers.
# In case required to generate drop triggers script

```
SELECT Concat('DROP TRIGGER ', Trigger_Name, ';') FROM information_schema.TRIGGERS WHERE TRIGGER_SCHEMA not in ('sys','mysql');
```

## Resources

For more information, see the following references:

- Amazon Aurora Migration Handbook
- What is Database Migration Service?
- Best practices for Database Migration Service
- Using a MySQL-compatible database as a source
- Using a MySQL-compatible database as a target

## Migrating PostgreSQL Databases to Amazon RDS for PostgreSQL or Amazon Aurora PostgreSQL

This walkthrough gets you started with homogeneous database migration from PostgreSQL to Amazon Relational Database Service (Amazon RDS) for PostgreSQL or Amazon Aurora PostgreSQL-Compatible Edition. This guide provides a quick overview of the data migration process and provides suggestions on how to select the best option to use.

Customers looking to migrate self-managed PostgreSQL databases to Amazon RDS for PostgreSQL or Aurora PostgreSQL, can use one of the three main approaches.

- Use a native or third-party database migration method such as pg_dump and pg_restore for full load only migrations.
- Use a managed service such as AWS Database Migration Service (AWS DMS) for full load and ongoing replication.
- Use a native tool for full load and a managed AWS DMS service for ongoing replication. We call this strategy the **hybrid approach**.

This document describes the hybrid approach. The following diagram shows the components of the hybrid approach.
The hybrid approach provides the following advantages.

- Automation of the creation of secondary database objects such as views, indexes, and constraints.
- AWS DMS data validation to ensure that your target data matches with the source, row by row and column by column.
- Other capabilities provided by AWS DMS, for example CloudWatch monitoring and table statistics. It may be simpler to use AWS DMS to track migration progress, transactional workload, receive and transmit throughput, source and target latency, and so on.

This document describes the native options for the full load. It also includes a comparison so that you can evaluate the options for your migration requirements. In conclusion, you can find a brief description of how to use AWS DMS for ongoing replication.

Topics
- Full Load (p. 10)
- Performance Comparison (p. 18)
- Ongoing Replication (p. 20)
- Summary (p. 20)
- Resources (p. 20)
Full Load

The full load migration phase populates the target database with a copy of the source data. This chapter describes the following native methods to help you choose the one that best matches your migration scenario.

- pg_dump and pg_restore
- Publisher and Subscriber
- pglogical

We recommend that you begin by reviewing the following table to understand the tools suitable for your use case.

<table>
<thead>
<tr>
<th>Method</th>
<th>Supported versions</th>
<th>Support of metadata migration</th>
<th>Suitable database sizes</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_dump and pg_restore</td>
<td>All versions of PostgreSQL</td>
<td>Yes</td>
<td>100 GB or less</td>
<td>Medium</td>
</tr>
<tr>
<td>Publisher and Subscriber</td>
<td>PostgreSQL 10.0 and higher</td>
<td>No</td>
<td>Any size</td>
<td>High</td>
</tr>
<tr>
<td>pglogical</td>
<td>PostgreSQL 9.4 and higher</td>
<td>Yes</td>
<td>Any size</td>
<td>High</td>
</tr>
</tbody>
</table>

The suitable database sizes provided in the preceding table are the AWS DMS recommendations. These recommendations are based on customer migration experiences and aren't the limitation of the native tools.

For more information about the performance of these tools, see Performance Comparison (p. 18).

Topics
- Preparing for Ongoing Replication (p. 10)
- pg_dump and pg_restore (p. 11)
- Publisher and Subscriber (p. 13)
- pglogical (p. 15)

Preparing for Ongoing Replication

Before you start full load, make sure that you record the current log sequence number (LSN) as the starting position for ongoing replication. Use this LSN when you configure the ongoing replication task in AWS DMS.

To avoid data loss or duplication with hybrid approach, make sure of the following:

- You create a replication slot on the source database before you start the full load using the following command:

  ```sql
  SELECT * FROM pg_create_logical_replication_slot('test_slot', 'test_decoding');
  ```

- When you create a replication slot, your source database doesn't have open transactions. To confirm, use the following command:
Database Migration Guide Step-by-Step Walkthroughs
Full Load

SELECT * FROM pg_stat_activity where state <> 'idle';

- If you have open transactions, wait for them to complete or cancel them.

To capture the LSN, use the following command.

SELECT slot_name, confirmed_flush_lsn from pg_replication_slots where slot_name like 'test_slot';

<table>
<thead>
<tr>
<th>slot_name</th>
<th>confirmed_flush_lsn</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_slot</td>
<td>12/68000000</td>
</tr>
</tbody>
</table>

From the output of the preceding command, copy the confirmed_flush_lsn value. In the example preceding, this value is set to 12/68000000. After you complete the full load, you can use this value as the start position for the AWS DMS task.

**pg_dump and pg_restore**

pg_dump and pg_restore is a native PostgreSQL client utility. You can find this utility as part of the database installation. It produces a set of SQL statements that you can run to reproduce the original database object definitions and table data.

The pg_dump and pg_restore utility is suitable for the following use cases if:

- Your database size is less than 100 GB.
- You plan to migrate database metadata as well as table data.
- You have a relatively large number of tables to migrate.

The pg_dump and pg_restore utility may not be suitable for the following use cases if:

- Your database size is greater than 100 GB.
- You want to avoid downtime.

**Example**

At a high level, you can use the following steps to migrate the dms_sample database.

1. Export data to one or more dump files.
2. Create a target database.
3. Import the dump file or files.
4. (Optional) Migrate database roles and users.

**Export Data**

You can use the following command to create dump files for your source database.

```
pg_dump -h <hostname> -p 5432 -U <username> -Fc -b -v -f <dumpfilelocation.sql> -j 2 -d <database_name>
```

-h is the name of source server where you would like to migrate your database.
-U is the name of the user present on the source server
-Fc: Sets the output as a custom-format archive suitable for input into pg_restore.
-b: Include large objects in the dump.
-v: Specifies verbose mode
-f: Dump file path
-j: Number of jobs to dump the data in parallel

Create a Database on Your Target Instance

First, login to your target database server.

```
psql -h <hostname> -p 5432 -U <username> -d <database_name>
```

-h is the name of target server where you would like to migrate your database.
-U is the name of the user present on the target server.
-d is the name of database name present on target already.

Then, use the following command to create a database.

```
create database migrated_database;
```

Import Dump Files

You can use the following command to import the dump file into your Amazon RDS instance.

```
pg_restore -v -h <hostname> -U <username> -d <database_name> -j 2 <dumpfilelocation.sql>
```

-h is the name of target server where you would like to migrate your database.
-U is the name of the user present on the target server.
-d is the name of database name present on target already.

-Migrate Database Roles and Users

To export such database objects as roles and users, you can use the `pg_dumpall` utility.

To generate a script for users and roles, run the following command on the source database.

```
pg_dumpall -U <username> -h <hostname> -f <dumpfilelocation.sql> --no-role-passwords -g
```

-h is the name of source server where you would like to migrate your database.
-U is the name of the user present on the source server.
-f: Dump file path.
-g: Dump only global objects (roles and tablespaces), no databases.

To restore users and roles, run the following command on your target database.

```
psql -h <hostname> -U <username> -f <dumpfilelocation.sql>
```

-h is the name of target server where you would like to migrate your database.
-U is the name of the user present on the target server.
-f: Dump file path.

To complete the export and import operations, the `pg_dump` and `pg_restore` requires some time. This time depends on the following parameters.

- The size of your source database.
- The number of jobs.
- The resources that you provision for your instance used to invoke `pg_dump` and `pg_restore`. 


Publisher and Subscriber

In PostgreSQL, logical replication uses a publisher and subscriber model. In this model, one or more subscribers subscribe to one or more publications on a publisher node. Subscribers pull data from the publications they subscribe to and may subsequently re-publish data to allow cascading replication or more complex configurations. PostgreSQL version 10.0 and higher supports the native publisher and subscriber model.

The publisher and subscriber model is suitable for the following use cases if:

- Your database size is greater than 100 GB.
- You have an existing schema in your target database and migrate only data.
- You want to capture ongoing changes.
- You want to minimize downtime.

The publisher and subscriber may not be suitable for the following use cases if:

- You plan to migrate database metadata.
- Your source PostgreSQL database version is lower than 10.x.
- You want to replicate the schema, DDL, and sequences.

Example

The migration process involves copying a snapshot of the publishing database to the subscriber. This step is also called the table synchronization phase. To reduce the time spent in this phase, you can spawn multiple table synchronization workers. However, you can only have one synchronization worker for each table.

The following example shows how to migrate all tables from a public schema.

Configure the Source Database

To configure the source database with built-in logical replication, complete the following steps.

1. In the source database, edit the postgresql.conf file to add the following parameters.

```plaintext
wal_level = 'logical'
max_replication_slots = 10
max_wal_senders = 10
```

Make sure that you set wal_level to logical. This adds information necessary to support logical decoding. Also, make sure that you set max_replication_slots to at least the number of subscriptions expected to connect, plus some reserve for table synchronization. Finally, make sure that you set max_wal_senders to at least the same value as max_replication_slots plus the number of physical replicas that are connected at the same time.

2. Restart the source PostgreSQL instance for these parameters to take effect.

3. To make sure that you configured parameters on your source database correctly, run the following command.

```bash
psql -h <hostname> -p 5432 -U <username> -d <database_name>
-1c "select name, setting from pg_settings where name in
('wal_level','max_worker_processes','max_replication_slots','max_wal_senders');"

-h is the name of source server where you would like to migrate your database.
```
Set Up the Logical Replication

Now, you can configure built-in logical replication between your self-managed PostgreSQL databases and Amazon RDS for PostgreSQL or Aurora PostgreSQL.

To create the publication on the source database server, run the following command.

```
CREATE PUBLICATION my_publication FOR ALL TABLES;
```

You can specify only the tables that you want to publish. You can also limit the changes that will be published.

To replicate DELETE and UPDATE operations, make sure that the published table has a replica identity, which can be a primary key. This makes it possible for the subscriber to identify the modified rows. You can replicate INSERT operations without a replica identity. After you created the publications, you can create subscriptions in the subscriber node.

Before you create a subscriber node, make sure that you created the target Amazon RDS for PostgreSQL or Aurora PostgreSQL database. For more information, see Creating a PostgreSQL DB instance and connecting to a database on a PostgreSQL DB instance.

To create a database on Amazon RDS for PostgreSQL or Aurora PostgreSQL, run the following command.

```
psql -h <hostname> -p 5432 -U <username> -d <database_name> -c "create database migrated_database;"
```

- `h` is the name of target server where you would like to migrate your database.
- `U` is the name of the user present on the target server.
- `d` is the name of database name present on target already.

To create the subscription on your target database, run the following command.

```
CREATE SUBSCRIPTION <subscription_name>
CONNECTION 'host=<host> port=<port_number> dbname=<database_name> user=<username> password=<password>' PUBLICATION <publication_name> WITH copy_data=true;
```

- `subscription_name`: Provide the name of the subscription created at target.
- `host`: Provide the hostname of the source database.
- `port_number`: Provide the port on which the source database is running.
- `database_name`: Provide the name of the database where the publication is created.
- `publication_name`: Provide the name of the publication created at source.
- `copy_data`: Specifies whether the existing data in the publications that are being subscribed to should be copied once the replication starts. The default is true.

Verify that the Data Replication Is Running

Make sure that no active transactions or data changes are happening on your source database. Then, check the status of your replication by running the following statement on your source database.

Make sure that the WAL locations are the same for the `sent_location`, `write_location`, and `replay_location`. This indicates that the target database is at the same LSN position as the source database.

```
SELECT * FROM pg_stat_replication;
```
Stop the Replication

When the data is in sync between your source and target databases, stop the subscriber on your target database.

```
ALTER SUBSCRIPTION <subscription_name> DISABLE;
```
- `<subscription_name>`: Provide the name of the subscription created at target.

Capture the slot name created by the publisher and subscriber on the target database.

```
select subslotname from pg_subscription where subname like 'subscription_name';
```
- `<subscription_name>`: Provide the name of the subscription created at target.

Capture the `confirmed_flush_lsn` value from the replication slot fetched. You can use this value as the start position for the AWS DMS task.

```
SELECT slot_name, confirmed_flush_lsn from pg_replication_slots where slot_name like 'replication_slot_name';
```

Drop Publication and Subscription Artifacts

To drop the subscription on your target database, run the following command.

```
DROP SUBSCRIPTION <subscription_name>;
```
- `<subscription_name>`: Provide the name of the subscription created at target.

To drop the publication on your source database, run the following command.

```
DROP PUBLICATION <publication_name>;
```
- `<publication_name>`: Provide the name of the publication created at source.

pglogical

The pglogical extension for PostgreSQL implements logical streaming replication, using a similar publish and subscribe built-in approach.

The pglogical extension is suitable for the following use cases if:

- Your database size is greater than 100 GB.
- You want to replicate the schema, DDL, sequences, and table data.
- You want to capture ongoing changes.
- You want to avoid downtime.

The pglogical extension may not be suitable for the following use cases if:

- You have UNLOGGED and TEMPORARY tables.
- You plan to migrate database metadata.

Example

The following example shows how to migrate the public schema.
Configure the Source Database

To configure the source database with logical replication, complete the following steps.

1. In the source database, edit the `postgresql.conf` file to add the following parameters.

```sql
wal_level = 'logical'
max_worker_processes = 10
max_replication_slots = 10
max_wal_senders = 10
shared_preload_libraries = 'pglogical'
```

2. Restart the source PostgreSQL instance for these parameters to take effect.

3. To make sure that you configured parameters on your source database correctly, run the following command.

```
psql -h <hostname> -p 5432 -U <username> -d <database_name> -c "select name, setting from pg_settings where name in ('wal_level','track_commit_timestamp','max_worker_processes','max_replication_slots','max_wal_senders','shared_preload_libraries');" -h is the name of source server where you would like to migrate your database.
-U is the name of the user present on the source server.
-d is the name of database name present on source already.
```

Configure the Target Database

By default, Amazon RDS for PostgreSQL and Aurora PostgreSQL have the `pglogical` extension. To configure the target DB parameter group, complete the following steps.

1. To turn on the logical replication in the target database, set the following parameters in the database parameter group. For more information, see Working with parameter groups.

```sql
rds.logical_replication=1
shared_preload_libraries = 'pglogical'
```

2. Reboot your Amazon RDS instance for these parameters to take effect.

3. To make sure that you configured parameters on your target database correctly, run the following command.

```
psql -h <hostname> -p 5432 -U <username> -d <database_name> -c "select name, setting from pg_settings where name in ('wal_level','track_commit_timestamp','max_worker_processes','max_replication_slots','max_wal_senders','shared_preload_libraries');" -h is the name of target server where you would like to migrate your database.
-U is the name of the user present on the target server.
-d is the name of database name present on target already.
```

Set Up the Logical Replication

Now, you can configure the logical replication between your self-managed PostgreSQL databases and Amazon RDS for PostgreSQL or Aurora PostgreSQL.

1. Download the `pglogical rpm` and install it on your source database.
   - For PostgreSQL 9.6, run the following command.

```
curl https://access.2ndquadrant.com/api/repository/dl/default/release/9.6.rpm | bash
```
yum install postgresql96-pglogical

- For PostgreSQL 10, run the following command.

```bash
curl https://access.2ndquadrant.com/api/repository/dl/default/release/10/rpm | bash
yum install postgresql10-pglogical
```

2. Create the `pglogical` extension on your provider and subscriber.

```sql
CREATE EXTENSION pglogical;
```

3. Create the publisher node on your source database.

```sql
SELECT pglogical.create_node(
    node_name := 'publisher_name',
    dsn := 'host=<publisher_hostname> port=port_number dbname=<database_name>'
);
```

- `publisher_name`: Provide the name of the publication created at source.
- `publisher_hostname`: Provide the hostname of the source database.
- `port_number`: Provide the port on which the source database is running.
- `database_name`: Provide the name of the database where the publication is created.

4. Add tables in public schema to the default replication set.

```sql
SELECT pglogical.replication_set_add_all_tables('default', ARRAY['public']);
```

5. Create the subscriber node on target database.

```sql
SELECT pglogical.create_node(
    node_name := 'subscriber_name',
    dsn := 'host=<subscriber_hostname> port=port_number dbname=<database_name>'
);
```

- `subscriber_hostname`: Provide the hostname of the target database.
- `port_number`: Provide the port on which the target database is running.
- `database_name`: Provide the name of the database where the subscription is created.
- `subscriber_name`: Provide the name of the subscription created at target.

6. Create the subscription on the subscriber node. This subscription starts synchronization and replication processes in background.

```sql
SELECT pglogical.create_subscription(
    subscription_name := 'subscriber_name',
    provider_dsn := 'host=<subscriber_hostname> port=port_number dbname=<database_name>'
);

SELECT pglogical.wait_for_subscription_sync_complete('subscriber_name');
```

- `subscriber_hostname`: Provide the hostname of the target database.
- `port_number`: Provide the port on which the target database is running.
- `database_name`: Provide the name of the database where the subscription is created.
- `subscriber_name`: Provide the name of the subscription created at target.

### Verify that the Data Replication Is Running

Make sure that no active transactions or data changes are happening on your source database. Then, check the status of your replication by running the following statement on your source database.

Make sure that the WAL locations are the same for the `sent_location`, `write_location`, and
replay_location. This indicates that the target database is at the same LSN position as the source database.

```sql
SELECT * FROM pg_stat_replication;
```

**Stop the Replication**

When the data is in sync between your source and target databases, stop the subscriber on your target database.

```sql
select pglogical.alter_subscription_disable('subscriber_name');
```

- `subscriber_name`: Provide the name of the subscription created at target.

Capture the `confirmed_flush_lsn` value from the replication slot created by the `pglogical` setup. You can use this value as the start position for the AWS DMS task.

```sql
SELECT slot_name, confirmed_flush_lsn from pg_replication_slots where slot_name like 'replication_slot_name';
```

**Drop the Subscription**

To drop the subscription on your target database, run the following command.

```sql
select pglogical.drop_subscription('subscriber_name');
```

- `subscriber_name`: Provide the name of the subscription created at target.

**Performance Comparison**

We analyzed the performance of `pg_dump` and `pg_restore`, publisher and subscriber, and `pglogical` in a full load migration. We migrated a 70 GB database that includes 6 tables and LOB data. We lifted and shifted this database from the source to the target.

The following image represents the performance comparison of the three migration methods. We expect similar performance trends for larger datasets.
We performed this test to provide a basic overview of the full load performance. This performance may vary because it depends on such factors as network bandwidth, data structure, parallel jobs, data size, and so on.

The elapsed time shown in the diagram is the actual migration time. It doesn't include the time spent on implementing prerequisites.

You can compare the results of all three methods.

- 35 minutes is the total elapsed time for pglogical.
- 37 minutes is the total elapsed time for publisher and subscriber.
- 46 minutes is the total elapsed time for pg_dump and pg_restore. This time includes:
  - 19 minutes to unload data using pg_dump.
  - 27 minutes to load data using pg_restore.

From the comparison, you can see that pglogical has the best performance among the three full load options. Consider this approach if you don't need to migrate secondary database objects such as views, stored procedures, triggers, and so on. This is the preferred approach where the database size is greater than 100 GB and when you don't have transformation or filtering requirements.
Publisher and subscriber may be appropriate if you don't need to migrate secondary database objects such as views, stored procedures, triggers, and so on. You can use publisher and subscriber for smaller migrations where ease of use considerations override the minor performance gains provided by pglogical.

Using pg_dump and pg_restore is slower than both pglogical and publisher and subscriber. pg_dump is the only option that migrates your secondary database objects. Additionally, data files created by pg_dump may be orders of magnitude larger than the original table size. You might need a significant amount of storage space when you use pg_dump to migrate in parallel.

Ongoing Replication

After you complete the full load, make sure that you perform ongoing replication using AWS DMS to keep the source and target databases in sync. To configure the ongoing replication task, open the Database Migration Service console. On the Create database migration task page, follow these steps to create a migration task.

1. For Migration type, choose Replicate ongoing changes.
2. Under CDC start mode for source transactions, choose Enable custom CDC start mode.
3. Under Custom CDC start point, paste the native start point you captured when you prepared for ongoing replication. For more information, see Preparing for Ongoing Replication (p. 10).

Note

PostgreSQL as a source doesn’t support a custom CDC start time. This is because the PostgreSQL database engine doesn’t have a way to map a timestamp to an LSN or SCN as Oracle and SQL Server do.

For more information, see Creating tasks for ongoing replication and Migrate from PostgreSQL to Amazon RDS.

Summary

This document describes the hybrid approach for migrating between PostgreSQL databases. We analyzed three options for full load and demonstrated the relative performance of each for a test database.

If you need to create secondary database objects, then pg_dump and pg_restore is the most appropriate option. However, this option incurs a performance tradeoff compared to other options.

pglogical has a slight performance advantage over publisher and subscriber. However, you need to install the pglogical extension on your source database server.

You can use these guidelines to choose the option that best matches your migration goal.

Resources

For more information, see the following references:

- What is Database Migration Service?
- Best practices for Database Migration Service
- Using a PostgreSQL database as a source
- Using a PostgreSQL database as a target
- Migrate from PostgreSQL on Amazon EC2 to Amazon RDS for PostgreSQL using pglogical
Migrating an Oracle Database to Amazon RDS for Oracle

You can use these three main approaches to migrate self-managed Oracle databases to Amazon Relational Database Service (Amazon RDS) for Oracle.

- Using a native database tool such as Oracle Data Pump.
- Using a managed service such as the AWS Database Migration Service (AWS DMS).
- Using a native tool for the full load phase and AWS DMS for ongoing replication.

This document describes the third strategy — we call this the hybrid approach. The following diagram shows the components of the hybrid approach.
The hybrid approach provides the following advantages.

- To automate the creation of secondary database objects such as views, indexes, and constraints.
- To use AWS DMS data validation to ensure your target data matches with the source, row by row and column by column.
To use some of the other capabilities that AWS DMS provides, for example data filtering or renaming tables and columns.

This document describes the native options for the full load. It also includes a comparison so you can evaluate the options against your migration requirements. In conclusion, you can find a brief description of how to use AWS DMS for ongoing replication.

Topics
- Full Load (p. 23)
- Performance Comparison (p. 32)
- Ongoing Replication (p. 33)
- Summary (p. 34)
- Resources (p. 34)

Full Load

The full load phase populates the target database with a copy of the source. This chapter describes the following methods to help you choose the one that best matches your migration scenario.

- Oracle Export/Import.
- Oracle Data Pump.
- Database link.
- Oracle SQL Developer database copy.
- Oracle materialized views.
- Oracle SQL*Loader.

We recommend that you begin by reviewing the following table to understand the tools suitable for your use case.

<table>
<thead>
<tr>
<th>Oracle native tools</th>
<th>Supports metadata migration</th>
<th>Suitable for large number of tables</th>
<th>Suitable database sizes</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Export/Import</td>
<td>Yes</td>
<td>Yes</td>
<td>Less than 10 GB</td>
<td>Medium</td>
</tr>
<tr>
<td>Oracle Data Pump</td>
<td>Yes</td>
<td>Yes</td>
<td>Any size</td>
<td>High</td>
</tr>
<tr>
<td>Database link</td>
<td>No</td>
<td>No</td>
<td>Less than 10 GB</td>
<td>Medium</td>
</tr>
<tr>
<td>Oracle SQL*Loader</td>
<td>No</td>
<td>No</td>
<td>Less than 10 GB</td>
<td>High</td>
</tr>
<tr>
<td>Oracle materialized views</td>
<td>No</td>
<td>No</td>
<td>Less than 500 MB</td>
<td>Low</td>
</tr>
<tr>
<td>SQL Developer database copy</td>
<td>Yes</td>
<td>Yes</td>
<td>Less than 200 MB</td>
<td>Low</td>
</tr>
</tbody>
</table>

Note
The sizes that are provided in the table are AWS DMS recommendations based on customer migration experiences and not the limitations of the native tools.
This document doesn’t discuss Oracle external tables because their use is similar to Oracle Data Pump. AWS DMS recommends using Oracle Data Pump instead.

Oracle Data Guard and RMAN are the excellent options to migrate an Oracle instance into EC2, however, Amazon RDS doesn’t support these options.

You can use the Amazon DMS Sample Database for Oracle: version 1.0 to run the following migration examples.

The Performance Comparison (p. 32) section to know the performance of these native tools.

The AWS DMS ongoing replication task requires a start position such as system change number (SCN) from the source database. AWS DMS ongoing replication task replicates changes from a position that you specify during the task configuration. If all exported objects are consistent as of same SCN or if there is no data modification after this SCN, we call it as consistent full load.

To avoid data loss and/or duplication with hybrid approach, make sure of the following:

- Full load is consistent.
- You captured the SCN from the source database before you start the full load when there are no open transactions in the database.

To capture the SCN, use the following command before you start the full load.

```
select current_scn from v$database;
```

**Oracle Export/Import**

Oracle Export/Import is a native database migration tool set that is provided as part of the database installation. Oracle replaced Export/Import with Oracle Data Pump, but many DBAs are familiar with Export/Import because of its long history and usage.

Export/Import may be suitable for your use case if:

- Your database size is less than 10 GB.
- You plan to migrate your database metadata, as well as table data.
- You have a relatively large number of tables to migrate.

Export/Import may not be suitable for your use case if:

- Your database size is greater than 10 GB.
- You have LOBs or other binary data values.

One limitation of Export/Import is that it performs a serial migration. If you have a large data volume and/or large objects like LOB or CLOB values, then using Export/Import may be slower than the other options.

**Example**

You use Oracle exp to unload the source database into a dump file, and Oracle imp to load the dump file into the target database. You can run exp and imp on the same host to avoid copying the dump file between database servers.

For example, you can perform the following steps on the source database to migrate the dms_sample schema using exp and imp. The schema includes tables, views, indexes, packages, stored procedures, and other database objects.
First, capture the current SCN on the source database. Use this value to get a consistent image in the database export.

```
select current_scn from v$database;
```

**CURRENT_SCN**

```
86409924
```

Then, export the dms_sample schema using the `exp` utility. Set `flashback_scn` to the SCN value that you obtained in the previous step to get a consistent export.

```
exp <user>/<password> file=export_file.dmp owner=dms_sample log=explog_file.log
flashback_scn=86409924 statistics=none
```

Finally, import the `export_file.dmp` dump file into the target Amazon RDS database

```
imp <user>/<password>@targetdb fromuser=dms_sample touser=dms_sample file=export_file.dmp
log=implog_file.log
```

This command imports the dump file into the target `dms_user` schema.

**Oracle Data Pump**

Oracle Data Pump is a native database migration tool set that is provided as part of the database installation. Such as `exp` and `imp`, Oracle Data Pump has separate utilities for export and import. These utilities are `expdp` and `impdp`. You can use Oracle Data Pump to migrate data and database objects such as tables, indexes, views, procedure, packages, and functions, and so on.

Oracle Data Pump may be suitable for your use case if:

- You want to automate creation of secondary database objects such as views, indexes, and constraints.
- You have a large number of tables to migrate. Because Oracle Data Pump works at different levels such as table, schema, and database level, it doesn't add additional steps or complexity with an increase of the number of tables.

Oracle Data Pump may not be suitable for your use case if you have a small database, which is less than 10 GB. Though Oracle Data Pump works well for databases of any size, for smaller data sets you have many native tools available as well.

**Example**

At a high level, we can use the following steps to migrate the dms_sample database.

- Export data to one or more dump files.
- Move the dump file or files.
- Import the dump file or files.

**Export data**

To perform a consistent export, make sure that you use the `flashback_scn` parameter in `expdp`. Run the following commands on the source database to get the current SCN, and create a database directory to store the dump and log files.

```
SQL> select current_scn from v$database;
```
CURRENT_SCN
60045321

SQL> create directory expdp_sample_bkp as '/u001/oraarch';
Directory created.

Now, use the 60045321 SCN and the expdp_sample_bkp directory in the following command to dump the dms_sample schema.

expdp userid=<user>/<password> directory=expdp_sample_bkp dumpfile=exp_dms_sample.dmp
logfile=exp_dms_sample.log schemas=dms_sample flashback_scn=60045321

Move the dump file or files

After you export your data, you have the exp_dms_sample.dmp dump file. You can move this file to Amazon RDS using different methods. In this example, we use the Amazon RDS and Amazon S3 integration feature to transfer the dump file to the Amazon RDS instance. For more information, see Amazon S3 integration in the Amazon Relational Database Service User Guide.

The first step is to copy the dump file into the Amazon S3 bucket in the same account and region as the target database.

aws s3 cp exp_dms_sample.dmp s3://YourBucket

Now, use the rdsadmin_s3_tasks package to copy the file from Amazon S3 to your Amazon RDS instance. The file will be copied to the DATA_PUMP_DIR directory which was created as part of your Amazon RDS instance.

select rdsadmin.rdsadmin_s3_tasks.download_from_s3(
    p_bucket_name => 'YourBucket',
    p_directory_name => 'DATA_PUMP_DIR', p_s3_prefix => 'exp%')
as task_id from dual;

The preceding statement returns the ID of the task. You can view the result by displaying the task's output file.

SELECT text FROM table(rdsadmin.rds_file_util.read_text_file('BDUMP','dbtask-*task-
id*.log'));

Replace task-id with the task ID returned by the download_from_s3.

Import the dump file or files

You can use the following command to import the dump file into your Amazon RDS instance.

DECLARE
    v_hdnl NUMBER;
BEGIN
    v_hdnl := DBMS_DATAPUMP.OPEN( operation => 'IMPORT', job_mode => 'SCHEMA', job_name => null);
    DBMS_DATAPUMP.ADD_FILE(
        handle => v_hdnl,
        filename => 'exp_dms_sample.dmp',
        directory => 'DATA_PUMP_DIR',
        filetype => dbms_datapump.ku$_file_type_dump_file);
**Database Link**

In Oracle, a database link enables you to access objects in another database. To migrate tables using a database link, you first create the database link and then run the insert and select statements for individual tables.

Using a database link may suit your use case if:

- You have a relatively small database size, which is less than 10 GB.
- You need to migrate table data only.
- You have a relatively small number of tables to migrate. For large number of tables you need to create a script to perform the migration.

Database link may not suit your use case if:

- Your database size is greater than 10 GB.
- You need to migrate schema objects other than table.

**Example**

The following example migrates the `sporting_event_ticket` table from the `dms_sample` schema.

First, create a database link to the source on the target database.

```
SQL> create database link rdsmigration_link
    2 CONNECT TO <user> identified by <password>
    3 USING '(description=(address=(protocol=tcp) (host=<Self Managed Database Hostname>)
    4 (port=<Listener Port number>)) (connect_data=(sid=<sourcedb sid>)))';
```

Use the following query to verify that the database link works correctly.

```
SQL> select sysdate from dual@rdsmigration_link;
```

Run the following query on the source to generate the DDL statement for the table. You can use the dbms_metadata system package to extract the DDL. After generating the DDL, create the table on the target database.

```
select dbms_metadata.get_ddl('TABLE','SPORTING_EVENT_TICKET') from dual;
```

Run the following query on target database to transfer data using the insert and select statements.

```
insert into dms_sample.sporting_event_ticket select * from
    sporting_event_ticket@rdsmigration_link;
Commit;
```
Oracle SQL*Loader

Oracle SQL*Loader or `sqlldr` is a native database utility. Oracle provides this utility as part of the Oracle installation. Oracle SQL*Loader loads data from flat files into an Oracle database.

Oracle SQL*Loader may be suitable for your use case if:

- Your database size is less than 10 GB.
- You need to migrate data only.
- You have a small number of tables to migrate, the tool requires data export and control file creation for each table.

Oracle SQL*Loader may not be suitable for your use case if:

- Your database size is greater than 10 GB.
- You need to migrate database objects along with data.

Example

The migration process includes the following steps.

- Export the table data and create configuration files.
- Create the table on the target database.
- Load the exported files into the target database.

The following example migrates the `sporting_event_ticket` table from the `dms_sample` schema.

Export the table data and create configuration files

Use the SQL Developer to connect to your source database. Choose **Tools**, and then choose **Database Export** to open the **Export Wizard**. Choose the following option to generate the data file, control file, and table DDL for the target database.

- Select your connection from the **Connect Panel**.
- Turn on **Export DDL**.
- Turn on **Export Data**.
- Make sure that the data format is set to **loader**.
- Save as separate files.

To export the `sporting_event_ticket` table, open the **Types of Export** page and do the following:

- Choose **Table**.
- Search for the `sporting_event_ticket` table.
- Use default options for other pages.
- Choose **Finish**.

This step exports the `sporting_event_ticket` data, generates the control file, and the table DDL in separate operating system files.

Create the table on the target database
On the target Amazon RDS database, run the DDL script that was generated by Oracle SQL Developer to create the table.

**Load the exported files into the target database**

Use SQL*Loader from same host to import data. If required, you can copy the dump files to an Amazon EC2 instance to perform the import.

```
sqlldr userid=<user>/<password>@targetdb control=sporting_event_ticket.ctl log=load.log bad=load.bad discard=load.dsc direct=y skip_index_maintenance=true
```

This example uses Oracle SQL Developer to generate the configuration file. If you prefer, you can use SQL*Plus to generate the configuration file. For more information, see [Oracle SQL*Loader](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_sqlldr.html) in the *Amazon Relational Database Service User Guide*.

**Oracle SQL Developer Database Copy**

Oracle SQL Developer is a graphical SQL client available from Oracle and can be installed on Windows, Linux, or macOS. You can use the Database Copy option in SQL Developer to transfer the data from one database to another; you can choose to copy individual objects or an entire schema.

Oracle SQL Developer database copy may be suitable for your use case if:

- You have a relatively small database size, which is less than 200 MB.
- You need to migrate both data and metadata.

**Example**

Use the following steps to migrate the `sporting_event_ticket` table to the target database.

1. Install Oracle SQL Developer.
2. Open the Database Copy wizard on the Tools menu.
3. Connect to your source and target databases.
4. Select the appropriate options to migrate the tables in your source database.
5. On the next page, choose the `sporting_event_ticket` table, and then choose **Finish** to start the migration.
Database Copy Wizard - Step 3 of 5

Specify Objects

Object Types

Specify Objects

Specify Data

Copy Summary

Name

More... Lookup

SPORTING_EVENT_TICKET

Proceed to summary.

Help

< Back
Oracle Materialized Views

A materialized view is an object that contains the results of a query. You can use Oracle materialized views to migrate data over a database link. With Oracle materialized views, you can perform a full load migration and keep your target tables continuously in-sync with the source.

Oracle materialized views may be suitable for your use case if:

- You have a relatively small database size, which is less than 500 MB.
- You need to migrate data only.
- You have a few tables to migrate as it requires preparation for each table.

Oracle materialized views may not be suitable for your use case if you need to migrate objects other than table data.

Example

The following example shows how to migrate the sporting_event_ticket table.

On the target Amazon RDS instance, create a database link to the source database.

```
cREATE DATABASE LINK rdsmigration_link
CONNECT TO <user> IDENTIFIED BY <password>
USING '(description=(address=(protocol=tcp, host=<Self Managed Database Hostname>, port=<Listener Port number>)) (connect_data=(sid=<sourcedb sid>)))';
```

Test the database link to make sure you can access the source database.

```
SELECT systime FROM dual@rdsmigration_link;
```

On source database, create a materialized view log.

```
cREATE MATERIALIZED VIEW LOG ON sporting_event_ticket;
```

On the target Amazon RDS instance, create a materialized view.

```
cREATE MATERIALIZED VIEW sporting_event_ticket
BUILD IMMEDIATE
REFRESH FAST
AS (SELECT * FROM dms_sample.sporting_event_ticket@rdsmigration);
```

When you're ready to switch to the new database, drop the materialized view using the PRESERVE TABLE clause to retain the underlying table and its contents.

```
DROP MATERIALIZED VIEW sporting_event_ticket preserve table;
```

For more information, see CREATE MATERIALIZED VIEW in the Oracle documentation.

Performance Comparison

We analyzed the Oracle Export/Import, Oracle Data Pump, database link, and SQL*Loader tools for their performance in a full load migration. We populated the sporting_event_ticket table with 10 GB of data to use as a test environment.
We expect the similar trend for larger data sets too. We didn’t include Oracle materialized views or SQL Developer database copy because those tools aren’t recommended for data sets larger than 1 GB.

• 18:08 minutes is the total elapsed time for Oracle Data Pump. This time includes:
  • 3:07 minutes to unload data and metadata using expdp.
  • 2:00 minutes to upload the data dump to Amazon S3 from Amazon EC2.
  • 3:01 minutes to download the data dump to Amazon RDS instance.
  • 7:00 Minutes to load data and metadata into Amazon RDS using impdp.
• 22 minutes is the total elapsed time for Oracle SQL*Loader. This time includes:
  • 15 minutes to unload data.
  • 7 minutes to load data into the target using direct=y.
• 29 minutes is the total elapsed time for database link.
• 52 minutes is the total elapsed time for Oracle Export/Import. This time includes:
  • 14 minutes to unload data using exp.
  • 38 minutes to load data using imp.

Ongoing Replication

After you complete the full load, make sure that you perform ongoing replication using AWS DMS to keep the source and target databases in sync. To configure the ongoing replication task, sign in to the AWS Management Console and follow these steps.

1. Choose Database Migration Service, and then choose Database migration tasks.
2. Choose Create task.
3. For Migration type, choose Replicate data changes only.
4. For CDC start mode for source transactions, choose Enable custom CDC start mode.
5. For **Custom CDC start point**, choose **Specify a log sequence number** and enter the SCN that you captured before starting the full load.

For more information, see Continuous replication tasks and Migrate from Oracle to Amazon RDS.

**Summary**

To migrate database objects and data, use either Oracle Export/Import or Oracle Data Pump. Oracle Export/Import and Oracle Data Pump automate schema object creation. Oracle Data Pump has better performance than Oracle Export/Import and it’s a newer version of Oracle Export/Import.

You can still choose to work with Oracle Export/Import for relatively small data sets which are less than 10 GB because of the ease of use. To migrate table data only, choose any native full load option described in the full load and performance comparison sections.

For full load and ongoing replication, use the hybrid approach. AWS DMS recommends using Oracle Data Pump for full load because it’s faster than other tools, and it automates target object creation.

**Resources**

For more information, see the following references:

- What is Database Migration Service?
- Best practices for Database Migration Service
- Using an Oracle database as a source
- Using an Oracle database as a target
- Migrate an on-premises Oracle database to Amazon RDS for Oracle
- Importing data into Oracle on Amazon RDS

**Migrating a SQL Server Always On Database to AWS**

Microsoft SQL Server Always On is a high-availability feature for Microsoft SQL Server databases. With the synchronous-commit secondary replica, your application remains transparent to a failover. If the primary node in the Always On Availability Group (AAG) fails due to unforeseen circumstances or due to maintenance, your applications remain unaware of the failure, and automatically redirect to a functional node. You can use AWS Database Migration Service (AWS DMS) to migrate a SQL Server Always On database to all supported target engines. AWS DMS has the flexibility to adapt to your Always On configuration, but it may be unclear how to set up the optimal AWS DMS configuration.

Using this guide, you can learn how to configure AWS DMS to migrate a SQL Server Always On database to AWS. This guide also describes specific configuration and troubleshooting issues and best practices to resolve them.

The guide includes a customer use case and covers the issues that the customer encountered when configuring AWS DMS, along with the solutions employed.

**Topics**

- Prerequisites (p. 35)
- SQL Server Always On Availability Groups (p. 35)
Prerequisites

The following prerequisites are required to complete this walkthrough:

- Understand how to work with Microsoft SQL Server as a source for AWS DMS. For information about working with SQL Server as a source, see Using a SQL Server Database as a Source.

- Understand how to work with SQL Server Always On availability groups. For more information about working with SQL Server Always On availability groups, see Working with SQL Server Always On availability groups.

- Understand how to run prerequisite tasks for AWS DMS, such as setting up your source and target databases. For information about prerequisites for AWS DMS, see Prerequisites.

- Understand the supported features and limitations of AWS DMS. For information about AWS DMS, see What Is Database Migration Service?.

For more information about AWS DMS, see the user guide.

SQL Server Always On Availability Groups

Always On availability groups provide high availability, disaster recovery, and read-scale balancing. These availability groups require a cluster manager. The Always On availability groups feature provides an enterprise-level alternative to database mirroring. Introduced in SQL Server 2012 (11.x), Always On availability groups maximizes the availability of a set of user databases for an enterprise. An availability group supports a fail-over environment for a discrete set of user databases, known as availability databases, that fail over together. An availability group supports a set of read-write primary databases and sets of corresponding secondary databases. Optionally, secondary databases can be made available for read-only access and/or some backup operations.

AWS DMS Use Case

A customer used AWS DMS to migrate data from a SQL Server 2017 source database. This database was clustered in a 4-node Always On Availability Group (AAG) configuration. The customer configured the AWS DMS source endpoint to connect directly to the IP address of the primary node of the AAG by using an IP address. With this setup, the customer used the AAG HA/DR functionality for internal applications. In this case, AWS DMS can't use the secondary database if a failover happens. The customer used the target endpoint to populate an Operational Data Store (ODS) of the Amazon RDS for SQL Server database and an Amazon Simple Storage Service (Amazon S3) data lake.

The following diagram displays the customer's existing architecture.
Issues with This Approach

Maintenance activities (operating system patching, RDBMS patching) can cause a server failover and AWS DMS will not be able to connect to the source.

Activity and transactions continue to occur on the failover database as shown in the preceding image. Because of this, the change data capture task becomes out of sync when the cluster fails back to the primary node.

At the start of the task, AWS DMS polls all the nodes in Always On cluster for transaction backups. The AWS DMS task can also fail if transaction backup happens from any other node than the primary.

The Solution Recommended by AWS DMS

To address connectivity design deficiencies, AWS DMS recommended to configure the AWS DMS source endpoint to connect to the AAG listener IP address or a canonical name record instead of connecting directly to the IP address of the primary node. In case of a failover, AWS DMS will interact with the secondary databases, like any other application. Without using the AAG listener IP address, AWS DMS will not be aware of the secondary replica to connect in case of a failover.
The following diagram displays the proposed architecture.

AWS DMS recommended to set the extra connection attribute `MultiSubnetFailover=Yes` in the customer’s AWS DMS endpoint. This ODBC driver attribute helps AWS DMS connect to the new primary in case of an Availability Group failover. This attribute is designed for situations when the connection is broken. In these situations, AWS DMS attempts to connect to all IP addresses associated with the AAG listener. For more information, see [Multi-subnet failovers](#).

Also, AWS DMS recommended to set the extra connection attribute `alwaysOnSharedSynchedBackupIsEnabled=false` to poll all the nodes in Always On cluster for transaction backups.

For more information on extra connection attributes for SQL Server as source, see [Extra connection attributes when using SQL Server as a source](#).
Migrating SQL Server Databases to Amazon RDS for SQL Server

This walkthrough gets you started with homogeneous database migration from Microsoft SQL Server to Amazon Relational Database Service (Amazon RDS) for SQL Server. This guide provides a quick overview of the data migration process and provides suggestions on how to select the best option to use.

Customers looking to migrate self-managed SQL Server databases to Amazon RDS for SQL Server, can use one of the three main approaches.

- Use a native database migration method such as backup and restore.
- Use a managed service such as AWS DMS.
- Use a native tool for full load and a managed AWS DMS service for ongoing replication. We call this strategy the *hybrid approach*.

The following diagram shows the hybrid approach. Here, we use one of the three native tools for full load, and AWS DMS for ongoing replication.
The hybrid approach provides the simplicity of the native tools with additional built-in capabilities of AWS DMS. These include:

- Data validation
- Customizable source object selection rules
- Data filtering
- Renaming target tables or columns
- Data transformations
- Data partitioning
This document describes in detail the three full load migration methods. This guide helps you evaluate each method for your migration requirements. In the end, you can find a brief description of how to use AWS DMS for ongoing replication.

Topics
- Full Load (p. 40)
- Performance Comparison (p. 47)
- Ongoing Replication (p. 49)
- Summary (p. 49)
- Resources (p. 49)

Full Load

The full load migration phase populates the target database with a copy of the source data. In each section, you can find detailed information about the full load method and their results to help you choose the one that fits your use case. For all three methods, we use the dms_sample database as an example. The dms_sample database includes tables, views, indexes, stored procedures, and other database objects.

Topics
- Backup and Restore Using Amazon S3 (p. 40)
- SQL Server Import and Export Wizard (p. 43)
- Generate and Publish Scripts Wizard and Bulk Copy Program Utility (p. 45)

Backup and Restore Using Amazon S3

Backup and restore is the easiest and usually the preferred method for the initial load of the target database. In this method, you create a full backup of your self-managed SQL Server database, transfer it to an Amazon S3 bucket, and restore it to your Amazon RDS for SQL Server instance. For more information, see Importing and exporting SQL Server databases using native backup and restore in the Amazon RDS User Guide.

The backup and restore method is suitable for the following use cases:
- Your database size is less than 16 TB.
- You want to carry out a lift and shift migration with no changes or minimal changes to the database. For example, you want to migrate secondary database objects such as users, views, stored procedures, triggers, and so on in addition to your data.
- Network connectivity between your on-premises data center and AWS is often congested or has frequent disconnects. Backup and restore gives you the flexibility to transmit backup files during non-business hours.

The backup and restore method has the following limitations:
- Amazon RDS for SQL Server supports native restore of databases up to 16 TB in size. For SQL Server Express Edition databases, Amazon RDS supports native restore of up to 10 GB.
- On Multi-AZ database instances, you can only natively restore databases that are backed up in full recovery model.
- The Amazon S3 bucket where you store your data, has to be located in the same AWS Region as your target Amazon RDS for SQL Server database instance.
- Restoring backups from one time zone to a different time zone isn't recommended.
You can't transform or filter data at a table-level when you use backup and restore. When you need to migrate a subset of tables, you can't use backup and restore.

**Migration Steps**

At a high level, the steps involved in backup and restore are the following:

- Perform a full backup of the source database.
- Copy the backup file to an Amazon S3 bucket.
- Restore the backup from the Amazon S3 bucket onto the target Amazon RDS for SQL Server database.

We use the **dms_sample** database in the following example.

**Perform Full Backup**

First, perform a full back up of the source database. Amazon S3 currently limits data files to 5 TB. If the database backup size is less than 5 TB, you can use the following command.

```sql
Use [dms_sample]
GO
BACKUP DATABASE [dms_sample] TO
DISK = 'C:\Backup\dms_sample.bak'
WITH NOFORMAT, NOINIT,
NAME = 'Full Backup of dms_sample', SKIP, NOREWIND, NOUNLOAD, STATS = 10
Go
```

If your database is larger than 5 TB, split the backup files. Make sure that each file is less than 5 TB in size. For example, the size of the **dms_sample** database is 15 TB. This means that we use three backup files.

```sql
Use [dms_sample]
GO
BACKUP DATABASE [dms_sample] TO
DISK = 'C:\Backup\dms_sample1.bak',
DISK = 'C:\Backup\dms_sample2.bak',
DISK = 'C:\Backup\dms_sample3.bak'
WITH NOFORMAT, NOINIT,
NAME = 'Full Backup of dms_sample', SKIP, NOREWIND, NOUNLOAD, STATS = 10
Go
```

**Copy Backup Files to Amazon S3**

Now, use the AWS CLI to upload the backup file to an Amazon S3 bucket.

```
aws s3 cp C:\Backup\dms_sample.bak s3://sampledatabaseuswest2/
```

For multiple backup files, use the folder path to copy the backup files to an Amazon S3 bucket.

```
aws s3 cp "C:\Backup" s3://sampledatabaseuswest2/ --recursive
```

Make sure that you define an AWS Identity and Access Management (IAM) role to access the option group. An option group can specify features, called options, that are available for a particular Amazon
When you associate a DB instance with an option group, the specified options and option settings are enabled for that DB instance.

When you create this IAM role, attach a trust relationship and a permissions policy. For more information, see Manually creating an IAM role for native backup and restore.

We create the sql-server-backup-restore role, and then use it when we configure the target Amazon RDS database.

### Restore Your Backup to the Target Database

To restore your backup, do the following:

1. Create an option group for the target database.
   a. In the Amazon RDS console, choose **Option groups**, and then choose **Create option group**.
   b. For **Name**, enter **SQLServerrestore**.
   c. For **Description**, enter **SQLServerrestore**.
   d. For **Engine**, choose **sqlserver-se**.
   e. For **Major engine version**, choose **14.00**.
   f. Choose **Create**.

2. Add the SQLSERVER_BACKUP_RESTORE option and the sql-server-backup-restore role to this option group to access S3 bucket.
   a. On the **Option groups** page, choose the option group that you created.
   b. For **Options**, choose **Add option**. The **Add option** page opens.
   c. For **Option name**, choose **SQLSERVER_BACKUP_RESTORE**.
   d. For **IAM role**, choose the sql-server-backup-restore role.

3. Modify your Amazon RDS for SQL Server DB instance and attach this option group.
   a. In the Amazon RDS console, choose **Databases**, and then choose your target database.
   b. Choose **Modify**. The **Modify DB instance** page opens.
   c. In the **Additional configuration** section, choose **SQLServerrestore** for **Option group**.

Now, you can restore the backup file from Amazon S3 into the target Amazon RDS for SQL Server database. To restore your database, call the rds_restore_database stored procedure. For more information, see Restoring a database.

```
exec msdb.dbo.rds_restore_database
@restore_db_name='DMS',
@s3_arn_to_restore_from='arn:aws:s3:::sampledatabaseuswest2/dms_sample.bak';
```

To restore multiple backup files, use the following command.

```
exec msdb.dbo.rds_restore_database
@restore_db_name='DMS',
@s3_arn_to_restore_from='arn:aws:s3:::sampledatabaseuswest2/dms_sample*';
```

The preceding statement returns the ID of the task. You can use the following command to check the status of the restore using this task ID.

```
exec msdb.dbo.rds_task_status
[@db_name='DMS'],
[@task_id=<ID_number>];
```
Finally, use the following SQL command to get the log sequence number (LSN) of the on-premises source database backup. Then use this LSN to set up the change data capture (CDC) task in AWS DMS.

```
Use [dms_sample]
GO
SELECT [Current LSN], [Begin Time], Description FROM fn_dblog(NULL, NULL) Where
[Transaction Name] = 'Backup:CommitDifferentialBase'
```

**SQL Server Import and Export Wizard**

Microsoft SQL Server Import and Export Wizard is a high-performance option for data migration. It uses the SQL Server Integration Services (SSIS) framework. For more information, see Import and Export Data with the SQL Server Import and Export Wizard and SQL Server Integration Services.

The Import and Export Wizard is suitable for the following use cases:

- To achieve high migration performance.
- To transform data during the migration. You can use the wizard to create SSIS packages and modify them in Visual Studio with an SSIS extension to achieve this.
- To rename the target tables or schemas during the migration.
- To migrate only the tables and avoid the migration of the secondary database objects such as users, views, stored procedures, triggers, foreign keys or functions.

The migration performance is affected by resource constraints of the host where you run the wizard. During the migration, all data is funneled through this host.

**Migration Steps**

Use the following steps to migrate all the tables and views from the `dms_sample` database to your target database.

Disable all constraints on the target DB instance before to the migration. The Import and Export Wizard copies tables in a random order. This may lead to failures if you enforce referential integrity on the target.

```
EXEC sp_msforeachtable 'ALTER TABLE ? NOCHECK CONSTRAINT all'
```

Make sure that you capture the current log sequence number (LSN) from the source database before your start the full load. To capture the current LSN, use the following command.

```
SELECT max([Current LSN]) FROM fn_dblog(NULL, NULL)
```

Then you can use this LSN to set up the change data capture (CDC) task in AWS DMS.

Open the SQL Server Import and Export Wizard from the Windows Start menu. Connect to your source and target databases and select the source tables and views. The following image shows the SQL Server Import and Export Wizard application window.
Choose Next, then choose Run immediately, and then choose Finish. The SQL Server Import and Export Wizard starts the migration. You can monitor the progress of your migration using the Performing Operation screen. For more information, see Performing Operation (SQL Server Import and Export Wizard).

Make sure that you turn on constraints after you complete the migration.
EXEC sp_msforeachtable 'ALTER TABLE ? CHECK CONSTRAINT all'

For more information, see Get started with this simple example of the Import and Export Wizard.

Generate and Publish Scripts Wizard and Bulk Copy Program Utility

You can use the SQL Server Generate and Publish Scripts wizard to create Transact-SQL scripts for objects in your database. Then you can run the Bulk Copy Program Utility (bcp) to copy data from your Microsoft SQL Server instance into data files. Also, you can use bcp to import data into a table from data files. For more information, see How to: Generate a Script (SQL Server Management Studio) and bcp Utility.

This approach is suitable for the following use cases:

• You don't transform data during the migration.
• You don't rename tables or schemas during the migration.
• You use referential integrity on target tables. In this case, bcp automatically suspends RI constraints on target tables during data import.
• You can script and migrate all database objects using the SQL Server Generate and Publish Scripts wizard.

This approach has the following limitations:

• You need to create schemas on your target database before you can use migration scripts.
• This approach is slower than the Import and Export Wizard.
• Data transformations aren't supported.
• In bcp, the error messages are limited to 512 bytes. This can make troubleshooting complicated.
• You run the bcp command for each table. This increases the complexity for large migrations.

Migration Steps

At a high level, the steps involved in this approach are the following:

• Use Microsoft Generate and Publish Scripts wizard to create Transact-SQL scripts from source database.
• Use the created Transact-SQL scripts to create database objects in Target database.
• Use SQL Server Bulk Copy Program Utility (bcp) to export data from the source database to data files. Then, use bcp to import data from the data files into the target database table.

The following example shows how to migrate the dms_sample database using Generate and Publish Scripts wizard and Bulk Copy Program Utility.

Generate a Transact-SQL script for the source database tables. You can save the script as single file or save in a new query window.
Next, create database objects on the target database using the script that you generated in the previous step.

Run the following command on the source database to capture the current log sequence number (LSN). Then use this LSN to set up the change data capture (CDC) task in AWS DMS.

```sql
SELECT max([Current LSN]) FROM fn_dblog(NULL, NULL)
```

Use the Windows command prompt to export the source tables to data files with the bcp utility.
Import the data files created in the previous step into the target database with the bcp utility.

```
bcp [database_name.] schema.table_name out "data_file" -c -t -S [server_name] -d [database_name] -U [login] -P [password]
```

You can create a .bat file with all the bcp scripts to avoid running script one by one. The following code example shows the contents of this .bat file.

```
bcp dbo.export1 out C:\BCP\export1.dat -c -t -S source-server-name -d dms_sample -U dms_user -P password
bcp dbo.export2 out C:\BCP\export2.dat -c -t -S source-server-name -d dms_sample -U dms_user -P password
bcp dbo.export3 out C:\BCP\export3.dat -c -t -S source-server-name -d dms_sample -U dms_user -P password
bcp dbo.export1 in C:\BCP\export1.dat -c -t -S target-server-name -d dms_sample -U dms_user -P password
bcp dbo.export2 in C:\BCP\export2.dat -c -t -S target-server-name -d dms_sample -U dms_user -P password
bcp dbo.export3 in C:\BCP\export3.dat -c -t -S target-server-name -d dms_sample -U dms_user -P password
```

**Performance Comparison**

To compare the full load migration performance for all three methods, we used a test environment. In this environment, we populated the dms_sample database with 410.90 GB of data. We used the same on-premise SQL Server source and RDS SQL Server target databases to load data three times. For these data loads, we used the following methods:

- Backup and restore.
- Import and export wizard.
- Generate and publish scripts wizard and bulk copy program utility (bcp).

The following image represents the performance comparison of the three migration methods. We expect similar performance trends for larger datasets.
The elapsed time shown in the diagram is the actual migration time. It doesn’t include the time spent on implementing prerequisites.

For the backup and restore method, we spent 4.24 hours. This time includes:

- 1.66 hours to backup the database.
- 1.75 hours to copy the data from backup location to Amazon S3.
- 0.88 hours to restore the data from the S3 bucket to Amazon RDS for SQL Server.

For the import and export wizard, we spent 8.58 hours.

For the bcp method, we spent 199 hours. This time includes:

- 0.01 hours to generate scripts.
- 0.01 hours to run the generated script on Amazon RDS for SQL Server.
- 27.88 hours to run the bcp statements for unloading data from on-premise SQL Server.
- 171.1 hours to run the bcp statements for loading data into Amazon RDS for SQL Server.
Ongoing Replication

After you complete the full load, set up ongoing replication using AWS DMS to keep the source and target databases synchronized. To configure the ongoing replication task, open the AWS DMS console. On the Create database migration task page, follow these three steps.

- For Migration type, select Replicate ongoing changes.
- Under CDC start mode for source transactions, select Specify a log sequence number.
- Under System change number, enter the SQL Server log sequence number that you captured during the full load.

For more information, see Continuous replication tasks.

Summary

The following table helps understand how each migration approach fits to different use cases.

<table>
<thead>
<tr>
<th>SQL Server native tools</th>
<th>Data transformation</th>
<th>Table filtering</th>
<th>Metadata rename</th>
<th>Migration of secondary objects</th>
<th>Data validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backup and restore</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Import and export wizard</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL Server - Generate and Publish Scripts Wizard and bulk copy program utility (bcp)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

You can see that the SQL Server backup and restore has the best performance among the three full load options. This is the preferred approach where the database size is less than 16 TB and when you don't have transformation or filtering requirements. Backup and restore has the additional advantage of migrating your secondary database objects such as stored procedures, functions, and so on.

SQL Server Import and Export Wizard supports a wide range of features. Consider this approach as the next option to evaluate if you don't need to migrate secondary database objects such as views, stored procedures, triggers, and so on. Also, use this approach to overcome the backup and restore limitations. SQL Server Import and Export can also be used for smaller migrations where ease of use considerations override the minor performance gains provided by SQL Server Backup and Restore.

Using Generate and Publish Scripts Wizard and bulk copy program utility (bcp) is slower than SQL Server Import and Export Wizard. You can use this approach in some cases because in bcp you can parallelize the load. That said, data files created by bcp may be orders of magnitude larger than the original table size. Because of this, you might need a significant amount of storage space when you use bcp to migrate in parallel.

Resources

For more information, see the following references:
Migrating from Amazon RDS for Oracle to Amazon RDS for PostgreSQL and Aurora PostgreSQL

Amazon Relational Database Service (Amazon RDS) for PostgreSQL and Amazon Aurora PostgreSQL-Compatible Edition have evolved as a strong and cost-effective alternatives to Oracle without the need for a software license or a server to manage. The journey from Amazon RDS for Oracle to Amazon RDS for PostgreSQL and Aurora PostgreSQL has never been easier. This guide provides a quick overview of the process and considerations to be made when moving existing workloads to Amazon RDS for PostgreSQL or Aurora PostgreSQL and some of the tools that can assist in the process. It complements a large body of detailed online reference guidance on every aspects of a migration, and serves to provide a birds eye view of the process.

This document focuses on migrating custom applications where you control the source code. If you operate a packaged vendor application on Oracle, you must determine if the vendor supports the new platform.

Topics

- Can My Oracle Database Migrate? (p. 50)
- Migration Strategies (p. 51)
- The 12 Step Migration Process (p. 52)
  - Future State Architecture Design (p. 54)
  - Database Schema Conversion (p. 56)
  - Application Conversion or Remediation (p. 57)
  - Script/ETL/Report Conversion (p. 58)
  - Integration with Third-Party Applications (p. 59)
  - Data Migration Mechanism (p. 59)
- Testing and Bug Fixing (p. 60)
- Performance Tuning (p. 62)
- Setup, DevOps, Integration, Deployment, and Security (p. 62)
- Documentation and Knowledge Transfer (p. 65)
- Project Management and Version Control (p. 65)
- Post-Production Support (p. 66)
- Automation (p. 54)
- Platform Differences (p. 66)

Can My Oracle Database Migrate?

To quickly see if your workload qualifies as a migration candidate, please use the DMA Connect Application and Database Questionnaire to sort out migration obstacles specific to your application. Consider the following questions. The more you answer No, the easier the migration to Amazon RDS for PostgreSQL or Aurora PostgreSQL will be.
### Application Questions

<table>
<thead>
<tr>
<th>Question</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Are there Oracle dependent parts of the application that you can’t modify by yourself?</td>
<td>If you don’t control all of the code it can be difficult to change the underlying database.</td>
</tr>
<tr>
<td>Is the application commercial off the shelf, and not available for PostgreSQL?</td>
<td>Unless the commercial off-the shelf software (COTS) application also supports PostgreSQL, it will not be able to migrate.</td>
</tr>
<tr>
<td>Does the application use specific methods to connect to an Oracle database such as Oracle Call Interface (OCI)?</td>
<td>Refactoring OCI calls to ODBC is not impossible, but typically an involved process.</td>
</tr>
<tr>
<td>Does the application use Oracle specific libraries?</td>
<td>It could be challenging finding PostgreSQL replacements for Oracle specific libraries.</td>
</tr>
</tbody>
</table>

### Database Questions

<table>
<thead>
<tr>
<th>Question</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does the database use any third party packages?</td>
<td>It could be challenging finding PostgreSQL replacements for Oracle specific packages.</td>
</tr>
<tr>
<td>Does the database use any data cartridges?</td>
<td>It could be challenging finding PostgreSQL replacements for Oracle specific cartridges.</td>
</tr>
<tr>
<td>Does the application use Oracle Forms or Application Express (APEX)?</td>
<td>Completely replacing Forms or APEX with a non-Oracle solution is substantial.</td>
</tr>
<tr>
<td>Does the database use SQLJ or .NET stored procedures?</td>
<td>You can refactor external stored procedure code for use with PostgreSQL, but it adds development work.</td>
</tr>
<tr>
<td>Does the database use Oracle Streams?</td>
<td>Some refactoring is required to replace Oracle Streams with a PostgreSQL-compatible solution.</td>
</tr>
<tr>
<td>Does the database use Oracle Multi Media?</td>
<td>Some refactoring is required to replace Oracle Multi Media with a PostgreSQL-compatible solution.</td>
</tr>
<tr>
<td>Does the database use Oracle Locator?</td>
<td>Depending on feature use, such a solution may be refactored to work with PostGIS 3.1.</td>
</tr>
<tr>
<td>Does the database use Oracle Java Virtual Machine (JVM)?</td>
<td>Detaching a Java application from Oracle JVM can be involved development work.</td>
</tr>
<tr>
<td>Does the database use Oracle Machine Learning or formerly Advanced Analytics?</td>
<td>The solution will have to be refactored to use similar functionality on AWS.</td>
</tr>
</tbody>
</table>

### Migration Strategies

The options for dealing with a legacy application have often been described as the 6 R’s. For more information, see [6 Strategies for Migrating Applications to the Cloud](#).

- Re-host
- Re-platform
- Repurchase

---

51
Refactor/Re-architect
Retire
Retain

This document describes the steps to migrate database instances running on Amazon RDS for Oracle to Aurora PostgreSQL or Amazon RDS for PostgreSQL. This also details out the steps to Re-platform and Refactor the application(s) running on these databases.

Re-platforming and re-architecting a database application ranges from modifying the code to work with a different cloud-native database to also adopting other cloud-native operations such as serverless application architectures like Kubernetes. This document deal with the changes necessary to migrate to a new database with pointers to other available documentation.

The 12 Step Migration Process

You may have an Oracle database in Amazon RDS for both production or non-production purposes, and it may just be convenience and familiarity that steered you to Oracle even though there is a licensing cost to this choice. It is certainly easier to continue with the database you know than something new, but sometimes there are few remaining reasons do so.

Everyone's Oracle application is special, and nobody has the same setup and needs for the future. To provide a single framework for migration, this guide organizes the work in 12 topics in order to cover what is in scope for most applications. These topics will be used in sequence for multiple purposes and should not be seen as a strictly linear process, but rather an overall arch of a migration project where individual topics and activities can be overlapped or swapped to fit specific project conditions. The following image shows the 12 steps with an approximate share of effort in a typical project.

Steps in migration

<table>
<thead>
<tr>
<th>Steps in migration</th>
<th>Effort breakdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Future State Architecture Design</td>
<td>2%</td>
</tr>
<tr>
<td>2. Database Schema Conversion</td>
<td>14%</td>
</tr>
<tr>
<td>3. Application Conversion / Remediation</td>
<td></td>
</tr>
<tr>
<td>4. Scripts / ETL / Reports Conversion</td>
<td>7%</td>
</tr>
<tr>
<td>5. Integration with 3rd-Party Applications</td>
<td>3%</td>
</tr>
<tr>
<td>6. Data Migration Mechanism</td>
<td>4%</td>
</tr>
<tr>
<td>7. Testing and Bug Fixing</td>
<td></td>
</tr>
<tr>
<td>8. Performance Tuning</td>
<td>2%</td>
</tr>
<tr>
<td>9. Setup, DevOps, Integration, Deployment, and Security</td>
<td>7%</td>
</tr>
<tr>
<td>10. Documentation and Knowledge Transfer</td>
<td>2%</td>
</tr>
<tr>
<td>11. Project Management and Version Control</td>
<td>2%</td>
</tr>
<tr>
<td>12. Post-Production Support</td>
<td>3%</td>
</tr>
</tbody>
</table>

Each step will be described at a high level in order to allow the reader to skip to relevant topics in the following chapters.

1. Future State Architecture Design
The understanding of the current design and its requirements together with those of the future state are addressed here with deployment diagrams and feature or component mappings for things that will change as a result of the migration. This step defines the scope and architectural view of the migration.

2. **Database Schema Conversion**

Because we are migrating a database application from Oracle to Amazon RDS or Aurora, the database schema needs to change. Subtle differences in functionality and syntax need to accommodate the new platform and comprehensive tooling exists to automate this step. In this step we include replacements for any Oracle specific database feature which works differently on PostgreSQL.

3. **Application Conversion or Remediation**

The Oracle application may be implemented in any programming language like Java or C# and often abstracts from the nature of the underlying database through an object relational model (ORM). But it is also common to have some reliance on the database syntax directly in the application code, and this step covers the necessary changes to the application code to work with Amazon RDS for PostgreSQL. In this step we also include operating system dependencies like direct file access which may need to change on the new platform.

4. **Script/ETL/Report Conversion**

An Oracle application may move data in and out sideways in addition to the application for the purpose of reporting or data import/export. This can happen by executing a stored procedure or through external scripting and SQL*Loader. Such scripts and PL/SQL statements and the operational framework will need to be modified to work with PostgreSQL.

5. **Integration with Third-Party Applications**

Few applications are islands and often connect to other applications and monitoring. These dependencies may be affected by the move to the PostgreSQL database platform. The monitoring of the database may need to use native AWS tools or the third-party applications use Oracle specific means of communication. These dependencies may already support PostgreSQL or suitable replacements will need to be found.

6. **Data Migration Mechanism**

As we move from one database platform to another the data needs to move as well. This will happen several times through the migration, first for testing purposes and later for production cutover. If there are multiple customers of the database application they may need to be migrated at different times once the application has been migrated.

7. **Testing and Bug Fixing**

Migration touches all the stored procedures and functions and may affect substantial parts of the application code. For this reason good testing is required both at the unit and system functional level.

8. **Performance Tuning**

Due to database platform differences and syntax, certain constructs or combinations of data objects may perform differently on the new platform. The performance tuning part of the migration resolves any bottlenecks that might have been created.

9. **Setup, DevOps, Integration, Deployment, and Security**

How the application is put together and deployed may be impacted by the migration, and many customers take the opportunity to embrace infrastructure as code for the first time in the context of a migration. In this step we also focus on the impact to application security. In this step we also address cutover planning.

10. **Documentation and Knowledge Transfer**
In order to support the application going forward it may be necessary to document the changes that happened to the application and the operational environment. Maintenance of the application will have been impacted by the change of database and certain application behavior may have changed. This is especially important if the migration is done by a different team from those maintaining the application.

11 Project Management and Version Control

A migration certainly involves people with different skills and often entirely different teams, and maybe an outside party. A successful project needs to be well planned and coordinated to execute on a predictable schedule. Version control is a crucial foundation for a migration since database code may not be managed in the same way as application code.

12 Post-Production Support

After the application is live, the migration team may need to stay around for a while to address any emerging problems on the new platform that were not caught by testing.

Automation

This document references the freely available AWS Schema Conversion Tool (AWS SCT) for code conversion and the AWS Database Migration Service for data migration. For more information, see https://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_Installing.html and https://aws.amazon.com/dms/.

Future State Architecture Design

When you migrate an Oracle application to use a different database like PostgreSQL you must capture the architecture of the existing application to ensure that all considerations are covered, we call that the current state architecture. The current state architecture describes the part of the application that matters to the migration from an architectural point of view. The same is true for the future state architecture which takes the new database platform into account. We don't need to describe everything, but some things, like external dependencies, are very important, and help us determine what work to do.

You may already have some favorite drawing tools for architecture diagrams such as Lucidchart, Visio or the freely available Diagrams.net which are all great choices as they supports AWS infrastructure symbols along with many others to describe the current and future environment. But the tool is less important than what is captured in the diagrams.

The architecture diagrams also serve the important role of defining the context of what is inside and outside the scope of work as a team collaborates on the task.

Current State Architecture

There may be existing documentation on the database application which should be examined for currency and relevance. Let us review what is important for the migration work before we decide if more documentation is needed.

A network diagram is useful because it typically connects servers to each other, and servers to databases. It may also show the division into multiple availability or disaster recovery zones. This is useful because it shows potential server and network dependencies that must be addressed in the new architecture. A network diagram may also highlight important security considerations like multiple networks and internet connectivity.

A component diagram is useful if the application is comprised of multiple parts using different technologies which each may present the migration with their own challenges to address.
A **class diagram** is useful if it shows a specific persistence layer or a query factory where the migration can focus while leaving the rest of the application untouched.

A **data flow diagram** is useful because it directly shows parts of the value chain of information flowing inside and outside the application highlighting what additional code may needs to be changed.

The following image shows a simple network diagram that can help easily communicate current architecture.

---

**Future State Architecture**

The future state architecture envisions the application using the new database, and potentially other services in the environment. It’s a new version of the current state architecture diagrams with certain parts replaced with the new components. This document will focus mainly on replacing Amazon RDS for Oracle with Amazon RDS for PostgreSQL or Aurora PostgreSQL.

**Transition Architecture**

Depending on how involved your migration is, you may need a transition architecture by which we mean, infrastructure that is there only for migration purposes. Examples of transition architecture includes AWS DMS servers and other mediating or transformation platforms. Such infrastructure has to be provisioned, secured and removed after the migration to avoid additional vulnerability and cost.

The following image shows a transition architecture diagram.
Database Schema Conversion

Relational databases contain a tabular structure for data using basic data types and procedural code in the form of triggers, functions and stored procedures. Oracle uses the PL/SQL dialect which is different from the PL/pgSQL dialect of PostgreSQL and while some table definitions and queries may look the same, others will require modification. Doing so manually would be a substantial task, but fortunately there are freely available AWS tools to automate this job (See Automation section in the Introduction).

The AWS Schema Conversion Tool (AWS SCT) is capable of connecting to Oracle and reading all PL/SQL code directly from the source Oracle database and converting it to PostgreSQL PL/pgSQL. AWS SCT will retrieve the DDL for tables, views, triggers, stored procedures and functions from the database, parse it and generate the equivalent PostgreSQL code.

Based on experience, AWS SCT fully converts 90+% of the database code which leaves less than 10% for the database expert to improve.

Process

At a high level, the database conversion process works like this:

- Download and install AWS SCT (Linux or Windows).
- Download and install Oracle database drivers (you probably have those already).
- Download and install the PostgreSQL database drivers for Amazon RDS or Aurora PostgreSQL.
- Run AWS SCT and create a migration assessment report. For more information, see Creating migration assessment reports.
- Run AWS SCT and automatically convert the database code. For more information, see Converting database schemas.
- Fix any warnings and error in the database code conversion.

AWS SCT operates with default assumptions about mappings between Oracle and PostgreSQL which may or may not be optimal for your particular application due to the data you have in the database.
Certain data type mappings may need to be changed to ensure good performance. As an example, a NUMBER datatype in Oracle is an extremely versatile container which without further qualification may be too expensive for the application. In this case you would look at the type of data contained in the NUMERIC column and its requirements for precision and scale, and then determine the best match for that in PostgreSQL with the appropriate precision and scale.

Once AWS SCT has automatically converted the DDL code, the developer needs to investigate any warnings and errors which need manual remediation. Warnings and Errors can happen for many reasons. AWS SCT does not have 100% coverage of all syntactical situations, and code inside the database can be corrupted or encrypted preventing AWS SCT from reading it. In these situations, the output DDL code is marked up with comments about the problem AWS SCT had with conversion, and ask the developer for help.

Exceptions

There are exceptions to the automatic code conversion by AWS SCT like SQLJ, .NET Stored Procedures, Spatial data, RDF Graphs. But in each case there are good candidate replacement features like Lambda functions, PostGIS and Neptune.

Interactive and Batch Modes

AWS SCT offers both an interactive GUI and a command line interface (CLI) which are useful in different situations. The user interface is good in a more interactive situations where the user needs to explore the schema and perhaps select only part of it for conversion. The CLI is good for automation in situations where DDL code might be coming from a different source such as reports. For more information, see Script/ETL/Report Conversion (p. 58).

Schema Drift

If the original database schema changes during the timeframe of migration, this can be detected in AWS SCT which can compare the old and the new database schema and highlight the object that need to be updated. If an object was converted 100% or with few manual changes, that object can be converted again and remediated.

For more information, see AWS Schema Conversion Tool User Guide, Oracle Database 19c To Amazon Aurora with PostgreSQL Compatibility (12.4) Migration Playbook, and AWS Schema Conversion Tool CLI and Interactive Mode Reference.

Application Conversion or Remediation

The Oracle application may be written in a variety of languages like C++, C# and Java, each with their own patterns for calling Oracle. A common case is the use of an object relational model (ORM) between the application code and the database which reduce the amount of PL/SQL that needs to be changed. Examples include Entity Framework and Hibernate which are also supported on PostgreSQL.

Oracle uses the PL/SQL dialect which is different from the PL/pgSQL dialect of PostgreSQL and while some table definitions and queries may look the same, others will require modification. Doing so manually would be a substantial task, but the freely available AWS Schema Conversion Tool (AWS SCT).

AWS SCT is capable of identifying and replacing embedded PL/SQL in the application code with the equivalent PostgreSQL code. For more information, see Automation (p. 54).

In addition to using AWS SCT, you must also examine the source code for possible issues like:

- Specific ORM or other data access framework and versions or in use and confirm its compatibility with the target engine.
• Modify database connection as appropriate for the new engine.
• Modify any table/entity mapping configuration or code as appropriate for the converted schema.
• Identify and refactor any vendor-specific driver functionality in use in the code.

Process

At a high level, the application conversion process works like this:

1. Perform the database conversion. This is necessary because the PL/SQL conversion needs to know the schema of the database. For more information, see Database Schema Conversion (p. 56).
2. Run AWS SCT and automatically convert the application code. For more information, see Converting application SQL.
3. Fix any warnings and errors in the application code conversion.

Exceptions

There are exceptions to the automated application code conversion process if the application uses the native Oracle Call Interface (OCI). In this case the developer must refactor the code to use ODBC or JDBC.

Script/ETL/Report Conversion

ETL is an acronym that stands for Extract, Transform and Load. The ETL process plays a central role in data integration strategies. ETL allows businesses to gather data from multiple sources and consolidate it into a single, centralized location. ETL also makes it possible for different types of data to work together.

ELT is similar to ETL. However, the primary difference between them is that the data transformation processes occur after the Raw data from the source have been extracted and loaded into a staging area. The transformation of the data may occur in the destination database or in the middle tier or via serverless tools that might reduce the cost of the data processing.

Transforming the data is a critical process that may provide significant value to the data. It’s also the stage where the data could be cleansed, standardized, deduplicated, verified, sorted, shared, and much more.

The role of ELT or ETL in database migration projects is critical for any successful migration.

For the remainder of this document, ETL will also refer to ELT patterns.

ETL can be implemented in the database itself, in external scripts or in third-party tools such as Informatica, Talend, and so on. If the ETL is done using Oracle stored procedure, the freely available AWS Schema Conversion Tool (AWS SCT) is capable of converting the ETL code to AWS Glue. For more information, see Automation (p. 54).

Process for Conversion to AWS Glue

If Python/Glue is a desired future state architecture for ETL code, and the ETL is implemented in the database, the conversion process works like this:

1. Perform the database conversion. This is necessary because the PL/SQL conversion needs to know the schema of the database. For more information, see Database Schema Conversion (p. 56).
2. Run AWS SCT, select the code involved in ETL and automatically convert the ETL code to AWS Glue. For more information, see Converting ETL processes.
3. Fix any warnings and errors in the ETL code conversion.
Process for Conversion of Stored Procedures

If ETL or report process is implemented in the database, then the database conversion takes care of converting the code, and only the method to call the stored procedures need to change.

Process for Conversion of Scripts, Reports, and Third-Party ETL

If the ETL or Report code is available in scripts or hosted in third-party tools and those tools will be used in the future, then a custom process will have to be implemented:

1. Perform the database conversion. This is necessary because the PL/SQL conversion needs to know the schema of the database. For more information, see Database Schema Conversion (p. 56).
2. Extract PL/SQL statements from the third-party ETL or reporting tool into flat files, unless already available.
3. Write YAML configuration files for AWS SCT CLI to convert external files.
4. Run AWS SCT CLI on the external scripts using the YAML configuration files. For more information, see AWS Schema Conversion Tool CLI and Interactive Mode Reference.
5. Fix any warnings and errors in the ETL or report code conversion.
6. Insert the converted PL/pgSQL code back into the third-party ETL or reporting tool, unless they stay as flat files.

Integration with Third-Party Applications

Few applications are islands and your Oracle application is likely to integrate with other applications that are not themselves going to be migrated. Examples include ETL, reporting, and monitoring applications for alerts and logs. For more information, see Script/ETL/Report Conversion (p. 58).

If these third-party applications connect directly to the Oracle database, they are going to be affected by the migration. If they are packaged applications, the vendor may offer support for Amazon RDS and Aurora PostgreSQL and if they are custom, you may need to modify them to work with the migrated application. There are a wealth of resources on the partner network which complement any solution from AWS.

AWS native tools such as Amazon Simple Notification Service, Amazon RDS Performance Insights, Amazon CloudWatch, and Amazon Relational Database Service are already integrated with the Amazon RDS and Aurora PostgreSQL database platform and are recommended for a full picture of the ongoing performance.

For more information, see Engage with Amazon Web Services Partners.

Data Migration Mechanism

For testing purposes and for production cutover, data needs to be migrated from the old Amazon RDS for Oracle instance to the new Amazon RDS or Aurora PostgreSQL instance. Such a data migration requires knowledge of data type mapping and possibly incremental loading, depending on the size of the data and migration window.

For this purpose AWS Database Migration Service (AWS DMS) can be used to connect source and target databases to replicate the contents of the data in the most optimal way.

Process

1. Create a replication server.
2. Create source and target endpoints that have connection information about your data stores.
3. Create one or more migration tasks to migrate data between the source and target data stores.

After you configured AWS DMS, you can perform the following operations:

- A full data migration from Oracle to PostgreSQL.
- An ongoing replication from Oracle to PostgreSQL.

Depending on the type of data in the database, you may need to optimize AWS DMS for handling certain data types like LOBS which you can read more about in the product guidance.

**Reverse Migration**

Normally you just fall back to the old system if a migration fails during smoke testing, and in most cases you may decide to fix forward after cutover, in which case you fix any unforeseen bugs in the migrated system. But in some cases you may decide to have the option of migrating production data back from the new system to the original system after having been in production for a time. In those cases, a reverse data migration mechanism must be configured.

For more information, see What is Database Migration Service? and Migrating Oracle databases with near-zero downtime.

**Testing and Bug Fixing**

Testing can be manual or automated. We recommend that you use an automated framework for testing. During migration, you will need to run the test multiple times, so having an automated testing framework helps speed up the bug fixing and optimization cycles.

**Unit Testing**

Unit testing at the data level after migration can range from comparing every last bit of data in source and target by comparing extracted CSV files, but more realistically, custom aggregation queries should be constructed to incorporate large amounts of the migrated data and compare the results.
Unit tests validate individual units of your code, independent from any other components. Unit tests check the smallest unit of functionality and should have few reasons to fail.

Database objects need to be validated after migrating the DDL of an Oracle database to PostgreSQL. Database objects includes packages, tables, views, sequences, triggers, primary and foreign keys, indexes, constraints.

A typical way to perform unit testing on the converted database is to script out calls to stored procedures and functions and compare the returned data with external tools such as standard Linux/Unix tooling of diff.

The application needs to be validated with new and existing test case scenarios based on documented changes on database objects such as field names, types, minimum and maximum values, length, mandatory fields, field level validations etc.

**Functional Testing**

Functional testing of the application is done by exercising user stories and comparing the results on the source and target system. This is typically a manual process, but third-party tools do exist to make automated regression tests of the UI (e.g. Selenium).

Functional testing of the database is largely done through the application, but there may be additional direct database use cases that can only be done directly on the database such as ETL for imports and extracts. In these cases, the data can be compared automatically before and after using standard Linux/Unix tooling like diff on extracted CSV files for example.

Functional testing of reports involve visual inspection to see that all fields are correctly displayed and comparison of the semantic values between the old and the new reports.

**Load Testing**

In order to stress the migrated system and test its performance you may perform load testing which is typically done on a system that is scaled the same as production and requires a means of simulating load on the system. It is sometimes limited to running specific well-known expensive operations rather than user traffic.

**Standard Operating Procedures**

Standard Operating Procedures (SOP) may be affected by a migration of database application. Database management procedures change when going to PostgreSQL and some procedures may be unnecessary when going to the highly managed Aurora PostgreSQL.

In any case, all existing operational procedures need to be tested and their language updated to reflect the new environment. For more information, see Managing Amazon Aurora PostgreSQL.

**Monitoring**

Monitoring of the database will be affected by the migration and some metrics may change which could affect how SLA is monitored. The way operational staff go from detecting a problem to diving into the underlying details may be affected. For more information, see Monitor Amazon RDS for PostgreSQL and Amazon Aurora for PostgreSQL database log errors and set up notifications using Amazon CloudWatch.

**Cutover**

Cutover procedures are the planned event where everything goes the way you want, but it still needs to be tested.
**Fallback**

Fallback is when you have both old and new systems in sync with the new one operating as primary and you decide to switch back to the original which is still in sync.

**Rollback**

Rollback is usually the scenario when you don’t have an ongoing replication mechanism to keep old and new in sync, so in the event of a no-go decision during the cutover, you abandon the new system and go back to the original.

**Migrate Back**

In some rare cases, you may decide to include the option of migrating production from the new system back to the old system after having the cutover. If you include this scenario, it must be tested.

For more information, see [Testing Amazon Aurora PostgreSQL by using fault injection queries](#), [Automate benchmark tests for Amazon Aurora PostgreSQL](#), [Validating database objects after migration](#), and [Validate database objects after migrating from Oracle to Amazon Aurora PostgreSQL](#).

**Performance Tuning**

Any migration is likely to slightly change the performance of individual queries in the application and in stored procedures and functions. Depending on the context, those small differences may not matter in reality. But it is a good idea to deliberately compare the performance of operations that are known to be slow in the original system because any difference in performance is likely to be greater. Such testing is usually confined to specific long running ETL jobs and reports. Other performance issues may show up during functional or load testing and will be addressed as bugs.

**Setup, DevOps, Integration, Deployment, and Security**

Deployment to production is the culmination of the migration activity and is a high stakes effort which requires good planning and benefits from well tested automation.

With DevOps, you can create a process that helps easily deploy and update your virtual architecture in a scalable and repeatable way. This reduces the risk of human error. Furthermore, DevOps allow us to deploy much faster than humans which may be a factor in large deployments.

**Wave Planning**

For any application or cluster of applications there is an important question of sequencing because every cutover window, for example, a weekend can only accommodate so much work. This means that larger portfolio may need to be migrated in multiple waves, and this makes wave planning necessary.

Wave planning considers that some parts of the application will move while other stay behind under different network, connectivity and security conditions. Different parts of the application may also be under different ownership, so wave planning becomes the place where all stakeholders coordinate their efforts. Wave planning is a matter of minimizing risk during the overall migration.

**Infrastructure Automation**

Infrastructure automation is a code layer that wraps API calls to a cloud provider with commands to provision infrastructure. Such code typically in YAML scripts are easily learned with any coding experience and are scalable and powerful. This layer will allow you to spin up one or one hundred web nodes nearly
simultaneously. This layer is not designed to configure files on a server, install software on a server, or run commands on a server - That comes in the next section.

**Terraform** represents a cross cloud incarnation of this idea. The downside of Terraform is that its cross-cloud and open source nature makes it slower to adopt new features and provide detailed provisioning, often months after a new feature or configuration is released.

**CloudFormation** is a native AWS language in JSON or YAML format. Use AWS CloudFormation to write infrastructure code more specifically to specific features because it doesn't have to work with other clouds.

**Configuration Management**

Configuration management systems manages configuration of software and state of files on a server or group of servers. These systems however are capable of much more than that, they also allow functionality such as installing software, running local commands, starting services and more in the same scalable way.

**Ansible** is a lightweight and easily installed tool that is configured using YAML files. It doesn't require a local daemon to be installed on the instances that Ansible is managing. The way that ansible does all this is through open source functions that essentially wrap cli commands that are run on remote hosts over Secure Shell Protocol (SSH). This allows for a plethora of functionality from database manipulation to package instillation through simple changes in pre-written functions at the YAML level. Beyond a large library of open source function one can easily write custom functions (in python) or simply use the cli function to run any cli command through ansible remotely and in a scalable fashion. Some environments could be prevented from using Ansible due to limited or highly restricted SSH access to resources due to security protocols and standards.

**Puppet** works in a primary and secondary system that communicates over https (443) and is configured using its own language called **puppet**. It's often found in enterprise level deployments, configuration management platforms based on a locally deployed daemon called a node. Puppet differs from other similar platforms like Chief is its methodology in regards to how a resource acquires a desired state. Puppet takes a declarative approach, which is to say it defines the end state it requires, but makes not design on how it is achieved. Due to its fairly high level of technical investment in regards to its programming language, puppet is generally not recommended for smaller deployments as the investment.

**Chef** has a lot in common with puppet like a similar primary and secondary model, they both communicate over https, and are configured using a programming language. Where they differ is in terms of how they handles state. Chef takes an imperative approach, which is to say you as the end user have nearly full control on how a resource acquires a desired state which it achieved through Ruby as its configuration system. This type of deployment provides more flexibility as well as being easy to adopt if you are already using Ruby.

**Code Repository**

A code repository offers safe storage of code and a change capture log which facilitates parallel development of a codebase by many developers simultaneously with the use of code branches, and integration with CI/CD pipelines. Other files than application source code might be stored in a Git repository such as infrastructure as code (IaC), database reports; recorded state changes or even short logs. Importantly you should never store credentials in the code repository. Credentials should be handled in other ways discussed below to avoid sensitive files being deleted or scrubbed, remnants left behind of the original state. The two most widely used code repositories are GitHub and GitLab with similar features and functionality.

**Secrets Management**

A vault is for credentials. There are several options when it comes to Vault, many of which are baked into a lot of the technologies already described. Ansible has ansible vault, Jenkins has a functionality
for storing and parametrize credentials which can be used at a smaller scale as a vault. However most of
these baked in vaults don’t generally have the effectiveness and capabilities of a dedicated vault.

**HashiCorp Vault** is a dedicated vault that differs from a secrets manager that comes packaged with
another product is its varied capabilities around secret management. HashiCorp is an industry leader in
this regards with capabilities such as dynamic secrets that can be generated on the fly for database or
application credentials, data encryption, leasing and renewal which always credentials to be expired and
rotated as well as the ability to revoke credentials remotely. In general, if an enterprise requires a wide
range of credentials stored across a range a technologies, Vault is generally a good option to start.

**Orchestration**

Orchestration is the glue that binds DevOps together. Many of the described technologies can be
executed manually or from a scheduled script, however building on this idea of removing one of the
largest points of failures such as humans from the actual deployment and management of infrastructure
we can eliminate many of the pitfalls that can arise in the process of executing deployment scripts.
Orchestration allows you to create a repeatable timeline, with logic gates, to deploy your infrastructure
in exactly the order with the configurations chosen. This also allows deployments themselves to be
tested before a change or deployment to production. Each of the configuration management platforms
discussed above generally have an orchestration platform, they are generally focused on scheduling jobs
within their particular vertical. For example, AWX for Ansible is mostly limited to scheduling ansible jobs.

**Jenkins** is managed through a GUI running on a primary node that is deployed on a resource within the
company. There is also functionality to allow secondary nodes deployed on micro services for larger
scale.

The process is arranged in a series of **Groovy** files called Jenkinsfile that dictate what action will be taken
during each step in the pipeline. These jobs then can be scheduled jobs that periodically run and kick
any job, from a ansible jobs that runs periodically to prevent configuration drift, to reporting jobs that
execute a series of database calls. Generally Jenkins can connect any modern codebase and technology
together.

An example pipeline you might run on Jenkins: use git to pull configuration scripts for the pipeline, then
use a Terraform job from the cloned codebase to deploy a server, use Ansible to install and configure a
database, then push a status file with information on the pipeline run back to git all while using Vault to
manage the secrets for both access right for Jenkins and configuration of users on the database.
Documentation and Knowledge Transfer

As a result of the migration, the operation of the database and the future development of the application and database will have been affected due to infrastructure and technology changes. If the migration is done by a separate team, it is vital that these changes be documented.

There may be a need for additional PostgreSQL training to operate the database and develop for PostgreSQL going forward.

Project Management and Version Control

Experience tells us that the steps take different amounts of efforts across a typical project.

When planning a migration project, tooling like AWS SCT can provide an important data point in the form of how much manual work needs to be done to fix database and application code that was not automatically converted. Using the above rules of thumb shares of the overall project, an initial plan can be created.
Post-Production Support

After production cutover, there are a few possibilities for what can happen. You will have either decided to fix forward as the old system is being decommissioned, or you have decided to way a certain amount of time, a bake-in time, with production on the new system, during which a decision to abandon the new system can be made. Abandoning the new system has the following flavors:

- Roll back, all new data is lost.
- Roll back and reapply all new transactions.
- Roll back and migrate new production data back.
- Maintain a live replication back to the old system until bake-in period is over.

During this time, defects are tracked and triaged for possibly triggering the rollback or being fixed forward. Help desk will have been trained in the new system differences and will be able to detect if an end user inquiry just requires training or it may be a defect.

Beyond acceptance migration criteria, the application may have well defined KPIs defined already which can be observed when in production on the new system and compared to historical KPIs.

For more information, see [How to Migrate Your Oracle Database to PostgreSQL](#).

Platform Differences

This section discusses some of the differences between Oracle and PostgreSQL to illustrate opportunities and challenges with migrating an Oracle application. This overview is by no means an exhaustive, however these are common challenges you may encounter when administering PostgreSQL after a background with Oracle.

Range and List Partitions

Along with possible performance difference, architecturally partitions on Oracle and PostgreSQL act quite differently. On Oracle you can define a Range, for example each month or year being a partition, or a list, where every occurrence of say the letter “N” or “Y” in a char field is partitioned at the table definition level and PostgreSQL handles these operations differently. PostgreSQL operates with a “parent” table that holds no data and a “child” table that defines the partitions themselves and holds the data. The parent table is created first, the child tables is then defined with corresponding constraints to create the partition. You must supply a trigger to insert into the parent table and have the data be routed to the correct partition. For more information, see [Strategy for Migrating Partitioned Tables from Oracle to Amazon RDS for PostgreSQL and Amazon Aurora with PostgreSQL Compatibility](#).

Data Types

Some PostgreSQL data types are much easier to work with than their corresponding Oracle types. For example, the Text type can store up to 1 GB of text and can be handled in SQL just like the char and varchar fields. They don’t require special large object functions like character large objects (CLOBs) do.

However, there are some important differences to note. You can use the Numeric field in PostgreSQL to map any Number data types. But when you use it for joins (such as for a foreign key), it is less performant than using an int or bigint data type. This is a typical area where custom data type mapping should be considered.

The PostgreSQL Timestamp with time zone field is slightly different from and corresponds to the Oracle Timestamp with local time zone. These small differences can cause either performance issues or subtle application bugs that require thorough testing.
For more information, see Migration tips for developers converting Oracle and SQL Server code to PostgreSQL.

Transaction Control and Exception Handling

PostgreSQL Multiversion Concurrency Control (MVCC) is very different from Oracle rollback segments, even though they both provide ACID transactions. PostgreSQL creates a snapshot state taken at the start of the transaction, and essentially copies the data to a temporary page while the transaction is in flight. This can affect both the queries and the application, as well as hardware considerations. Because open transactions require temporary space to hold their snapshot state during the transaction, a workload that requires many open transactions in addition to possible deadlocks, needs to consider the location and parameters surrounding temporary table space.

Unlike Oracle, PostgreSQL uses auto-commit for transactions by default. However, there are two options to support explicit transactions, which are similar to the default behavior in Oracle (non-auto-commit). You can use the START TRANSACTION or BEGIN TRANSACTION statements and then COMMIT or ROLLBACK; or you can simply set AUTOCOMMIT to OFF at the session or system level.

PostgreSQL does not allow transaction control inside of PL/pgSQL like commit or roll back inside a stored procedure. The caller must perform the transaction management. If your existing PL/SQL contains explicit commit and rollback code it must be modified. When a run-time exception has occurred in a transaction, you must roll back that transaction before you can execute any new statement on the connection. Finally, exception handling in PL/pgSQL, using a BEGIN...EXCEPTION...END block to let your code catch any errors that occur. This block automatically creates a savepoint before the block, and rolls back to that savepoint when an exception occurs. You can then determine what logic to execute based on whether there was an error. Exception blocks are expensive however, due to the created savepoint. If you don’t need to catch an error, or if you are planning to simply raise the error back to the calling application, don’t use the exception block at all. Let the original error flow up to the application.

For more information, see Oracle Database 19c To Amazon Aurora with PostgreSQL Compatibility (12.4) Migration Playbook.

Migrating from SAP ASE to Amazon Aurora MySQL

Following, you can find a high-level outline and a step-by-step walkthrough that show the migration process of an on-premises SAP ASE database to Amazon Aurora MySQL-Compatible Edition using AWS Database Migration Service (AWS DMS). Amazon Aurora is a highly available and managed relational database service with automatic scaling and high-performance features. The combination of MySQL compatibility with Aurora enterprise database capabilities provides an ideal target for commercial database migrations.

This walkthrough covers all steps in the migration from initial analysis of the source database to final cutover of applications to the target database.

The following diagram shows the basic architecture for the migration.
We use the **pubs2** database for SAP ASE as the example database in the rest of this document.

**Topics**
Database Migration Guide Step-by-Step Walkthroughs

Prerequisites

The following prerequisites are required to complete this walkthrough:

- Familiarity with Amazon Relational Database Service (Amazon RDS), the applicable database technologies, and SQL.
- Understand the supported features and limitations of AWS Database Migration Service (AWS DMS). For more information, see What Is Database Migration Service?
- Accomplish the prerequisites required for using an SAP ASE database as a source for AWS DMS. For more information, see Prerequisites for using an SAP ASE database as a source.
- Understand the limitations on using SAP ASE as a source and MySQL as a target for AWS DMS. For more information, see Limitations on using SAP ASE as a source and Limitations on using a MySQL-compatible database as a target.
- Accomplish the prerequisites required for using a MySQL-compatible database as a target for AWS DMS. For more information, see Using a MySQL-compatible database as a target.
- Set up the network for AWS DMS replication. This includes configuring VPC, private subnets, availability zone, and adding connections on the source firewall if it exists. For more information, see Setting up a network for a replication instance.
- Download and install AWS Schema Conversion Tool (AWS SCT) with the required SAP ASE and MySQL JDBC drivers. For more information, see Installing, verifying, and updating the Schema Conversion Tool.
- Know the recommendations on the most efficient way to use AWS DMS. For more information, see Best practices.

Preparation and Assessment

Preparation and assessment of your source database is the initial phase. Before you start moving data, you should monitor and analyze the source database schema for the data lifecycle. To provide the best migration solution, you should have a good understanding on the workload, data access patterns, and data dependencies. Make sure to consider the following items:

- Character set.
- Largest table size.
- Largest LOB size.
- Integration with other databases or OS.

Determine the Character Set

To find out the default character set and sort order for your SAP ASE database, run the following query:

```
exec sp_default_charset
```

If your application uses a different character set, you can find it from your session by checking the global variable @@client_csname or @@client_csid.
If you have a non-default character set that you want to migrate, use an extra connect attribute to specify the character set for the source database. For example, if your default character set is UTF8, specify `charset=utf8` as an extra connect attribute to correctly migrate data.

### Determine the Largest Table Size

Explore your largest and busiest tables to find out their size and rate of change. This gives you an accurate estimate of where time will be spent when you do the initial data migration using the AWS Database Migration Service (AWS DMS) full load feature.

You can parallelize the load on the table level with one task to save time by using parallel load feature. For more information, see Using parallel load for selected tables, views, and collections.

In SAP ASE, you can run only one replication thread for each database. Because of that, you can start one AWS DMS task at one time for each database. You can't run multiple tasks, which is common when you migrate from other database engines. For more information, see Limitations on using SAP ASE as a source.

For SAP ASE version 15 and later, query `sysobjects` to list the top 10 in row count and space used. Use the following query:

```sql
select top 10 convert(varchar(30), o.name) AS table_name, row_count(db_id(), o.id) AS row_count, data_pages(db_id(), o.id, 0) AS pages, data_pages(db_id(), o.id, 0) * (@@maxpagesize/1024) AS kbs
from sysobjects o
where type = 'U'
order by kbs DESC, table_name ASC
```

The output of this query is shown following.

<table>
<thead>
<tr>
<th>table_name</th>
<th>row_count</th>
<th>pages</th>
<th>kbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>salesdetail</td>
<td>116</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

### Determine the Largest LOB Size

Large objects (LOBs) typically take the longest to migrate, unlike data types such as number and character, because of time spent encoding, storing, decoding, and retrieving them. You should identify tables with the TEXT, UNITEXT, and IMAGE data types, because AWS DMS converts these objects to LOB.

AWS DMS recommends to identify the size of LOB columns and choose limited or full mode, and max LOB size in the LOB settings appropriately. You can use the following dynamic SQL to generate a query for each table.

```sql
select 'select max(datalength('+ c.name +'))/1024 KB_SIZE from dbo.'+ o.name+';' from sysobjects o, syscolumns c
where o.type = 'U' and o.id = c.id and c.type in (34,35,174);
```

The output of this query is shown following.

```sql
select max(datalength(pic))/1024 KB_SIZE from dbo.au_pix;
select max(datalength(copy))/1024 KB_SIZE from dbo.blurbs;
```

After you run the preceding queries, compare the results and choose the top one.
For example, we found the largest LOB column in our SAP ASE database is 51 KB. We used this number as input in our task settings with limited LOB mode.

The speed of the full load is improved with the limited LOB mode compared to the full LOB mode. For performance reasons, AWS DMS recommends to use limited LOB mode and increase the maximum LOB size to cover the actual size you find from your query. For more information, see How can I improve the speed of a migration task that has LOB data?

Document Integrations with Other Databases or Applications

Replace remote objects or interfaces in your code with other AWS services or equivalent external services.

For example, in the SAP ASE database, you may send out email using the `xp_sendmail` procedure. Because Amazon Aurora MySQL doesn't provide native support for sending emails, redesign the process. For example, use an AWS Lambda function to send email from the database. For more information, see AWS Lambda and Sending notifications from Amazon Aurora MySQL.

Note
For database links from the source database to a remote server in SAP ASE, update data using foreign data wrappers (FDW).

Database Migration

This section covers two major database migration tasks: code conversion and data load. You can use AWS Schema Conversion Tool (AWS SCT) to convert database schema objects such as tables, views, procedures, functions, and so on. Then you can use AWS Database Migration Service (AWS DMS) to load data.

**Database Schema Conversion**

To convert your database schema and code objects from SAP ASE to Amazon Aurora MySQL, follow these steps.

1. Download and install AWS Schema Conversion Tool (AWS SCT) with the required SAP ASW and MySQL JDBC drivers. For more information, see https://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_Installing.html.
2. Create a new AWS SCT project, add your source and target databases, and add a mapping rule. For more information, see Creating a new project, Adding database servers, and Creating mapping rules.
3. Convert your database schema. For more information, see Converting database schemas.
4. Save the converted SQL scripts. For more information, see Saving and applying your converted schema.
5. Run these scripts against your target MySQL database. First, create tables with primary keys only. Then add the foreign keys and secondary indexes after you complete the full load.

**Migrate an SAP ASE Database to Amazon Aurora MySQL Using AWS DMS**

This section covers the steps that you follow to migrate an SAP ASE database to Amazon Aurora MySQL using AWS DMS.

AWS DMS creates the schema in the target if the schema doesn't exist. However, AWS DMS only creates the tables with primary keys. AWS DMS doesn't create foreign keys or secondary indexes. Even the default values may be missing.
The best practice is to create the schema objects using the scripts that AWS SCT generated in the prior step, then start AWS DMS to load table data.

Create a Replication Instance

To start data migration, create an AWS DMS replication instance. For performance reasons, AWS DMS recommends creating the replication instance in the same AWS Region as your target Amazon Aurora database. For more information, see Creating a replication instance.

Create a Source Endpoint

Create a source endpoint for SAP ASE and test the connection using the preceding replication instance.

- On the AWS DMS console, choose Endpoints.
- Choose Create endpoint.
- For Endpoint type, select Source endpoint.
- Enter your desired endpoint configuration.
## Endpoint configuration

### Endpoint identifier  
A label for the endpoint to help you identify it.

**Source-SAP-ASE-Database**

### Descriptive Amazon Resource Name (ARN) - optional  
A friendly name to override the default DMS ARN. You cannot modify it after creation.

**Friendly-ARN-name**

### Source engine  
The type of database engine this endpoint is connected to.

**SAP Sybase ASE**

### Access to endpoint database  

- [ ] **AWS Secrets Manager**  
- [x] **Provide access information manually**

### Server name  

**Server name**

### Port  
The port the database runs on for this endpoint.

**5000**

### Secure Socket Layer (SSL) mode  
The type of Secure Socket Layer enforcement.

**none**

You can use your own on-premises name server and a hostname instead of the IP address. For more information, see Using your own on-premises name server.

- Select the endpoint that you created, and choose **Test connection** from the **Actions** drop-down menu.

To use Transport Layer Security (TLS) for an SAP ASE database version 15.7 and higher, use the Adaptive Server Enterprise 16.03.06 extra connection attribute (ECA) provider. Use the following example:

```plaintext
provider=Adaptive Server Enterprise 16.03.06;
```

Make sure that you open the database port to the IP range of your replication instance before you test the connection. If the firewall is open but you still experience a connection issue, please contact AWS support.
Create a Target Endpoint

Create a target endpoint for your Amazon Aurora MySQL database.

- On the AWS DMS console, choose **Endpoints**.
- Choose **Create endpoint**.
- For **Endpoint type**, select **Target endpoint**.
- Enter your desired endpoint configuration.

**Endpoint configuration**

**Endpoint identifier**  
*Info*  
A label for the endpoint to help you identify it.

Target-Aurora-MySQL-Database

**Descriptive Amazon Resource Name (ARN) – optional**  
A friendly name to override the default DMS ARN. You cannot modify it after creation.

*Friendly-ARN-name*

**Target engine**  
The type of database engine this endpoint is connected to.

Amazon Aurora MySQL

**Access to endpoint database**

- [ ] AWS Secrets Manager
- [x] Provide access information manually

**Server name**

aurora-mysql.cluster-exampleclustername.eu-west-1.rds.amazonaws.com

**Port**  
The port the database runs on for this endpoint.

3306

**Secure Socket Layer (SSL) mode**  
The type of Secure Socket Layer enforcement.

none

**User name**  
*Info*  

mysqluser

**Password**  
*Info*  

[Redacted]

- Test the connection using the preceding replication instance.
To establish the connection, make sure that you edit the security group for your Amazon Aurora DB instance. Also, open the 3306 port on your MySQL database to the private IP or IP range of the replication instance.

- On the Amazon Relational Database Service (Amazon RDS) console, choose your Amazon Aurora MySQL DB instance.
- On the Connectivity & security tab, locate your security group name under Security.
- Choose the security group link. A new security group interface page opens.
- Choose Inbound rules.
- Choose Edit inbound rules.
- Add the IP range of the replication instance.

**Create a Migration Task**

Create a migration task using the source and target endpoints that you created on the preceding step. For more information, see Creating a task. After you create your task, AWS DMS sets its status to Ready. When you start or resume the task, AWS DMS changes the status to Starting or Running.

To monitor the process, choose Task Monitoring, Table Statistics, Logs. For more information, see Monitoring Database Migration Service metrics and How can I enable monitoring for an database migration task?.

**Cutover Procedures**

When the AWS DMS task finishes the full load and applies cached changes, the task moves to the change data capture (CDC) stage. At this point, you can perform the cutover to Amazon Aurora. You run SQL queries to validate data and use AWS services to set up backup and monitor jobs.

To perform the cutover, do the following:

- Analyze the database queries in Amazon Aurora MySQL and test the performance of critical queries.
- Shut down all the application servers and stop all the client connections to SAP ASE. Close any user sessions if necessary.
- Verify that the target data has been synced with the source database.
- Stop the AWS DMS task.
- Create the foreign keys and secondary indexes in Amazon Aurora MySQL if you didn’t create them before the CDC stage started.
- Validate tables, views, procedures, functions, and triggers within your schema.
- Switch the application servers, clients, and jobs to the Amazon Aurora MySQL database.
- Create the CloudWatch alarms based on your desired DB metrics. For more information, see Key Metrics for Amazon Aurora and Monitoring an Amazon Aurora DB Cluster.
- Add a reader node to an existing Amazon Aurora MySQL cluster. By default, Amazon Aurora replicates data across three Availability Zones in one Region at the storage level. This architecture is fault tolerant by design. For enhanced availability, add a reader node for a production database to automate failover in case of instance failure. Modify the database cluster to enable failover in case of instance failure. For more information, see High Availability for Amazon Aurora.

**Troubleshooting**

For more information about troubleshooting issues with AWS DMS, see Troubleshooting migration tasks in Database Migration Service.
For more information about troubleshooting issues specific to using AWS DMS with SAP ASE databases, see Troubleshooting issues with SAP ASE.

For more information about troubleshooting issues specific to using AWS DMS with Amazon Aurora MySQL databases, see Troubleshooting issues with MySQL and Troubleshooting issues with Amazon Aurora MySQL.

Best Practices

- When the AWS Database Migration Service (AWS DMS) task completes the full load, AWS DMS stops this task. You can take this opportunity to add or enable your foreign keys or constraints and triggers in your target. If your migration type is Migrate existing data and replicate ongoing changes, resume the task to pick up the cached changes.

- When you have tasks running, you can monitor the on-premises source host, your replication instance, and your target Amazon Aurora database. Make sure that you create alarms and get notified on key metrics, such as central processing unit (CPU) utilization, freeable memory, and IOPS.

- Before you start the AWS DMS migration, make sure that you disables foreign keys and triggers on the target database. Additionally, make sure that your AWS account has privileges on AWS DMS and Amazon Relational Database Service (Amazon RDS).

- AWS DMS recommends to periodically monitor the exception tables using the following query:

```
SELECT STATEMENT FROM admin."awsdms_apply_exceptions" WHERE TASK_NAME IN ('TASK NAME')
```

- Monitoring the AWS DMS CloudWatch log for errors and warnings. In case of any problem with migration, AWS DMS records a corresponding warning or error message in the log.

- Set up monitoring of source and target database latency to understand the replication lag.

- Use the AWS DMS data validation feature to detect data issues.

- Test the steps designed for migration to understand any unforeseen issues.

This walkthrough covers proven end-to-end steps for migrating an SAP ASE database to Amazon Aurora MySQL using AWS DMS. The walkthrough also includes basic instructions that show how to perform a similar migration.
Migrating Databases to the Amazon Web Services Cloud Using the Database Migration Service

You can use several AWS tools and services to migrate data from an external database to AWS. Depending on the type of database migration you are doing, you may find that the native migration tools for your database engine are also effective.

AWS Database Migration Service (AWS DMS) helps you migrate databases to AWS efficiently and securely. The source database can remain fully operational during the migration, minimizing downtime to applications that rely on the database. AWS DMS can migrate your Oracle data to the most widely used commercial and open-source databases on AWS.

AWS DMS migrates data, tables, and primary keys to the target database. All other database elements are not migrated. If you are migrating an Oracle database to Amazon Aurora MySQL-Compatible Edition, for example, you would want to use the AWS Schema Conversion Tool in conjunction with AWS DMS.

The AWS Schema Conversion Tool (AWS SCT) makes heterogeneous database migrations easy by automatically converting the source database schema and a majority of the custom code, including views, stored procedures, and functions, to a format compatible with the target database. Any code that cannot be automatically converted is clearly marked so that it can be manually converted. You can use this tool to convert your source Oracle databases to an Amazon Aurora MySQL, MySQL, or PostgreSQL target database on either Amazon RDS or EC2.

It is important to understand that AWS DMS and AWS SCT are two different tools and serve different needs.

- AWS DMS takes a minimalist approach and creates only those objects required to efficiently migrate the data for example tables with primary key. Therefore, we will use DMS to load the tables with data without any foreign keys or constraints. (We can also use the SCT to generate the table scripts and create it on the target before performing the load via DMS).
- We use AWS SCT:
  - To identify the issues, limitations and actions for the schema conversion
  - To generate the target schema scripts including foreign key and constraints
  - To convert code such as procedures and views from source to target and apply it on target

The size and type of Oracle database migration you want to do greatly determines the tools you should use. For example, a heterogeneous migration, where you are migrating from an Oracle database to a different database engine on AWS, is best accomplished using AWS DMS. A homogeneous migration, where you are migrating from an Oracle database to an Oracle database on AWS, is best accomplished using native Oracle tools.

Topics
- Migrating an On-Premises Oracle Database to Amazon Aurora MySQL (p. 78)
- Migrating an Amazon RDS for Oracle Database to Amazon Aurora MySQL (p. 97)
Migrating an On-Premises Oracle Database to Amazon Aurora MySQL

Following, you can find a high-level outline and also a complete step-by-step walkthrough that both show the process for migrating an on-premises Oracle database (the source endpoint) to an Amazon Aurora MySQL-Compatible Edition (the target endpoint) using AWS Database Migration Service (AWS DMS) and the AWS Schema Conversion Tool (AWS SCT).

AWS DMS migrates your data from your Oracle source into your Aurora MySQL target. AWS DMS also captures data manipulation language (DML) and data definition language (DDL) changes that happen on your source database and apply these changes to your target database. This way, AWS DMS helps keep your source and target databases in sync with each other. To facilitate the data migration, DMS creates tables and primary key indexes on the target database if necessary.

However, AWS DMS doesn't migrate your secondary indexes, sequences, default values, stored procedures, triggers, synonyms, views and other schema objects not specifically related to data migration. To migrate these objects to your Aurora MySQL target, use the AWS Schema Conversion Tool.

We highly recommend that you follow along using the Amazon sample database. To find a tutorial that uses the sample database and instructions on how to get a copy of the sample database, see Working with the Sample Database for Migration (p. 96).

If you've used AWS DMS before or you prefer clicking a mouse to reading, you probably want to work with the high-level outline. If you need the details and want a more measured approach (or run into questions), you probably want the step-by-step guide.

| Topic: Migration from On-Premises Oracle to Aurora MySQL or Amazon RDS for MySQL |
| Time: |
| Cost: |
| Source Database: Oracle |
| Target Database: Amazon Aurora MySQL/MySQL |
| Restrictions: |
| Oracle Edition: Enterprise, Standard, Express and Personal |
Topic: Migration from On-Premises Oracle to Aurora MySQL or Amazon RDS for MySQL

Oracle Version: 10g (10.2 and later), 11g, 12c, (On Amazon Relational Database Service (Amazon RDS), 11g or higher is required.)

MySQL or Related Database Version: 5.5, 5.6, 5.7, MariaDB, Amazon Aurora MySQL

Costs

Because AWS DMS isn't incorporated into the calculator yet, see the following table for a pricing estimate.

In addition to the setup on your own PC, you must create several AWS components to complete the migration process. The AWS components include:

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Aurora MySQL DB instance</td>
<td>db.r3.large</td>
<td>Single AZ, 10 GB storage, 1 million I/O</td>
</tr>
<tr>
<td>AWS DMS replication instance</td>
<td>T2.large</td>
<td>50 GB of storage for keeping replication logs included</td>
</tr>
<tr>
<td>AWS DMS data transfer</td>
<td>Free, based on the amount of data transferred for the sample database.</td>
<td></td>
</tr>
<tr>
<td>Data transfer out</td>
<td>First 1 GB per month free</td>
<td></td>
</tr>
</tbody>
</table>

Migration High-Level Outline

To migrate your data from Oracle to Aurora MySQL using AWS DMS, you take the following steps. If you've used AWS DMS before or prefer clicking a mouse to reading, the following summary should help you kick-start your migration. To get the details about migration or if you run into questions, see the step-by-step guide.

Step 1: Prepare Your Oracle Source Database

To use AWS DMS to migrate data from an Oracle source database requires some preparation and we also recommend a few additional steps as best practices.

- AWS DMS account — It's a good practice to create a separate account for the specific purpose of migrating your data. This account should have the minimal set of privileges required to migrate your data. You can find specific details regarding those privileges later. If you are simply interested in testing AWS DMS on a non-production database, any DBA account will be sufficient.
- Supplemental logging — To capture changes, you must enable supplemental logging in order to use DMS. To enable supplemental logging at the database level issue the following command.

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA
```

Additionally, AWS DMS requires for each table being migrated, you set at least key-level supplemental logging. AWS DMS automatically adds this supplemental logging for you if you include the following extra connection parameter for your source connection.
Database Migration Guide Step-by-Step Walkthroughs
Migration High-Level Outline

addSupplementalLogging=Y

- Source database – To migrate your data, the AWS DMS replication server needs access to your source database. Make sure that your firewall rules give the AWS DMS replication server ingress.

Step 2: Launch and Prepare Your Aurora MySQL Target Database

Following are some things to consider when launching your Aurora MySQL instance:

- For best results, we recommend that you locate your Aurora MySQL instance and your replication instance in the same VPC and, if possible, the same Availability Zone.
- We recommend that you create a separate account with minimal privileges for migrating your data. The AWS DMS account needs the following privileges on all databases to which data is being migrated.

ALTER, CREATE, DROP, INDEX, INSERT, UPDATE, DELETE, SELECT

Additionally, AWS DMS needs complete access to the awsdms_control database. This database holds information required by AWS DMS specific to the migration. To provide access, run the following command.

ALL PRIVILEGES ON awsdms_control.* TO 'dms_user'

Step 3: Launch a Replication Instance

The AWS DMS service connects to your source and target databases from a replication instance. Here are some things to consider when launching your replication instance:

- For best results, we recommend that you locate your replication instance in the same VPC and Availability Zone as your target database, in this case Aurora MySQL.
- If either your source or target database is outside of the VPC where you launch your replication server, the replication server must be publicly accessible.
- AWS DMS can consume a fair bit of memory and CPU. However, it’s easy enough to scale up if necessary. If you anticipate running several tasks on a single replication server or if your migration involves a large number of tables, consider using one of the larger instances.
- The default storage is usually enough for most migrations.

Step 4: Create a Source Endpoint

For AWS DMS to access your Oracle source database you’ll need to create a source endpoint. The source endpoint defines all the information required for AWS DMS to connect to your source database from the replication server. Following are some requirements for the source endpoint.

- Your source endpoint needs to be accessible from the replication server. To allow this, you will likely need to modify your firewall rules to whitelist the replication server. You can find the IP address of your replication server in the AWS DMS Management Console.
- For AWS DMS to capture changes, Oracle requires supplemental logging be enabled. If you want AWS DMS to enable supplemental logging for you, add the following to the extra connection attributes for your Oracle source endpoint.

addSupplementalLogging=Y
Step 5: Create a Target Endpoint

For AWS DMS to access your Aurora MySQL target database you'll need to create a target endpoint. The target endpoint defines all the information required for DMS to connect to your Aurora MySQL database.

- Your target endpoint needs to be accessible from the replication server. You might need to modify your security groups to make the target endpoint accessible.
- If you've pre-created the database on your target, it's a good idea to disable foreign key checks during the full load. To do so, add the following to your extra connection attributes.

```
initstmt=SET FOREIGN_KEY_CHECKS=0
```

Step 6: Create and Run a Migration Task

A migration task tells AWS DMS where and how you want your data migrated. When you create a migration task, consider setting migration parameters as shown following.

Endpoints and replication server — Choose the endpoints and replication server created before.

Migration type — In most cases you'll want to choose migrate existing data and replication ongoing changes. With this option, AWS DMS loads your source data while capturing changes to that data. When the data is fully loaded, AWS DMS applies any outstanding changes and keeps the source and target databases in sync until the task is stopped.

Target table preparation mode — If you're having AWS DMS create your tables, choose drop tables on target. If you're using some other method to create your target tables such as the AWS Schema Conversion Tool, choose truncate.

LOB parameters — If you're just trying AWS DMS, choose include LOB columns in replication, Limited LOB mode, and set your max LOB size to 16 (which is 16k.) For more information regarding LOBs, read the details in the step-by-step guide.

Enable logging — To help with debugging migration issues, always enable logging.

Table mappings — When migrating from Oracle to Aurora MySQL, we recommend that you convert your schema, table, and column names to lowercase. To do so, create a custom table mapping. The following example migrates the schema DMS_SAMPLE and converts schema, table and column names to lowercase.

```json
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "DMS_SAMPLE",
                "table-name": "%"
            },
            "rule-action": "include"
        },
        {
            "rule-type": "transformation",
            "rule-id": "6",
            "rule-name": "6",
            "rule-action": "convert-lowercase",
            "rule-target": "schema",
            "object-locator": {
```
Migration Step-by-Step Guide

Following, you can find step-by-step instructions for migrating an Oracle database from an on-premises environment to Amazon Aurora MySQL. These instructions assume that you have already done the setting up steps for using AWS DMS located at https://docs.aws.amazon.com/dms/latest/userguide/CHAP_SettingUp.html.

Topics

• Step 1: Configure Your Oracle Source Database (p. 82)
• Step 2: Configure Your Aurora Target Database (p. 84)
• Step 3: Creating a Replication Instance (p. 85)
• Step 4: Create Your Oracle Source Endpoint (p. 87)
• Step 5: Create Your Aurora MySQL Target Endpoint (p. 90)
• Step 6: Create a Migration Task (p. 91)
• Step 7: Monitor Your Migration Task (p. 96)
• Troubleshooting (p. 96)

Step 1: Configure Your Oracle Source Database

To use Oracle as a source for AWS Database Migration Service (AWS DMS), you must first ensure that ARCHIVELOG MODE is on to provide information to LogMiner. AWS DMS uses LogMiner to read information from the archive logs so that AWS DMS can capture changes.

For AWS DMS to read this information, make sure the archive logs are retained on the database server as long as AWS DMS requires them. If you configure your task to begin capturing changes immediately, you should only need to retain archive logs for a little longer than the duration of the longest running transaction. Retaining archive logs for 24 hours is usually sufficient. If you configure your task to begin from a point in time in the past, archive logs need to be available from that time forward. For more
specific instructions for enabling ARCHIVELOG MODE and ensuring log retention for your on-premises Oracle database see the Oracle documentation.

To capture change data, AWS DMS requires supplemental logging to be enabled on your source database for AWS DMS. Minimal supplemental logging must be enabled at the database level. AWS DMS also requires that identification key logging be enabled. This option causes the database to place all columns of a row's primary key in the redo log file whenever a row containing a primary key is updated (even if no value in the primary key has changed). You can set this option at the database or table level.

If your Oracle source is in Amazon RDS, your database will be placed in ARCHIVELOG MODE if, and only if, you enable backups. The following command will ensure archive logs are retained on your RDS source for 24 hours:

```
exec rdsadmin.rdsadmin_util.set_configuration('archivelog retention hours',24);
```

To configure your Oracle source database, do the following:

1. **Enable database-level supplemental logging**

Run the following command to enable supplemental logging at the database level, which AWS DMS requires:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

For RDS:
```
exec rdsadmin.rdsadmin_util.alter_supplemental_logging('ADD');
```

2. **Enable identification key supplemental logging**

Use the following command to enable identification key supplemental logging at the database level. AWS DMS requires supplemental key logging at the database level unless you allow AWS DMS to automatically add supplemental logging as needed or enable key-level supplemental logging at the table level:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
```

For RDS:
```
exec rdsadmin.rdsadmin_util.alter_supplemental_logging('ADD','PRIMARY KEY');
```

3. **(Optional) Enable key level supplemental logging at the table level**

Your source database incurs a small bit of overhead when key level supplemental logging is enabled. Therefore, if you are migrating only a subset of your tables, you might want to enable key level supplemental logging at the table level. To enable key level supplemental logging at the table level, use the following command.

```
alter table table_name add supplemental log data (PRIMARY KEY) columns;
```

If a table does not have a primary key you have two options:

- You can add supplemental logging to all columns involved in the first unique index on the table (sorted by index name.)
- You can add supplemental logging on all columns of the table.

To add supplemental logging on a subset of columns in a table, that is those involved in a unique index, run the following command.

```
ALTER TABLE table_name ADD SUPPLEMENTAL LOG GROUP example_log_group (ID,NAME)
```
Always;

To add supplemental logging for all columns of a table, run the following command.

```sql
alter table table_name add supplemental log data (ALL) columns;
```

4. Create or configure a database account to be used by AWS DMS

We recommend that you use an account with the minimal privileges required by AWS DMS for your AWS DMS connection. AWS DMS requires the following privileges.

```sql
CREATE SESSION
SELECT ANY TRANSACTION
SELECT on V_$ARCHIVED_LOG
SELECT on V_$LOG
SELECT on V_$LOGFILE
SELECT on V_$DATABASE
SELECT on V_$THREAD
SELECT on V_$PARAMETER
SELECT on V_$NLS_PARAMETERS
SELECT on V_$TIMEZONE_NAMES
SELECT on V_$TRANSACTION
SELECT on ALL_INDEXES
SELECT on ALL_OBJECTS
SELECT on ALL_TABLES
SELECT on ALL_USERS
SELECT on ALL_CATALOG
SELECT on ALL_CONSTRAINTS
SELECT on ALL_CONS_COLUMNS
SELECT on ALL_TAB_COLS
SELECT on ALL_IND_COLUMNS
SELECT on ALL_LOG_GROUPS
SELECT on SYS.DBA_REGISTRY
SELECT on SYS.OBJ$
SELECT on DBA_TABLESPACES
SELECT on ALL_TAB_PARTITIONS
SELECT on ALL_ENCRYPTED_COLUMNS
* SELECT on all tables migrated
```

If you want to capture and apply changes (CDC) you also need the following privileges.

```sql
EXECUTE on DBMS_LOGMNR
SELECT on V_$LOGMNR_LOGS
SELECT on V_$LOGMNR_CONTENTS
LOGMINING /* For Oracle 12c and higher. */
* ALTER for any table being replicated (if you want to add supplemental logging)
```

For Oracle versions before 11.2.0.3, you need the following privileges. If views are exposed, you need the following privileges.

```sql
SELECT on DBA_OBJECTS /* versions before 11.2.0.3 */
SELECT on ALL_VIEWS (required if views are exposed)
```

**Step 2: Configure Your Aurora Target Database**

As with your source database, it's a good idea to restrict access of the user you're connecting with. You can also create a temporary user that you can remove after the migration.

```sql
CREATE USER 'dms_user'@'%' IDENTIFIED BY 'dms_user';
```
GRANT ALTER, CREATE, DROP, INDEX, INSERT, UPDATE, DELETE, SELECT ON <target database(s)>.* TO 'dms_user'@'%';

AWS DMS uses some control tables on the target in the database awsdms_control. The following command ensures that your dms_user has the necessary access to the awsdms_control database:

GRANT ALL PRIVILEGES ON awsdms_control.* TO 'dms_user'@'%';
flush privileges;

### Step 3: Creating a Replication Instance

An AWS DMS replication instance performs the actual data migration between source and target. The replication instance also caches the changes during the migration. How much CPU and memory capacity a replication instance has influences the overall time required for the migration. Use the following procedure to set the parameters for a replication instance.

To create an AWS DMS replication instance, do the following:

1. Sign in to the AWS Management Console, and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/ and choose Replication instances. If you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information on the permissions required, see IAM Permissions.
2. Choose Create replication instance.
3. On the Create replication instance page, specify your replication instance information as shown following.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>If you plan to launch multiple replication instances or share an account, choose a name that helps you quickly differentiate between the different replication instances.</td>
</tr>
<tr>
<td>Description</td>
<td>A good description gives others an idea of what the replication instance is being used for and can prevent accidents.</td>
</tr>
<tr>
<td>Instance class</td>
<td>AWS DMS can use a fair bit of memory and CPU. If you have a large database (many tables) or use a number of LOB data types, setting up a larger instance is probably better. As described following, you might be able to boost your throughput by running multiple tasks. Multiple tasks consume more resources and require a larger instance. Keep an eye on CPU and memory consumption as you run your tests. If you find you are using the full capacity of the CPU or swap space, you can easily scale up.</td>
</tr>
<tr>
<td>VPC</td>
<td>Here you can choose the VPC where your replication instance will be launched. We recommend that, if possible, you select the same VPC where either your source or target database is (or both). AWS DMS needs to access your source and target database from within this VPC. If either or both of your database endpoints are</td>
</tr>
</tbody>
</table>
### Multi-AZ
If you choose Multi-AZ, AWS DMS launches a primary and secondary replication instance in separate Availability Zones. In the case of a catastrophic disk failure, the primary replication instance automatically fails over to the secondary, preventing an interruption in service. In most situations, if you are performing a migration, you won't need Multi-AZ. If your initial data load takes a long time and you need to keep the source and target databases in sync for a significant portion of time, you might consider running your migration server in a Multi-AZ configuration.

### Publicly accessible
If either your source or your target database are outside of the VPC where your replication instance is, you need to make your replication instance publicly accessible.

4. In the **Advanced** section, set the **Allocated storage (GB)** parameter, and then choose **Next**.

### Allocated storage (GB)
Storage is primarily consumed by log files and cached transactions. For caches transactions, storage is used only when the cached transactions need to be written to disk. Therefore, AWS DMS doesn't use a significant amount of storage. Some exceptions include the following:

- **Very large tables that incur a significant transaction load.** Loading a large table can take some time, so cached transactions are more likely to be written to disk during a large table load.

- **Tasks that are configured to pause prior to loading cached transactions.** In this case, all transactions are cached until the full load completes for all tables. With this configuration, a fair amount of storage might be consumed by cached transactions.

- **Tasks configured with tables being loaded into Amazon Redshift.** However, this configuration isn't an issue when Aurora MySQL is the target.

In most cases, the default allocation of storage is sufficient. However, it’s always a good idea to pay attention to storage related metrics and scale up your storage if you find you are consuming more than the default allocation.
<table>
<thead>
<tr>
<th>For This Option</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication Subnet Group</td>
<td>If you run in a Multi-AZ configuration, you need at least two subnet groups.</td>
</tr>
<tr>
<td>Availability Zone</td>
<td>If possible, locate your primary replication server in the same Availability Zone as your target database.</td>
</tr>
<tr>
<td>VPC Security group(s)</td>
<td>With security groups you can control ingress and egress to your VPC. With AWS DMS you can associate one or more security groups with the VPC where your replication server launches.</td>
</tr>
<tr>
<td>KMS key</td>
<td>With AWS DMS, all data is encrypted at rest using a KMS encryption key. By default, AWS DMS creates a new encryption key for your replication server. However, you can use an existing key if desired.</td>
</tr>
</tbody>
</table>

**Step 4: Create Your Oracle Source Endpoint**

While your replication instance is being created, you can specify the Oracle source endpoint using the AWS Management Console. However, you can only test connectivity after the replication instance has been created, because the replication instance is used to test the connection.

To specify source or target database endpoints, do the following:

1. In the AWS DMS console, choose **Endpoints** on the navigation pane.
2. Choose **Create endpoint**. The **Create database endpoint page** appears, as shown following.
3. Specify your connection information for the source Oracle database. The following table describes the source settings.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint type</td>
<td>Choose Source.</td>
</tr>
<tr>
<td>Endpoint Identifier</td>
<td>Enter an identifier for your Oracle endpoint. The identifier for your endpoint must be unique within an AWS Region.</td>
</tr>
<tr>
<td>Source Engine</td>
<td>Choose oracle.</td>
</tr>
<tr>
<td>Server name</td>
<td>If your database is on-premises, type an IP address that AWS DMS can use to connect to your database from the replication server. If your database is running on Amazon Elastic Compute Cloud (Amazon EC2) or Amazon RDS, type the public Domain Name Service (DNS) address.</td>
</tr>
<tr>
<td>Port</td>
<td>Enter the port which your database is listening for connections (the Oracle default is 1521).</td>
</tr>
<tr>
<td>SSL mode</td>
<td>Choose a Secure Sockets Layer (SSL) mode if you want to enable connection encryption for this endpoint. Depending on the mode you select,</td>
</tr>
</tbody>
</table>
For This Parameter | Do This
---|---
you might need to provide certificate and server certificate information.

Username | Enter the AWS account user name. We recommend that you create an AWS account specific to your migration.

Password | Provide the password for the user name preceding.

4. Choose the **Advanced** tab to set values for extra connection strings and the encryption key.

<table>
<thead>
<tr>
<th>For This Option</th>
<th>Do This</th>
</tr>
</thead>
</table>
| Extra connection attributes | Here you can add values for extra attributes that control the behavior of your endpoint. A few of the most relevant attributes are listed here. For the full list, see the documentation. Separate multiple entries from each other by using a semi-colon (;).

* **addSupplementalLogging**: AWS DMS will automatically add supplemental logging if you enable this option (addSupplementalLogging=Y).

* **useLogminerReader**: By default AWS DMS uses Oracle LogMiner to capture change data from the logs. AWS DMS can also parse the logs using its proprietary technology. If you use Oracle 12c and need to capture changes to tables that include LOBS, set this to No (useLogminerReader=N).

* **numberDataTypeScale**: Oracle supports a NUMBER data type that has no precision or scale. By default, NUMBER is converted to a number with a precision of 38 and scale of 10, number(38,10). Valid values are 0—38 or -1 for FLOAT.

* **archivedLogDestId**: This option specifies the destination of the archived redo logs. The value should be the same as the DEST_ID number in the $archived_log table. When working with multiple log destinations (DEST_ID), we recommend that you specify a location identifier for archived redo logs. Doing so improves performance by ensuring that the correct logs are accessed from the outset. The default value for this option is 0.

| KMS key | Choose the encryption key to use to encrypt replication storage and connection information. If you choose **(Default) aws/dms**, the default |
Before you save your endpoint, you can test it. To do so, select a VPC and replication instance from which to perform the test. As part of the test AWS DMS refreshes the list of schemas associated with the endpoint. (The schemas are presented as source options when creating a task using this source endpoint.)

**Step 5: Create Your Aurora MySQL Target Endpoint**

Next, you can provide information for the target Amazon Aurora MySQL database by specifying the target endpoint settings.

To specify a target database endpoint, do the following:

1. In the AWS DMS console, choose **Endpoints** on the navigation pane.
2. Choose **Create endpoint**. The **Create endpoint** appears, as shown following.
3. Specify your connection information for the target Aurora MySQL database. The following table describes the target settings.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Endpoint type</strong></td>
<td>Choose <strong>Target endpoint</strong>.</td>
</tr>
<tr>
<td><strong>Endpoint identifier</strong></td>
<td>Enter an identifier for your Aurora MySQL endpoint. The identifier for your endpoint must be unique within an AWS Region.</td>
</tr>
<tr>
<td><strong>Target engine</strong></td>
<td>Choose <strong>Amazon Aurora MySQL</strong>.</td>
</tr>
<tr>
<td><strong>Access to endpoint database</strong></td>
<td>Choose <strong>Provide access information manually</strong>.</td>
</tr>
<tr>
<td><strong>Server name</strong></td>
<td>Enter the writer endpoint for your Aurora MySQL instance. The writer endpoint is the primary instance.</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>Enter the port assigned to the instance.</td>
</tr>
<tr>
<td><strong>Secure Socket Layer (SSL) mode</strong></td>
<td>Choose an SSL mode if you want to enable connection encryption for this endpoint. Depending on the mode you select, you might need to provide certificate and server certificate information.</td>
</tr>
<tr>
<td><strong>User name</strong></td>
<td>Enter the user name for the account you are using for the migration. We recommend that you create an account specific to your migration.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Provide the password for the user name preceding.</td>
</tr>
</tbody>
</table>

4. Define additional specific settings for your endpoints using wizard or editor in **Endpoint settings**.
5. Choose the encryption key to use to encrypt replication storage and connection information in **KMS key**. If you choose **(Default) aws/dms**, the default AWS Key Management Service (AWS KMS) key associated with your account and region is used.
6. Add tags to organize your DMS resources in **Tags**. You can use tags to manage your IAM roles and policies, and track your DMS costs.

Prior to saving your endpoint, you have an opportunity to test it in **Test endpoint connection (optional)**. To do so you'll need to choose a VPC and replication instance from which to perform the test.

**Step 6: Create a Migration Task**

When you create a migration task you tell AWS DMS exactly how you want your data migrated. Within a task you define which tables you’d like migrated, where you’d like them migrated, and how you’d like them migrated. If you’re planning to use the change capture and apply capability of AWS DMS it’s important to know transactions are maintained within a single task. In other words, you should migrate all tables that participate in a single transaction together in the same task.

Using an AWS DMS task, you can specify what schema to migrate and the type of migration. You can migrate existing data, migrate existing data and replicate ongoing changes, or replicate data changes only. This walkthrough migrates existing data only.

To create a migration task, do the following:

1. On the navigation pane, choose **Tasks**.
2. Choose **Create Task**.
3. On the **Create Task** page, specify the task options. The following table describes the settings.

<table>
<thead>
<tr>
<th>For This Option</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Task name</strong></td>
<td>It’s always a good idea to give your task a descriptive name that helps organization.</td>
</tr>
<tr>
<td><strong>Task description</strong></td>
<td>Enter a description for the task.</td>
</tr>
<tr>
<td><strong>Source endpoint</strong></td>
<td>Select your source endpoint.</td>
</tr>
<tr>
<td><strong>Target endpoint</strong></td>
<td>Select your target endpoint.</td>
</tr>
<tr>
<td><strong>Replication instance</strong></td>
<td>Select a replication instance on which to run the task. Remember, your source and target endpoints must be accessible from this instance.</td>
</tr>
<tr>
<td><strong>Migration type</strong></td>
<td>You can use three different migration types with AWS DMS.</td>
</tr>
<tr>
<td></td>
<td>1. <strong>Migrate existing data</strong></td>
</tr>
<tr>
<td></td>
<td>If you select this option, AWS DMS migrates only your existing data. Changes to your source data aren’t captured and applied to your target. If you can afford taking an outage for the duration of the full load, migrating with this option is simple and straightforward. This method is also good to use when creating test copies of your database.</td>
</tr>
<tr>
<td></td>
<td>2. <strong>Migrate existing data and replicate ongoing changes</strong></td>
</tr>
<tr>
<td></td>
<td>With this option, AWS DMS captures changes while migrating your existing data. AWS DMS</td>
</tr>
</tbody>
</table>
### For This Option

<table>
<thead>
<tr>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>continues to capture and apply changes even after the bulk data has been loaded. Eventually the source and target databases will be in sync, allowing for a minimal downtime migration. To do this, take the following steps:</td>
</tr>
<tr>
<td>* Shut the application down</td>
</tr>
<tr>
<td>* Let the final change flow through to the target</td>
</tr>
<tr>
<td>* Perform any administrative tasks such as enabling foreign keys and triggers</td>
</tr>
<tr>
<td>* Start the application pointing to the new target database</td>
</tr>
</tbody>
</table>

Note that AWS DMS loads the bulk data table-by-table, `<n>` tables at a time. As the full load progresses, AWS DMS begins applying cached changes to the target tables as soon as possible. During the bulk load, referential integrity is violated, therefore existing foreign keys must be disabled for the full load. Once the full load is complete, your target database has integrity and changes are applied as transactions.

### 3. Replicate data changes only

In some cases you might choose to load bulk data using a different method. This approach generally only applies to homogeneous migrations.

#### Start task on create

<table>
<thead>
<tr>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>In most situations having the task start immediately is fine. Sometimes you might want to delay the start of a task, for instance, to change logging levels.</td>
</tr>
</tbody>
</table>

4. Next, set the Advanced settings as shown following.

<table>
<thead>
<tr>
<th>For This Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target table preparation mode</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS DMS allows you to specify how you would like your target tables prepared prior to loading.</td>
</tr>
</tbody>
</table>

**Do nothing** - When you select this option, AWS DMS does nothing to prepare your tables. Your table structure remains as is and any existing data is left in the table. You can use this method to consolidate data from multiple systems.

**Drop tables on target** - Typically you use this option when you want AWS DMS to create your target table for you. When you select this option, AWS DMS drops and recreates the tables to migrate before migration.
### For This Option

<table>
<thead>
<tr>
<th>For This Option</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Truncate</strong></td>
<td>Select this option if you want to pre-create some or all of the tables on your target system, maybe with the AWS Schema Conversion Tool. When you select this option, AWS DMS truncates a target table prior to loading it. If the target table doesn't exist, AWS DMS creates the table for you.</td>
</tr>
</tbody>
</table>

### Include LOB columns in replication

| Include LOB columns in replication | Large objects, (LOBs) can sometimes be difficult to migrate between systems. AWS DMS offers a number of options to help with the tuning of LOB columns. To see which and when datatypes are considered LOBS by AWS DMS, see the AWS DMS documentation. |
|-----------------------------------| Don't include LOB columns - When you migrate data from one database to another, you might take the opportunity to rethink how your LOBs are stored, especially for heterogeneous migrations. If you want to do so, there's no need to migrate the LOB data. |
| Full LOB mode                    | Full LOB mode - In full LOB mode AWS DMS migrates all LOBs from source to target regardless of size. In this configuration, AWS DMS has no information about the maximum size of LOBs to expect. Thus, LOBs are migrated one at a time, piece by piece. Full LOB mode can be quite slow. |
| Limited LOB mode                 | Limited LOB mode - In limited LOB mode, you set a maximum size LOB that AWS DMS should accept. Doing so allows AWS DMS to pre-allocate memory and load the LOB data in bulk. LOBs that exceed the maximum LOB size are truncated and a warning is issued to the log file. In limited LOB mode you get significant performance gains over full LOB mode. We recommend that you use limited LOB mode whenever possible. |
| Max LOB size (K)                 | When a task is configured to run in limited LOB mode, this option determines the maximum size LOB that AWS DMS accepts. Any LOBs that are larger than this value will be truncated to this value. |

Note that with Oracle, LOBs are treated as VARCHAR data types whenever possible. This approach means AWS DMS fetches them from the database in bulk, which is significantly faster than other methods. The maximum size of a VARCHAR in Oracle is 64K, therefore a limited LOB size of less than 64K is optimal when Oracle is your source database.
5. Set additional parameters.

<table>
<thead>
<tr>
<th>For This Option</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LOB chunk size (K)</strong></td>
<td>When a task is configured to use full LOB mode, AWS DMS retrieves LOBs in pieces. This option determines the size of each piece. When setting this option, pay particular attention to the maximum packet size allowed by your network configuration. If the LOB chunk size exceeds your maximum allowed packet size, you might see disconnect errors.</td>
</tr>
<tr>
<td><strong>Custom CDC start time</strong></td>
<td>This parameter pertains to tasks configured to replicate data changes only. It tells AWS DMS where to start looking for changes in the change stream.</td>
</tr>
<tr>
<td><strong>Enable logging</strong></td>
<td>Always enable logging.</td>
</tr>
</tbody>
</table>


Table mappings tell AWS DMS which tables a task should migrate from source to target. Table mappings are expressed in JSON, though some settings can be made using the AWS Management Console. Table mappings can also include transformations such as changing table names from the upper case to lower case.

AWS DMS generates default table mappings for each (non-system) schema in the source database. In most cases you’ll want to customize your table mapping. To customize your table mapping select the custom radio button. For details on creating table mappings see the AWS DMS documentation. The following table mapping does these things:

- It includes the DMS_SAMPLE schema in the migration.
- It excludes the tables NFL_DATA, MLB_DATA, NAME_DATE, and STADIUM_DATA.
- It converts the schema, table, and column names to lower case.

```json
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "DMS_SAMPLE",
        "table-name": "*"
      }
    }
  ]
}
```
"rule-action": "include"
],

[ "rule-type": "selection",
  "rule-id": "2",
  "rule-name": "2",
  "object-locator": {
    "schema-name": "DMS_SAMPLE",
    "table-name": "MLB_DATA"
  },
  "rule-action": "exclude"
],

[ "rule-type": "selection",
  "rule-id": "3",
  "rule-name": "3",
  "object-locator": {
    "schema-name": "DMS_SAMPLE",
    "table-name": "NAME_DATA"
  },
  "rule-action": "exclude"
],

[ "rule-type": "selection",
  "rule-id": "4",
  "rule-name": "4",
  "object-locator": {
    "schema-name": "DMS_SAMPLE",
    "table-name": "NFL_DATA"
  },
  "rule-action": "exclude"
],

[ "rule-type": "selection",
  "rule-id": "5",
  "rule-name": "5",
  "object-locator": {
    "schema-name": "DMS_SAMPLE",
    "table-name": "NFL_STADIUM_DATA"
  },
  "rule-action": "exclude"
],

[ "rule-type": "transformation",
  "rule-id": "6",
  "rule-name": "6",
  "rule-action": "convert-lowercase",
  "rule-target": "schema",
  "object-locator": {
    "schema-name": "%
  }
],

[ "rule-type": "transformation",
  "rule-id": "7",
  "rule-name": "7",
  "rule-action": "convert-lowercase",
  "rule-target": "table",
  "object-locator": {
    "schema-name": "%
  }
]
Step 7: Monitor Your Migration Task

Three sections in the console provide visibility into what your migration task is doing:

- Task monitoring — The **Task Monitoring** tab provides insight into your full load throughput and also your change capture and apply latencies.
- Table statistics — The **Table Statistics** tab provides detailed information on the number of rows processed, type and number of transactions processed, and also information on DDL operations.
- Logs — From the **Logs** tab you can view your task's log file, (assuming you turned logging on,) If for some reason your task fails, search this file for errors. Additionally, you can look in the file for any warnings. Any data truncation in your task appears as a warning in the log file. If you need to, you can increase the logging level by using the AWS Command Line Interface (AWS CLI).

Troubleshooting

The two most common areas people have issues with when working with Oracle as a source and Aurora MySQL as a target are: supplemental logging and case sensitivity.

- Supplemental logging — With Oracle, in order to replication change data supplemental logging must be enabled. However, if you enable supplemental logging at the database level, it sometimes still need to enable it when creating new tables. The best remedy for this is to allow DMS to enable supplemental logging for you using the extra connection attribute:

  ```
  addSupplementalLogging=Y
  ```

- Case sensitivity — Oracle is case-insensitive (unless you use quotes around your object names). However, text appears in the upper case. Thus, AWS DMS defaults to naming your target objects in the upper case. In most cases, you’ll want to use transformations to change schema, table and column names to lower case.

For more tips, see the AWS DMS troubleshooting section in the [AWS DMS User Guide](https://aws.amazon.com/dms/user-guide/).

To troubleshoot issues specific to Oracle, see the Oracle troubleshooting section: [Troubleshooting issues with Oracle](https://aws.amazon.com/dms/user-guide/).

To troubleshoot Aurora MySQL issues, see the MySQL troubleshooting section: [Troubleshooting issues with MySQL](https://aws.amazon.com/dms/user-guide/).

Working with the Sample Database for Migration

We recommend working through the preceding outline and guide by using the sample Oracle database provided by Amazon. This database mimics a simple sporting event ticketing system. The scripts
to generate the sample database can be found at https://github.com/aws-samples/aws-database-migration-samples/tree/master/oracle/sampledb/v1.

To build the sample database, go to the oracle/sampledb/v1 folder and follow the instructions in the README.md file.

The sample creates approximately 8-10 GB of data. The sample database also includes a ticketManagement package, which you can use to generate some transactions. To generate transactions, log into SQL*Plus or SQL Developer and run the following as dms_sample:

```
SQL> call generateTicketActivity(1000,0.01);
```

The first parameter is the transaction delay in seconds, the second is the number of transactions to generate. The preceding procedure simply sells tickets to people. You'll see updates to the tables: sporting_event_ticket and ticket_purchase_history.

Once you've sold some tickets, you can transfer them using the command following:

```
SQL> call generateTransferActivity(100,0.1);
```

The first parameter is the transaction delay in seconds, the second is the number of transactions to generate. This procedure also updates sporting_event_ticket and ticket_purchase_history.

Migrating an Amazon RDS for Oracle Database to Amazon Aurora MySQL

This walkthrough gets you started with heterogeneous database migration from Amazon RDS for Oracle to Amazon Aurora MySQL-Compatible Edition using AWS Database Migration Service (AWS DMS) and the AWS Schema Conversion Tool (AWS SCT). This is an introductory exercise so does not cover all scenarios but will provide you with a good understanding of the steps involved in executing such a migration.

It is important to understand that AWS DMS and AWS SCT are two different tools and serve different needs. They don't interact with each other in the migration process. At a high level, the steps involved in this migration are:

1. Using AWS SCT to:
   - Run the conversion report for Oracle to Amazon Aurora MySQL to identify the issues, limitations, and actions required for the schema conversion.
   - Generate the schema scripts and apply them on the target before performing the data load via AWS DMS. AWS SCT will perform the necessary code conversion for objects like procedures and views.

2. Identify and implement solutions to the issues reported by AWS SCT. For example, an object type like Oracle Sequence that is not supported in the Amazon Aurora MySQL can be handled using the auto_increment option to populate surrogate keys or develop logic for sequences at the application layer.

3. Disable foreign keys or any other constraints which may impact the AWS DMS data load.

4. AWS DMS loads the data from source to target using the Full Load approach. Although AWS DMS is capable of creating objects in the target as part of the load, it follows a minimalistic approach to efficiently migrate the data so it doesn't copy the entire schema structure from source to target.

5. Perform post-migration activities such as creating additional indexes, enabling foreign keys, and making the necessary changes in the application to point to the new database.
This walkthrough uses a custom AWS CloudFormation template to create an Amazon RDS DB instances for Oracle and Amazon Aurora MySQL. It then uses a SQL command script to install a sample schema and data onto the Amazon RDS Oracle DB instance that you then migrate to Amazon Aurora MySQL.

This walkthrough takes approximately two hours to complete. The estimated cost to complete it, using AWS resources, is about $5.00. Be sure to follow the instructions to delete resources at the end of this walkthrough to avoid additional charges.

**Topics**
- Costs (p. 98)
- Prerequisites (p. 99)
- Migration Architecture (p. 99)
- Step-by-Step Migration (p. 101)
- Next Steps (p. 131)

**Costs**

For this walkthrough, you provision Amazon Relational Database Service (Amazon RDS) resources by using AWS CloudFormation and also AWS Database Migration Service (AWS DMS) resources. Provisioning these resources will incur charges to your AWS account by the hour. The AWS Schema Conversion Tool incurs no cost; it is provided as a part of AWS DMS.

Although you'll need only a minimum of resources for this walkthrough, some of these resources are not eligible for AWS Free Tier. At the end of this walkthrough, you'll find a section in which you delete the resources to avoid additional charges. Delete the resources as soon as you complete the walkthrough.

To estimate what it will cost to run this walkthrough on AWS, you can use the AWS Simple Monthly Calculator. However, AWS DMS is not incorporated into the calculator yet. The following table shows both AWS DMS and Amazon RDS for Oracle Standard Edition Two pricing.

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Instance Type</th>
<th>Storage and I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon RDS for Oracle DB instance</td>
<td>db.m3.medium</td>
<td>Single AZ, 10 GB storage, GP2</td>
</tr>
<tr>
<td>instance, License Included</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Standard Edition Two), Single AZ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amazon Aurora MySQL DB instance</td>
<td>db.r3.large</td>
<td>Single AZ, 10 GB storage, 1 million I/O</td>
</tr>
<tr>
<td>AWS DMS replication instance</td>
<td>t2.small</td>
<td>50 GB of storage for keeping</td>
</tr>
<tr>
<td>AWS DMS data transfer</td>
<td>Free—data transfer</td>
<td>replication logs included</td>
</tr>
<tr>
<td>between AWS DMS and databases in RDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>instances in the same</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Availability Zone is free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data transfer out</td>
<td>First 1 GB per</td>
<td></td>
</tr>
<tr>
<td></td>
<td>month free</td>
<td></td>
</tr>
</tbody>
</table>

Assuming you run this walkthrough for two hours, we estimate the following pricing for AWS resources:

- Amazon Aurora MySQL + 10 GB storage pricing estimated by using the link to the Simple Monthly Calculator that you can access from the pricing site is $1.78.
Prerequisites

The following prerequisites are also required to complete this walkthrough:

- Amazon RDS for Oracle SE2 (license included) + 10 GB GP2 storage cost, estimated as per the aws.amazon.com at ($0.226) * 2 hours + ($0.115) * 10 GB, is $1.602.
- AWS DMS service cost for the t2.small instance with 50 GB GP2 storage, estimated as per the pricing site at ($0.036) * 2 hours, is $0.072.

Total estimated cost to run this project = $1.78 + $1.602 + $0.072 = $3.454—approximately $5.00.

This pricing is based on the following assumptions:

- We assume the total data transfer to the Internet is less than a gigabyte. The preceding pricing estimate assumes that data transfer and backup charges associated with the RDS and DMS services are within Free Tier limits.
- Storage consumed by the Aurora MySQL database is billed in per GB-month increments, and I/Os consumed are billed in per-million request increments.
- Data transfer between DMS and databases in RDS instances in the same Availability Zone is free.

Migration Architecture

This walkthrough uses AWS CloudFormation to create a simple network topology for database migration that includes the source database, the replication instance, and the target database in the same VPC. For more information on AWS CloudFormation, see the AWS CloudFormation documentation.
We will provision the AWS resources that are required for this AWS Database Migration Service (AWS DMS) walkthrough through AWS CloudFormation. These resources include a VPC and Amazon Relational Database Service (Amazon RDS) instances for Oracle and Amazon Aurora MySQL-Compatible Edition. We provision through AWS CloudFormation because it simplifies the process, so we can concentrate on tasks related to data migration. When you create a stack from the AWS CloudFormation template, it provisions the following resources:

- A VPC with CIDR (10.0.0.0/24) with two public subnets in your region, DBSubnet1 at the address 10.0.0.0/26 in Availability Zone 1 (AZ 1) and DBSubnet2 at the address 10.0.0.64/26, in AZ 2.
- A DB subnet group that includes DBSubnet1 and DBSubnet2.
- Oracle RDS Standard Edition Two with these deployment options:
  - License Included
  - Single-AZ setup
  - db.m3.medium or equivalent instance class
  - Port 1521
  - Default option and parameter groups
- Amazon Aurora MySQL DB instance with these deployment options:
  - No replicas
  - db.r3.large or equivalent instance class
  - Port 3306
  - Default option and parameter groups
- A security group with ingress access from your computer or 0.0.0.0/0 (access from anywhere) based on the input parameter

We have designed the AWS CloudFormation template to require few inputs from the user. It provisions the necessary AWS resources with minimum recommended configurations. However, if you want to change some of the configurations and parameters, such as the VPC CIDR block and Amazon RDS instance types, feel free to update the template.

We will use the AWS Management Console to provision the AWS DMS resources, such as the replication instance, endpoints, and tasks. You will install client tools such as SQL Workbench/J and the AWS Schema Conversion Tool (AWS SCT) on your local computer to connect to the Amazon RDS instances.

Following is an illustration of the migration architecture for this walkthrough.
Step-by-Step Migration

In the following sections, you can find step-by-step instructions for migrating an Amazon Relational Database Service (Amazon RDS) for Oracle database to Amazon Aurora MySQL-Compatible Edition. These steps assume that you have already prepared your source database as described in preceding sections.

Topics
- Step 1: Launch the RDS Instances in a VPC by Using the AWS CloudFormation Template (p. 101)
- Step 2: Install the SQL Tools and AWS Schema Conversion Tool on Your Local Computer (p. 109)
- Step 3: Test Connectivity to the Oracle DB Instance and Create the Sample Schema (p. 112)
- Step 4: Test the Connectivity to the Aurora MySQL DB Instance (p. 114)
- Step 5: Use the AWS Schema Conversion Tool to Convert the Oracle Schema to Aurora MySQL (p. 116)
- Step 6: Validate the Schema Conversion (p. 120)
- Step 7: Create an AWS DMS Replication Instance (p. 121)
- Step 8: Create AWS DMS Source and Target Endpoints (p. 122)
- Step 9: Create and Run Your AWS DMS Migration Task (p. 125)
- Step 10: Verify That Your Data Migration Completed Successfully (p. 128)
- Step 11: Delete Walkthrough Resources (p. 130)

Step 1: Launch the RDS Instances in a VPC by Using the AWS CloudFormation Template

Before you begin, you’ll need to download an AWS CloudFormation template. Follow these instructions:
1. Download the following archive to your computer: dms-sbs-RDSOracle2Aurora.zip.
2. Extract the AWS CloudFormation template (Oracle_Aurora_For_DMSDemo.template) from the archive.
3. Copy and paste the Oracle_Aurora_For_DMSDemo.template file into your current directory.

Now you need to provision the necessary AWS resources for this walkthrough. Do the following:

2. Choose **Create stack** and then choose **With new resources (standard)**.
3. On the **Specify template** section of the **Create stack** page, choose **Upload a template file**.
4. Click **Choose file**, and then choose the Oracle_Aurora_For_DMSDemo.template file that you extracted from the dms-sbs-RDSOracle2Aurora.zip archive.
5. Choose **Next**. On the **Specify Details** page, provide parameter values as shown following.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stack Name</strong></td>
<td>Enter DMSdemo.</td>
</tr>
<tr>
<td><strong>OracleDBName</strong></td>
<td>Provide a unique name for your database. The name should begin with a letter. The default is ORCL.</td>
</tr>
<tr>
<td><strong>OracleDBUsername</strong></td>
<td>Specify the admin (DBA) user for managing the Oracle instance. The default is oraadmin.</td>
</tr>
<tr>
<td><strong>OracleDBPassword</strong></td>
<td>Provide the password for the admin user. The default is oraadmin123.</td>
</tr>
<tr>
<td><strong>AuroraDBUsername</strong></td>
<td>Specify the admin (DBA) user for managing the Aurora MySQL instance. The default is auradmin.</td>
</tr>
<tr>
<td><strong>AuroraDBPassword</strong></td>
<td>Provide the password for the admin user. The default is auradmin123.</td>
</tr>
<tr>
<td><strong>ClientIP</strong></td>
<td>Specify the IP address in CIDR (x.x.x/32) format for your local computer. You can get your IP address from whatsmyip.org. Your RDS instances' security group will allow ingress to this IP address. The default is access from anywhere (0.0.0.0/0), which is not recommended; you should use your IP address for this walkthrough.</td>
</tr>
</tbody>
</table>
Specify stack details

Stack name

Stack name
DMSDemo
Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

Parameters
Parameters are defined in your template and allow you to input custom values when you create or update a stack.

Source Oracle Database Configuration
OracleDBName
Enter Oracle Database name
ORCL

OracleDBUsername
Enter database Admin username for Oracle
oraadmin

OracleDBPassword
Enter password for Oracle Admin user

Target Aurora Database Configuration
AuroraDBUsername
Enter database Admin username for Aurora
auradmin

AuroraDBPassword
Enter password for Aurora Admin user

Enter IP address for DB Security group Configuration
ClientIP
The IP address range that can be used to connect to the RDS instances from your local machine. It must be a valid range of the form x.x.x.x. Pls get your address using checkip.amazonaws.com or whatismyip.org
54.89.73/32
6. Choose Next. On the Configure stack options page, shown following, choose Next.
Configure stack options

**Tags**
You can specify tags (key-value pairs) to apply to resources in your stack. You can add up to 50 unique tags for each stack. Learn more

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Add tag

**Permissions**
Choose an IAM role to explicitly define how CloudFormation can create, modify, or delete resources in the stack. If you don’t choose a role, CloudFormation uses permissions based on your user credentials. Learn more

IAM role - optional
Choose the IAM role for CloudFormation to use for all operations performed on the stack.

IAM role ... Sample-role-name

**Stack failure options**
Behavior on provisioning failure
Specify the roll back behavior for a stack failure. Learn more

- **Roll back all stack resources**
  Roll back the stack to the last known stable state.

- **Preserve successfully provisioned resources**
  Preserves the state of successfully provisioned resources, while rolling back failed resources to the last known stable state. A last known stable state will be deleted upon the next stack operation.

**Advanced options**
You can set additional options for your stack, like notification options and a stack policy. Learn more

- **Stack policy**
  Defines the resources that you want to protect from unintentional updates during a stack update.

- **Rollback configuration**
  Specify alarms for CloudFormation to monitor when creating and updating the stack. If the operation breaches an alarm, CloudFormation rolls it back. Learn more
7. On the **Review** page, review the details, and if they are correct, scroll down and choose **Create stack**. You can get the estimated cost of running this AWS CloudFormation template by choosing **Estimate cost** at the **Template** section on top of the page.
Review DMSdemo

Step 1: Specify template

Template

Template URL
https://s3-external-1.amazonaws.com/cf-templates-grtw7rybb40t-us-east-1/20212514ci-Oracle_Aurora_For_DMSDemo.template

Stack description
This CloudFormation sample template OracleToAurora_DMS creates Oracle and Aurora instances in a VPC with 2 private subnets and 1 public subnet to test the migration using AWS DMS service. You will be billed for the AWS resources used if you create a stack from this template.

Estimate cost

Step 2: Specify stack details

Parameters (6)

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuroraDBPassword</td>
<td>**********</td>
</tr>
<tr>
<td>AuroraDBUsername</td>
<td>auradmin</td>
</tr>
<tr>
<td>ClientIP</td>
<td>54.239.6.187/32</td>
</tr>
<tr>
<td>OracleDBName</td>
<td>ORCL</td>
</tr>
<tr>
<td>OracleDBPassword</td>
<td>**********</td>
</tr>
<tr>
<td>OracleDBUsername</td>
<td>oraadmin</td>
</tr>
</tbody>
</table>
8. AWS can take about 20 minutes or more to create the stack with Amazon RDS for Oracle and Amazon Aurora MySQL instances.

9. After the stack is created, choose Stack, select the DMSdemo stack, and then choose Outputs. Record the JDBC connection strings, OracleJDBCConnectionString and AuroraJDBCConnectionString, for use later in this walkthrough to connect to the Oracle and Aurora MySQL DB instances.
Note
Oracle 12c SE Two License version 12.1.0.2.v4 is available in all regions. However, Amazon Aurora MySQL is not available in all regions. Amazon Aurora MySQL is currently available in US East (N. Virginia), US West (Oregon), EU (Ireland), Asia Pacific (Tokyo), Asia Pacific (Mumbai), Asia Pacific (Sydney), and Asia Pacific (Seoul). If you try to create a stack in a region where Aurora MySQL is not available, creation fails with the error Invalid DB Engine for AuroraCluster.

Step 2: Install the SQL Tools and AWS Schema Conversion Tool on Your Local Computer

Next, you need to install a SQL client and the AWS Schema Conversion Tool (AWS SCT) on your local computer.

This walkthrough assumes you will use the SQL Workbench/J client to connect to the RDS instances for migration validation. A few other software tools you might want to consider are the following:

- JACK DB, an online web interface to work with RDS databases (Oracle and Aurora MySQL) over JDBC
- DBVisualizer
- Oracle SQL Developer

To install the SQL client software, do the following:

1. Download SQL Workbench/J from the SQL Workbench/J website, and then install it on your local computer. This SQL client is free, open-source, and DBMS-independent.
2. Download the JDBC driver for your Oracle database release. For more information, go to https://www.oracle.com/jdbc.

3. Download the MySQL JDBC driver (.jar file). For more information, go to https://dev.mysql.com/downloads/connector/j/.

4. Using SQL Workbench/J, configure JDBC drivers for Oracle and Aurora MySQL to set up connectivity, as described following.
   a. In SQL Workbench/J, choose File, then choose Manage Drivers.
   b. From the list of drivers, choose Oracle.
   c. Choose the Open icon, then choose the .jar file for the Oracle JDBC driver that you downloaded in the previous step. Choose OK.
   d. From the list of drivers, choose MySQL.
   e. Choose the Open icon, then choose the MySQL JDBC driver that you downloaded in the previous step. Choose OK.

To install the AWS Schema Conversion Tool and the required JDBC drivers, do the following:

2. Launch the AWS Schema Conversion Tool.

3. In the AWS Schema Conversion Tool, choose Global settings from Settings.

4. In Global settings, choose Driver, and then choose Browse for Oracle driver path. Locate the JDBC Oracle driver and choose OK. Next, choose Browse for MySQL driver path. Locate the JDBC MySQL driver and choose OK. Choose OK to close the dialog box.
Step 3: Test Connectivity to the Oracle DB Instance and Create the Sample Schema

After the AWS CloudFormation stack has been created, test the connection to the Oracle DB instance by using SQL Workbench/J and then create the HR sample schema.

To test the connection to your Oracle DB instance and create the sample schema, do the following:

1. In SQL Workbench/J, choose **File**, then choose **Connect window**. Create a new connection profile using the following information as shown following:

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>New profile name</strong></td>
<td>Enter RDSOracleConnection.</td>
</tr>
<tr>
<td><strong>Driver</strong></td>
<td>Choose Oracle (oracle.jdbc.OracleDriver).</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td>Use the <code>OracleJDBCConnectionString</code> value you recorded when you examined the output details of the DMSdemo stack in a previous step.</td>
</tr>
<tr>
<td><strong>Username</strong></td>
<td>Enter oraadmin.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Provide the password for the admin user that you assigned when creating the Oracle DB instance using the AWS CloudFormation template.</td>
</tr>
</tbody>
</table>

2. To test the connection, choose **Test**. Choose **OK** to close the dialog box, then choose **OK** to create the connection profile.
Note

If your connection is unsuccessful, ensure that the IP address you assigned when creating the AWS CloudFormation template is the one you are attempting to connect from. This is the most common issue when trying to connect to an instance.

3. Create the HR schema you will use for migration using a custom SQL script (Oracle-HR-Schema-Build.sql). To obtain this script, do the following:
   a. Download the following archive to your computer: dms-sbs-RDSOracle2Aurora.zip.
   b. Extract the SQL script(Oracle-HR-Schema-Build.sql) from the archive.
   c. Copy and paste the Oracle-HR-Schema-Build.sql file into your current directory.

4. Open the provided SQL script in a text editor. Copy the entire script.

5. In SQL Workbench/J, paste the SQL script in the Default.wksp window showing Statement 1.

6. Choose SQL, then choose Execute All.

   When you run the script, you will get an error message indicating that user HR does not exist. You can ignore this error and run the script. The script drops the user before creating it, which generates the error.

7. Verify the object types and count in HR Schema were created successfully by running the following SQL query.

   ```sql
   SELECT OBJECT_TYPE, COUNT(*) FROM dbaOBJECTS WHERE owner='HR'
   GROUP BY OBJECT_TYPE;
   ```

   The results of this query should be similar to the following:

<table>
<thead>
<tr>
<th>OBJECT_TYPE</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX</td>
<td>8</td>
</tr>
<tr>
<td>PROCEDURE</td>
<td>2</td>
</tr>
<tr>
<td>SEQUENCE</td>
<td>3</td>
</tr>
</tbody>
</table>
8. Verify the number of constraints in the HR schema by running the following SQL query:

```sql
SELECT CONSTRAINT_TYPE, COUNT(*) FROM dba_constraints WHERE owner='HR'
AND (CONSTRAINT_TYPE IN ('P', 'R') OR SEARCH_CONDITION_VC NOT LIKE '%NOT NULL%')
GROUP BY CONSTRAINT_TYPE;
```

The results of this query should be similar to the following:

<table>
<thead>
<tr>
<th>CONSTRAINT_TYPE</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>10</td>
</tr>
<tr>
<td>P</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>

9. Analyze the HR schema by running the following:

```sql
BEGIN
  dbms_stats.gather_schema_stats('HR');
END;
```

10. Verify the total number of tables and number of rows for each table by running the following SQL query:

```sql
SELECT table_name, num_rows FROM dba_tables WHERE owner='HR' ORDER BY 1;
```

The results of this query should be similar to the following:

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>NUM_ROWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTRIES</td>
<td>25</td>
</tr>
<tr>
<td>DEPARTMENTS</td>
<td>27</td>
</tr>
<tr>
<td>EMPLOYEES</td>
<td>107</td>
</tr>
<tr>
<td>JOBS</td>
<td>19</td>
</tr>
<tr>
<td>JOB_HISTORY</td>
<td>10</td>
</tr>
<tr>
<td>LOCATIONS</td>
<td>23</td>
</tr>
<tr>
<td>REGIONS</td>
<td>4</td>
</tr>
</tbody>
</table>

11. Verify the relationships of the tables. Check the departments with employees greater than 10 by running the following SQL query:

```sql
SELECT b.department_name, COUNT(*) FROM HR.Employees a, HR.departments b WHERE a.department_id=b.department_id
GROUP BY b.department_name HAVING COUNT(*) > 10
ORDER BY 1;
```

The results of this query should be similar to the following:

<table>
<thead>
<tr>
<th>DEPARTMENT_NAME</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>34</td>
</tr>
<tr>
<td>Shipping</td>
<td>45</td>
</tr>
</tbody>
</table>

**Step 4: Test the Connectivity to the Aurora MySQL DB Instance**

Next, test your connection to your Aurora MySQL DB instance.
1. In SQL Workbench/J, choose **File**, then choose **Connect window**. Choose the Create a new connection profile icon, using the following information: Connect to the Aurora MySQL DB instance in SQL Workbench/J by using the information as shown following:

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>New profile</strong> name</td>
<td>Enter RDSAuroraConnection.</td>
</tr>
<tr>
<td><strong>Driver</strong></td>
<td>Choose MySQL (com.mysql.jdbc.Driver).</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td>Use the <strong>AuroraJDBCConnectionString</strong> value you recorded when you examined the output details of the DMSdemo stack in a previous step.</td>
</tr>
<tr>
<td><strong>Username</strong></td>
<td>Enter auradmin.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Provide the password for the admin user that you assigned when creating the Aurora MySQL DB instance using the AWS CloudFormation template.</td>
</tr>
</tbody>
</table>

2. Test the connection by choosing **Test**. Choose **OK** to close the dialog box, then choose OK to create the connection profile.

3. Log on to the Aurora MySQL instance by using the master admin credentials.

4. Verify your connectivity to the Aurora MySQL DB instance by running a sample SQL command, such as `SHOW DATABASES;`. 

**Note**

If your connection is unsuccessful, ensure that the IP address you assigned when creating the AWS CloudFormation template is the one you are attempting to connect from. This is the most common issue when trying to connect to an instance.
Step 5: Use the AWS Schema Conversion Tool to Convert the Oracle Schema to Aurora MySQL

Before you migrate data to Aurora MySQL, you convert the Oracle schema to an Aurora MySQL schema. This video covers all the steps of this process.

To convert an Oracle schema to an Aurora MySQL schema using AWS Schema Conversion Tool (AWS SCT), do the following:

1. Launch AWS SCT. In AWS SCT, choose File, then choose New Project. Create a new project named DMSDemoProject, specify the Location of the project folder, and then choose OK.
2. Choose Add source to add a source Oracle database to your project, then choose Oracle, and choose Next.
3. Enter the following information, and then choose Test Connection.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection name</td>
<td>Enter Amazon RDS for Oracle. AWS SCT displays this name in the tree in the left panel.</td>
</tr>
<tr>
<td>Type</td>
<td>Choose SID.</td>
</tr>
<tr>
<td>Server name</td>
<td>Use the OracleJDBCConnectionString value you used to connect to the Oracle DB instance, but remove the JDBC prefix information. For example, a sample connection string you use with SQL Workbench/J might be &quot;jdbc:oracle:thin:@do1xa4grferti8y.cqi4tcs0mg7.us-west-2.rds.amazonaws.com:1521:ORCL&quot;. For AWS SCT Server name, you remove &quot;jdbc:oracle:thin:@/&quot; and &quot;:1521&quot; to use just the server name: &quot;do1xa4grferti8y.cqi4tcs0mg7.us-west-2.rds.amazonaws.com&quot;</td>
</tr>
<tr>
<td>Server port</td>
<td>Enter 1521.</td>
</tr>
</tbody>
</table>
For This Parameter | Do This
---|---
Oracle SID | Enter ORCL.
User name | Enter oraadmin.
Password | Enter the password for the admin user that you assigned when creating the Oracle DB instance using the AWS CloudFormation template.

4. Choose OK to close the alert box, then choose Connect to close the dialog box and to connect to the Oracle DB instance.

5. Choose Add target to add a target Amazon Aurora MySQL database to your project, then choose Amazon Aurora (MySQL compatible), and choose Next.

6. Enter the following information and then choose Test Connection.
For This Parameter | Do This
--- | ---
Connection name | Enter Aurora MySQL. AWS SCT displays this name in the tree in the right panel.

Server name | Use the `AuroraJDBCConnectionString` value you used to connect to the Aurora MySQL DB instance, but remove the JDBC prefix information and the port suffix. For example, a sample connection string you use with SQL Workbench/J might be "jdbc:mysql://dmsdemo-auroracluster-1u1ogdfg35v.cluster-cqiw4tcs0mg7.us-west-2.rds.amazonaws.com:3306". For AWS SCT Server name, you remove "jdbc:mysql://" and ":3306" to use just the server name: "dmsdemo-auroracluster-1u1ogdfg35v.cluster-cqiw4tcs0mg7.us-west-2.rds.amazonaws.com"

Server port | Enter 3306.

User name | Enter auradmin.

Password | Enter the password for the admin user that you assigned when creating the Oracle DB instance using the AWS CloudFormation template.

7. Choose OK to close the alert box, then choose Connect to connect to the Amazon Aurora MySQL DB instance.

8. In the tree in the left panel, select only the HR schema. In the tree in the right panel, select your target Aurora MySQL database. Choose Create mapping.

9. Choose Main view. In the tree in the left panel, right-click the HR schema and choose Create report.
10. Check the report and the action items it suggests. The report discusses the type of objects that can be converted by using AWS SCT, along with potential migration issues and actions to resolve these issues. For this walkthrough, you should see something like the following:

**Database objects with conversion actions for Amazon Aurora (MySQL compatible)**

Of the total 27 database storage object(s) and 51 database code object(s) in the source database, we identified:

- An aggregate of 10 (37%) database storage object(s) require 11 complex user action(s) to complete the conversion.
- An aggregate of 11 (22%) database code object(s) require 11 complex user action(s) to complete the conversion.

**Figure: Conversion statistics for database storage objects**

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Conversion Rate</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema (1: 1/0/0/0)</td>
<td>100%</td>
<td>1</td>
</tr>
<tr>
<td>Table (10: 4/1/0/5)</td>
<td>40%</td>
<td>10</td>
</tr>
<tr>
<td>Constraint (7: 7/0/0/0)</td>
<td>100%</td>
<td>7</td>
</tr>
<tr>
<td>Index (2: 1/0/0/1)</td>
<td>50%</td>
<td>2</td>
</tr>
<tr>
<td>Type (1: 0/0/0/1)</td>
<td>50%</td>
<td>1</td>
</tr>
<tr>
<td>Collection type (1: 0/0/0/1)</td>
<td>100%</td>
<td>1</td>
</tr>
<tr>
<td>Sequence (2: 2/0/0/0)</td>
<td>100%</td>
<td>2</td>
</tr>
<tr>
<td>Synonym (1: 0/1/0/0)</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Database link (1: 0/0/0/1)</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

You can optionally save the report as .csv or .pdf format for later analysis.

11. Choose **Action Items**, and review any recommendations that you see.

12. In the tree in the left panel, right-click the HR schema and then choose **Convert schema**.
13. Choose Yes for the confirmation message. AWS SCT then converts your schema to the target database format.

14. In the tree in the right panel, choose the converted hr schema, and then choose Apply to database to apply the schema scripts to the target Aurora MySQL instance.

15. Choose the hr schema, and then choose Refresh from Database to refresh from the target database.

The database schema has now been converted and imported from source to target.

**Step 6: Validate the Schema Conversion**

To validate the schema conversion, you compare the objects found in the Oracle and Aurora MySQL databases using SQL Workbench/J.

1. In SQL Workbench/J, choose File, then choose Connect window. Choose the RDSAuroraConnection you created in an earlier step. Click OK.

2. Run the following script to verify the number of object types and count in the HR schema in the target Aurora MySQL database. These values should match the number of objects in the source Oracle database:

```sql
SELECT a.OBJECT_TYPE, COUNT(*)
FROM
  (SELECT OBJECT_TYPE
    ,OBJECT_SCHEMA
    ,OBJECT_NAME
    FROM (
      SELECT 'TABLE' AS OBJECT_TYPE
        ,TABLE_NAME AS OBJECT_NAME
        ,TABLE_SCHEMA AS OBJECT_SCHEMA
      FROM information_schema.TABLES
      where TABLE_TYPE='BASE TABLE'
      UNION
      SELECT 'VIEW' AS OBJECT_TYPE
        ,TABLE_NAME AS OBJECT_NAME
        ,TABLE_SCHEMA AS OBJECT_SCHEMA
      FROM information_schema.VIEWS
      UNION
      SELECT 'INDEX' AS OBJECT_TYPE
        ,CONCAT (
            CONSTRAINT_TYPE
        ,':'
        ,CONSTRAINT_NAME
        ,':'
        ,TABLE_NAME
        ) AS OBJECT_NAME
        ,TABLE_SCHEMA AS OBJECT_SCHEMA
      FROM information_schema.TABLE_CONSTRAINTS
      where constraint_type='PRIMARY KEY'
      UNION
      SELECT ROUTINE_TYPE AS OBJECT_TYPE
        ,ROUTINE_NAME AS OBJECT_NAME
        ,ROUTINE_SCHEMA AS OBJECT_SCHEMA
      FROM information_schema.ROUTINES
      UNION
      SELECT 'TRIGGER' AS OBJECT_TYPE
        ,CONCAT (TRIGGER_NAME
        ,':'
        ,EVENT_OBJECT_SCHEMA
        ,':'
        ) AS OBJECT_NAME
        ,TABLE_SCHEMA AS OBJECT_SCHEMA
      FROM information_schema.EVENT_TRIGGERS
    )
  ) AS a
```

120
CREATE OR REPLACE VIEW EVENT_OBJECT_TABLE AS

(SELECT EVENT_SCHEMA AS OBJECT_SCHEMA,
 EVENT_TABLE AS OBJECT_NAME
) AS R

FROM information_schema.views

WHERE R.EVENT_SCHEMA = 'HR'

ORDER BY 1)

GROUP BY a.OBJECT_TYPE;

The output from this query should be similar to the following:

<table>
<thead>
<tr>
<th>OBJECT_TYPE</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX</td>
<td>7</td>
</tr>
<tr>
<td>PROCEDURE</td>
<td>2</td>
</tr>
<tr>
<td>TABLE</td>
<td>7</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>10</td>
</tr>
<tr>
<td>VIEW</td>
<td>1</td>
</tr>
</tbody>
</table>

Next, run the following query to get table constraints information:

```
SELECT CONSTRAINT_TYPE,COUNT(*)
FROM information_schema.TABLE_CONSTRAINTS where constraint_schema='HR'
GROUP BY CONSTRAINT_TYPE;
```

The output from this query should be similar to the following:

<table>
<thead>
<tr>
<th>CONSTRAINT_TYPE</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOREIGN KEY</td>
<td>10</td>
</tr>
<tr>
<td>PRIMARY KEY</td>
<td>7</td>
</tr>
<tr>
<td>UNIQUE</td>
<td>7</td>
</tr>
</tbody>
</table>

Step 7: Create an AWS DMS Replication Instance

After we validate the schema structure between source and target databases, as described preceding, we proceed to the core part of this walkthrough, which is the data migration. The following illustration shows a high-level view of the migration process.

A DMS replication instance performs the actual data migration between source and target. The replication instance also caches the transaction logs during the migration. How much CPU and memory capacity a replication instance has influences the overall time required for the migration.

To create an AWS DMS replication instance, do the following:

1. Sign in to the AWS Management Console, select AWS Database Migration Service (AWS DMS) and choose Create replication instance. If you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information on the permissions required, see IAM Permissions.
2. On the **Create replication instance** page, specify your replication instance information as shown following.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter DMSdemo-repserver.</td>
</tr>
<tr>
<td>Descriptive Amazon Resource Name (ARN)</td>
<td>Skip this optional field.</td>
</tr>
<tr>
<td>Description</td>
<td>Enter a brief description, such as DMS demo replication server.</td>
</tr>
<tr>
<td>Instance class</td>
<td>Choose <code>dms.t3.medium</code>. This instance class is large enough to migrate a small set of tables.</td>
</tr>
<tr>
<td>Engine version</td>
<td>Choose 3.4.5. This is the latest AWS DMS version, which includes all new features and enhancements.</td>
</tr>
<tr>
<td>Allocated storage (GiB)</td>
<td>Choose 50. This storage space is enough for your migration project.</td>
</tr>
<tr>
<td>VPC</td>
<td>Choose DMSDemoVPC, which is the VPC that was created by the AWS CloudFormation stack.</td>
</tr>
<tr>
<td>Multi-AZ</td>
<td>Choose Dev or test workload (Single-AZ).</td>
</tr>
<tr>
<td>Publicly accessible</td>
<td>Leave this item selected.</td>
</tr>
</tbody>
</table>

3. For the **Advanced**, **Maintenance**, and **Tags** sections, leave the default settings as they are, and choose **Create**.

### Step 8: Create AWS DMS Source and Target Endpoints

While your replication instance is being created, you can specify the source and target database endpoints using the **AWS Management Console**. However, you can only test connectivity after the replication instance has been created, because the replication instance is used in the connection.

1. Specify your connection information for the source Oracle database and the target Amazon Aurora MySQL database. The following table describes the source settings.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint Identifier</td>
<td>Enter Orasource (the Amazon RDS for Oracle endpoint).</td>
</tr>
<tr>
<td>Source Engine</td>
<td>Choose oracle.</td>
</tr>
<tr>
<td>Server name</td>
<td>Provide the Oracle DB instance name. This is the <strong>Server name</strong> you used for AWS SCT, such as &quot;do1xa4grferti8y.cqiwh4tcs0mg7.us-west-2.rds.amazonaws.com&quot;.</td>
</tr>
<tr>
<td>Port</td>
<td>Enter 1521.</td>
</tr>
<tr>
<td>SSL mode</td>
<td>Choose None.</td>
</tr>
</tbody>
</table>
### Step-by-Step Migration

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Username</td>
<td>Enter <code>oraadmin</code>.</td>
</tr>
<tr>
<td>Password</td>
<td>Provide the password for the Oracle DB instance.</td>
</tr>
<tr>
<td>SID</td>
<td>Provide the Oracle database name.</td>
</tr>
</tbody>
</table>

The following table describes the target settings.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint Identifier</td>
<td>Enter <code>Aurtarget</code> (the Amazon Aurora MySQL endpoint).</td>
</tr>
<tr>
<td>Target Engine</td>
<td>Choose <code>aurora</code>.</td>
</tr>
<tr>
<td>Servername</td>
<td>Provide the Aurora MySQL DB instance name. This is the <strong>Server name</strong> you used for AWS SCT, such as &quot;dmsdemo-auror-cluster-cluster-cqiwc0mg7.us-west-2.rds.amazonaws.com&quot;.</td>
</tr>
<tr>
<td>Port</td>
<td>Enter 3306.</td>
</tr>
<tr>
<td>SSL mode</td>
<td>Choose <strong>None</strong>.</td>
</tr>
<tr>
<td>Username</td>
<td>Enter <code>auradmin</code>.</td>
</tr>
<tr>
<td>Password</td>
<td>Provide the password for the Aurora MySQL DB instance.</td>
</tr>
</tbody>
</table>

The completed page should look like the following:
2. In order to disable foreign key checks during the initial data load, you must add the following commands to the target Aurora MySQL DB instance. In the Advanced section, shown following, type the following commands for Extra connection attributes: `initstmt=SET FOREIGN_KEY_CHECKS=0;autocommit=1`

The first command disables foreign key checks during a load, and the second command commits the transactions that DMS executes.
3. Choose **Next**.

### Step 9: Create and Run Your AWS DMS Migration Task

Using a AWS DMS task, you can specify what schema to migrate and the type of migration. You can migrate existing data, migrate existing data and replicate ongoing changes, or replicate data changes only. This walkthrough migrates existing data only.

1. On the **Create Task** page, specify the task options. The following table describes the settings.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task name</td>
<td>Enter migratehrschema.</td>
</tr>
<tr>
<td>Task description</td>
<td>Enter a description for the task.</td>
</tr>
<tr>
<td>Source endpoint</td>
<td>Shows orasource (the Amazon RDS for Oracle endpoint).</td>
</tr>
<tr>
<td>Target endpoint</td>
<td>Shows aurtarget (the Amazon Aurora MySQL endpoint).</td>
</tr>
</tbody>
</table>
2. Under Task Settings, choose Do nothing for Target table preparation mode, because you have already created the tables through Schema Migration Tool. Because this migration doesn’t contain any LOBs, you can leave the LOB settings at their defaults.

Optionally, you can select Enable logging. If you enable logging, you will incur additional Amazon CloudWatch charges for the creation of CloudWatch logs. For this walkthrough, logs are not necessary.
3. Leave the Advanced settings at their default values.

4. Choose Table mappings, choose Default for Mapping method, and then choose HR for Schema to migrate.

The completed section should look like the following.

5. Choose Create task. The task will begin immediately.
The Tasks section shows you the status of the migration task.

You can monitor your task if you choose **Enable logging** when you set up your task. You can then view the CloudWatch metrics by doing the following:

1. On the navigation pane, choose **Tasks**.
2. Choose your migration task (migrateschema).
3. Choose the **Task monitoring** tab, and monitor the task in progress on that tab.

**Step 10: Verify That Your Data Migration Completed Successfully**

When the migration task completes, you can compare your task results with the expected results.

1. On the navigation pane, choose **Tasks**.
2. Choose your migration task (migrateschema).
3. Choose the **Table statistics** tab, shown following.
4. Connect to the Amazon Aurora MySQL instance by using SQL Workbench/J, and then check if the database tables were successfully migrated from Oracle to Aurora MySQL by running the SQL script shown following.

```
SELECT TABLE_NAME, TABLE_ROWS
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'HR' AND TABLE_TYPE = 'BASE TABLE' ORDER BY 1;
```
5. Run the following query to check the relationship in tables; this query checks the departments with employees greater than 10.

```
SELECT B.DEPARTMENT_NAME, COUNT(*)
FROM HR.EMPLOYEES A, HR.DEPARTMENTS B
WHERE A.DEPARTMENT_ID = B.DEPARTMENT_ID
GROUP BY B.DEPARTMENT_NAME HAVING COUNT(*) > 10
ORDER BY 1;
```

The output from this query should be similar to the following.

<table>
<thead>
<tr>
<th>department_name</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>34</td>
</tr>
<tr>
<td>Shipping</td>
<td>45</td>
</tr>
</tbody>
</table>

Now you have successfully completed a database migration from an Amazon RDS for Oracle database instance to Amazon Aurora MySQL.

**Step 11: Delete Walkthrough Resources**

After you have completed this walkthrough, perform the following steps to avoid being charged further for AWS resources used in the walkthrough. It's necessary that you do the steps in order, because some resources cannot be deleted if they have a dependency upon another resource.

1. On the navigation pane, choose **Tasks**, choose your migration task (**migratethrschema**), and then choose **Delete**.
2. On the navigation pane, choose **Endpoints**, choose the Oracle source endpoint (**orarsource**), and then choose **Delete**.
3. Choose the Amazon Aurora MySQL target endpoint (**aurtarget**), and then choose **Delete**.
4. On the navigation pane, choose **Replication instances**, choose the replication instance (**DMSdemo-repserver**), and then choose **Delete**.
Next, you must delete your AWS CloudFormation stack, DMSdemo.

1. Sign in to the AWS Management Console and open the AWS CloudFormation console.
   
   Note that if you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS CloudFormation.
2. Choose your AWS CloudFormation stack, DMSdemo.
3. For Actions, choose Delete stack.

The status of the stack changes to DELETE_IN_PROGRESS while AWS CloudFormation AWS CloudFormation cleans up the resources associated with the DMSdemo stack. When AWS CloudFormation is finished cleaning up resources, it removes the stack from the list.

Next Steps

You can explore several other features of AWS DMS that were not included in this walkthrough, including the following:

• The AWS DMS change data capture (CDC) feature, for ongoing replication of data.
• Transformation actions that let you specify and apply transformations to the selected schema or table as part of the migration process.

For more information, see https://docs.aws.amazon.com/dms/latest/userguide/CHAP_GettingStarted.html.

Migrating a SQL Server Database to Amazon Aurora MySQL

Using this walkthrough, you can learn how to migrate a Microsoft SQL Server database to an Amazon Aurora MySQL-Compatible Edition database using the AWS Schema Conversion Tool (AWS SCT) and AWS Database Migration Service (AWS DMS). AWS DMS migrates your data from your SQL Server source into your Aurora MySQL target.

AWS DMS doesn’t migrate your secondary indexes, sequences, default values, stored procedures, triggers, synonyms, views, and other schema objects that aren’t specifically related to data migration. To migrate these objects to your Aurora MySQL target, use AWS SCT.

Topics

• Prerequisites (p. 131)
• Step-by-Step Migration (p. 132)
• Troubleshooting (p. 147)

Prerequisites

The following prerequisites are required to complete this walkthrough:

• Understand Amazon Relational Database Service (Amazon RDS), the applicable database technologies, and SQL.
• Create an AWS account with AWS Identity and Access Management (IAM) credentials that allows you to launch Amazon RDS and AWS Database Migration Service (AWS DMS) instances in your AWS Region. For information about IAM credentials, see Create an IAM User.
• Understand the Amazon Virtual Private Cloud (Amazon VPC) service and security groups. For information about using Amazon VPC with Amazon RDS, see Amazon Virtual Private Cloud (VPCs) and Amazon RDS. For information about Amazon RDS security groups, see Amazon RDS Security Groups.

• Understand the supported features and limitations of AWS DMS. For information about AWS DMS, see https://docs.aws.amazon.com/dms/latest/userguide/Welcome.html.

• Understand how to work with Microsoft SQL Server as a source and Amazon Aurora MySQL as a target. For information about working with SQL Server as a source, see Using a SQL Server Database as a Source. Aurora MySQL is a MySQL-compatible database. For information about working with Aurora MySQL as a target, see Using a MySQL-Compatible Database as a Target.

• Understand the supported data type conversion options for SQL Server and Aurora MySQL. For information about data types for SQL Server as a source, see Source Data Types for Microsoft SQL Server. For information about data types for Aurora MySQL as a target, see Target Data Types for MySQL.

• Size your target Aurora MySQL database host. DBAs should be aware of the load profile of the current source SQL Server database host. Consider CPU, memory, and IOPS. With Amazon RDS, you can size up the target database host, or reduce it, after the migration. If this is the first time that you're migrating to Aurora MySQL, we recommend that you have extra capacity to account for performance issues and tuning opportunities.

• Audit your source SQL Server database. For each schema and all the objects under each schema, determine whether any of the objects are no longer being used. Deprecate these objects on the source SQL Server database, because there’s no need to migrate them if they aren’t being used.

• Decide between these migration options: migrate existing data only or migrate existing data and replicate ongoing changes.

  • If you migrate existing data only, the migration is a one-time data transfer from a SQL Server source database to the Aurora MySQL target database. If the source database remains open to changes during the migration, these changes must be applied to the target database after the migration is complete.

    Note
    If the SQL Server database is an Amazon RDS database, replication is not supported, and you must use the option to migrate existing data only.

  • If you migrate existing data and replicate ongoing changes, one option is to replicate the source database changes. Replication keeps the source and target databases in sync with each other during the migration process and can reduce database downtime. With this option, you complete an initial sync operation and then configure MS-REPLICATION. This option requires the Standard, Enterprise, or Developer SQL Server edition. You enable MS-REPLICATION for each SQL Server instance that you want to use as a database source.

  • If you want to migrate existing data and replicate ongoing changes, another option is change data capture (CDC) instead of replication. This option allows AWS DMS to perform ongoing migration of data. In the case of CDC, AWS DMS uses the CDC tables to enable ongoing database migration. This option requires the Enterprise or Developer edition of SQL Server.

For more information about AWS DMS, see https://docs.aws.amazon.com/dms/latest/userguide/CHAP_GettingStarted.html.

Step-by-Step Migration

The following steps provide instructions for migrating a Microsoft SQL Server database to an Amazon Aurora MySQL database. These steps assume that you have already prepared your source database as described in Prerequisites (p. 131).

Topics
• Step 1: Install the SQL Drivers and AWS Schema Conversion Tool on Your Local Computer (p. 133)
• Step 2: Configure Your Microsoft SQL Server Source Database (p. 134)
Step-by-Step Walkthroughs

Step-by-Step Migration

- Step 3: Configure Your Aurora MySQL Target Database (p. 135)
- Step 4: Use AWS SCT to Convert the SQL Server Schema to Aurora MySQL (p. 135)
- Step 5: Create an AWS DMS Replication Instance (p. 138)
- Step 6: Create AWS DMS Source and Target Endpoints (p. 140)
- Step 7: Create and Run Your AWS DMS Migration Task (p. 144)
- Step 8: Cut Over to Aurora MySQL (p. 147)

Step 1: Install the SQL Drivers and AWS Schema Conversion Tool on Your Local Computer

First, install the SQL drivers and the AWS Schema Conversion Tool (AWS SCT) on your local computer. Do the following:

1. Download the JDBC driver for Microsoft SQL Server `mssql-jdbc-7.2.2.jre11.jar`.
2. Download the JDBC driver for Aurora MySQL. Amazon Aurora MySQL uses the MySQL driver.
3. Install AWS SCT and the required JDBC drivers.
   a. See https://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_Installing.html, and choose the appropriate link to download AWS SCT.
   b. Start AWS SCT, and choose Settings, Global settings.
   c. In Global settings, choose Drivers, and then choose Browse for Microsoft SQL Server driver path. Locate the JDBC driver for SQL Server, and choose OK.
   d. Choose Browse for MySQL driver path. Locate the JDBC driver you downloaded for Aurora MySQL, and choose OK.

   ![Global Settings Dialog Box](image)

   e. Choose OK to close the Global settings dialog box.
Step 2: Configure Your Microsoft SQL Server Source Database

After installing the SQL drivers and AWS Schema Conversion Tool, you can configure your Microsoft SQL Server source database using one of several options, depending on how you plan to migrate your data.

When configuring your source database, you can choose to migrate existing data only, migrate existing data and replicate ongoing changes, or migrate existing data and use change data capture (CDC) to replicate ongoing changes. For more information about these options, see Prerequisites (p. 131).

Migrating existing data only

No configuration steps are necessary for the SQL Server database. You can move on to Step 3: Configure Your Aurora MySQL Target Database (p. 135).

Note

If the SQL Server database is an Amazon RDS database, replication is not supported, and you must use the option for migrating existing data only.

Migrating existing data and replicating ongoing changes

To configure MS-REPLICATION, complete the following steps:

1. In Microsoft SQL Server Management Studio, open the context (right-click) menu for the Replication folder, and then choose Configure Distribution.
2. In the Distributor step, choose db_name will act as its own distributor. SQL Server creates a distribution database and log.

For more information, see Microsoft documentation.

When the configuration is complete, your server is enabled for replication. Either a distribution database is in place, or you have configured your server to use a remote distribution database.

Note

Replication requires a primary key for all tables that are being replicated. If your tables don’t have primary keys defined, consider using CDC instead.

Migrating existing data and using change data capture (CDC) to replicate ongoing changes

To configure MS-CDC, complete the following steps:

1. Connect to SQL Server with a login that has SYSADMIN role membership.
2. For each database containing data that is being migrated, run the following command within the database context:

   ```sql
   use [DBname]
   EXEC sys.sp_cdc_enable_db
   ```

3. For each table that you want to configure for ongoing migration, run the following command:

   ```sql
   EXEC sys.sp_cdc_enable_table @source_schema = N'schema_name', @source_name = N'table_name', @role_name = NULL;
   ```

For more information, see Microsoft documentation.

Note

- If you are migrating databases that participate in an Always On Availability Group, it is best practice to use replication for migration. To use this option, publishing must be enabled, and
a distribution database must be configured for each node of the Always On Availability Group. Additionally, ensure you are using the name of the availability group listener for the database rather than the name of the server currently hosting the availability group database for the target server name. These requirement apply to each instance of SQL Server in the cluster and must not be configured using the availability group listener.

- If your database isn't supported for MS-REPLICATION or MS-CDC (for example, if you are running the Workgroup Edition of SQL Server), some changes can still be captured, such as INSERT and DELETE statements, but other DML statements such as UPDATE and TRUNCATE TABLE will not be captured. Therefore, a migration with continuing data replication is not recommended in this configuration, and a static one time migration (or repeated one time full migrations) should be considered instead.

For more information about using MS-REPLICATION and MS-CDC, see Configuring a Microsoft SQL Server Database as a Replication Source.

Step 3: Configure Your Aurora MySQL Target Database

AWS DMS migrates the data from the SQL Server source into an Amazon Aurora MySQL target. In this step, you configure the Aurora MySQL target database.

1. Create the AWS DMS user to connect to your target database, and grant Superuser or the necessary individual privileges (or for Amazon RDS, use the master username).

   Alternatively, you can grant the privileges to an existing user.

   ```sql
   CREATE USER 'aurora_dms_user' IDENTIFIED BY 'password';
   GRANT ALTER, CREATE, CREATE TEMPORARY TABLES, DROP, INDEX, INSERT, UPDATE, DELETE, SELECT ON target_database.* TO 'aurora_dms_user';
   ```

2. AWS DMS uses control tables on the target in the database awsdms_control. Use the following command to ensure that the user has the necessary access to the awsdms_control database:

   ```sql
   GRANT ALL PRIVILEGES ON awsdms_control.* TO 'aurora_dms_user';
   FLUSH PRIVILEGES;
   ```

Step 4: Use AWS SCT to Convert the SQL Server Schema to Aurora MySQL

Before you migrate data to Amazon Aurora MySQL, convert the Microsoft SQL Server schema to an Aurora MySQL schema using the AWS Schema Conversion Tool (AWS SCT). This video covers all the steps of this process.

To convert a SQL Server schema to an Aurora MySQL schema, do the following:

1. Launch AWS SCT. In AWS SCT, choose File, then choose New Project. Create a new project named AWS Schema Conversion Tool SQL Server to Aurora MySQL, specify the Location of the project folder, and then choose OK.

2. Choose Add source to add a source Microsoft SQL Server database to your project, then choose Microsoft SQL Server, and choose Next.

3. Enter the following information, and then choose Test connection.
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection name</strong></td>
<td>Enter Microsoft SQL Server. AWS SCT displays this name in the tree in the left panel.</td>
</tr>
<tr>
<td><strong>Server name</strong></td>
<td>Enter the server name.</td>
</tr>
<tr>
<td><strong>Server port</strong></td>
<td>Enter the SQL Server port number. The default is 1433.</td>
</tr>
<tr>
<td><strong>Instance name</strong></td>
<td>Enter the SQL Server database instance name.</td>
</tr>
<tr>
<td><strong>User name</strong></td>
<td>Enter the SQL Server admin user name.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Enter the password for the admin user.</td>
</tr>
</tbody>
</table>

4. Choose **OK** to close the alert box. Then choose **Connect** to close the dialog box and connect to the Microsoft SQL Server database instance. AWS SCT displays the structure of the Microsoft SQL Server database instance in the left panel.
5. Choose **Add target** to add a target Amazon Aurora MySQL database to your project, then choose **Amazon Aurora (MySQL compatible)**, and choose **Next**.

6. Enter the following information and then choose **Test Connection**.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection name</strong></td>
<td>Enter Aurora MySQL. AWS SCT displays this name in the tree in the right panel.</td>
</tr>
<tr>
<td><strong>Server name</strong></td>
<td>Enter the server name.</td>
</tr>
<tr>
<td><strong>Server port</strong></td>
<td>Enter the SQL Server port number. The default is 3306.</td>
</tr>
<tr>
<td><strong>User name</strong></td>
<td>Enter the Aurora MySQL admin user name.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Enter the password for the admin user.</td>
</tr>
</tbody>
</table>

7. Choose **OK** to close the alert box. Then choose **Connect** to close the dialog box and connect to the Aurora MySQL database instance.

8. In the tree in the left panel, select the schema to migrate. In the tree in the right panel, select your target Aurora MySQL database. Choose **Create mapping**.

9. Choose **Main view**. In the tree in the left panel, right-click the HR schema and choose **Create report**.

10. Open the context (right-click) menu for the schema to migrate, and then choose **Convert schema**.

11. Choose **Yes** for the confirmation message. AWS SCT analyzes the schema, creates a database migration assessment report, and converts your schema to the target database format.

12. Choose **Assessment Report View** from the menu to check the database migration assessment report. The report breaks down by each object type and by how much manual change is needed to convert it successfully.

    Generally, packages, procedures, and functions are more likely to have some issues to resolve because they contain the most custom Transact-SQL code. AWS SCT also provides hints about how to fix these objects.

13. Choose the **Action Items** tab.
The **Action Items** tab shows each issue for each object that requires attention.

For each conversion issue, you can complete one of the following actions:

a. Modify the objects on the source SQL Server database so that AWS SCT can convert the objects to the target Aurora MySQL database.
   i. Modify the objects on the source SQL Server database.
   ii. Repeat the previous steps to convert the schema and check the assessment report.
   iii. If necessary, repeat this process until there are no conversion issues.
   iv. Choose **Main View** from the menu. Open the context (right-click) menu for the target Aurora MySQL schema, and choose **Apply to database** to apply the schema changes to the Aurora MySQL database, and confirm that you want to apply the schema changes.

b. Instead of modifying the source schema, modify scripts that AWS SCT generates before applying the scripts on the target Aurora MySQL database.
   i. Choose **Main View** from the menu. Open the context (right-click) menu for the target Aurora MySQL schema name, and choose **Save as SQL**. Next, choose a name and destination for the script.
   ii. In the script, modify the objects to correct conversion issues.

   You can also exclude foreign key constraints, triggers, and secondary indexes from the script because they can cause problems during the migration. After the migration is complete, you can create these objects on the Aurora MySQL database.
   iii. Run the script on the target Aurora MySQL database.

For more information, see [Converting Database Schema to Amazon RDS](#).

14. (Optional) Use AWS SCT to create migration rules.
   a. Choose **Mapping view** and then choose **New migration rule**.
   b. Create additional migration transformation rules that are required based on the action items.
   c. Save the migration rules.
   d. Choose **Export script for DMS** to export a JSON format of all the transformations that the AWS DMS task will use. Choose **Save**.

### Step 5: Create an AWS DMS Replication Instance

After validating the schema structure between source and target databases, continue with the core part of this walkthrough, which is the data migration. The following illustration shows a high-level view of the migration process.

![Diagram of AWS DMS replication instance](#)

An AWS DMS replication instance performs the actual data migration between source and target. The replication instance also caches the transaction logs during the migration. The amount of CPU and memory capacity a replication instance has influences the overall time that is required for the migration.

For information about best practices for using AWS DMS, see [AWS Database Migration Service Best Practices](#).
To create an AWS DMS replication instance, do the following:

1. Sign in to the AWS Management Console, and open the AWS DMS console.
2. In the console, choose Create migration. If you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information about the permissions required, see IAM Permissions.
3. On the Welcome page, choose Next to start a database migration.
4. On the Create replication instance page, specify your replication instance information.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Select a name for your replication instance. If you are using multiple replication servers or sharing an account, choose a name that helps you quickly differentiate between the different servers.</td>
</tr>
<tr>
<td>Description</td>
<td>Enter a brief description.</td>
</tr>
<tr>
<td>Instance class</td>
<td>Select the type of replication server to create. Each size and type of instance class has increasing CPU, memory, and I/O capacity. Generally, t2 instances are for lower load tasks, and the c4 instances are for higher load and more tasks.</td>
</tr>
<tr>
<td>VPC</td>
<td>Choose the virtual private cloud (VPC) in which your replication instance will launch. If possible, select the same VPC in which either your source or target database resides (or both).</td>
</tr>
<tr>
<td>Multi-AZ</td>
<td>If you choose Yes, AWS DMS creates a second replication server in a different Availability Zone for failover if there is a problem with the primary replication server.</td>
</tr>
<tr>
<td>Publicly accessible</td>
<td>If either your source or target database resides outside of the VPC in which your replication server resides, you must make your replication server policy publicly accessible.</td>
</tr>
</tbody>
</table>

5. For the Advanced section, specify the following information.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocated storage (GB)</td>
<td>Amount of storage on the replication server for the AWS DMS task logs, including historical tasks logs. AWS DMS also uses disk storage to cache certain data while it replicates it from the source database to the target. Additionally, more storage generally enables better IOPS on the server.</td>
</tr>
<tr>
<td>Replication Subnet Group</td>
<td>If you are running in a Multi-AZ configuration, you need at least two subnet groups.</td>
</tr>
</tbody>
</table>
### Step-by-Step Migration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability zone</td>
<td>Generally, performance is better if you locate your primary replication server in the same Availability Zone as your target database.</td>
</tr>
<tr>
<td>VPC Security Group(s)</td>
<td>Security groups enable you to control ingress and egress to your VPC. AWS DMS lets you associate one or more security groups with the VPC in which your replication server is launched.</td>
</tr>
<tr>
<td>KMS key</td>
<td>With AWS DMS, all data is encrypted at rest using a KMS encryption key. By default, AWS DMS creates a new encryption key for your replication server. However, you might choose to use an existing key. For information about the KMS key, see Setting an Encryption Key and Specifying KMS Permissions.</td>
</tr>
</tbody>
</table>

6. Click Next.

### Step 6: Create AWS DMS Source and Target Endpoints

While your replication instance is being created, you can specify the source and target database endpoints using the AWS Management Console. However, you can test connectivity only after the replication instance has been created, because the replication instance is used in the connection.

1. In the AWS DMS console, specify your connection information for the source SQL Server database and the target Aurora MySQL database. The following table describes the source settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint Identifier</td>
<td>Enter a name, such as SQLServerSource.</td>
</tr>
<tr>
<td>Source Engine</td>
<td>Choose sqlserver.</td>
</tr>
<tr>
<td>Server name</td>
<td>Provide the SQL Server DB instance server name.</td>
</tr>
<tr>
<td>Port</td>
<td>Enter the port number of the database. The default for SQL Server is 1433.</td>
</tr>
<tr>
<td>SSL mode</td>
<td>Choose an SSL mode if you want to enable encryption for your connection's traffic.</td>
</tr>
<tr>
<td>User name</td>
<td>Enter the name of the user you want to use to connect to the source database.</td>
</tr>
<tr>
<td>Password</td>
<td>Provide the password for the user.</td>
</tr>
<tr>
<td>Database name</td>
<td>Provide the SQL Server database name.</td>
</tr>
</tbody>
</table>

The following table describes the advanced source settings.
### Extra connection attributes

Extra parameters that you can set in an endpoint to add functionality or change the behavior of AWS DMS. A few of the most relevant attributes are listed here. Use a semicolon (;) to separate multiple entries.

- **safeguardpolicy**: Changes the behavior of SQL Server by opening transactions to prevent the transaction log from being truncated while AWS DMS is reading the log. Valid values are `EXCLUSIVE_AUTOMATIC_TRUNCATION` or `RELY_ON_SQL_SERVER_REPLICATION_AGENT` (default).
- **useBCPFullLoad**: Directs AWS DMS to use BCP (bulk copy) for data loading. Valid values are `Y` or `N`. When the target table contains an identity column that does not exist in the source table, you must disable the use of BCP for loading the table by setting the parameter to `N`.
- **BCPPacketSize**: If BCP is enabled for data loads, then enter the maximum packet size used by BCP. Valid values are `1` – `100000` (default 16384).
- **controlTablesFileGroup**: Specifies the file group to use for the control tables that the AWS DMS process creates in the database.

### KMS key

Enter the KMS key if you choose to encrypt your replication instance's storage.

The following table describes the target settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint Identifier</td>
<td>Enter a name, such as <code>Auroratarget</code>.</td>
</tr>
<tr>
<td>Target Engine</td>
<td>Choose <code>aurora</code>.</td>
</tr>
<tr>
<td>Server name</td>
<td>Provide the Aurora MySQL DB server name for the primary instance.</td>
</tr>
<tr>
<td>Port</td>
<td>Enter the port number of the database. The default for Aurora MySQL is 3306.</td>
</tr>
<tr>
<td>SSL mode</td>
<td>Choose <code>None</code>.</td>
</tr>
<tr>
<td>User name</td>
<td>Enter the name of the user that you want to use to connect to the target database.</td>
</tr>
<tr>
<td>Password</td>
<td>Provide the password for the user.</td>
</tr>
</tbody>
</table>
The following table describes the advanced target settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Extra connection attributes** | Extra parameters that you can set in an endpoint to add functionality or change the behavior of AWS DMS. A few of the most relevant attributes are listed here. Use a semicolon to separate multiple entries.  
  - **targetDbType** - By default, AWS DMS creates a different database for each schema that is being migrated. If you want to combine several schemas into a single database, set this option to `targetDbType=SPECIFIC_DATABASE`.  
  - **initstmt** - Use this option to invoke the MySQL `initstmt` connection parameter and accept anything MySQL `initstmt` accepts. For an Aurora MySQL target, it's often useful to disable foreign key checks by setting this option to `initstmt=SET FOREIGN_KEY_CHECKS=0`. |
| **KMS key**                | Enter the KMS key if you choose to encrypt your replication instance's storage.                                                 |
For information about extra connection attributes, see Using Extra Connection Attributes.

2. After the endpoints and replication instance are created, test the endpoint connections by choosing **Run test** for the source and target endpoints.

3. Drop foreign key constraints and triggers on the target database.

   During the full load process, AWS DMS does not load tables in any particular order, so it might load the child table data before parent table data. As a result, foreign key constraints might be violated if they are enabled. Also, if triggers are present on the target database, they might change data loaded by AWS DMS in unexpected ways.

   ```sql
   ALTER TABLE 'table_name' DROP FOREIGN KEY 'fk_name';
   DROP TRIGGER 'trigger_name';
   ```

4. If you dropped foreign key constraints and triggers on the target database, generate a script that enables the foreign key constraints and triggers.

   Later, when you want to add them to your migrated database, you can just run this script.

5. (Optional) Drop secondary indexes on the target database.

   Secondary indexes (as with all indexes) can slow down the full load of data into tables because they must be maintained and updated during the loading process. Dropping them can improve the
performance of your full load process. If you drop the indexes, you must add them back later, after the full load is complete.

```
ALTER TABLE 'table_name' DROP INDEX 'index_name';
```

6. Choose Next.

### Step 7: Create and Run Your AWS DMS Migration Task

Using an AWS DMS task, you can specify what schema to migrate and the type of migration. You can migrate existing data, migrate existing data and replicate ongoing changes, or replicate data changes only.

1. In the AWS DMS console, on the **Create task** page, specify the task options. The following table describes the settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task name</td>
<td>Enter a name for the migration task.</td>
</tr>
<tr>
<td>Task description</td>
<td>Enter a description for the task.</td>
</tr>
<tr>
<td>Source endpoint</td>
<td>Shows the SQL Server source endpoint. If you have more than one endpoint in the account, choose the correct endpoint from the list.</td>
</tr>
<tr>
<td>Target endpoint</td>
<td>Shows the Aurora MySQL target endpoint.</td>
</tr>
<tr>
<td>Replication instance</td>
<td>Shows the AWS DMS replication instance.</td>
</tr>
<tr>
<td>Migration type</td>
<td>Choose an option.</td>
</tr>
</tbody>
</table>

- **Migrate existing data** - AWS DMS migrates only your existing data. Changes to your source data aren’t captured and applied to your target. If you can afford to take an outage for the duration of the full load, then this is the simplest option. You can also use this option to create test copies of your database. If the source SQL Server database is an Amazon RDS database, you must choose this option.

- **Migrate existing data and replicate ongoing changes** - AWS DMS captures changes while migrating your existing data. AWS DMS continues to capture and apply changes even after the bulk data has been loaded. Eventually the source and target databases are in sync, allowing for a minimal downtime.

- **Replicate data changes only** - Bulk load data using a different method. This approach generally applies only to homogeneous migrations.
Database Migration Guide Step-by-Step Walkthroughs
Step-by-Step Migration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Start task on create</strong></td>
<td>In most situations, you should choose this option. Sometimes, you might want to delay the start of a task, for example, if you want to change logging levels.</td>
</tr>
</tbody>
</table>

The page should look similar to the following:

2. Under **Task settings**, specify the settings. The following table describes the settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Target table preparation mode</strong></td>
<td>Choose an option.</td>
</tr>
<tr>
<td>* Do nothing</td>
<td>AWS DMS does nothing to prepare your tables. Your table structure remains the same, and any existing data remains in the table. You can use this method to consolidate data from multiple systems.</td>
</tr>
<tr>
<td>* Drop tables on target</td>
<td>AWS DMS creates your target tables for you. AWS DMS drops and re-creates the tables to migrate before migration. AWS DMS creates the table and a primary key only for heterogeneous migrations.</td>
</tr>
<tr>
<td>* Truncate</td>
<td>AWS DMS truncates a target table before loading it. If the target table doesn't exist, then AWS DMS creates it.</td>
</tr>
</tbody>
</table>

IMPORTANT: If the AWS Schema Conversion Tool already created the tables on the target, choose **Do nothing** or **Truncate**.

<table>
<thead>
<tr>
<th>Include LOB columns in replication</th>
<th>Choose an option.</th>
</tr>
</thead>
</table>

145
### Database Migration Guide Step-by-Step Walkthroughs

**Step-by-Step Migration**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Don’t include LOB columns*</td>
<td>Do not migrate LOB data.</td>
</tr>
<tr>
<td>* Full LOB mode*</td>
<td>AWS DMS migrates all LOBs (large objects) from the source to the target regardless of size. In this configuration, AWS DMS has no information about the maximum size of LOBs to expect. Thus, LOBs are migrated one at a time, piece by piece. Full LOB mode can be relatively slow.</td>
</tr>
<tr>
<td>* Limited LOB mode*</td>
<td>You set a maximum size LOB that AWS DMS accepts. This option enables AWS DMS to pre-allocate memory and load the LOB data in bulk. LOBs that exceed the maximum LOB size are truncated, and a warning is issued to the log file. In limited LOB mode, you get significant performance gains over full LOB mode. We recommend that you use limited LOB mode whenever possible.</td>
</tr>
</tbody>
</table>

**Max LOB size (kb)**  
When **Limited LOB mode** is selected, this option determines the maximum LOB size that AWS DMS accepts. Any LOBs that are larger than this value are truncated to this value.

**Enable logging**  
It’s best to select **Enable logging**. If you enable logging, you can see any errors or warnings that the task encounters, and you can troubleshoot those issues.

3. Leave the Advanced settings at their default values.
4. If you created and exported mapping rules with AWS SCT in the last step in Step 4: Convert the SQL Server Schema to Aurora MySQL (p. 135), choose **Table mappings**, and select the **JSON** tab. Then select **Enable JSON editing**, and enter the table mappings you saved.

If you did not create mapping rules, then proceed to the next step.
5. Choose **Create task**. The task starts immediately.

The **Tasks** section shows you the status of the migration task.

If you chose **Enable logging** during setup, you can monitor your task. You can then view the Amazon CloudWatch metrics.

1. On the navigation pane, choose **Tasks**.
2. Choose your migration task.
3. Choose the **Task monitoring** tab, and monitor the task in progress on that tab.
When the full load is complete and cached changes are applied, the task stops on its own.

4. On the target Aurora MySQL database, if you disabled foreign key constraints and triggers, enable them using the script that you saved previously.

5. On the target Aurora MySQL database, re-create the secondary indexes if you removed them previously.

6. If you chose to use AWS DMS to replicate changes, in the AWS DMS console, start the AWS DMS task by choosing Start/Resume for the task.

   Important replication instance metrics to monitor include the following:
   • CPU
   • FreeableMemory
   • DiskQueueDepth
   • CDCLatencySource
   • CDCLatencyTarget

   The AWS DMS task keeps the target Aurora MySQL database up to date with source database changes. AWS DMS keeps all the tables in the task up to date until it's time to implement the application migration. The latency is zero, or close to zero, when the target has caught up to the source.

   For more information, see https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Monitoring.html.

**Step 8: Cut Over to Aurora MySQL**

To move connections from your Microsoft SQL Server database to your Amazon Aurora MySQL database, do the following:

1. End all SQL Server database dependencies and activities, such as running scripts and client connections. Ensure that the SQL Server Agent service is stopped.

   The following query should return no results other than your connection:

   ```sql
   SELECT session_id, login_name from sys.dm_exec_sessions where session_id > 50;
   ```

2. Kill any remaining sessions (other than your own).

   ```sql
   KILL session_id;
   ```

3. Shut down the SQL Server service.

4. Let the AWS DMS task apply the final changes from the SQL Server database on the Amazon Aurora MySQL database.

5. In the AWS DMS console, stop the AWS DMS task by choosing Stop for the task, and then confirming that you want to stop the task.

**Troubleshooting**

When you work with Microsoft SQL Server as a source database and Amazon Aurora MySQL as a target database, the two most common problem areas are SQL Server change data capture (CDC) and foreign keys.

- MS-CDC: If you are using MS-CDC with SQL Server for the migration, errors that are related to permissions or errors during change data capture are common. These types of errors usually result
when one of the prerequisites was not met. For example, the most common overlooked prerequisite is a full database backup.

- **Foreign keys:** During the full load process, AWS DMS does not load tables in any particular order, so it might load the child table data before parent table data. As a result, foreign key constraints might be violated if they are enabled. You should disable foreign keys on the Aurora MySQL target database. You can enable the foreign keys on the target after the migration is complete.

For more tips, see the AWS DMS troubleshooting section in the User Guide.

To troubleshoot issues specific to SQL Server, see the SQL Server troubleshooting section:

- **Troubleshooting Microsoft SQL Server Specific Issues**

To troubleshoot Aurora MySQL issues, see the Aurora MySQL troubleshooting section and the MySQL troubleshooting section:

- **Troubleshooting Amazon Aurora MySQL Specific Issues**
- **Troubleshooting MySQL Specific Issues**

---

**Migrating an Amazon RDS for SQL Server Database to an Amazon S3 Data Lake**

This walkthrough gets you started with the process of migrating from an Amazon Relational Database Service (Amazon RDS) for Microsoft SQL Server to Amazon Simple Storage Service (Amazon S3) cloud data lake using AWS Database Migration Service (AWS DMS).

For most organizations, data is distributed across multiple systems and data stores to support varied business needs. On-premises data stores struggle to scale performance as data sizes and formats grow exponentially for analytics and reporting purposes. These limitations in data storage and management limit efficient and comprehensive analytics.

Amazon S3 based data lakes provide reliable and scalable storage, where you can store structured, semi-structured and unstructured datasets for varying analytics needs. You can integrate Amazon S3 based data lakes with distributed processing frameworks such as Apache Spark, Apache Hive, and Presto to decouple compute and storage, so that both can scale independently.

**Topics**

- Why Amazon S3? (p. 148)
- Why AWS DMS? (p. 149)
- Solution Overview (p. 149)
- Prerequisites (p. 151)
- Step-by-Step Migration (p. 151)

**Why Amazon S3?**

Amazon S3 is an object storage service for structured, semi-structured, and unstructured data that offers industry-leading scalability, data availability, security, and performance. With a data lake built on Amazon S3, you can use native AWS services, optimize costs, organize data, and configure fine-tuned access controls to meet specific business, organizational, and compliance requirements.
Amazon S3 is designed for 99.999999999% (11 9s) of data durability. The service automatically creates and stores copies of all uploaded S3 objects across multiple systems. This means your data is available when needed and protected against failures, errors, and threats.

Amazon S3 is secure by design, scalable on demand, and durable against the failure of an entire AWS Availability Zone. You can use AWS native services and integrate with third-party service providers to run applications on your data lake.

Why AWS DMS?

Data lakes typically require building, configuring, and maintaining multiple data ingestion pipelines from cloud and on-premises data stores.

Traditionally, databases can be loaded once with data ingestion tools such as import, export, bulk copy, and so on. Ongoing changes are either not possible or are implemented by bookmarking the initial state. Setting up a data lake using these methods can present challenges ranging from increased load on the source database to overheads while carrying schema changes.

AWS DMS supports a one-time load and near-real-time ongoing replication making the data migration seamless, while supporting multiple source and target database platforms. One of the common use cases is the need to derive insights on data stored in several sources. For example, you may need to identify monthly sales for a specific year on sales data stored on different database instances.

As a part of this walkthrough, we will configure AWS DMS to move data from an Amazon RDS for SQL Server database instance to Amazon S3 for a sales analytics use case.

**Note**
This introductory exercise doesn't cover all use cases of migrating to Amazon S3 but provides an overview of the migration process using AWS DMS. This example covers commonly faced problems and describes best practices to follow when migrating to an Amazon S3 data lake.

Solution Overview

The following diagram displays a high-level architecture of the solution, where we use AWS DMS to move data from Microsoft SQL Server databases hosted on Amazon Relational Database Service (Amazon RDS) to Amazon Simple Storage Service (Amazon S3).
The following diagram shows the structure of the Amazon S3 bucket from the preceding diagram.

To replicate data, you need to create and configure the following artifacts in AWS DMS:

- **Replication Instance** — An AWS managed instance that hosts the AWS DMS engine. You control the type or size of the instance based on the workload you plan to migrate.
- **Source Endpoint** — An endpoint that provides connection details, data store type, and credentials to connect to a source database. For this use case, we will configure the source endpoint to point to the Amazon RDS for SQL Server database.
- **Target Endpoint** — AWS DMS supports several target systems including Amazon RDS, Amazon Aurora, Amazon Redshift, Amazon Kinesis Data Streams, Amazon S3, and more. For the use case, we will configure Amazon S3 as the target endpoint.
Database Migration Guide Step-by-Step Walkthroughs

Prerequisites

- **Replication Task** — A task that runs on the replication instance and connects to endpoints to replicate data from the source database to the target database.

For this walkthrough, we will use the AdventureWorks sample database on an Amazon RDS for SQL Server instance as the base data for the walkthrough. The AdventureWorks database holds sales, marketing, and order data. We will use AWS DMS to move sales data from the source database to Amazon S3 object store, which can be used as a data lake for downstream analytics needs.

**Note**
You can refer to the section called “Migrating a SQL Server Always On Database to AWS” (p. 34) for details on migrating from a Microsoft SQL Server Always On database instance.

We will create an AWS DMS task, which will perform a one-time full load to migrate a point in time snapshot and will then stream incremental data to the target Amazon S3 bucket. This way, sales data in the S3 bucket will be kept in sync with the source database.

**Prerequisites**

The following prerequisites are required to complete this walkthrough:

- Understand Amazon Relational Database Service (Amazon RDS), the applicable database technologies, and SQL.
- Create an AWS account with AWS Identity and Access Management (IAM) credentials that allows you to launch Amazon RDS and AWS Database Migration Service (AWS DMS) instances in your AWS Region. For information about IAM credentials, see [Create an IAM user](https://docs.aws.amazon.com/iam/latest/userguide/create-user.html).
- Understand the Amazon Virtual Private Cloud (Amazon VPC) service and security groups. For information about using Amazon VPC with Amazon RDS, see [Amazon Virtual Private Cloud (VPCs) and Amazon RDS](https://docs.aws.amazon.com/vpc/latest/userguide/). For information about Amazon RDS security groups, see [Controlling access with security groups](https://docs.aws.amazon.com/rds/latest/userguide/rds-security-groups.html).
- Understand the supported features and limitations of AWS DMS. For information about AWS DMS, see [Getting started with Database Migration Service](https://docs.aws.amazon.com/dms/latest/userguide/Welcome.html).
- Understand how to work with Microsoft SQL Server as a source and Amazon S3 data lake as a target. For information about working with SQL Server as a source, see [Using a Microsoft SQL Server database as a source](https://docs.aws.amazon.com/dms/latest/userguide/). For information about working with Amazon S3 as a target, see [Using Amazon S3 as a target](https://docs.aws.amazon.com/dms/latest/userguide/).
- Understand the supported data type conversion options for SQL Server and Amazon S3. For information about data types for SQL Server as a source, see [Source data types for SQL Server](https://docs.aws.amazon.com/dms/latest/userguide/dms-database-data-type-transformations-instancesource.html). For information about data types for Amazon S3 as a target (Parquet only), see [Target data types for S3 Parquet](https://docs.aws.amazon.com/dms/latest/userguide/dms-database-data-type-transformations-target.html).
- Audit your source SQL Server database. For each schema and all the objects under each schema, determine whether any of the objects are no longer being used. Deprecate these objects on the source SQL Server database, because there’s no need to migrate them if they aren’t being used.

For more information about AWS DMS, see [Getting started with Database Migration Service](https://docs.aws.amazon.com/dms/latest/userguide/Welcome.html).

**Step-by-Step Migration**

The following steps provide instructions for migrating an Amazon RDS for SQL Server database to an Amazon S3 data lake. These steps assume that you have already prepared your source database as described in the section called “Prerequisites” (p. 151).

**Topics**

- **Step 1: Create an AWS DMS Replication Instance** (p. 152)
- **Step 2: Configure a Source Amazon RDS for SQL Server Database** (p. 153)
Step 1: Create an AWS DMS Replication Instance

To create an AWS Database Migration Service (AWS DMS) replication instance, see Creating a replication instance. Usually, the full load phase is multi-threaded (depending on task configurations) and has a greater resource footprint than ongoing replication. Consequently, it's advisable to start with a larger instance class and then scale down once the task is in the ongoing replication phase. Moreover, if you intend to migrate your workload using multiple tasks, monitor your replication instance metrics and resize your instance accordingly.

For this use case, we will migrate a subset (the Sales schema) of the AdventureWorks database, which is over 3 GB in size. Because we perform a heterogenous migration without many LOB columns, we can start with a compute optimized instance like c5.xlarge running the latest AWS DMS engine version. We can later scale up or down based on resource utilization during task execution.

Note
Scaling replication instance during full load and ongoing replication phases is usually based on CloudWatch metrics such as CPU, memory, I/O, and so on. Choosing the appropriate replication instance class and size depends on several factors such as number of tasks, table size, DML activity, size of transactions, Large Objects (LOB), and so on. This is out of scope for this walkthrough. To learn more about these topics, see Choosing replication instance types and Sizing a replication instance.

To create an AWS DMS replication instance, do the following:

1. Sign in to the AWS Management Console, and open the AWS DMS console.
2. If you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information about the permissions required, see IAM permissions.
3. On the Welcome page, choose Create replication instance to start a database migration.
4. On the Create replication instance page, specify your replication instance information.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter datalake-migration-ri. If you are using multiple replication servers or sharing an account, choose a name that helps you quickly differentiate between the different servers.</td>
</tr>
<tr>
<td>Description</td>
<td>Enter Migrate SQL Server to Amazon S3 data lake.</td>
</tr>
<tr>
<td>Instance class</td>
<td>Choose dms.c5.xlarge. Each size and type of instance class has increasing CPU, memory, and I/O capacity.</td>
</tr>
<tr>
<td>Engine version</td>
<td>Leave the default value, which is the latest stable version of the AWS DMS replication engine.</td>
</tr>
<tr>
<td>Allocated storage (GiB)</td>
<td>Choose 50.</td>
</tr>
<tr>
<td>For This Parameter</td>
<td>Do This</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------</td>
</tr>
<tr>
<td>VPC</td>
<td>Choose the virtual private cloud (VPC) in which your replication instance will launch. If possible, select the same VPC in which either your source or target database resides (or both).</td>
</tr>
<tr>
<td>Multi AZ</td>
<td>If you choose Yes, AWS DMS creates a second replication server in a different Availability Zone for failover if there is a problem with the primary replication server.</td>
</tr>
<tr>
<td>Publicly accessible</td>
<td>If either your source or target database resides outside of the VPC in which your replication server resides, you must make your replication server policy publicly accessible.</td>
</tr>
</tbody>
</table>

5. Choose Create.

**Step 2: Configure a Source Amazon RDS for SQL Server Database**

One of the primary considerations when setting up AWS DMS replication is the load that it induces on the source database. During full load, AWS DMS tasks initiate two or three connections for each table that is configured for parallel load. Because AWS DMS settings and data volumes vary across tasks, workloads, and even across different runs of the same task, providing an estimate of resource utilization that applies for all use cases is difficult.

Ongoing replication is single-threaded and it usually consumes less resources than full load. Providing estimates for change data capture (CDC) resource utilization has the same challenges described before.

That said, you can estimate the expected increase in load on your source Amazon RDS instance, by running test AWS DMS tasks on replicas of your source Amazon RDS for SQL Server instance and monitoring the CPU, memory, IO and throughput metrics.

For our source database, we use an m5.xlarge Amazon RDS instance running Microsoft SQL Server 2019. While the steps for Amazon RDS for SQL Server creation are out of scope for this walkthrough (for more information, see the section called “Prerequisites” (p. 151)), make sure that your Amazon RDS instance has **Automatic Backups** turned on so that the recovery model for the database is set to **FULL**. This is a pre-requisite for ongoing replication with AWS DMS. You can turn on these settings when you create or modify an existing Amazon RDS instance.

The following image displays the database settings required for ongoing replication with AWS DMS.
To perform the full load phase, AWS DMS requires read privileges to the tables in scope for migration. For more information on required permissions, see Permissions for full load only tasks.

Connect to the Amazon RDS for SQL Server instance and run the following queries. Use a login with master user privileges for both full load and CDC.

```sql
USE AdventureWorks;
CREATE LOGIN dms_user WITH PASSWORD = 'password'
CREATE USER dms_user FOR LOGIN dms_user
ALTER ROLE [db_datareader] ADD MEMBER dms_user
ALTER ROLE [db_owner] ADD MEMBER dms_user
GRANT VIEW DATABASE STATE to dms_user
USE master;
GRANT VIEW SERVER STATE TO dms_user
```

**Note**
Here, we create a new user to perform the migration. You can skip this step if you plan to use existing logins and users that have the required privileges.

Turn on MS-CDC for your Amazon RDS for SQL Server database instance at the database level.

```
exec msdb.dbo.rds_cdc_enable_db 'AdventureWorks'
```

Because we migrate all tables in the Sales schema of the AdventureWorks database, we need to identify the total number of tables.

```sql
SELECT TABLE_SCHEMA, TABLE_NAME, TABLE_TYPE
FROM information_schema.tables
WHERE TABLE_SCHEMA = 'Sales'
ORDER BY TABLE_NAME
```

Then we need to divide tables in the following groups:
- Tables with a primary key.
- Tables with a unique index without primary key.
- Tables without a primary key and unique index.

We use the information_schema to identify tables that have a primary key or a unique index without a primary key.

```sql
SELECT a.TABLE_SCHEMA, a.TABLE_NAME, a.CONSTRAINT_TYPE, CONSTRAINT_NAME
FROM information_schema.table_constraints a
JOIN information_schema.tables b ON a.TABLE_SCHEMA = b.TABLE_SCHEMA
AND a.TABLE_NAME  = b.TABLE_NAME
WHERE b.TABLE_TYPE = 'BASE TABLE'
AND a.TABLE_SCHEMA = 'Sales'
AND a.CONSTRAINT_TYPE in ('UNIQUE','PRIMARY KEY')
ORDER BY a.TABLE_SCHEMA, a.TABLE_NAME
```

The query results show that the task has 19 tables and all of them have primary keys. For all these tables, run the following query to turn on MS-CDC at the table level.

```sql
exec sys.sp_cdc_enable_table
@source_schema = N'Sales',
@source_name = N'table_name',
@role_name = NULL,
@supports_net_changes = 1
```

Now, set the retention period for changes to be available on the source using the following commands. Set the pollinginterval value to 86399 seconds to increase the retention of changes on the Amazon RDS for SQL Server instance.

```sql
EXEC sys.sp_cdc_change_job @job_type = 'capture' ,@pollinginterval = 86399
exec sp_cdc_stop_job 'capture'
exec sp_cdc_start_job 'capture'
exec sys.sp_cdc_help_jobs
```

Set the polling interval on your secondary database to 86399 seconds too. For most use cases these settings should be enough. For databases that have a large number of transactions, you need to make additional configuration changes to make sure that the transaction log has optimal retention. For more information, see Optional settings when using Amazon RDS for SQL Server as a source.

For more information about ongoing replication, see Setting up ongoing replication on a Cloud SQL Server DB instance.

**Note**

AWS DMS does not support replicating ongoing changes from views. For more information, see Selection rules and actions.

In this walkthrough, we focus on migrating the tables and do not include views in the migration scope. You should also look at estimating the number of records in the tables you are going to migrate as this is a useful consideration while configuring AWS DMS tasks.

**Step 3: Create an AWS DMS Source Endpoint**

After you configured the AWS Database Migration Service (AWS DMS) replication instance and the source Amazon RDS for SQL Server instance, ensure connectivity between these two instances. To ensure that the replication instance can access the server and the port for the database, make changes to the relevant security groups and network access control lists. For more information about your network configuration, see Setting up a network for a replication instance.
After you completed the network configurations, you can create a source endpoint.

To create a source endpoint, do the following:

2. Choose **Endpoints**.
3. Choose **Create endpoint**.
4. On the **Create endpoint** page, enter the following information.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Endpoint type</strong></td>
<td>Choose <strong>Source endpoint</strong>, turn on <strong>Select RDS DB instance</strong>, and choose <strong>datalake-source-db</strong> RDS instance.</td>
</tr>
<tr>
<td><strong>Endpoint identifier</strong></td>
<td>Enter <strong>datalake-source-db</strong>.</td>
</tr>
<tr>
<td><strong>Source engine</strong></td>
<td>Choose <strong>Microsoft SQL Server</strong>.</td>
</tr>
<tr>
<td><strong>Access to endpoint database</strong></td>
<td>Choose <strong>Provide access information manually</strong>.</td>
</tr>
<tr>
<td><strong>Server name</strong></td>
<td>Enter the database server name on Amazon RDS.</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>Enter <strong>1433</strong>.</td>
</tr>
<tr>
<td><strong>Secure Socket Layer (SSL) mode</strong></td>
<td>Choose <strong>none</strong>.</td>
</tr>
<tr>
<td><strong>User name</strong></td>
<td>Enter <strong>dms_user</strong>.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Enter the password that you created for the <strong>dms_user</strong> user.</td>
</tr>
<tr>
<td><strong>Database name</strong></td>
<td>Enter <strong>AdventureWorks</strong>.</td>
</tr>
</tbody>
</table>

5. Choose **Create endpoint**.

**Note**
To migrate a Microsoft SQL Server Always On database, you need to use different configurations. For more information, see the section called “Migrating a SQL Server Always On Database to AWS” (p. 34).

**Step 4: Configure a Target Amazon S3 Bucket**

You can integrate Amazon S3 with other AWS and third-party services to take advantage of the following:

- Data analysis using Amazon Athena query engine. This service helps reduce cost as you do not pay for dedicated resources and instead pay based on the amount data being scanned.
- Perform extract, transform, and load (ETL) operations using distributed processing frameworks such as Spark with Amazon EMR or AWS Glue.
- Implement machine learning use cases, because Amazon S3 can store granular time series data spanning years in raw form, in conjunction with Amazon SageMaker.

Because in this use case we migrate the Sales schema to Amazon S3, we need to account for future use cases of the migrated data before we set up Amazon S3 bucket and AWS DMS endpoints.

To create the Amazon S3 bucket, do the following:
2. Choose Create bucket.
3. For Bucket name, enter adventure-works-datalake.
4. For AWS Region, choose the region that hosts your AWS DMS replication instance.
5. Leave the default values in the other fields and choose Create bucket.

Step 5: Configure an AWS DMS Target Endpoint

To use Amazon S3 as an AWS Database Migration Service (AWS DMS) target endpoint, create an IAM role with write and delete access to the S3 bucket. Then add DMS (dms.amazonaws.com) as trusted entity in this IAM role. For more information, see Prerequisites for using Amazon S3 as a target.

When using AWS DMS to migrate data to an Amazon Simple Storage Service (Amazon S3) data lake, you can change the default task behavior, such as file formats, partitioning, file sizing, and so on. This leads to minimizing post-migration processing and helps downstream applications consume data efficiently. You can customize task behavior using endpoint settings and extra connection attributes (ECA). Most of the Amazon S3 endpoint settings and ECA settings overlap, except for a few parameters. In this walkthrough, we will configure Amazon S3 endpoint settings.

Choose File Format

AWS DMS supports data replication through comma-separated values (CSV) or Apache Parquet file formats. Each file format has its own benefits. Choose the right file format depending on your consumption pattern.

Apache Parquet is a columnar format, which is built to support efficient compression and encoding schemes providing storage space savings and performance benefits. With Parquet, you can specify compression schemes for each column to improve query performance when using avg(), max(), or other column level aggregation operations. That is why Parquet is popular for data lake and analytics use cases.

CSV files are helpful when you plan to keep data in human readable format, share or transfer Amazon S3 files into other downstream systems for further processing.

For this walkthrough, we will use the Parquet file format. Specify the following endpoint settings.

```
DataFormat=parquet
ParquetVersion=PARQUET_2_0
```

Determine File Size

By default, during ongoing replication AWS DMS tasks writes to Amazon S3 are triggered either if the file size reaches 32 KB or if the previous file write was more than 60 seconds ago. These settings ensure that the data capture latency is less than a minute. However, this approach creates a large number of small files in target Amazon S3 bucket.

Because we migrate our source Sales database schema for an analytics use case, some latency is acceptable. However, we need to optimize this schema for cost and performance. When you use distributed processing frameworks such as Amazon Athena, AWS Glue or Amazon EMR, it is recommended to avoid too many small files (less than 64 MB). Small files create management overhead for the driver node of the distributed processing framework.

Because we plan to use Amazon Athena to query data from our Amazon S3 bucket, we need to make sure our target file size is at least 64 MB. Specify the following endpoint settings: CdcMaxBatchInterval=3600 and CdcMinFileSize=64000. These settings ensure that AWS DMS writes the file until its size reaches 64 MB or if the last file write was more than an hour ago.
Note
Parquet files created by AWS DMS are usually smaller than the specified CdcMinFileSize setting because Parquet data compression ratio varies depending on the source data set. The size of CSV files created by AWS DMS is equal to the value specified in CdcMinFileSize.

Turn on S3 Partitioning
Partitioning in Amazon S3 structures your data by folders and subfolders that help efficiently query data. For example, if you receive sales record data daily from different regions and you query data for a specific region and find stats for a few months, then it is recommended to partition data by region, year, and month. In Amazon S3, the path for our use case looks as following:

`s3://<sales-data-bucket-name>/<region>/<schemaname>/<tablename>/<year>/<month>/<day>`

- `s3://adventure-works-datalake`
- `s3://adventure-works-datalake/US-WEST-DATA`
- `s3://adventure-works-datalake/US-WEST-DATA/Sales/CreditCard/LOAD00000001.parquet`

Partitioning provides performance benefits because data scanning will be limited to the amount of data in the specific partition based on the filter condition in your queries. For our sales data example, your queries might look as follows:

```sql
SELECT <column-list> FROM <sales-table-name> WHERE <region> = <region-name> AND <year> = <year-value>
```

If you use Amazon Athena to query data, partitioning helps reduce cost as Athena pricing is based on the amount of data that you scan when running queries.

To turn on partitioning for ongoing changes in the preceding format, use the following queries.

```sql
bucketFolder=US-WEST-DATA
DatePartitionedEnabled=true
DatePartitionSequence=YYYYMMDD
DatePartitionDelimiter=SLASH
```

Other Considerations
The preceding settings help optimize performance and cost. We also need to configure additional settings because:

- Our use case does not have a fixed end-date.
- We need to minimize issues arising from misconfigurations or retroactive changes.
- We want to minimize recovery time in case of unforeseen issues.

Serialize ongoing replication events
A common challenge when using Amazon S3 as a target involves identifying the ongoing replication event sequence when multiple records are updated at the same time on the source database.
AWS DMS provides two options to help serialize such events for Amazon S3. You can use the TimeStampColumnName endpoint setting or use transformation rules to include LSN column. Here, we will discuss the first option. For more information about the second option, see the section called “Step 6: Create an AWS DMS Task” (p. 162).

**Use the TimeStampColumnName endpoint setting**

The TimeStampColumnName setting adds an additional STRING column to the target Parquet file created by AWS DMS. During the ongoing replication, the column value represents the commit timestamp of the event in SQL Server. For the full load phase, the columns values represent the timestamp of data transfer to Amazon S3.

The default format is `yyyy-MM-dd HH:mm:ss.SSSSSS`. This format provides a microsecond precision but depends on the source database transaction log timestamp precision. The following image shows the seven microseconds difference between two operations in the `sourceRecordTime` field.
Note
Because TimeStampColumnName is an endpoint setting, all tasks that use this endpoint, will include this column for all tables.

Include full load operation field

All files created during the ongoing replication, have the first column marked with I, U, or D. These symbols represent the DML operation on the source and stand for Insert, Update, or Delete operations.

For full load files, you can add this column by configuring the endpoint setting.
This ensures that all full load files are marked with an I operation.

When you use this approach, new subscribers can consume the entire data set or prepare a fresh copy in case of any downstream processing issues.

**Create a Target Endpoint**

After you completed all settings configurations, you can create a target endpoint.

To create a target endpoint, do the following:

2. Choose **Endpoints**, and then choose **Create endpoint**.
3. On the **Create endpoint** page, enter the following information.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint type</td>
<td>Choose <strong>Target endpoint</strong>, and turn off <strong>Select RDS DB instance</strong>.</td>
</tr>
<tr>
<td>Endpoint identifier</td>
<td>Enter <strong>adventure-works-datalake-target</strong>.</td>
</tr>
<tr>
<td>Target engine</td>
<td>Choose <strong>Amazon S3</strong>.</td>
</tr>
<tr>
<td>Service access role ARN</td>
<td>Enter the IAM role that can access your Amazon S3 data lake.</td>
</tr>
<tr>
<td>Bucket name</td>
<td>Enter <strong>adventure-works-datalake</strong>.</td>
</tr>
<tr>
<td>Bucket folder</td>
<td>Enter <strong>US-WEST-DATA</strong>.</td>
</tr>
</tbody>
</table>

4. Expand the **Endpoint settings** section, choose **Wizard**, and then choose **Add new setting** to add the settings as shown on the following image.
Step 6: Create an AWS DMS Task

After you configured the replication instance and endpoints, you need to analyze your source database. A good understanding of the workload helps plan an effective migration approach and minimize configuration issues. Find some of the important considerations following and learn how they apply to our walkthrough.

**Size and number of records**

The volume of migrated records affects the full load completion time. It is difficult to predict the full load time upfront, but testing with a replica of a production instance should provide a baseline. Use this estimate to decide whether you should parallelize full load by using multiple tasks or by using the parallel load option.

The `Sales` schema includes 19 tables. The `CreditCard` table is the largest table containing 100,000 records. We can increase the number of tables loaded in parallel to 19 if the full load is slow. The default value for the number of tables loaded in parallel is eight.

**Transactions per second**

While full load is affected by the number of records, the ongoing replication performance relies on the number of transactions on the source Amazon RDS for SQL Server database. Performance issues during change data capture (CDC) generally stem from resource constraints on the source database, replication instance, target database, and network bandwidth or throughput. Knowing average and peak TPS on the source and recording CDC throughput and latency metrics help baseline (AWS DMS) performance and identify an optimal task configuration. For more information, see [Replication task metrics](#).
In this walkthrough, we will track the CDC latency and throughput values after the task moves into the ongoing replication phase to baseline AWS DMS performance.

LOB columns

AWS DMS handles large binary objects (LOBs) columns differently compared to other data types. For more information, see [Migrating large binary objects (LOBs)](https://docs.aws.amazon.com/dms/latest/migration-guide/cdc-latency-throughput.html).

Because AWS DMS does not support **Full LOB mode** for Amazon S3 endpoints, we need to identify a suitable **LOB Max Size**.

A detailed explanation of LOB handling by AWS DMS is out of scope for this walkthrough. However, remember that increasing the **LOB Max Size** increases the tasks memory utilization. Because of that, it is recommended not to set **LOB Max Size** to a large value.

For more information about LOB settings, see the section called “Task Configuration” (p. 163).

Unsupported data types

Identify data types used in tables and check that AWS DMS supports these data types. For more information, see [Source data types for SQL Server](https://docs.aws.amazon.com/dms/latest/migration-guide/unsupported-data-type.html).

Validate that the target Amazon S3 has the corresponding data types. For more information, see [Target data types for S3 Parquet](https://docs.aws.amazon.com/dms/latest/migration-guide/unsupported-data-type.html).

After running the initial load test, validate that AWS DMS converted data as you expected. You can also initiate a pre-migration assessment to identify any unsupported data types in the migration scope. For more information, see [Specifying individual assessments](https://docs.aws.amazon.com/dms/latest/migration-guide/unsupported-data-type.html).

**Note**

The preceding list is not complete. For more information, see [Best practices](https://docs.aws.amazon.com/dms/latest/migration-guide/unsupported-data-type.html).

Combining the considerations from the preceding list, we start with a single task that migrates all 19 tables. Based on the full load run time and resource utilization metrics on the source SQL Server database instance and replication instance, we can evaluate if we should parallelize the load further to improve performance.

**Task Configuration**

In an AWS DMS task, you can specify the schema or table to migrate, the type of migration, and the configurations for the migration. You can choose one of the following options for your task.

- **Full Load only** — migrate existing data.
- **Full Load + CDC** — migrate existing data and replicate ongoing changes.
- **CDC only** — replicate ongoing changes.

For more information about the task creation steps and available configuration options, see [Creating a task](https://docs.aws.amazon.com/dms/latest/migration-guide/unsupported-data-type.html).

In this walkthrough, we will focus on the following settings.

**Table mappings**

Use selection rules to define the schemas and tables that the AWS DMS task will migrate. For more information, see [Selection rules and actions](https://docs.aws.amazon.com/dms/latest/migration-guide/unsupported-data-type.html).

Because we need to identify monthly sales for a specific year, one possible approach can restrict the migration to `SalesOrder%` tables in the `Sales` schema and keep adding new tables to the task.
when additional reporting is required. This approach saves cost and minimizes the load, but increases operational overhead by requiring repeated configurations, performance baselining, and so on. For the walkthrough, we will migrate all tables (%) from the Sales schema.

**Using transformation rules to include LSN column**

In the previous section we discussed using the TimestampColumnName endpoint setting to serialize ongoing replication events. For more information about using the TimestampColumnName endpoint setting, see the section called “Serialize ongoing replication events” (p. 158).

Because the source database transaction log precision is limited to milliseconds, multiple events can have the same timestamp. To address this issue, you can use task level transformation rules to include source table headers to the Amazon S3 target files as described in the task creation section.

Source table headers add an additional column that contains the log sequence number (LSN) value of the operation from the source SQL Server database instance. You can use this information in our Amazon S3 data lake scenario for downstream serialization. For more information about source table headers, see Replicating source table headers using expressions.

To include headers, add the following transformation rule in the JSON editor in table mapping. This rule adds a new transact.id column with the LSN to all tables that the task migrates. For more information, see Specifying table selection and transformations rules using JSON.

```
{
  "rule-type": "transformation",
  "rule-id": "2",
  "rule-name": "2",
  "rule-target": "column",
  "object-locator": {
    "schema-name": "%",
    "table-name": "%"
  },
  "rule-action": "add-column",
  "value": "transact_id",
  "expression": "$AR_H_STREAM_POSITION",
  "data-type": {
    "type": "string",
    "length": 50
  }
}
```

**Note**

Mapping rules are applied at the task level. You need to add a mapping rule to each task that replicates data to your data lake.

**LOB settings**

Use the sys schema to identify the LOB columns in the tables of the Sales schema.

```
SELECT s.name AS SchemaName,
     t.name AS TableName,
     c.name AS ColumnName,
     y.name AS DataType
FROM sys.tables AS t
INNER JOIN sys.schemas AS s ON s.schema_id = t.schema_id
INNER JOIN sys.columns AS c ON t.object_id = c.object_id
INNER JOIN sys.types AS y ON y.user_type_id = c.user_type_id
WHERE (c.user_type_id in (34,35,99,129,130,241,256) OR (c.user_type_id in (165,167,231) AND c.max_length = -1))
AND s.name = 'Sales'
ORDER BY t.name;
```
The Sales.Store table includes one LOB column. Use the following query to identify the size of the largest LOB in the migrated tables.

```
select max(datalength(Demographics)) as "Size in Bytes" from Sales.Store
```

The size of the largest LOB is 1,000 bytes. Because of that, we will leave the default value for LOB Max Size, which is 32 KB. If the size of the largest LOB is more than 32 KB, it is recommended to factor in LOB growth over time, include some buffer, and set that as the LOB Max Size value.

**Other task settings**

Choose **Enable CloudWatch Logs** to upload the AWS DMS task execution log to Amazon CloudWatch. You can use these logs to troubleshoot issues because they include error and warning messages, start and end times of the run, configuration issues, and so on. Changes to the task logging setting, such as enabling debug or trace can also be helpful to diagnose performance issues.

**Note**

CloudWatch log usage is charged at standard rates. For more information, see [Amazon CloudWatch pricing](https://aws.amazon.com/cloudwatch/pricing/).

For **Target table preparation mode**, choose one of the following options: Do nothing, truncate, and Drop. Use Truncate in data pipelines where the downstream systems rely on a fresh dump of clean data and do not rely on historical data. In this walkthrough, we choose **Do nothing** because we want to control the retention of files from previous runs.

For **Maximum number of tables to load in parallel**, enter the number of parallel threads that AWS DMS initiates during full load. You can increase this value to improve the full load performance and minimize the load time when you have numerous tables.

**Note**

Increasing this parameter induces additional load on the source database, replication instance, and target database.

**Create an AWS DMS Task**

After you completed all settings configurations, you can create an AWS DMS database migration task.

To create a database migration task, do the following:

2. Choose **Database migration tasks**, and then choose **Create task**.
3. On the **Create database migration task** page, enter the following information.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task identifier</td>
<td>Enter <code>AdventureWorks-to-S3-data-lake</code>.</td>
</tr>
<tr>
<td>Replication instance</td>
<td>Choose <code>datalake-migration-ri</code> (the value that you configured on Step 1).</td>
</tr>
<tr>
<td>Source database endpoint</td>
<td>Choose <code>datalake-source-db</code> (the value that you configured on Step 3).</td>
</tr>
<tr>
<td>Target database endpoint</td>
<td>Choose <code>adventure-works-datalake</code> (the value that you configured on Step 4).</td>
</tr>
<tr>
<td>Migration type</td>
<td>Choose <code>Migrate existing data and replicate ongoing changes</code>.</td>
</tr>
</tbody>
</table>
### Step-by-Step Migration

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editing mode</td>
<td>Choose Wizard.</td>
</tr>
<tr>
<td>Custom CDC stop mode for source transactions</td>
<td>Choose Disable custom CDC stop mode.</td>
</tr>
<tr>
<td>Target table preparation mode</td>
<td>Choose Do nothing.</td>
</tr>
<tr>
<td>Stop task after full load completes</td>
<td>Choose Don't stop.</td>
</tr>
<tr>
<td>Include LOB columns in replication</td>
<td>Choose Limited LOB mode.</td>
</tr>
<tr>
<td>Maximum LOB size (KB)</td>
<td>Enter 32.</td>
</tr>
<tr>
<td>Enable validation</td>
<td>Turn off because Amazon S3 does not support validation.</td>
</tr>
<tr>
<td>Enable CloudWatch logs</td>
<td>Turn on.</td>
</tr>
</tbody>
</table>

4. Leave the default values in the other fields and choose Create task.

5. The task begins immediately. The Database migration tasks section shows you the status of the migration task.

### Step 7: Run the AWS DMS Task

After you created your AWS Database Migration Service (AWS DMS) task, run the task a few times to identify the full load run time and ongoing replication performance. You can validate that initial configurations work as expected. You can do this by monitoring and documenting resource utilization on the source database, replication instance, and target database. These details make up the initial baseline and help determine if you need further optimizations.

After you started the task, the full load operation starts loading tables. You can see the table load completion status in the Table Statistics section and the corresponding target files in the Amazon S3 bucket. Because in our case the overall number of records is less than 200,000, the full load operation finishes in less than a minute. We can increase the value of Maximum number of tables to load in parallel, but it will not provide any meaningful gain in this scenario.

After the AWS DMS task completes full load, the status changes to the Load complete, replication ongoing phase. The following image shows the updated status of the task.

```
<table>
<thead>
<tr>
<th>Status</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load complete, replication ongoing</td>
<td>Full load, ongoing replication</td>
</tr>
</tbody>
</table>
```

During this phase, AWS DMS partitions data by the year, month, and day of generation. The following image shows the structure of folders.
Following, find some of the common errors and unexpected results you might see while following this walkthrough.

**Files are not written to the Amazon S3 target even though changes are visible in the table statistics section of the c console**

- This happens due to the target endpoint configuration. After you set CdcMaxBatchInterval=3600 and CdcMinFileSize=64000, AWS DMS waits for an hour or for the file size to reach 64 MB before writing data to Amazon S3.
- To write the output to Amazon S3 sooner, reduce CdcMaxBatchInterval to a smaller value. Alternatively, you can stop and resume the task. This will force Amazon S3 to flush events to Amazon S3 disregarding the extra connection attributes settings. Using these options means that the size of CDC files will be much smaller than the expected 64 MB.

**Parquet file sizes are less than 64 MB despite setting CdcMinFileSize=64000**

AWS DMS creates 64 MB files in memory. When this data is encoded as Parquet the resulting file size is smaller. The file sizes vary based on the level of compression possible.

**AWS DMS captures only inserts and deletes and does not migrate update records to the target**

You can see the following warning in the task logs:

```
00008570: 2021-12-07T19:52:52 [SOURCE_CAPTURE ]W: MS-REPLICATION is not enabled for table 'Sales].[SalesPerson]'. Therefore, UPDATE changes to it will not be captured. If you want UPDATE changes to be captured, either define a Primary Key for the table (if missing) or enable Microsoft CDC instead. (sqlserver_log_utils.c:1292)
```
This log message indicates MS-Replication. However, for Amazon RDS for SQL Server you can use MS-CDC. This error occurs when you have not turned on MS-CDC for the table. For more information, see the section called “Step 2: Configure a Source Amazon RDS for SQL Server Database” (p. 153).

In this walkthrough, we covered most prerequisites that help avoid configuration related errors. If you observe issues when running the task, see Troubleshooting migration tasks, Best practices, or reach out to AWS Support for further assistance.

After you completed the migration, validate that your data migrated successfully and delete the cloud resources that you created.

Migrating an Oracle Database to PostgreSQL

Using this walkthrough, you can learn how to migrate an Oracle database to a PostgreSQL database using AWS Database Migration Service (AWS DMS) and the AWS Schema Conversion Tool (AWS SCT).

AWS DMS migrates your data from your Oracle source into your PostgreSQL target. AWS DMS also captures data manipulation language (DML) and supported data definition language (DDL) changes that happen on your source database and applies these changes to your target database. This way, AWS DMS keeps your source and target databases in sync with each other. To facilitate the data migration, AWS SCT creates the migrated schemas on the target database, including the tables and primary key indexes on the target if necessary.

AWS DMS doesn’t migrate your secondary indexes, sequences, default values, stored procedures, triggers, synonyms, views, and other schema objects not specifically related to data migration. To migrate these objects to your PostgreSQL target, use AWS SCT.

Topics
- Prerequisites (p. 168)
- Step-by-Step Migration (p. 169)
- Rolling Back the Migration (p. 185)
- Troubleshooting (p. 185)

Prerequisites

The following prerequisites are required to complete this walkthrough:

- Understand Amazon Relational Database Service (Amazon RDS), the applicable database technologies, and SQL.
- Create an AWS account with AWS Identity and Access Management (IAM) credentials that allows you to launch Amazon RDS and AWS Database Migration Service (AWS DMS) instances in your AWS Region. For information about IAM credentials, see Create an IAM User.
- Understand the Amazon Virtual Private Cloud (Amazon VPC) service and security groups. For information about using Amazon VPC with Amazon RDS, see Amazon Virtual Private Cloud (VPCs) and Amazon RDS. For information about Amazon RDS security groups, see Amazon RDS Security Groups.
- Understand the supported features and limitations of AWS DMS. For information about AWS DMS, see https://docs.aws.amazon.com/dms/latest/userguide/Welcome.html.
- Understand the supported data type conversion options for Oracle and PostgreSQL. For information about data types for Oracle as a source, see Using an Oracle Database as a Source. For information about data types for PostgreSQL as a target, see Using a PostgreSQL Database as a Target.
- Size your target PostgreSQL database host. DBAs should be aware of the load profile of the current source Oracle database host. Consider CPU, memory, and IOPS. With RDS, you can size up the
target database host, or reduce it, after the migration. If this is the first time you are migrating to PostgreSQL, then we recommend that you have extra capacity to account for performance issues and tuning opportunities.

- Audit your source Oracle database. For each schema and all the objects under each schema, determine if any of the objects are no longer being used. Deprecate these objects on the source Oracle database, because there’s no need to migrate them if they are not being used.
- If load capacity permits, then get the max size (kb) for each LOB type on the source database, and keep this information for later.
- If possible, move columns with BLOB, CLOB, NCLOB, LONG, LONG RAW, and XMLTYPE to Amazon S3, Dynamo DB, or another data store. Doing so simplifies your source Oracle database for an easier migration. It will also lower the capacity requirements for the target PostgreSQL database.

For more information on AWS DMS, see [https://docs.aws.amazon.com/dms/latest/userguide/CHAP_GettingStarted.html](https://docs.aws.amazon.com/dms/latest/userguide/CHAP_GettingStarted.html).

### Step-by-Step Migration

The following steps provide instructions for migrating an Oracle database to a PostgreSQL database. These steps assume that you have already prepared your source database as described in Prerequisites (p. 168).

#### Topics

- **Step 1: Install the SQL Drivers and AWS Schema Conversion Tool on Your Local Computer** (p. 169)
- **Step 2: Configure Your Oracle Source Database** (p. 170)
- **Step 3: Configure Your PostgreSQL Target Database** (p. 173)
- **Step 4: Use AWS SCT to Convert the Oracle Schema to PostgreSQL** (p. 173)
- **Step 5: Create an AWS DMS Replication Instance** (p. 176)
- **Step 6: Create AWS DMS Source and Target Endpoints** (p. 178)
- **Step 7: Create and Run Your AWS DMS Migration Task** (p. 181)
- **Step 8: Cut Over to PostgreSQL** (p. 184)

#### Step 1: Install the SQL Drivers and AWS Schema Conversion Tool on Your Local Computer

To install the SQL drivers and the AWS Schema Conversion Tool (AWS SCT) on your local computer, do the following:

1. Download the JDBC driver for your Oracle database release. For more information, go to [https://www.oracle.com/jdbc](https://www.oracle.com/jdbc).
2. Download the PostgreSQL driver (`postgresql-42.2.19.jar`).
3. Install AWS SCT and the required JDBC drivers.
   b. Launch AWS SCT.
   c. In AWS SCT, choose **Global settings** from **Settings**.
   d. In **Global settings**, choose **Driver**, and then choose **Browse for Oracle driver path**. Locate the JDBC Oracle driver and choose **OK**.
   e. Choose **Browse** for **PostgreSQL driver path**. Locate the JDBC PostgreSQL driver and choose **OK**.
f. Choose OK to close the dialog box.

Step 2: Configure Your Oracle Source Database

To use Oracle as a source for AWS Database Migration Service (AWS DMS), you must first ensure that ARCHIVELOG MODE is on to provide information to LogMiner. AWS DMS uses LogMiner to read information from the archive logs so that AWS DMS can capture changes.

For AWS DMS to read this information, make sure the archive logs are retained on the database server as long as AWS DMS requires them. If you configure your task to begin capturing changes immediately, then you should only need to retain archive logs for a little longer than the duration of the longest running transaction. Retaining archive logs for 24 hours is usually sufficient. If you configure your task to
begin from a point in time in the past, then archive logs must be available from that time forward. For more specific instructions about enabling ARCHIVELOG MODE and ensuring log retention for your Oracle database, see the Oracle documentation.

To capture change data, AWS DMS requires supplemental logging to be enabled on your source database. Minimal supplemental logging must be enabled at the database level. AWS DMS also requires that identification key logging be enabled. This option causes the database to place all columns of a row’s primary key in the redo log file whenever a row containing a primary key is updated. This result occurs even if no value in the primary key has changed. You can set this option at the database or table level.

1. Create or configure a database account to be used by AWS DMS. We recommend that you use an account with the minimal privileges required by AWS DMS for your AWS DMS connection. AWS DMS requires the following privileges.

   ```sql
   CREATE SESSION
   SELECT ANY TRANSACTION
   SELECT on V_$ARCHIVED_LOG
   SELECT on V_$LOG
   SELECT on V_$LOGFILE
   SELECT on V_$DATABASE
   SELECT on V_$THREAD
   SELECT on V_$PARAMETER
   SELECT on V_$NLS_PARAMETERS
   SELECT on V_$TIMEZONE_NAMES
   SELECT on V_$TRANSACTION
   SELECT on ALL_INDEXES
   SELECT on ALL_OBJECTS
   SELECT on ALL_TABLES
   SELECT on ALL_USERS
   SELECT on ALL_CATALOG
   SELECT on ALL_CONSTRAINTS
   SELECT on ALL_CONS_COLUMNS
   SELECT on ALL_TAB_COLS
   SELECT on ALL_IND_COLUMNS
   SELECT on ALL_LOG_GROUPS
   SELECT on SYS.DBA_REGISTRY
   SELECT on SYS.OBJ$ SELECT on DBA_TABLESPACES
   SELECT on ALL_TAB_PARTITIONS
   SELECT on ALL_ENCRYPTED_COLUMNS
   * SELECT on all tables migrated
   ```

   If you want to capture and apply changes (CDC), then you also need the following privileges.

   ```sql
   EXECUTE on DBMS_LOGMNR
   SELECT on V_$LOGMNR_LOGS
   SELECT on V_$LOGMNR_CONTENTS
   LOGMINING /* For Oracle 12c and higher. */
   * ALTER for any table being replicated (if you want to add supplemental logging)
   ```

   For Oracle versions before 11.2.0.3, you need the following privileges.

   ```sql
   SELECT on DBA_OBJECTS /* versions before 11.2.0.3 */
   SELECT on ALL_VIEWS (required if views are exposed)
   ```

2. If your Oracle database is an Amazon RDS database, then connect to it as an administrative user, and run the following command to ensure that archive logs are retained on your RDS source for 24 hours:
exec rdsadmin.rdsadmin_util.set_configuration('archivelog retention hours',24);

If your Oracle source is an Amazon RDS database, it will be placed in ARCHIVELOG MODE if, and only if, you enable backups.

3. Run the following command to enable supplemental logging at the database level, which AWS DMS requires:
   a. In Oracle SQL:

   ```sql
   ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
   ```
   b. In RDS:

   ```sql
   exec rdsadmin.rdsadmin_util.alter_supplemental_logging('ADD');
   ```

4. Use the following command to enable identification key supplemental logging at the database level. AWS DMS requires supplemental key logging at the database level. The exception is if you allow AWS DMS to automatically add supplemental logging as needed or enable key-level supplemental logging at the table level:
   a. In Oracle SQL:

   ```sql
   ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
   ```
   b. In RDS:

   ```sql
   exec rdsadmin.rdsadmin_util.alter_supplemental_logging('ADD','PRIMARY KEY');
   ```

   Your source database incurs a small bit of overhead when key level supplemental logging is enabled. Therefore, if you are migrating only a subset of your tables, then you might want to enable key level supplemental logging at the table level.

5. To enable key level supplemental logging at the table level, use the following command:

   ```sql
   ALTER TABLE table_name ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
   ```

   If a table does not have a primary key, then you have two options.
   * You can add supplemental logging on all columns involved in the first unique index on the table (sorted by index name).
   * You can add supplemental logging on all columns of the table.

   To add supplemental logging on a subset of columns in a table, such as those involved in a unique index, run the following command.

   ```sql
   ALTER TABLE table_name
   ADD SUPPLEMENTAL LOG GROUP example_log_group (column_list) ALWAYS;
   ```

   To add supplemental logging for all columns of a table, run the following command.

   ```sql
   ALTER TABLE table_name
   ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
   ```

6. Create a user for AWS SCT.
CREATE USER oracle_sct_user IDENTIFIED BY password;
GRANT CONNECT TO oracle_sct_user;
GRANT SELECT_CATALOG_ROLE TO oracle_sct_user;
GRANT SELECT ANY DICTIONARY TO oracle_sct_user;

Step 3: Configure Your PostgreSQL Target Database

1. If the schemas you are migrating do not exist on the PostgreSQL database, then create the schemas.
2. Create the AWS DMS user to connect to your target database, and grant Superuser or the necessary individual privileges (or use the master username for RDS).

CREATE USER postgresql_dms_user WITH PASSWORD 'password';
ALTER USER postgresql_dms_user WITH SUPERUSER;

3. Create a user for AWS SCT.

CREATE USER postgresql_sct_user WITH PASSWORD 'password';
GRANT CONNECT ON DATABASE database_name TO postgresql_sct_user;
GRANT USAGE ON SCHEMA schema_name TO postgresql_sct_user;
GRANT SELECT ON ALL TABLES IN SCHEMA schema_name TO postgresql_sct_user;
GRANT ALL ON ALL SEQUENCES IN SCHEMA schema_name TO postgresql_sct_user;

Step 4: Use AWS SCT to Convert the Oracle Schema to PostgreSQL

Before you migrate data to PostgreSQL, you convert the Oracle schema to a PostgreSQL schema. This video covers all the steps of this process.

To convert an Oracle schema to a PostgreSQL schema using AWS Schema Conversion Tool (AWS SCT), do the following:

1. Launch AWS SCT. In AWS SCT, choose File, then choose New Project. Create a new project named AWS Schema Conversion Tool Oracle to PostgreSQL, specify the Location of the project folder, and then choose OK.
2. Choose Add source to add a source Oracle database to your project, then choose Oracle, and choose Next.
3. Enter the following information, and then choose Test Connection.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection name</td>
<td>Enter Oracle. AWS SCT displays this name in the tree in the left panel.</td>
</tr>
<tr>
<td>Type</td>
<td>Choose SID.</td>
</tr>
<tr>
<td>Server name</td>
<td>Enter the server name.</td>
</tr>
<tr>
<td>Server port</td>
<td>Enter the Oracle port number. The default is 1521.</td>
</tr>
<tr>
<td>Oracle SID</td>
<td>Enter the database SID.</td>
</tr>
</tbody>
</table>
**Database Migration Guide Step-by-Step Walkthroughs**

**Step-by-Step Migration**

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>User name</td>
<td>Enter the Oracle admin username.</td>
</tr>
<tr>
<td>Password</td>
<td>Enter the password for the admin user.</td>
</tr>
</tbody>
</table>

4. Choose **OK** to close the alert box, then choose **Connect** to close the dialog box and to connect to the Oracle DB instance.

5. Choose **Add target** to add a target PostgreSQL database to your project, then choose **Amazon RDS for PostgreSQL**, and choose **Next**.

6. Enter the following information and then choose **Test Connection**.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection name</td>
<td>Enter Amazon RDS for PostgreSQL. AWS SCT displays this name in the tree in the right panel.</td>
</tr>
<tr>
<td>Server name</td>
<td>Enter the server name.</td>
</tr>
</tbody>
</table>
### Parameter | Description
--- | ---
Server port | Enter the PostgreSQL port number. The default is 5432.
Database | Enter the database name.
User name | Enter the PostgreSQL admin username.
Password | Enter the password for the admin user.

7. Choose **OK** to close the alert box, then choose **Connect** to connect to the Amazon RDS for PostgreSQL DB instance.

8. In the tree in the left panel, select the schema to migrate. In the tree in the right panel, select your target Amazon RDS for PostgreSQL database. Choose **Create mapping**. For more information, see **Creating mapping rules** in the *Schema Conversion Tool User Guide*.

9. Choose **Main view**. In the tree in the left panel, right-click the schema to migrate and choose **Convert schema**.

10. Choose **Yes** for the confirmation message. AWS SCT analyzes the schema, creates a database migration assessment report, and converts your schema to the target database format.

11. Choose **Assessment Report View** from the menu to check the database migration assessment report. The report breaks down by each object type and by how much manual change is needed to convert it successfully.

   Generally, packages, procedures, and functions are more likely to have some issues to resolve because they contain the most custom PL/SQL code. AWS SCT also provides hints about how to fix these objects.

12. Choose the **Action Items** tab.

   The **Action Items** tab shows each issue for each object that requires attention.

   For each conversion issue, you can complete one of the following actions:

   a. Modify the objects on the source Oracle database so that AWS SCT can convert the objects to the target Amazon RDS for PostgreSQL database.
      i. Modify the objects on the source Oracle database.
ii. Repeat the previous steps to convert the schema and check the assessment report.
iii. If necessary, repeat this process until there are no conversion issues.

iv. Choose Main View from the menu. Open the context (right-click) menu for the target Amazon RDS for PostgreSQL schema, and choose Apply to database to apply the schema changes to the Amazon RDS for PostgreSQL database, and confirm that you want to apply the schema changes.

b. Instead of modifying the source schema, modify scripts that AWS SCT generates before applying the scripts on the target Amazon RDS for PostgreSQL database.

i. Choose Main View from the menu. Open the context (right-click) menu for the target Amazon RDS for PostgreSQL schema name, and choose Save as SQL. Next, choose a name and destination for the script.

ii. In the script, modify the objects to correct conversion issues.

You can also exclude foreign key constraints, triggers, and secondary indexes from the script because they can cause problems during the migration. After the migration is complete, you can create these objects on the Amazon RDS for PostgreSQL database.

iii. Run the script on the target Amazon RDS for PostgreSQL database.

For more information, see Converting Database Schema to Amazon RDS.

13 (Optional) Use AWS SCT to create migration rules.

a. Choose Mapping view and then choose New migration rule.

b. Create additional migration transformation rules that are required based on the action items.

c. Save the migration rules.

d. Choose Export script for DMS to export a JSON format of all the transformations that the AWS DMS task will use. Choose Save.

Step 5: Create an AWS DMS Replication Instance

After validating the schema structure between source and target databases, continue with the core part of this walkthrough, which is the data migration. The following illustration shows a high-level view of the migration process.

An AWS DMS replication instance performs the actual data migration between source and target. The replication instance also caches the transaction logs during the migration. How much CPU and memory capacity a replication instance has influences the overall time required for the migration.

1. Sign in to the AWS Management Console, and select AWS DMS at https://console.aws.amazon.com/dms/v2/. Next, choose Create Migration. If you are signed in as an AWS Identity and Access Management (IAM) user, then you must have the appropriate permissions to access AWS DMS. For more information on the permissions required, see IAM Permissions.

2. Choose Next to start a database migration from the console’s Welcome page.

3. On the Create replication instance page, specify your replication instance information.
### Parameter | Description
--- | ---
Name | Select a name for your replication instance. If you will be using multiple replication servers or sharing an account, then choose a name that will help you quickly differentiate between the different servers.
Description | Enter a brief description.
Instance class | Select the type of replication server to create. Each size and type of instance class will have increasing CPU, memory, and I/O capacity. Generally, the t2 instances are for lower load tasks, and the c4 instances are for higher load and more tasks.
VPC | Choose the VPC in which your replication instance will be launched. If possible, select the same VPC in which either your source or target database resides (or both).
Multi-AZ | When Yes is selected, AWS DMS creates a second replication server in a different Availability Zone for failover if there is a problem with the primary replication server.
Publicly accessible | If either your source or target database resides outside of the VPC in which your replication server resides, then you must make your replication server policy publicly accessible.

4. For the Advanced section, specify the following information.

### Parameter | Description
--- | ---
Allocated storage (GB) | Amount of storage on the replication server for the AWS DMS task logs, including historical tasks logs. AWS DMS also uses disk storage to cache certain data while it replicates it from the source to the target. Additionally, more storage generally enables better IOPS on the server.
Replication Subnet Group | If you are running in a Multi-AZ configuration, then you will need at least two subnet groups.
Availability zone | Generally, performance is better if you locate your primary replication server in the same Availability Zone as your target database.
VPC Security Group(s) | Security groups enable you to control ingress and egress to your VPC. AWS DMS allows you to associate one or more security groups with the VPC in which your replication server is launched.
KMS key | With AWS DMS, all data is encrypted at rest using a KMS encryption key. By default, AWS
Database Migration Guide Step-by-Step Walkthroughs
Step-by-Step Migration

Parameter | Description
----------|--------------------------------------------------
| DMS will create a new encryption key for your replication server. However, you may choose to use an existing key.

For information about the KMS key, see Setting an Encryption Key and Specifying KMS Permissions.

5. Click Next.

Step 6: Create AWS DMS Source and Target Endpoints

While your replication instance is being created, you can specify the source and target database endpoints using the AWS Management Console. However, you can only test connectivity after the replication instance has been created, because the replication instance is used in the connection.

1. Sign in to the AWS Management Console, open the AWS DMS console, and then choose Endpoints.
2. Specify your connection information for the source Oracle database and the target PostgreSQL database. The following table describes the source settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint Identifier</td>
<td>Enter a name, such as Orasource.</td>
</tr>
<tr>
<td>Source Engine</td>
<td>Choose oracle.</td>
</tr>
<tr>
<td>Server name</td>
<td>Provide the Oracle DB instance server name.</td>
</tr>
<tr>
<td>Port</td>
<td>The port of the database. The default for Oracle is 1521.</td>
</tr>
<tr>
<td>SSL mode</td>
<td>Choose an SSL mode if you want to enable encryption for your connection's traffic.</td>
</tr>
<tr>
<td>Username</td>
<td>The user you want to use to connect to the source database.</td>
</tr>
<tr>
<td>Password</td>
<td>Provide the password for the user.</td>
</tr>
<tr>
<td>SID</td>
<td>Provide the Oracle database name.</td>
</tr>
</tbody>
</table>

The following table describes the advanced source settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Extra connection attributes | Extra parameters that you can set in an endpoint to add functionality or change the behavior of AWS DMS. Some of the most common and convenient parameters to set for an Oracle source database are the following. Separate multiple entries from each other by using a semi-colon (;).
  • addSupplementalLogging - This parameter automatically configures supplemental logging when set to Y. |
Parameter | Description
---|---
useLogminerReader | By default, AWS DMS uses LogMiner on the Oracle database to capture all of the changes on the source database. The other mode is called Binary Reader. When using Binary Reader instead of LogMiner, AWS DMS copies the archived redo log from the source Oracle database to the replication server and reads the entire log in order to capture changes. The Binary Reader option is recommended if you are using ASM since it has performance advantages over LogMiner on ASM. If your source database is 12c, then the Binary Reader option is currently the only way to capture CDC changes in Oracle for LOB objects.
To use LogMiner, enter the following:
useLogminerReader=Y
To use Binary Reader, enter the following:
useLogminerReader=N; useBfile=Y

KMS key | Enter the KMS key if you choose to encrypt your replication instance's storage.

For information about extra connection attributes, see Using Extra Connection Attributes.

The following table describes the target settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint Identifier</td>
<td>Enter a name, such as Postgrestarget.</td>
</tr>
<tr>
<td>Target Engine</td>
<td>Choose postgres.</td>
</tr>
<tr>
<td>Servername</td>
<td>Provide the PostgreSQL DB instance server name.</td>
</tr>
<tr>
<td>Port</td>
<td>The port of the database. The default for PostgreSQL is 5432.</td>
</tr>
<tr>
<td>SSL mode</td>
<td>Choose None.</td>
</tr>
<tr>
<td>Username</td>
<td>The user you want to use to connect to the target database.</td>
</tr>
<tr>
<td>Password</td>
<td>Provide the password for the PostgreSQL DB instance.</td>
</tr>
</tbody>
</table>

The following is an example of the completed page.
3. After the endpoints and replication instance have been created, test each endpoint connection by choosing **Run test** for the source and target endpoints.

4. Drop foreign key constraints and triggers on the target database.

   During the full load process, AWS DMS does not load tables in any particular order, so it may load the child table data before parent table data. As a result, foreign key constraints might be violated if they are enabled. Also, if triggers are present on the target database, then it may change data loaded by AWS DMS in unexpected ways.

5. If you do not have one, then generate a script that enables the foreign key constraints and triggers.

   Later, when you want to add them to your migrated database, you can just run this script.

6. (Optional) Drop secondary indexes on the target database.

   Secondary indexes (as with all indexes) can slow down the full load of data into tables since they need to be maintained and updated during the loading process. Dropping them can improve the
performance of your full load process. If you drop the indexes, then you will need to add them back later after the full load is complete.

7. Choose Next.

**Step 7: Create and Run Your AWS DMS Migration Task**

Using an AWS DMS task, you can specify which schema to migrate and the type of migration. You can migrate existing data, migrate existing data and replicate ongoing changes, or replicate data changes only. This walkthrough migrates existing data and replicates ongoing changes.

1. On the Create Task page, specify the task options. The following table describes the settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task name</td>
<td>Enter a name for the migration task.</td>
</tr>
<tr>
<td>Task description</td>
<td>Enter a description for the task.</td>
</tr>
<tr>
<td>Source endpoint</td>
<td>Shows the Oracle source endpoint.</td>
</tr>
<tr>
<td></td>
<td>If you have more than one endpoint in the account, then choose the correct</td>
</tr>
<tr>
<td></td>
<td>endpoint from the list.</td>
</tr>
<tr>
<td>Target endpoint</td>
<td>Shows the PostgreSQL target endpoint.</td>
</tr>
<tr>
<td>Replication instance</td>
<td>Shows the AWS DMS replication instance.</td>
</tr>
<tr>
<td>Migration type</td>
<td>Choose <strong>Migrate existing data and replicate ongoing changes</strong>.</td>
</tr>
<tr>
<td>Start task on create</td>
<td>Select this option.</td>
</tr>
</tbody>
</table>

The page should look like the following:
2. Under Task Settings, choose Do nothing or Truncate for Target table preparation mode, because you have already created the tables using the AWS Schema Conversion Tool.

If the Oracle database has LOBs, then for Include LOB columns in replication, select Full LOB mode if you want to replicate the entire LOB for all tables. Select Limited LOB mode if you want to replicate the LOBs only up to a certain size. You specify the size of the LOB to migrate in Max LOB size (kb).

It is best to select Enable logging. If you enable logging, then you can see any errors or warnings that the task encounters, and you can troubleshoot those issues.

3. Leave the Advanced settings at their default values.
4. Choose **Table mappings**, and select the **JSON** tab. Next, select **Enable JSON editing**, and enter the table mappings you saved in the last step in Step 4: Convert the Oracle Schema to PostgreSQL (p. 173).

The following is an example of mappings that convert schema names and table names to lowercase.

```json
{
    "rules": [
        {
            "rule-type": "transformation",
            "rule-id": "100000",
            "rule-name": "Default Lowercase Table Rule",
            "rule-action": "convert-lowercase",
            "rule-target": "table",
            "object-locator": {
                "schema-name": "%",
                "table-name": "%"
            }
        },
        {
            "rule-type": "transformation",
            "rule-id": "100001",
            "rule-name": "Default Lowercase Schema Rule",
            "rule-action": "convert-lowercase",
            "rule-target": "schema",
            "object-locator": {
                "schema-name": "%"
            }
        }
    ]
}
```

5. Choose **Create task**. The task will begin immediately.

The Tasks section shows you the status of the migration task.

You can monitor your task if you chose **Enable logging** when you set up your task. You can then view the CloudWatch metrics by doing the following:

1. On the navigation pane, choose **Tasks**.
2. Choose your migration task.
3. Choose the **Task monitoring** tab, and monitor the task in progress on that tab.

When the full load is complete and cached changes are applied, the task will stop on its own.

4. On the target PostgreSQL database, enable foreign key constraints and triggers using the script you saved previously.
5. On the target PostgreSQL database, re-create the secondary indexes if you removed them previously.
6. In the AWS DMS console, start the AWS DMS task by clicking **Start/Resume** for the task.
The AWS DMS task keeps the target PostgreSQL database up-to-date with source database changes. AWS DMS will keep all of the tables in the task up-to-date until it is time to implement the application migration. The latency will be zero, or close to zero, when the target has caught up to the source.

**Step 8: Cut Over to PostgreSQL**

To move connections from your Oracle database to your PostgreSQL database, do the following:

1. End all Oracle database dependencies and activities, such as running scripts and client connections.

   The following query should return no results:

   ```sql
   SELECT MACHINE, COUNT(*) FROM V$SESSION GROUP BY MACHINE;
   ```

2. List any remaining sessions, and kill them.

   ```sql
   SELECT SID, SERIAL#, STATUS FROM V$SESSION;
   ALTER SYSTEM KILL 'sid, serial_number' IMMEDIATE;
   ```

3. Shut down all listeners on the Oracle database.

4. (Optional) Turn off automated jobs on the Oracle database. For your production database, check that this operation doesn’t influence the business logic.

   ```sql
   ALTER SYSTEM SET JOB_QUEUE_PROCESSES=0
   ```

5. (Optional) Turn off time monitoring on queue messages on the Oracle database. For your production database, check that this operation doesn’t influence the business logic.

   ```sql
   ALTER SYSTEM SET AQ_TM_PROCESSES=0
   ```

6. Let the AWS DMS task apply the final changes from the Oracle database on the PostgreSQL database.

   ```sql
   ALTER SYSTEM CHECKPOINT;
   ```

7. In the AWS DMS console, stop the AWS DMS task by clicking **Stop** for the task, and confirm that you want to stop the task.

8. (Optional) Set up a rollback.

   You can optionally set up a rollback task, in case you run into a show stopping issue, by creating a task going in the opposite direction. Because all tables should be in sync between both databases, you only need to set up a CDC task. Therefore, you do not have to disable any foreign key constraints. Now that the source and target databases are reversed, you must follow the instructions in the following sections:

   - **Using a PostgreSQL Database as a Source**
   - **Using an Oracle Database as a Target**
   
   a. Disable triggers on the source Oracle database.

   ```sql
   SELECT 'ALTER TRIGGER' || owner || '.' || trigger_name || 'DISABLE;' FROM DBA_TRIGGERS WHERE OWNER = 'schema_name';
   ```

   You do not have to disable the foreign key constraints. During the CDC process, foreign key constraints are updated in the same order as they are updated by application users.
b. Create a new CDC-only AWS DMS task with the endpoints reversed (source PostgreSQL endpoint and target Oracle endpoint database). See Step 7: Create and Run Your Migration Task (p. 181).

For the rollback task, set Migration type to Replicate data changes only and Target table preparation mode to Do nothing.

c. Start the AWS DMS task to enable you to push changes back to the original source Oracle database from the new PostgreSQL database if rollback is necessary.

9. Connect to the PostgreSQL database, and enable triggers.

```sql
ALTER TABLE table_name ENABLE TRIGGER ALL;
```

10. If you set up a rollback, then complete the rollback setup.

   a. Start the application services on new target PostgreSQL database (including scripts, client software, and so on).

   b. Add CloudWatch monitoring on your new PostgreSQL database. See Monitoring Amazon RDS.

### Rolling Back the Migration

If there are major issues with the migration that cannot be resolved in a timely manner, you can roll back the migration. These steps assume that you have already prepared for the rollback as described in Step 8: Cut Over to PostgreSQL (p. 184).

1. Stop all application services on the target PostgreSQL database.
2. Let the AWS DMS task replicate remaining changes back to the source Oracle database.
3. Stop the PostgreSQL to Oracle AWS DMS task.
4. Start all applications back on the source Oracle database.

### Troubleshooting

The two most common problem areas when working with Oracle as a source and PostgreSQL as a target are: supplemental logging and case sensitivity.

- **Supplemental logging** – With Oracle, in order to replicate change data, supplemental logging must be enabled. However, if you enable supplemental logging at the database level, it sometimes still needs to be enabled when new tables are created. The best remedy for this is to allow AWS DMS to enable supplemental logging for you by using the extra connection attribute:

  ```
  addSupplementalLogging=Y
  ```

- **Case sensitivity**: Oracle is case-insensitive (unless you use quotes around your object names). However, text appears in the upper case. Thus, AWS DMS defaults to naming your target objects in the upper case. In most cases, you’ll want to use transformations to change schema, table, and column names to lower case.

For more tips, see the AWS DMS troubleshooting section in the AWS DMS User Guide.

To troubleshoot issues specific to Oracle, see the Oracle troubleshooting section:

**Troubleshooting Oracle Specific Issues**

To troubleshoot PostgreSQL issues, see the PostgreSQL troubleshooting section:
Migrating Oracle Databases to Amazon RDS for PostgreSQL with DMS Schema Conversion

This walkthrough gets you started with heterogeneous database migration from Oracle to Amazon RDS for PostgreSQL. To automate the migration, we use the AWS DMS Schema Conversion. This service helps assess the complexity of your migration and converts source Oracle database schemas and code objects to a format compatible with PostgreSQL. Then, you apply the converted code to your target database. This introductory exercise shows how you can use DMS Schema Conversion for this migration.

At a high level, this migration includes the following steps:

- Use the AWS Management Console to do the following:
  - Create a VPC in the Amazon VPC console.
  - Create IAM roles in the the IAM console.
  - Create an Amazon S3 bucket in the Amazon S3 console.
  - Create your target Amazon RDS for PostgreSQL database in the Amazon RDS console.
  - Store database credentials in AWS Secrets Manager.
- Use the AWS DMS console to do the following:
  - Create an instance profile for your migration project.
  - Create data providers for your source and target databases.
  - Create a migration project.
- Use DMS Schema Conversion to do the following:
  - Assess the migration complexity and review the migration action items.
  - Convert your source database.
  - Apply the converted code to your target database.

This walkthrough takes approximately three hours to complete. Make sure that you delete resources at the end of this walkthrough to avoid additional charges.

Topics

- Prerequisites (p. 186)
- Migration Overview (p. 187)
- Step-by-Step Migration (p. 188)
- Next Steps (p. 196)

Prerequisites

The following prerequisites are also required to complete this walkthrough:

- Familiarity with the AWS Management Console, AWS Database Migration Service, and SQL.
- An AWS account with AWS Identity and Access Management (IAM) credentials. Make sure that you can use these credentials to create an Amazon S3 bucket in your AWS Region.
- Basic knowledge of the Amazon Virtual Private Cloud (Amazon VPC) service and of security groups.
- An understanding of the supported features and limitations of DMS Schema Conversion. For more information, see Schema conversion limitations.
Database Migration Guide Step-by-Step Walkthroughs

Migration Overview

We recommend that you don't use your production workloads for the migration in this walkthrough. After you get familiar with migration tools and AWS services, you can migrate your production workloads.

Make sure that you create all your AWS and DMS Schema Conversion resources in the AWS Regions that support DMS Schema Conversion. For more information, see the list of supported Regions. In other Regions, you can use the AWS Schema Conversion Tool. For an example of migration from Oracle to PostgreSQL with AWS SCT, see Use Schema Conversion Tool to Convert the Oracle Schema to PostgreSQL (p. 173).

For more information about DMS Schema Conversion, see the user guide.

Migration Overview

This section provides high-level guidance for customers looking to migrate from Oracle to PostgreSQL using DMS Schema Conversion.

DMS Schema Conversion automatically converts your source Oracle database schemas and most of the database code objects to a format compatible with PostgreSQL. This conversion includes tables, views, stored procedures, functions, data types, synonyms, and so on. Any objects that DMS Schema Conversion can’t convert automatically are clearly marked. To complete the migration, you can convert these objects manually.

At a high level, DMS Schema Conversion operates with the following three components: instance profiles, data providers, and migration projects. An instance profile specifies network and security settings. A data provider stores database connection credentials. A migration project contains data providers, an instance profile, and migration rules. AWS DMS uses data providers and an instance profile to design a process that converts database schemas and code objects.

The following diagram illustrates the DMS Schema Conversion process.

Start the walkthrough by creating the required resources (p. 188).
Step-by-Step Migration

In the following sections, you can find step-by-step instructions for migrating your Oracle database to Amazon RDS for PostgreSQL using DMS Schema Conversion.

Topics
- Step 1: Create AWS Resources (p. 188)
- Step 2: Configure Your Source Database (p. 190)
- Step 3: Create Your Target Amazon RDS for PostgreSQL Database (p. 191)
- Step 4: Store Database Credentials in AWS Secrets Manager (p. 192)
- Step 5: Create an Instance Profile (p. 193)
- Step 6: Configure Data Providers (p. 193)
- Step 7: Create a Migration Project (p. 194)
- Step 8: Convert Database Objects (p. 195)
- Step 9: Edit and Apply Your Converted Code (p. 196)

Step 1: Create AWS Resources

In this step, you create and configure the required AWS resources for DMS Schema Conversion.

First, you create a virtual private cloud (VPC) in your AWS account. This VPC is based on the Amazon Virtual Private Cloud (Amazon VPC) service and contains your AWS resources. Make sure that you create this VPC in one of the AWS Regions that support DMS Schema Conversion. For more information, see the list of supported Regions.

To create a VPC for DMS Schema Conversion

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. Choose your AWS Region.
3. Choose Create VPC.
4. On the Create VPC page, enter the following settings:
   - Resources to create — VPC and more
   - Name tag auto-generation — Choose Auto-generate and enter a globally unique name. For example, enter sc-vpc.
   - IPv4 CIDR block — 10.0.1.0/24
   - NAT gateways — In 1 AZ
   - VPC endpoints — None
5. Keep the rest of the settings as they are, and then choose Create VPC.
6. Choose Subnets.
   - For Filter by VPC, choose sc-vpc.
   - Take a note of your two private subnet IDs. Private subnet IDs don’t include Public in the name.
7. Choose NAT gateways.
   - Choose your NAT gateway.
   - Take a note of your Elastic IP address.

Use this VPC when you create your instance profile in Step 5 (p. 193) and your target Amazon RDS database in Step 3 (p. 191).
Next, you create AWS Identity and Access Management (IAM) roles to use in your DMS Schema Conversion migration project. AWS DMS uses these IAM roles to access your Amazon S3 bucket and database credentials stored in AWS Secrets Manager.

**To create an IAM role that provides access to your Amazon S3 bucket**

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. On the **Select trusted entity** page, choose **AWS service**. Choose **DMS**.
5. Choose **Next**. The **Add permissions** page opens.
6. For **Filter policies**, enter S3. Choose **AmazonS3FullAccess**.
7. Choose **Next**. The **Name, review, and create** page opens.
8. For **Role name**, enter a descriptive name. For example, enter **sc-s3-role**. Choose **Create role**.

Use this IAM role when you create your instance profile in **Step 5** (p. 193).

**To create an IAM role that provides access to AWS Secrets Manager**

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. On the **Select trusted entity** page, choose **AWS service**. Choose **DMS**.
5. Choose **Next**. The **Add permissions** page opens.
6. For **Filter policies**, enter Secret. Choose **SecretsManagerReadWrite**.
7. Choose **Next**. The **Name, review, and create** page opens.
8. For **Role name**, enter a descriptive name. For example, enter **sc-secrets-manager-role**. Choose **Create role**.

Use this IAM role when you create your migration project in **Step 7** (p. 194).

Next, you create an Amazon S3 bucket to use in your DMS Schema Conversion migration project. DMS Schema Conversion uses this Amazon S3 bucket to save assessment reports, SQL scripts with the converted code, and database metadata.

**To create an Amazon S3 bucket for DMS Schema Conversion**

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose **Create bucket**.
3. On the **Create bucket** page, select a globally unique name for your S3 bucket. For example, enter **sc-s3-bucket**.
4. For **AWS Region**, choose your Region.
5. For **Bucket Versioning**, choose **Enable**.
6. Keep the rest of the settings as they are, and then choose **Create bucket**.

Use this Amazon S3 bucket when you create your instance profile in **Step 5** (p. 193).
Step 2: Configure Your Source Database

In this step, you configure a new database user on your source Oracle database. Also, you configure the network to set up interaction for your source database with DMS Schema Conversion.

Use the credentials of this new user in DMS Schema Conversion. We encourage not using the admin user in the DMS Schema Conversion migration project.

Make sure that you grant the following privileges to this new user to complete the migration:

- **CONNECT** — includes only the CREATE SESSION privilege.
- **SELECT_CATALOG_ROLE** — provides SELECT privileges on all data dictionary views for Oracle 10g users.
- **SELECT ANY DICTIONARY** — provides query access to any object in the SYS schema.

You can use the following code example to create a database user and grant the privileges.

```
CREATE USER user_name IDENTIFIED BY your_password;
GRANT CONNECT TO user_name;
GRANT SELECT_CATALOG_ROLE TO user_name;
GRANT SELECT ANY DICTIONARY TO user_name;
```

In the preceding example, replace `user_name` with the name of your user. Then, replace `your_password` with a secure password.

After you configure your database user, make sure that DMS Schema Conversion can access your source Oracle database. To set up a network for DMS Schema Conversion, you can use different network configurations. These configurations depend on the settings of your source database and your network. For more information about available options, see Setting up a network for DMS Schema Conversion.

In this walkthrough, you configure a Site-to-Site VPN connection using a virtual private gateway.

**To configure a Site-to-Site VPN connection**

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. Choose your AWS Region.
3. Create a customer gateway.
   - In the navigation pane, choose Customer gateways, and then Create customer gateway.
   - For Name tag, enter a name for your customer gateway.
   - For BGP ASN, enter a Border Gateway Protocol (BGP) Autonomous System Number (ASN) for your customer gateway.
   - For IP address, enter the static, internet-routable IP address for your customer gateway device.
   - For Certificate ARN, choose the Amazon Resource Name of the private certificate.
   - For Device, enter a name for the device that hosts this customer gateway.
4. Create a virtual private gateway.
   - In the navigation pane, choose Virtual private gateways, and then Create virtual private gateway.
   - For Name tag, enter a name for your virtual private gateway.
   - For Autonomous System Number (ASN), choose Amazon default ASN.
   - Choose Create virtual private gateway.
   - Select the virtual private gateway you created, choose Actions, and then Attach to VPC.
   - Under Available VPCs, select your VPC from the list and choose Attach to VPC.
5. Configure route propagation in your route table.
• In the navigation pane, choose **Route tables**, and then select the route table that is associated with your subnet. By default, this is the main route table for the VPC.

• On the **Route propagation** tab in the details pane, choose **Edit route propagation**.

• Select the virtual private gateway that you created before, and then choose **Save**.

6. Add rules to your security group.

• In the navigation pane, choose **Security groups**, and then select the default security group for your VPC.

• On the **Inbound** tab in the details pane, add rules that allow inbound SSH, RDP, and ICMP access from your network.

• Choose **Save**.

7. Create a Site-to-Site VPN connection.

• In the navigation pane, choose **Site-to-Site VPN connections**, and then **Create VPN connection**.

• For **Name tag**, enter a name for your Site-to-Site VPN connection.

• For **Target gateway type**, choose either **Virtual private gateway**.

• For **Customer gateway**, select **Existing**.

• For **Customer gateway ID**, choose the customer gateway that you created before.

• Select the routing option. If your customer gateway device supports BGP, then choose **Dynamic (requires BGP)**. Alternatively, choose **Static** and specify IP prefixes for the private network of your Site-to-Site VPN connection.

• For **Outside IP address type**, keep the default option.

• Choose **Create VPN connection**.

8. Download the configuration file.

• In the navigation pane, choose **Site-to-Site VPN connections**, and then **Download configuration**.

• Select the vendor, platform, software, and IKE version that correspond to your customer gateway device. If your device isn't listed, choose **Generic**.

• Choose **Download**.

9. Use the sample configuration file to configure your customer gateway device.

---

**Step 3: Create Your Target Amazon RDS for PostgreSQL Database**

In this step, you create a new Amazon RDS for PostgreSQL database to use as a migration target for DMS Schema Conversion. Also, you configure a new database user on your target Amazon RDS for PostgreSQL database.

If you already created the target database, skip this step and proceed with the configuration of your database user.

**To create an Amazon RDS for PostgreSQL database for DMS Schema Conversion**

1. Sign in to the AWS Management Console and open the Amazon RDS console at [https://console.aws.amazon.com/rds/](https://console.aws.amazon.com/rds/).

2. Choose your AWS Region.

3. Choose **Create database**.

4. For **Engine type**, choose **PostgreSQL**.

5. For **Templates**, choose **Free tier**.

6. For **DB instance identifier**, enter a unique name for your PostgreSQL database.

7. For **Master password** and **Confirm master password**, enter a secure password that includes at least 8 printable characters.
8. For Virtual private cloud (VPC) under Connectivity, choose `sc-vpc`. You created this VPC in Step 1 (p. 188).

9. For Public access, choose Yes.

After you create your Amazon RDS for PostgreSQL database, configure a new database user. Then, use the credentials of this user in DMS Schema Conversion. We encourage not using the admin user in the DMS Schema Conversion migration project.

To configure your target database user, create a new user and grant CREATE ON DATABASE and the `rds_superuser` role.

You can use the following code example to create a database user and grant the privileges.

```
CREATE ROLE user_name LOGIN PASSWORD 'your_password';
GRANT CREATE ON DATABASE db_name TO user_name;
GRANT rds_superuser TO user_name;
ALTER DATABASE db_name OWNER TO user_name;
```

In the preceding example, replace `user_name` with the name of your user. Then, replace `your_password` with a secure password. Finally, replace `db_name` with the name of your target Amazon RDS for PostgreSQL database.

### Step 4: Store Database Credentials in AWS Secrets Manager

To connect to your source and target databases with DMS Schema Conversion, store your database credentials in AWS Secrets Manager. Make sure that you replicate these secrets to your AWS Region.

#### To store your source database credentials in AWS Secrets Manager

1. Sign in to the AWS Management Console and open the AWS Secrets Manager console at [https://console.aws.amazon.com/secretsmanager/](https://console.aws.amazon.com/secretsmanager/).
2. Choose your AWS Region.
3. Choose Store a new secret. The Choose secret type page opens.
4. For Secret type, choose Credentials for other database.
5. For User name and Password, enter the credentials of the database user that you created for your source database in Step 2 (p. 190).
6. For Database, choose Oracle.
7. For Server name, Database name, and Port, enter your Oracle database connection information.
9. For Secret name, enter `sc-oracle-secret`.
12. Choose Store.

#### To store your target database credentials in AWS Secrets Manager

1. Sign in to the AWS Management Console and open the AWS Secrets Manager console at [https://console.aws.amazon.com/secretsmanager/](https://console.aws.amazon.com/secretsmanager/).
2. Choose your AWS Region.
3. Choose Store a new secret. The Choose secret type page opens.
4. For **Secret type**, choose **Credentials for Amazon RDS database**.
5. For **User name** and **Password**, enter the credentials of the database user that you created for your target database in **Step 3** *(p. 191)*.
6. For **Database**, choose your Amazon RDS for PostgreSQL DB instance.
7. Choose **Next**. The **Configure secret** page opens.
8. For **Secret name**, enter sc-postgresql-secret.
9. Choose **Next**. The **Configure rotation** page opens.
10. Choose **Next**. The **Review** page opens.
11. Choose **Store**.

Use these secrets when you create your migration project in **Step 7** *(p. 194)*.

**Step 5: Create an Instance Profile**

Before you create your migration project in DMS Schema Conversion, you set up an instance profile. An instance profile specifies network and security settings for DMS Schema Conversion.

**To create an instance profile**

2. Choose your AWS Region.
3. In the navigation pane, choose **Instance profiles**, and then choose **Create instance profile**.
4. For **Name**, enter a unique name for your instance profile. For example, enter sc-instance.
5. For **Virtual private cloud (VPC)**, choose sc-vpc. You created this VPC in **Step 1** *(p. 188)*.
6. For **Subnet IDs**, choose two private subnet IDs for your instance profile. You noted these private subnet IDs in **Step 1** *(p. 188)*.
7. For **S3 bucket** under **Schema conversion settings - optional**, choose an Amazon S3 bucket that you created in **Step 1** *(p. 188)*.
8. For **IAM role**, choose the AWS Identity and Access Management (IAM) role that grants access to Amazon S3. You created this role in **Step 1** *(p. 188)*.
9. Choose **Create instance profile**.

Use this instance profile when you create your migration project in **Step 7** *(p. 194)*.

**Step 6: Configure Data Providers**

In this step, you create data providers that describe your source and target databases. A data provider stores a data store type and the location information about your database. Data providers don't include database credentials. You store database credentials in AWS Secrets Manager. Make sure that you include data providers and database secrets in your DMS Schema Conversion migration project.

You can create only one data provider for a single database. If you try to create a second data provider for the same database, DMS Schema Conversion displays an error message. However, you can use one data provider in multiple migration projects.

**To create a data provider for your Oracle database**

2. Choose your AWS Region.
3. In the navigation pane, choose Data providers, and then choose Create data provider.
4. For Configuration, choose Enter manually.
5. For Name, enter a unique name for your source data provider. For example, enter sc-oracle.
6. For Engine type, choose Oracle.
7. For Server name, enter the Domain Name Service (DNS) name or IP address of your database server.
8. For Port, enter the port used to connect to your database server.
9. For Service ID (SID) or service name, enter the Oracle System ID (SID). To find the Oracle SID, submit the following query to your Oracle database:

```
SELECT sys_context('userenv','instance_name') AS SID FROM dual;
```

10. For Secure Socket Layer (SSL) mode, choose none. Optionally, choose the type of your SSL enforcement, and provide the certificate information.
11. Choose Create data provider.

To create a data provider for your Amazon RDS for PostgreSQL database

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose your AWS Region.
3. In the navigation pane, choose Data providers, and then choose Create data provider.
4. For Configuration, choose RDS database instance.
5. For Database from RDS, choose your Amazon RDS for PostgreSQL database.
6. For Name, enter a unique name for your target data provider. For example, enter sc-postgresql.
7. Choose Create data provider.

Use these data providers when you create your migration project in Step 7 (p. 194).

Step 7: Create a Migration Project

Now you can create a migration project which is the foundation of your work with DMS Schema Conversion. A migration project describes your source and target data providers, your instance profile, and migration rules.

To create a migration project

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose your AWS Region.
3. Choose Migration projects, and then choose Create migration project.
4. For Name, enter a unique name for your migration project. For example, enter sc-project.
5. For Instance profile, choose sc-instance. You created this instance profile in Step 5 (p. 193).
6. For Source, choose Browse, and then choose sc-oracle. You created this data provider in Step 6 (p. 193).
8. For IAM role, choose sc-secrets-manager-role. You created this role in Step 1 (p. 188).
9. For Target, choose Browse, and then choose sc-postgresql. You created this data provider in Step 6 (p. 193).
11. For IAM role, choose `sc-secrets-manager-role`. You created this role in Step 1 (p. 188).
12. Choose Create migration project.

Use this migration project to convert your Oracle database schemas to PostgreSQL.

**Step 8: Convert Database Objects**

After you create the migration project, you can convert your Oracle database schemas to PostgreSQL. To start working with your migration project, you launch DMS Schema Conversion.

The first launch of DMS Schema Conversion requires some setup. AWS Database Migration Service (AWS DMS) starts a schema conversion instance, which can take 10-15 minutes. This process also reads the metadata from the source and target databases. After a successful first launch, you can access DMS Schema Conversion instantly.

**To convert your source Oracle database schema with DMS Schema Conversion**

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose your AWS Region.
4. Choose sc-project, and then choose Schema conversion.
5. Choose Launch schema conversion. If you launch schema conversion for the first time, then the notification appears. Choose Launch. The Schema conversion page opens. DMS Schema Conversion displays your source database schema in the left pane in a tree-view format.
6. In the source database pane, select the check box for the schema name.
7. Choose this schema in the left pane of the migration project. DMS Schema Conversion highlights the schema name in blue and activates the Actions menu.
8. For Actions, choose Convert schema. The conversion dialog box appears.
9. Choose Convert in the dialog box to confirm your choice.

After DMS Schema Conversion completes the conversion, you can review the converted code. After you choose a database object in the left pane of your project, DMS Schema Conversion automatically displays the source converted code for this object.

DMS Schema Conversion stores the converted code in your migration project and doesn't apply these code changes to your target database. You can apply the converted code in DMS Schema Conversion. Alternatively, you can save the converted code as a SQL script, edit it, and then apply to your target database. For more information, see Step 9 (p. 196).

In the settings of your migration project, you can customize your schema conversion view. Also, you can change conversion settings to improve the performance of converted code.

**To edit the settings of your DMS Schema Conversion migration project**

1. In the AWS DMS console, choose Migration projects. The Migration projects page opens.
2. Choose your migration project. Choose Schema conversion, then Launch schema conversion.
4. Change the settings to customize the schema conversion view. For more information, see Specifying migration project settings.
5. Change the settings to improve the performance of converted code. For more information, see Specifying Oracle to PostgreSQL conversion settings.
6. Choose **Apply**, and then choose **Schema conversion**.

After you change the settings, convert your source code again.

**Step 9: Edit and Apply Your Converted Code**

After you convert your source Oracle database objects, you can review the conversion statistics. DMS Schema Conversion converts most of the database objects, but some of the objects require manual conversion.

DMS Schema Conversion displays the objects that require manual conversion in the **Action items** tab. To convert these objects, you can save the converted code as a SQL script. Then you can edit it using your code editor and apply these scripts to your target database. Alternatively, you can apply the converted code as is to your target database and make the edits later.

**To save the converted code as a SQL script**

1. In the target database pane, choose the converted database schema.
2. Select the check box for the name of this schema. DMS Schema Conversion highlights the schema name in blue and activates the **Actions** menu.
3. For **Actions**, choose **Save as SQL**. The **Save** dialog box appears.
4. Choose **Save as SQL** to confirm your choice.
5. Choose **S3 bucket**. The **Amazon S3** console opens.
6. Choose **Download** to save your SQL scripts.

**To apply the converted code to your target database**

1. In the target database pane, choose the converted database schema.
2. Select the check box for the name of this schema. DMS Schema Conversion highlights the schema name in blue and activates the **Actions** menu.
3. For **Actions**, choose **Apply changes**. The **Apply changes** dialog box appears.
4. Choose **Apply** to confirm your choice.

Now you have successfully converted your source Oracle database schemas to PostgreSQL. To complete the database migration, move your data and connect your applications to the new database.

**Next Steps**

After you migrate your Oracle database to Amazon RDS for PostgreSQL using DMS Schema Conversion, you can explore several other resources:

- Use AWS DMS to migrate your source data. For more information, see the **Database Migration Service User Guide**.
- Use DMS Fleet Advisor to inventory your source databases and discover other candidates to move to the cloud. For more information, see the **DMS Fleet Advisor User Guide**.
- Learn more about Amazon RDS for PostgreSQL. For more information, see the **Amazon Relational Database Service User Guide**.

After you've finished using DMS Schema Conversion, clean up your resources. Amazon terminates the schema conversion instance that your migration project uses in three days after you complete the conversion. You can retrieve your converted schema and assessment report from the Amazon S3 bucket that you use for DMS Schema Conversion. However, you need to terminate other resources manually.
To clean up your DMS Schema Conversion resources

- Sign in to the AWS Management Console and open the AWS DMS console.
- In the navigation pane, choose Migration projects, and then choose your migration project. Choose Schema conversion, and then choose Stop schema conversion. Choose Delete and confirm your choice.
- Choose Instance profiles, and then choose sc-instance. Choose Delete and confirm your choice.
- Choose Data providers, and then select sc-oracle and sc-postgresql. Choose Delete and confirm your choice.

Also, make sure that you delete your Amazon S3 bucket, database secrets in AWS Secrets Manager, IAM roles, and virtual private cloud (VPC).

Migrating SQL Server Databases to Amazon RDS for MySQL with DMS Schema Conversion

This walkthrough gets you started with heterogeneous database migration from Microsoft SQL Server to Amazon RDS for MySQL. To automate the migration, we use the AWS DMS Schema Conversion. This service helps assess the complexity of your migration and converts source SQL Server database schemas and code objects to a format compatible with MySQL. Then, you apply the converted code to your target database. This introductory exercise shows how you can use DMS Schema Conversion for this migration.

At a high level, this migration includes the following steps:

- Use the AWS Management Console to do the following:
  - Create a VPC in the Amazon VPC console.
  - Create IAM roles in the the IAM console.
  - Create an Amazon S3 bucket in the Amazon S3 console.
  - Create your target Amazon RDS for MySQL database in the Amazon RDS console.
  - Store database credentials in AWS Secrets Manager.
- Use the AWS DMS console to do the following:
  - Create an instance profile for your migration project.
  - Create data providers for your source and target databases.
  - Create a migration project.
- Use DMS Schema Conversion to do the following:
  - Assess the migration complexity and review the migration action items.
  - Convert your source database.
  - Apply the converted code to your target database.

This walkthrough takes approximately three hours to complete. Make sure that you delete resources at the end of this walkthrough to avoid additional charges.

Topics
- Prerequisites (p. 198)
- Migration Overview (p. 198)
- Step-by-Step Migration (p. 199)
- Next Steps (p. 208)
Prerequisites

The following prerequisites are also required to complete this walkthrough:

• Familiarity with the AWS Management Console, AWS Database Migration Service, and SQL.
• An AWS account with AWS Identity and Access Management (IAM) credentials. Make sure that you can use these credentials to create an Amazon S3 bucket in your AWS Region.
• Basic knowledge of the Amazon Virtual Private Cloud (Amazon VPC) service and of security groups.
• An understanding of the supported features and limitations of DMS Schema Conversion. For more information, see Schema conversion limitations.

We recommend that you don’t use your production workloads for the migration in this walkthrough. After you get familiar with migration tools and AWS services, you can migrate your production workloads.

Make sure that you create all your AWS and DMS Schema Conversion resources in the AWS Regions that support DMS Schema Conversion. For more information, see the list of supported Regions. In other Regions, you can use the AWS Schema Conversion Tool (AWS SCT). For an example of migration from SQL Server to MySQL with AWS SCT, see Use Schema Conversion Tool to Convert the SQL Server Schema to MySQL (p. 173). To download AWS SCT, see Installing, verifying, and updating in the Schema Conversion Tool User Guide.

For more information about DMS Schema Conversion, see the user guide.

Migration Overview

This section provides high-level guidance for customers looking to migrate from SQL Server to MySQL using DMS Schema Conversion.

DMS Schema Conversion automatically converts your source SQL Server database schemas and most of the database code objects to a format compatible with MySQL. This conversion includes tables, views, stored procedures, functions, data types, synonyms, and so on. Any objects that DMS Schema Conversion can’t convert automatically are clearly marked. To complete the migration, you can convert these objects manually.

At a high level, DMS Schema Conversion operates with the following three components: instance profiles, data providers, and migration projects. An instance profile specifies network and security settings. A data provider stores database connection credentials. A migration project contains data providers, an instance profile, and migration rules. AWS DMS uses data providers and an instance profile to design a process that converts database schemas and code objects.

The following diagram illustrates the DMS Schema Conversion process for this walkthrough.
Database Migration Guide Step-by-Step Walkthroughs
Step-by-Step Migration

Start the walkthrough by creating the required resources (p. 199).

Step-by-Step Migration

In the following sections, you can find step-by-step instructions for migrating your SQL Server database to Amazon RDS for MySQL using DMS Schema Conversion.

Topics
- Step 1: Create AWS Resources (p. 199)
- Step 2: Configure Your Source Database (p. 201)
- Step 3: Create Your Target Amazon RDS for MySQL Database (p. 203)
- Step 4: Store Database Credentials in AWS Secrets Manager (p. 204)
- Step 5: Create an Instance Profile (p. 205)
- Step 6: Configure Data Providers (p. 205)
- Step 7: Create a Migration Project (p. 206)
- Step 8: Convert Database Objects (p. 207)
- Step 9: Edit and Apply Your Converted Code (p. 208)

Step 1: Create AWS Resources

In this step, you create and configure the required AWS resources for DMS Schema Conversion.

First, you create a virtual private cloud (VPC) in your AWS account. This VPC is based on the Amazon Virtual Private Cloud (Amazon VPC) service and contains your AWS resources. Make sure that you create this VPC in one of the AWS Regions that support DMS Schema Conversion. For more information, see the list of supported Regions.

To create a VPC for DMS Schema Conversion
1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. Choose your AWS Region.
3. Choose Create VPC.
4. On the Create VPC page, enter the following settings:
   - **Resources to create — VPC and more**
   - **Name tag auto-generation** — Choose Auto-generate and enter a globally unique name. For example, enter sc-vpc.
   - **IPv4 CIDR block** — 10.0.1.0/24
   - **NAT gateways** — In 1 AZ
   - **VPC endpoints** — None
5. Keep the rest of the settings as they are, and then choose Create VPC.
6. Choose Subnets.
   - For **Filter by VPC**, choose sc-vpc.
   - Take a note of your two private subnet IDs. Private subnet IDs don’t include Public in the name.
7. Choose NAT gateways.
   - Choose your **NAT gateway**.
   - Take a note of your Elastic IP address.

Use this VPC when you create your instance profile in Step 5 (p. 205) and your target Amazon RDS database in Step 3 (p. 203).

Next, you create AWS Identity and Access Management (IAM) roles to use in your DMS Schema Conversion migration project. AWS DMS uses these IAM roles to access your Amazon S3 bucket and database credentials stored in AWS Secrets Manager.

**To create an IAM role that provides access to your Amazon S3 bucket**

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles.
3. Choose Create role.
4. On the Select trusted entity page, choose AWS service. Choose DMS.
5. Choose Next. The Add permissions page opens.
6. For Filter policies, enter S3. Choose AmazonS3FullAccess.
7. Choose Next. The Name, review, and create page opens.
8. For Role name, enter a descriptive name. For example, enter sc-s3-role. Choose Create role.

Use this IAM role when you create your instance profile in Step 5 (p. 205).

**To create an IAM role that provides access to AWS Secrets Manager**

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles.
3. Choose Create role.
4. On the Select trusted entity page, choose AWS service. Choose DMS.
5. Choose Next. The Add permissions page opens.
6. For Filter policies, enter Secret. Choose SecretsManagerReadWrite.
7. Choose Next. The Name, review, and create page opens.
8. For **Role name**, enter a descriptive name. For example, enter `sc-secrets-manager-role`. Choose **Create role**.

Use this IAM role when you create your migration project in Step 7 (p. 206).

Next, you create an Amazon S3 bucket to use in your DMS Schema Conversion migration project. DMS Schema Conversion uses this Amazon S3 bucket to save assessment reports, SQL scripts with the converted code, and database metadata.

**To create an Amazon S3 bucket for DMS Schema Conversion**

1. Sign in to the AWS Management Console and open the Amazon S3 console at [https://console.aws.amazon.com/s3/](https://console.aws.amazon.com/s3/).
2. Choose **Create bucket**.
3. On the **Create bucket** page, select a globally unique name for your S3 bucket. For example, enter `sc-s3-bucket`.
4. For **AWS Region**, choose your Region.
5. For **Bucket Versioning**, choose **Enable**.
6. Keep the rest of the settings as they are, and then choose **Create bucket**.

Use this Amazon S3 bucket when you create your instance profile in Step 5 (p. 205).

**Step 2: Configure Your Source Database**

In this step, you configure a new database user on your source SQL Server database. Also, you configure the network to set up interaction for your source database with DMS Schema Conversion.

Use the credentials of this new user in DMS Schema Conversion. We encourage not using the admin user in the DMS Schema Conversion migration project.

Make sure that you grant the following privileges to this new user to complete the migration:

- **VIEW DEFINITION** — makes it possible for users that have public access to see object definitions.
- **VIEW DATABASE STATE** — makes it possible for users to check the features of the SQL Server Enterprise edition.

You can use the following code example to create a database user and grant the privileges.

```sql
USE db_name
CREATE USER user_name FOR LOGIN user_name
GRANT VIEW DEFINITION TO user_name
GRANT VIEW DATABASE STATE TO user_name
```

In the preceding example, replace `user_name` with the name of your user. Then, replace `db_name` with a name of your database.

Repeat the grant for each database whose schema you are converting.

Then, make sure that you grant the following privileges on the `master` database:

- **VIEW SERVER STATE** — makes it possible for users to collect server settings and configuration.
- **VIEW ANY DEFINITION** — makes it possible for users to view data providers.

You can use the following code example to create a database user and grant the privileges.
USE master
GRANT VIEW SERVER STATE TO user_name
GRANT VIEW ANY DEFINITION TO user_name

In the preceding example, replace user_name with the name of your user.

After you configure your database user, make sure that DMS Schema Conversion can access your source SQL Server database. To set up a network for DMS Schema Conversion, you can use different network configurations. These configurations depend on the settings of your source database and your network. For more information about available options, see Setting up a network for DMS Schema Conversion.

In this walkthrough, you configure a Site-to-Site VPN connection using a virtual private gateway.

To configure a Site-to-Site VPN connection

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. Choose your AWS Region.
3. Create a customer gateway.
   - In the navigation pane, choose Customer gateways, and then Create customer gateway.
   - For Name tag, enter a name for your customer gateway.
   - For BGP ASN, enter a Border Gateway Protocol (BGP) Autonomous System Number (ASN) for your customer gateway.
   - For IP address, enter the static, internet-routable IP address for your customer gateway device.
   - For Certificate ARN, choose the Amazon Resource Name of the private certificate.
   - For Device, enter a name for the device that hosts this customer gateway.
4. Create a virtual private gateway.
   - In the navigation pane, choose Virtual private gateways, and then Create virtual private gateway.
   - For Name tag, enter a name for your virtual private gateway.
   - For Autonomous System Number (ASN), choose Amazon default ASN.
   - Choose Create virtual private gateway.
   - Select the virtual private gateway you created before, and then choose Actions, and then Attach to VPC.
   - Under Available VPCs, select your VPC from the list and choose Attach to VPC.
5. Configure route propagation in your route table.
   - In the navigation pane, choose Route tables, and then select the route table that is associated with your subnet. By default, this is the main route table for the VPC.
   - On the Route propagation tab in the details pane, choose Edit route propagation.
   - Select the virtual private gateway that you created before, and then choose Save.
6. Add rules to your security group.
   - In the navigation pane, choose Security groups, and then select the default security group for your VPC.
   - On the Inbound tab in the details pane, add rules that allow inbound SSH, RDP, and ICMP access from your network.
   - Choose Save.
7. Create a Site-to-Site VPN connection.
   - In the navigation pane, choose Site-to-Site VPN connections, and then Create VPN connection.
   - For Name tag, enter a name for your Site-to-Site VPN connection.
   - For Target gateway type, choose either Virtual private gateway.
   - For Customer gateway, select Existing.
   - For Customer gateway ID, choose the customer gateway that you created before.
• Select the routing option. If your customer gateway device supports BGP, then choose Dynamic (requires BGP). Alternatively, choose Static and specify IP prefixes for the private network of your Site-to-Site VPN connection.
• For Outside IP address type, keep the default option.
• Choose Create VPN connection.

8. Download the configuration file.
• In the navigation pane, choose Site-to-Site VPN connections, and then Download configuration.
• Select the vendor, platform, software, and IKE version that correspond to your customer gateway device. If your device isn't listed, choose Generic.
• Choose Download.

9. Use the sample configuration file to configure your customer gateway device.

Step 3: Create Your Target Amazon RDS for MySQL Database

In this step, you create a new Amazon RDS for MySQL database to use as a migration target for DMS Schema Conversion. Also, you configure a new database user on your target Amazon RDS for MySQL database.

If you already created the target database, skip this step and proceed with the configuration of your database user.

To create an Amazon RDS for MySQL database for DMS Schema Conversion

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose your AWS Region.
3. Choose Create database.
4. For Engine type, choose MySQL.
5. For Templates, choose Free tier.
6. For DB instance identifier, enter a unique name for your MySQL database.
7. For Master password and Confirm master password, enter a secure password that includes at least 8 printable characters.
8. For Virtual private cloud (VPC) under Connectivity, choose sc-vpc. You created this VPC in Step 1 (p. 199).
9. For Public access, choose Yes.
10. Keep the rest of the settings as they are, and then choose Create database.

After you create your Amazon RDS for MySQL database, configure a new database user. Then, use the credentials of this user in DMS Schema Conversion. We encourage not using the admin user in the DMS Schema Conversion migration project.

To configure your target database user, create a new user and grant the following privileges:

• CREATE ON .
• ALTER ON .
• DROP ON .
• INDEX ON .
• REFERENCES ON .
• SELECT ON .
• CREATE VIEW ON .
• SHOW VIEW ON .
You can use the following code example to create a database user and grant the privileges.

```
CREATE USER 'user_name' IDENTIFIED BY 'your_password'; (1)
GRANT CREATE ON *.* TO 'user_name';
GRANT ALTER ON *.* TO 'user_name';
GRANT DROP ON *.* TO 'user_name';
GRANT REFERENCES ON *.* TO 'user_name';
GRANT INDEX ON *.* TO 'user_name';
GRANT SELECT ON *.* TO 'user_name';
GRANT CREATE VIEW ON *.* TO 'user_name';
GRANT SHOW VIEW ON *.* TO 'user_name';
GRANT TRIGGER ON *.* TO 'user_name';
GRANT CREATE ROUTINE ON *.* TO 'user_name';
GRANT ALTER ROUTINE ON *.* TO 'user_name';
GRANT EXECUTE ON *.* TO 'user_name';
GRANT INSERT, UPDATE ON AWS_SQLSERVER_EXT.* TO 'user_name';
GRANT INSERT, UPDATE, DELETE ON AWS_SQLSERVER_EXT_DATA.* TO 'user_name';
GRANT CREATE TEMPORARY TABLES ON AWS_SQLSERVER_EXT_DATA.* TO 'user_name';
```

In the preceding example, replace `user_name` with the name of your user. Then, replace `your_password` with a secure password.

### Step 4: Store Database Credentials in AWS Secrets Manager

To connect to your source and target databases with DMS Schema Conversion, store your database credentials in AWS Secrets Manager. Make sure that you replicate these secrets to your AWS Region.

#### To store your source database credentials in AWS Secrets Manager

1. Sign in to the AWS Management Console and open the AWS Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Choose your AWS Region.
3. Choose **Store a new secret**. The Choose secret type page opens.
4. For **Secret type**, choose **Credentials for other database**.
5. For **User name** and **Password**, enter the credentials of the database user that you created for your source database in Step 2 (p. 201).
6. For **Database**, choose **SQL Server**.
7. For **Server name**, **Database name**, and **Port**, enter your SQL Server database connection information.
8. Choose **Next**. The Configure secret page opens.
9. For **Secret name**, enter `sc-sql-server-secret`.
10. Choose **Next**. The Configure rotation page opens.
11. Choose **Next**. The Review page opens.
12. Choose **Store**.
**To store your target database credentials in AWS Secrets Manager**

1. Sign in to the AWS Management Console and open the AWS Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Choose your AWS Region.
3. Choose **Store a new secret**. The **Choose secret type** page opens.
4. For **Secret type**, choose **Credentials for Amazon RDS database**.
5. For **User name** and **Password**, enter the credentials of the database user that you created for your target database in Step 3 (p. 203).
6. For **Database**, choose your Amazon RDS for MySQL DB instance.
7. Choose **Next**. The **Configure secret** page opens.
8. For **Secret name**, enter **sc-mysql-secret**.
9. Choose **Next**. The **Configure rotation** page opens.
10. Choose **Next**. The **Review** page opens.
11. Choose **Store**.

Use these secrets when you create your migration project in Step 7 (p. 206).

**Step 5: Create an Instance Profile**

Before you create your migration project in DMS Schema Conversion, you set up an instance profile. An instance profile specifies network and security settings for DMS Schema Conversion.

**To create an instance profile**

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose your AWS Region.
3. In the navigation pane, choose **Instance profiles**, and then choose **Create instance profile**.
4. For **Name**, enter a unique name for your instance profile. For example, enter **sc-instance**.
5. For **Virtual private cloud (VPC)**, choose **sc-vpc**. You created this VPC in Step 1 (p. 199).
6. For **Subnet IDs**, choose two private subnet IDs for your instance profile. You noted these private subnet IDs in Step 1 (p. 199).
7. For **S3 bucket** under **Schema conversion settings - optional**, choose an Amazon S3 bucket that you created in Step 1 (p. 199).
8. For **IAM role**, choose the AWS Identity and Access Management (IAM) role that grants access to Amazon S3. You created this role in Step 1 (p. 199).
9. Choose **Create instance profile**.

Use this instance profile when you create your migration project in Step 7 (p. 206).

**Step 6: Configure Data Providers**

In this step, you create data providers that describe your source and target databases. A data provider stores a data store type and the location information about your database. Data providers don't include database credentials. You store database credentials in AWS Secrets Manager. Make sure that you include data providers and database secrets in your DMS Schema Conversion migration project.

You can create only one data provider for a single database. If you try to create a second data provider for the same database, DMS Schema Conversion displays an error message. However, you can use one data provider in multiple migration projects.
To create a data provider for your SQL Server database

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose your AWS Region.
3. In the navigation pane, choose Data providers, and then choose Create data provider.
4. For Configuration, choose Enter manually.
5. For Name, enter a unique name for your source data provider. For example, enter sc-sql-server.
6. For Engine type, choose Microsoft SQL Server.
7. For Server name, enter the Domain Name Service (DNS) name or IP address of your database server.
8. For Port, enter the port used to connect to your database server.
9. For Database name, enter the name of your database.
10. For Secure Socket Layer (SSL) mode, choose none. Optionally, choose the type of your SSL enforcement, and provide the certificate information.
11. Choose Create data provider.

To create a data provider for your Amazon RDS for MySQL database

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose your AWS Region.
3. In the navigation pane, choose Data providers, and then choose Create data provider.
4. For Configuration, choose RDS database instance.
5. For Database from RDS, choose your Amazon RDS for MySQL database.
6. For Name, enter a unique name for your target data provider. For example, enter sc-mysql.
7. Choose Create data provider.

Use these data providers when you create your migration project in Step 7 (p. 206).

Step 7: Create a Migration Project

Now you can create a migration project which is the foundation of your work with DMS Schema Conversion. A migration project describes your source and target data providers, your instance profile, and migration rules.

To create a migration project

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose your AWS Region.
3. Choose Migration projects, and then choose Create migration project.
4. For Name, enter a unique name for your migration project. For example, enter sc-project.
5. For Instance profile, choose sc-instance. You created this instance profile in Step 5 (p. 205).
6. For Source, choose Browse, and then choose sc-sql-server. You created this data provider in Step 6 (p. 205).
7. For Secret ID, choose sc-sql-server-secret. You created this secret in Step 4 (p. 204).
8. For IAM role, choose sc-secrets-manager-role. You created this role in Step 1 (p. 199).
9. For Target, choose Browse, and then choose sc-mysql. You created this data provider in Step 6 (p. 205).
For **Secret ID**, choose `sc-mysql-secret`. You created this secret in Step 4 (p. 204).

For **IAM role**, choose `sc-secrets-manager-role`. You created this role in Step 1 (p. 199).

Choose Create migration project.

Use this migration project to convert your SQL Server database schemas to MySQL.

**Step 8: Convert Database Objects**

After you create the migration project, you can convert your SQL Server database schemas to MySQL. To start working with your migration project, you launch DMS Schema Conversion.

The first launch of DMS Schema Conversion requires some setup. AWS Database Migration Service (AWS DMS) starts a schema conversion instance, which can take 10-15 minutes. This process also reads the metadata from the source and target databases. After a successful first launch, you can access DMS Schema Conversion instantly.

**To convert your source SQL Server database schema with DMS Schema Conversion**

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose your AWS Region.
4. Choose sc-project, and then choose Schema conversion.
5. Choose Launch schema conversion. If you launch schema conversion for the first time, then the notification appears. Choose Launch. The Schema conversion page opens. DMS Schema Conversion displays your source database schema in the left pane in a tree-view format.
6. In the source database pane, select the check box for the schema name.
7. Choose this schema in the left pane of the migration project. DMS Schema Conversion highlights the schema name in blue and activates the Actions menu.
8. For Actions, choose Convert schema. The conversion dialog box appears.
9. Choose Convert in the dialog box to confirm your choice.

After DMS Schema Conversion completes the conversion, you can review the converted code. After you choose a database object in the left pane of your project, DMS Schema Conversion automatically displays the source converted code for this object.

DMS Schema Conversion stores the converted code in your migration project and doesn't apply these code changes to your target database. You can apply the converted code in DMS Schema Conversion. Alternatively, you can save the converted code as a SQL script, edit it, and then apply to your target database. For more information, see Step 9 (p. 208).

In the settings of your migration project, you can customize your schema conversion view. Also, you can change conversion settings to improve the performance of converted code.

**To edit the settings of your DMS Schema Conversion migration project**

1. In the AWS DMS console, choose Migration projects. The Migration projects page opens.
2. Choose your migration project. Choose Schema conversion, then Launch schema conversion.
4. Change the settings to customize the schema conversion view. For more information, see Specifying migration project settings.
5. Change the settings to improve the performance of converted code. For more information, see Specifying SQL Server to MySQL conversion settings.
6. Choose **Apply**, and then choose **Schema conversion**.

After you change the settings, convert your source code again.

**Step 9: Edit and Apply Your Converted Code**

After you convert your source SQL Server database objects, you can review the conversion statistics. DMS Schema Conversion converts most of the database objects, but some of the objects require manual conversion.

DMS Schema Conversion displays the objects that require manual conversion in the **Action items** tab. To convert these objects, you can save the converted code as a SQL script. Then you can edit it using your code editor and apply these scripts to your target database. Alternatively, you can apply the converted code as is to your target database and make the edits later.

**To save the converted code as a SQL script**

1. In the target database pane, choose the converted database schema.
2. Select the check box for the name of this schema. DMS Schema Conversion highlights the schema name in blue and activates the **Actions** menu.
3. For **Actions**, choose **Save as SQL**. The **Save** dialog box appears.
4. Choose **Save as SQL** to confirm your choice.
5. Choose **S3 bucket**. The **Amazon S3** console opens.
6. Choose **Download** to save your SQL scripts.

**To apply the converted code to your target database**

1. In the target database pane, choose the converted database schema.
2. Select the check box for the name of this schema. DMS Schema Conversion highlights the schema name in blue and activates the **Actions** menu.
3. For **Actions**, choose **Apply changes**. The **Apply changes** dialog box appears.
4. Choose **Apply** to confirm your choice.

Now you have successfully converted your source SQL Server database schemas to MySQL. To complete the database migration, move your data and connect your applications to the new database.

**Next Steps**

After you migrate your SQL Server database to Amazon RDS for MySQL using DMS Schema Conversion, you can explore several other resources:

- Use AWS DMS to migrate your source data. For more information, see the [Database Migration Service User Guide](#).
- Use DMS Fleet Advisor to inventory your source databases and discover other candidates to move to the cloud. For more information, see the [DMS Fleet Advisor User Guide](#).
- Learn more about Amazon RDS for MySQL. For more information, see the [Amazon Relational Database Service User Guide](#).

After you've finished using DMS Schema Conversion, clean up your resources. Amazon terminates the schema conversion instance that your migration project uses in three days after you complete the conversion. You can retrieve your converted schema and assessment report from the Amazon S3 bucket that you use for DMS Schema Conversion. However, you need to terminate other resources manually.
To clean up your DMS Schema Conversion resources

- Sign in to the AWS Management Console and open the AWS DMS console.
- In the navigation pane, choose Migration projects, and then choose your migration project. Choose Schema conversion, and then choose Stop schema conversion. Choose Delete and confirm your choice.
- Choose Instance profiles, and then choose sc-instance. Choose Delete and confirm your choice.
- Choose Data providers, and then select sc-sql-server and sc-mysql. Choose Delete and confirm your choice.

Also, make sure that you delete your Amazon S3 bucket, database secrets in AWS Secrets Manager, IAM roles, and virtual private cloud (VPC).

Migrating an Amazon RDS for Oracle Database to Amazon Redshift

This walkthrough gets you started with heterogeneous database migration from Amazon RDS for Oracle to Amazon Redshift using AWS Database Migration Service (AWS DMS) and the AWS Schema Conversion Tool (AWS SCT). This introductory exercise doesn't cover all scenarios but provides you with a good understanding of the steps involved in such a migration.

It is important to understand that AWS DMS and AWS SCT are two different tools and serve different needs. They don’t interact with each other in the migration process. At a high level, the steps involved in this migration are the following:

1. Using AWS SCT to do the following:
   - Run the conversion report for Oracle to Amazon Redshift to identify the issues, limitations, and actions required for the schema conversion.
   - Generate the schema scripts and apply them on the target before performing the data load by using AWS DMS. AWS SCT performs the necessary code conversion for objects like procedures and views.
2. Identify and implement solutions to the issues reported by AWS SCT.
3. Disable foreign keys or any other constraints that might impact the AWS DMS data load.
4. AWS DMS loads the data from source to target using the Full Load approach. Although AWS DMS is capable of creating objects in the target as part of the load, it follows a minimalistic approach to efficiently migrate the data so that it doesn't copy the entire schema structure from source to target.
5. Perform postmigration activities such as creating additional indexes, enabling foreign keys, and making the necessary changes in the application to point to the new database.

This walkthrough uses a custom AWS CloudFormation template to create RDS DB instances for Oracle and Amazon Redshift. It then uses a SQL command script to install a sample schema and data onto the RDS Oracle DB instance that you then migrate to Amazon Redshift.

This walkthrough takes approximately two hours to complete. Be sure to follow the instructions to delete resources at the end of this walkthrough to avoid additional charges.

Topics

- Prerequisites (p. 210)
- Migration Architecture (p. 210)
- Step-by-Step Migration (p. 212)
Prerequisites

The following prerequisites are also required to complete this walkthrough:

- Familiarity with Amazon RDS, Amazon Redshift, the applicable database technologies, and SQL.
- The custom scripts that include creating the tables to be migrated and SQL queries for confirming the migration, as listed following:
  - Oracle_Redshift_For_DMSDemo.template — an AWS CloudFormation template.
  - Oraclesalesstarschema.sql — SQL statements to build the SH schema.

  These scripts are available at the following link: dms-sbs-RDSOracle2Redshift.zip.

  Each step in the walkthrough also contains a link to download the file involved or includes the exact query in the step.
- An AWS account with AWS Identity and Access Management (IAM) credentials that allow you to launch Amazon RDS, AWS Database Migration Service (AWS DMS) instances, and Amazon Redshift clusters in your AWS Region. For information about IAM credentials, see Creating an IAM User.
- Basic knowledge of the Amazon Virtual Private Cloud (Amazon VPC) service and of security groups. For information about using Amazon VPC with Amazon RDS, see Virtual Private Clouds (VPCs) and Amazon RDS. For information about Amazon RDS security groups, see Amazon RDS Security Groups. For information about using Amazon Redshift in a VPC, see Managing Clusters in an Amazon Virtual Private Cloud (VPC).
- An understanding of the supported features and limitations of AWS DMS. For information about AWS DMS, see https://docs.aws.amazon.com/dms/latest/userguide/Welcome.html.
- Knowledge of the supported data type conversion options for Oracle and Amazon Redshift. For information about data types for Oracle as a source, see Using an Oracle Database as a Source. For information about data types for Amazon Redshift as a target, see Using an Amazon Redshift Database as a Target.

For more information about AWS DMS, see https://docs.aws.amazon.com/dms/latest/userguide/CHAP_GettingStarted.html.

Migration Architecture

This walkthrough uses AWS CloudFormation to create a simple network topology for database migration that includes the source database, the replication instance, and the target database in the same VPC. For more information on AWS CloudFormation, see the AWS CloudFormation documentation.

We provision the AWS resources that are required for this AWS DMS walkthrough through AWS CloudFormation. These resources include a VPC and Amazon RDS instance for Oracle and an Amazon Redshift cluster. We provision through AWS CloudFormation because it simplifies the process, so we can concentrate on tasks related to data migration. When you create a stack from the AWS CloudFormation template, it provisions the following resources:

- A VPC with CIDR (10.0.0.0/24) with two public subnets in your region, DBSubnet1 at the address 10.0.0.0/26 in Availability Zone (AZ) 1 and DBSubnet2 at the address 10.0.0.64/26, in AZ 12.
- A DB subnet group that includes DBSubnet1 and DBSubnet2.
- Oracle RDS Standard Edition Two with these deployment options:
  - License Included
  - Single-AZ setup
  - db.m3.medium or equivalent instance class
• Port 1521
• Default option and parameter groups
• Amazon Redshift cluster with these deployment options:
  • dc1.large
  • Port 5439
  • Default parameter group
• A security group with ingress access from your computer or 0.0.0.0/0 (access from anywhere) based on the input parameter

We have designed the AWS CloudFormation template to require few inputs from the user. It provisions the necessary AWS resources with minimum recommended configurations. However, if you want to change some of the configurations and parameters, such as the VPC CIDR block and Amazon RDS instance types, feel free to update the template.

We use the AWS Management Console to provision the AWS DMS resources, such as the replication instance, endpoints, and tasks. You install client tools such as SQL Workbench/J and the AWS Schema Conversion Tool (AWS SCT) on your local computer to connect to the Amazon RDS instances.

Following is an illustration of the migration architecture for this walkthrough.
Step-by-Step Migration

In the following sections, you can find step-by-step instructions for migrating an Amazon RDS for Oracle database to Amazon Redshift. These steps assume that you have already prepared your source database as described in preceding sections.

Topics

- Step 1: Launch the RDS Instances in a VPC by Using the AWS CloudFormation Template (p. 212)
- Step 2: Install the SQL Tools and AWS Schema Conversion Tool on Your Local Computer (p. 216)
- Step 3: Test Connectivity to the Oracle DB Instance and Create the Sample Schema (p. 219)
- Step 4: Test the Connectivity to the Amazon Redshift Database (p. 223)
- Step 5: Use AWS SCT to Convert the Oracle Schema to Amazon Redshift (p. 224)
- Step 6: Validate the Schema Conversion (p. 229)
- Step 7: Create an AWS DMS Replication Instance (p. 230)
- Step 8: Create AWS DMS Source and Target Endpoints (p. 231)
- Step 9: Create and Run Your AWS DMS Migration Task (p. 234)
- Step 10: Verify That Your Data Migration Completed Successfully (p. 238)
- Step 11: Delete Walkthrough Resources (p. 240)

Step 1: Launch the RDS Instances in a VPC by Using the AWS CloudFormation Template

Before you begin, you'll need to download an AWS CloudFormation template. Follow these instructions:

1. Download the following archive to your computer:  [http://docs.aws.amazon.com/dms/latest/sbs/samples/dms-sbs-RDSOracle2Redshift.zip](http://docs.aws.amazon.com/dms/latest/sbs/samples/dms-sbs-RDSOracle2Redshift.zip)
2. Extract the AWS CloudFormation template (Oracle_Redshift_For_DMSDemo.template) from the archive.
3. Copy and paste the Oracle_Redshift_For_DMSDemo.template file into your current directory.

Now you need to provision the necessary AWS resources for this walkthrough.

2. Choose Create stack.
3. On the Select Template page, choose Upload a template to Amazon S3.
4. Click Choose File, and then choose the Oracle_Redshift_For_DMSDemo.template file that you extracted from the dms-sbs-RDSOracle2Redshift.zip archive.
5. Choose Next. On the Specify Details page, provide parameter values as shown following.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>*Stack Name *</td>
<td>Enter OracletoRedshiftDWusingDMS.</td>
</tr>
<tr>
<td>OracleDBName</td>
<td>Provide a unique name for your database. The name should begin with a letter. The default is ORCL.</td>
</tr>
<tr>
<td>OracleDBUsername</td>
<td>Specify the admin (DBA) user for managing the Oracle instance. The default is oraadmin.</td>
</tr>
<tr>
<td>For This Parameter</td>
<td>Do This</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OracleDBPassword</td>
<td>Provide the password for the admin user. The default is oraadmin123</td>
</tr>
<tr>
<td>RedshiftDBName</td>
<td>Provide any unique name for your database. The name should begin with a letter. The default is test.</td>
</tr>
<tr>
<td>RedshiftDBUsername</td>
<td>Provide the password for the master user. The default is Redshift#123.</td>
</tr>
<tr>
<td>ClientIP</td>
<td>Specify the IP address in CIDR (x.x.x.x/32) format for your local computer. You can get your IP address from whatsmyp.org. Your RDS instances' security group will allow ingress to this IP address. The default is access from anywhere (0.0.0.0/0), which is not recommended; you should use your IP address for this walkthrough.</td>
</tr>
</tbody>
</table>
6. Choose **Next**. On the **Options** page, choose **Next**.
7. On the **Review** page, review the details, and if they are correct choose **Create**.
Create stack

Select Template
Specify Details
Options
Review

Review

Template

Template URL: https://s3.amazonaws.com/cloudformation-examples/oracle-dw-to-redshift-dw-template.yaml
Description: This CloudFormation sample template OracleDWtoRedshift_DW can be used to test the data warehouse migration using AWS DMS service.

Estimate cost: Link is not available

Details

Stack name: OracleToRedshiftDWusingDMS

Source Oracle Database Configuration

OracleDBName: ORCL
OracleDBUsername: oraadmin
OracleDBPassword: ...

Target Redshift Database Configuration

RedshiftDBName: test
RedshiftDBUsername: masteruser
RedshiftDBPassword: ...

Enter IP address for DB Security group Configuration

ClientIP: 0.0.0.0/0

Options

Tags

No tags provided

Advanced

Notification
Timeout: none
Rollback on failure: yes
8. AWS can take about 20 minutes or more to create the stack with an Amazon RDS for Oracle instance and an Amazon Redshift cluster.

9. After the stack is created, select the OracleToRedshiftDWUsingDMS stack, and then choose the Outputs view. Record the JDBC connection strings, OracleJDBCConnectionString and RedshiftJDBCConnectionString, for use later in this walkthrough to connect to the Oracle and Amazon Redshift databases.

Step 2: Install the SQL Tools and AWS Schema Conversion Tool on Your Local Computer

Next, you need to install a SQL client and AWS SCT on your local computer.

This walkthrough assumes you will use the SQL Workbench/J client to connect to the RDS instances for migration validation.
1. Download SQL Workbench/J from the SQL Workbench/J website, and then install it on your local computer. This SQL client is free, open-source, and DBMS-independent.

2. Download the JDBC driver for your Oracle database release. For more information, go to https://www.oracle.com/jdbc.

3. Download the Amazon Redshift driver file, RedshiftJDBC41-1.1.17.1017.jar, as described following.
   a. Find the Amazon S3 URL to the file in Previous JDBC Driver Versions of the Amazon Redshift Cluster Management Guide.
   b. Download the driver as described in Download the Amazon Redshift JDBC Driver of the same guide.

4. Using SQL Workbench/J, configure JDBC drivers for Oracle and Amazon Redshift to set up connectivity, as described following.
   a. In SQL Workbench/J, choose File, then choose Manage Drivers.
   b. From the list of drivers, choose Oracle.
   c. Choose the Open icon, then choose the ojdbc.jar file that you downloaded in the previous step. Choose OK.
   d. From the list of drivers, choose Redshift.
   e. Choose the Open icon, then choose the Amazon Redshift JDBC driver that you downloaded in the previous step. Choose OK.
Next, install AWS SCT and the required JDBC drivers.


2. Follow the instructions to install AWS SCT.

3. Launch AWS SCT.

4. In AWS SCT, choose Global settings from Settings.

5. Choose Settings, Global settings, then choose Drivers, and then choose Browse for Oracle driver path. Locate the Oracle JDBC driver and choose OK.

6. Choose Browse for Amazon Redshift driver path. Locate the Amazon Redshift JDBC driver and choose OK. Choose OK to close the dialog box.
Step 3: Test Connectivity to the Oracle DB Instance and Create the Sample Schema

After the AWS CloudFormation stack has been created, test the connection to the Oracle DB instance by using SQL Workbench/J and then create the HR sample schema.

1. In SQL Workbench/J, choose File, then choose Connect window. Create a new connection profile using the following information.
<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>New profile name</td>
<td>Enter RDSOracleConnection.</td>
</tr>
<tr>
<td>Driver</td>
<td>Choose Oracle (oracle.jdbc.OracleDriver).</td>
</tr>
<tr>
<td>URL</td>
<td>Use the OracleJDBCConnectionString value you recorded when you examined the output details of the DMSdemo stack in a previous step.</td>
</tr>
<tr>
<td>Username</td>
<td>Enter oraadmin.</td>
</tr>
<tr>
<td>Password</td>
<td>Enter oraadmin123.</td>
</tr>
</tbody>
</table>

2. Test the connection by choosing Test. Choose OK to close the dialog box, then choose OK to create the connection profile.
3. Create the **SH** schema you will use for migration using a custom `Oraclesalesstarschema.sql` SQL script. To obtain this script, do the following:

   - Download the following archive to your computer: [http://docs.aws.amazon.com/dms/latest/sbs/samples/dms-sbs-RDSOracle2Redshift.zip](http://docs.aws.amazon.com/dms/latest/sbs/samples/dms-sbs-RDSOracle2Redshift.zip)
   - Extract the `Oraclesalesstarschema.sql` SQL script from the archive.
   - Copy and paste the `Oraclesalesstarschema.sql` file into your current directory.
     
     a. Open the SQL script in a text editor. Copy the entire script.
     
     b. In SQL Workbench/J, paste the SQL script in the Default.wksp window showing **Statement 1**.

---

**Note**

If your connection is unsuccessful, ensure that the IP address you assigned when creating the AWS CloudFormation template is the one you are attempting to connect from. This issue is the most common one when trying to connect to an instance.
c. Choose SQL, then choose **Execute All**.

4. Verify the object types and count in **SH** Schema were created successfully by running the following SQL query.

   ```sql
   Select OBJECT_TYPE, COUNT(*) from dba_OBJECTS where owner='SH'
   GROUP BY OBJECT_TYPE;
   ```

   The results of this query should be similar to the following.
5. Verify the total number of tables and number of rows for each table by running the following SQL query.

   ```sql
   SELECT table_name, num_rows FROM dba_tables WHERE owner='SH' ORDER BY 1;
   ```

   The results of this query should be similar to the following.

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>NUM_ROWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHANNELS</td>
<td>5</td>
</tr>
<tr>
<td>CUSTOMERS</td>
<td>8</td>
</tr>
<tr>
<td>PRODUCTS</td>
<td>66</td>
</tr>
<tr>
<td>PROMOTIONS</td>
<td>503</td>
</tr>
<tr>
<td>SALES</td>
<td>553</td>
</tr>
</tbody>
</table>

6. Verify the integrity in tables. Check the number of sales made in different channels by running the following SQL query.

   ```sql
   SELECT b.channel_desc, count(*) FROM SH.SALES a, SH.CHANNELS b WHERE a.channel_id = b.channel_id GROUP BY b.channel_desc ORDER BY 1;
   ```

   The results of this query should be similar to the following.

<table>
<thead>
<tr>
<th>CHANNEL_DESC</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Sales</td>
<td>710</td>
</tr>
<tr>
<td>Internet</td>
<td>52</td>
</tr>
<tr>
<td>Partners</td>
<td>344</td>
</tr>
</tbody>
</table>

**Note**

The preceding examples are representative of validation queries. When you perform actual migrations, you should develop similar queries to validate the schema and the data integrity.

## Step 4: Test the Connectivity to the Amazon Redshift Database

Next, test your connection to your Amazon Redshift database.

1. In SQL Workbench/J, choose **File**, then choose **Connect window**. Choose the **Create a new connection profile** icon. Connect to the Amazon Redshift database in SQL Workbench/J by using the information shown following.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>New profile name</td>
<td>Enter RedshiftConnection.</td>
</tr>
</tbody>
</table>
# Step-by-Step Migration

### For This Parameter | Do This
--- | ---
Driver | Choose Redshift (com.amazon.redshift.jdbc42.Driver).
URL | Use the `RedshiftJDBCConnectionString` value you recorded when you examined the output details of the DMSdemo stack in a previous step.
Username | Enter `redshiftadmin`.
Password | Enter `Redshift#123`.

2. Test the connection by choosing **Test**. Choose **OK** to close the dialog box, then choose **OK** to create the connection profile.

### Note
If your connection is unsuccessful, ensure that the IP address you assigned when creating the AWS CloudFormation template is the one you are attempting to connect from. This issue is the most common one when trying to connect to an instance.

3. Verify your connectivity to the Amazon Redshift DB instance by running a sample SQL command, such as `select current_date;`.

---

## Step 5: Use AWS SCT to Convert the Oracle Schema to Amazon Redshift

Before you migrate data to Amazon Redshift, you convert the Oracle schema to an Amazon Redshift schema. [This video covers all the steps of this process](https://aws.amazon.com/solutions/convert-oracle-to-redshift/).

To convert an Oracle schema to an Amazon Redshift schema using AWS Schema Conversion Tool (AWS SCT), do the following:
1. Launch AWS SCT. In AWS SCT, choose **File**, then choose **New Project**. Create a new project named `DWSchemaMigrationDemoProject`, specify the **Location** of the project folder, and then choose **OK**.

2. Choose **Add source** to add a source Oracle database to your project, then choose **Oracle**, and choose **Next**.

3. Enter the following information, and then choose **Test Connection**.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection name</strong></td>
<td>Enter <code>Oracle DW</code>. AWS SCT displays this name in the tree in the left panel.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Choose <code>SID</code>.</td>
</tr>
<tr>
<td><strong>Server name</strong></td>
<td>Use the <code>OracleJDBCConnectionString</code> value you used to connect to the Oracle DB instance, but remove the JDBC prefix information and the port and database name suffix. For example, a sample connection string you use with SQL Workbench/J might be <code>jdbc:oracle:thin:@abc12345678.cqi87654abc.us-west-2.rds.amazonaws.com:1521:ORCL</code>. For AWS SCT <strong>Server name</strong>, you remove <code>jdbc:oracle:thin:@</code> and <code>:1521:ORCL</code> and use just the server name: <code>abc12345678.cqi87654abc.us-west-2.rds.amazonaws.com</code>.</td>
</tr>
<tr>
<td><strong>Server port</strong></td>
<td>Enter <code>1521</code>.</td>
</tr>
<tr>
<td><strong>Oracle SID</strong></td>
<td>Enter <code>ORCL</code>.</td>
</tr>
<tr>
<td><strong>User name</strong></td>
<td>Enter <code>oraadmin</code>.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Enter <code>oraadmin123</code>.</td>
</tr>
</tbody>
</table>
4. Choose **OK** to close the alert box, then choose **Connect** to close the dialog box and to connect to the Oracle DB instance.

5. Choose **Add target** to add a target Amazon Redshift database to your project, then choose **Amazon Redshift**, and choose **Next**.

6. Enter the following information and then choose **Test Connection**.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection name</strong></td>
<td>Enter Amazon Redshift. AWS SCT displays this name in the tree in the right panel.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Choose <strong>SID</strong>.</td>
</tr>
<tr>
<td><strong>Server name</strong></td>
<td>Use the <code>RedshiftJDBCConnectionString</code> value you used to connect to the Amazon Redshift cluster, but remove the JDBC prefix information and the port suffix. For example, a sample connection string you</td>
</tr>
</tbody>
</table>
For This Parameter | Do This
---|---
use with SQL Workbench/J might be "jdbc:redshift://oracletoredshiftdwusingdms-redshiftcluster-abc123567.abc87654321.us-west-2.redshift.amazonaws.com:5439/test". For AWS SCT **Server name**, you remove "jdbc:redshift://" and :5439/test" to use just the server name: "oracletoredshiftdwusingdms-redshiftcluster-abc123567.abc87654321.us-west-2.redshift.amazonaws.com" | 

| Server port | Enter 5439. |
| User name | Enter redshiftadmin. |
| Password | Enter Redshift#123. |
| Use AWS Glue | Turn off this option. |

7. Choose **OK** to close the alert box, then choose **Connect** to connect to the Amazon Redshift DB instance.

8. In the tree in the left panel, select only the **SH** schema. In the tree in the right panel, select your target Amazon Redshift database. Choose **Create mapping**.

9. Choose **Main view**.

10. In the tree in the left panel, right-click the **SH** schema and choose **Collect Statistics**. AWS SCT analyzes the source data to recommend the best keys for the target Amazon Redshift database. For more information, see **Collecting or Uploading Statistics**.

   **Note**

   If the **SH** schema does not appear in the list, choose **Actions**, then choose **Refresh from Database**.

11. In the tree in the left panel, right-click the **SH** schema and choose **Create report**. AWS SCT analyzes the **SH** schema and creates a database migration assessment report for the conversion to Amazon Redshift.
12. Check the report and the action items it suggests. The report discusses the type of objects that can be converted by using AWS SCT, along with potential migration issues and actions to resolve these issues. For this walkthrough, you should see something like the following:

Database objects with conversion actions for Amazon Redshift

Of the total 25 database storage object(s) and 49 database code object(s) in the source database, we identified 16 (64%) database storage object(s) and 12 (24%) database code object(s) suitable for conversion to Amazon Redshift automatically or with minimal changes. We found 1 encrypted object(s).

9 (36%) database storage object(s) require 13 medium and 2 complex user action(s) to complete the conversion.

10 (20%) database code object(s) require 3 medium and 7 complex user action(s) to complete the conversion.

Figure: Conversion statistics for database storage objects

13. Review the report summary. To save the report, choose either Save to CSV or Save to PDF.

14. Choose the Action Items tab. The report discusses the type of objects that can be converted by using AWS SCT, along with potential migration issues and actions to resolve these issues.

15. In the tree in the left panel, right-click the SH schema and choose Convert schema.

16. Choose Yes for the confirmation message. AWS SCT then converts your schema to the target database format.

Note
The choice of the Amazon Redshift sort keys and distribution keys is critical for optimal performance. You can use key management in AWS SCT to customize the choice of keys. For this walkthrough, we use the defaults recommended by AWS SCT. For more information, see Optimizing Amazon Redshift.

17. In the tree in the right panel, choose the converted sh schema, and then choose Apply to database to apply the schema scripts to the target Amazon Redshift instance.

18. In the tree in the right panel, choose the sh schema, and then choose Refresh from Database to refresh from the target database.

The database schema has now been converted and imported from source to target.
Step 6: Validate the Schema Conversion

To validate the schema conversion, you compare the objects found in the Oracle and Amazon Redshift databases using SQL Workbench/J.

1. In SQL Workbench/J, choose File, then choose Connect window. Choose the RedshiftConnection you created in an earlier step. Choose OK.

2. Run the following script to verify the number of object types and count in SH schema in the target Amazon Redshift database. These values should match the number of objects in the source Oracle database.

   ```sql
   SELECT 'TABLE' AS OBJECT_TYPE,
          TABLE_NAME AS OBJECT_NAME,
          TABLE_SCHEMA AS OBJECT_SCHEMA
   FROM information_schema.TABLES
   WHERE TABLE_TYPE = 'BASE TABLE'
   AND OBJECT_SCHEMA = 'sh';
   ```

   The output from this query should be similar to the following.

<table>
<thead>
<tr>
<th>object_type</th>
<th>object_name</th>
<th>object_schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>channels</td>
<td>sh</td>
</tr>
<tr>
<td>TABLE</td>
<td>customers</td>
<td>sh</td>
</tr>
<tr>
<td>TABLE</td>
<td>products</td>
<td>sh</td>
</tr>
<tr>
<td>TABLE</td>
<td>promotions</td>
<td>sh</td>
</tr>
<tr>
<td>TABLE</td>
<td>sales</td>
<td>sh</td>
</tr>
</tbody>
</table>

3. Verify the sort and distributions keys that are created in the Amazon Redshift cluster by using the following query.

   ```sql
   set search_path to '$user', 'public', 'sh';
   SELECT tablename,
          "column", type, encoding, distkey, sortkey, "notnull"
   FROM pg_table_def
   WHERE (distkey = TRUE OR sortkey <> 0);
   ```

   The results of the query reflect the distribution key (distkey) and sort key (sortkey) choices made by using AWS SCT key management.

<table>
<thead>
<tr>
<th>tablename</th>
<th>column</th>
<th>type</th>
<th>encoding</th>
<th>distkey</th>
<th>sortkey</th>
<th>notnull</th>
</tr>
</thead>
<tbody>
<tr>
<td>channels</td>
<td>channel_id</td>
<td>numeric(38,18)</td>
<td>none</td>
<td>true</td>
<td></td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>customers</td>
<td>cust_id</td>
<td>numeric(38,18)</td>
<td>none</td>
<td>false</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>customers</td>
<td>cust_gender</td>
<td>character(2)</td>
<td>none</td>
<td>false</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>customers</td>
<td>cust_year_of_birth</td>
<td>smallint</td>
<td>none</td>
<td>false</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>customers</td>
<td>cust_marital_status</td>
<td>character varying(40)</td>
<td>none</td>
<td>false</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

229
Step 7: Create an AWS DMS Replication Instance

After we validate the schema structure between source and target databases, as described preceding, we proceed to the core part of this walkthrough, which is the data migration. The following illustration shows a high-level view of the migration process.

A DMS replication instance performs the actual data migration between source and target. The replication instance also caches the transaction logs during the migration. How much CPU and memory capacity a replication instance has influences the overall time required for the migration.

To create an AWS DMS replication instance, do the following:

1. Sign in to the AWS Management Console, open the AWS DMS console at https://console.aws.amazon.com/dms/v2/, and choose Create Migration. If you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information on the permissions required, see IAM Permissions.

2. Choose Create migration to start a database migration.

3. On the Welcome page, choose Next.

4. On the Create replication instance page, specify your replication instance information as shown following.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter DMSdemo-repserver.</td>
</tr>
</tbody>
</table>
### Step-by-Step Migration

1. **Description**  
Enter a brief description, such as DMS demo replication server.

2. **Instance class**  
Choose `dms.t2.medium`. This instance class is large enough to migrate a small set of tables.

3. **VPC**  
Choose `OracleToRedshiftUsingDMS`, which is the VPC that was created by the AWS CloudFormation stack.

4. **Multi-AZ**  
Choose No.

5. **Publicly accessible**  
Leave this item selected.

5. For the **Advanced** section, leave the default settings as they are, and choose **Next**.

### Step 8: Create AWS DMS Source and Target Endpoints

While your replication instance is being created, you can specify the source and target database endpoints using the **AWS Management Console**. However, you can only test connectivity after the replication instance has been created, because the replication instance is used in the connection.

1. **Specify your connection information for the source Oracle database and the target Amazon Redshift database.** The following table describes the source settings.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Endpoint Identifier</strong></td>
<td>Enter <code>Orasource</code> (the Amazon RDS for Oracle endpoint).</td>
</tr>
<tr>
<td><strong>Source Engine</strong></td>
<td>Choose <code>oracle</code>.</td>
</tr>
<tr>
<td><strong>Server name</strong></td>
<td>Provide the Oracle DB instance name. This name is the <strong>Server name</strong> value that you used for AWS SCT, such as &quot;abc123567.abc87654321.us-west-2.rds.amazonaws.com&quot;.</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>Enter 1521.</td>
</tr>
<tr>
<td><strong>SSL mode</strong></td>
<td>Choose <strong>None</strong>.</td>
</tr>
<tr>
<td><strong>Username</strong></td>
<td>Enter <code>oraadmin</code>.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Enter <code>oraadmin123</code>.</td>
</tr>
<tr>
<td><strong>SID</strong></td>
<td>Enter <code>ORCL</code>.</td>
</tr>
</tbody>
</table>

The following table describes the target settings.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Endpoint Identifier</strong></td>
<td>Enter <code>Redshifttarget</code> (the Amazon Redshift endpoint).</td>
</tr>
<tr>
<td>For This Parameter</td>
<td>Do This</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------</td>
</tr>
<tr>
<td><strong>Target Engine</strong></td>
<td>Choose <code>redshift</code>.</td>
</tr>
<tr>
<td><strong>Servername</strong></td>
<td>Provide the Amazon Redshift DB instance name. This name is the <strong>Server name</strong> value that you used for AWS SCT, such as &quot;oracletoredshiftdwusingdms-redshiftcluster-abc123567.abc87654321.us-west-2.redshift.amazonaws.com&quot;.</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>Enter 5439.</td>
</tr>
<tr>
<td><strong>SSL mode</strong></td>
<td>Choose <strong>None</strong>.</td>
</tr>
<tr>
<td><strong>Username</strong></td>
<td>Enter <code>redshiftadmin</code>.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Enter <code>Redshift#123</code>.</td>
</tr>
<tr>
<td><strong>Database name</strong></td>
<td>Enter <code>test</code>.</td>
</tr>
</tbody>
</table>

The completed page should look like the following.
2. Wait for the status to say **Replication instance created successfully**.

3. To test the source and target connections, choose **Run Test** for the source and target connections.

4. Choose **Next**.
Step 9: Create and Run Your AWS DMS Migration Task

Using an AWS DMS task, you can specify what schema to migrate and the type of migration. You can migrate existing data, migrate existing data and replicate ongoing changes, or replicate data changes only. This walkthrough migrates existing data only.

1. On the Create Task page, specify the task options. The following table describes the settings.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task name</td>
<td>Enter <code>migrateSHschema</code>.</td>
</tr>
<tr>
<td>Replication instance</td>
<td>Shows <code>DMSdemo-repserver</code> (the AWS DMS replication instance created in an earlier step).</td>
</tr>
<tr>
<td>Source endpoint</td>
<td>Shows <code>orасоurсе</code> (the Amazon RDS for Oracle endpoint).</td>
</tr>
<tr>
<td>Target endpoint</td>
<td>Shows <code>redshifttарget</code> (the Amazon Redshift endpoint).</td>
</tr>
<tr>
<td>Migration type</td>
<td>Choose <strong>Migrate existing data</strong>.</td>
</tr>
<tr>
<td>Start task on create</td>
<td>Choose this option.</td>
</tr>
</tbody>
</table>

The page should look like the following.
Create task

A task can contain one or more table mappings which define what data is moved from the source to the target automatically.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task name</td>
<td>migrateS1hchema</td>
</tr>
<tr>
<td>Replication instance</td>
<td>dmsdemo-repserver - vpc-809665e4</td>
</tr>
<tr>
<td>Source endpoint</td>
<td>orasource</td>
</tr>
<tr>
<td>Target endpoint</td>
<td>redshifttarget</td>
</tr>
<tr>
<td>Migration type</td>
<td>Migrate existing data</td>
</tr>
<tr>
<td>Target table preparation mode</td>
<td>Choose Do nothing.</td>
</tr>
<tr>
<td>Include LOB columns in replication</td>
<td>Choose Limited LOB mode.</td>
</tr>
<tr>
<td>Max LOB size (kb)</td>
<td>Accept the default (32).</td>
</tr>
</tbody>
</table>

The section should look like the following.
3. In the **Selection rules** section, specify the settings as shown in the following table.

<table>
<thead>
<tr>
<th>For This Parameter</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema name is</td>
<td>Choose Enter a schema.</td>
</tr>
<tr>
<td>Schema name is like</td>
<td>Enter SH%.</td>
</tr>
<tr>
<td>Table name is like</td>
<td>Enter %.</td>
</tr>
<tr>
<td>Action</td>
<td>Choose Include.</td>
</tr>
</tbody>
</table>

The section should look like the following:
4. Choose **Add selection rule**.

5. Choose **Create task**. The task begins immediately. The **Tasks** section shows you the status of the migration task.
Step 10: Verify That Your Data Migration Completed Successfully

When the migration task completes, you can compare your task results with the expected results.

1. On the navigation pane, choose Tasks.
2. Choose your migration task (migrateSHschema).
3. Choose the Table statistics tab, shown following.
4. Connect to the Amazon Redshift instance by using SQL Workbench/J, and then check whether the database tables were successfully migrated from Oracle to Amazon Redshift by running the SQL script shown following.

```sql
select "table", tbl_rows
from svv_table_info
where
  SCHEMA = 'sh'
```
order by 1;

Your results should look similar to the following.

<table>
<thead>
<tr>
<th>table</th>
<th>tbl_rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>channels</td>
<td>5</td>
</tr>
<tr>
<td>customers</td>
<td>8</td>
</tr>
<tr>
<td>products</td>
<td>66</td>
</tr>
<tr>
<td>promotions</td>
<td>503</td>
</tr>
<tr>
<td>sales</td>
<td>1106</td>
</tr>
</tbody>
</table>

5. To verify whether the output for tables and number of rows from the preceding query matches what is expected for RDS Oracle, compare your results with those in previous steps.

6. Run the following query to check the relationship in tables; this query checks the departments with employees greater than 10.

```
Select b.channel_desc,count(*) from SH.SALES a,SH.CHANNELS b where a.channel_id=b.channel_id group by b.channel_desc order by 1;
```

The output from this query should be similar to the following.

<table>
<thead>
<tr>
<th>channel_desc</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Sales</td>
<td>355</td>
</tr>
<tr>
<td>Internet</td>
<td>26</td>
</tr>
<tr>
<td>Partners</td>
<td>172</td>
</tr>
</tbody>
</table>

7. Verify column compression encoding.

DMS uses an Amazon Redshift COPY operation to load data. By default, the COPY command applies automatic compression whenever loading to an empty target table. The sample data for this walkthrough is not large enough for automatic compression to be applied. When you migrate larger data sets, COPY will apply automatic compression.

For more details about automatic compression on Amazon Redshift tables, see [Loading Tables with Automatic Compression](#).

To view compression encodings, run the following query.

```
SELECT * FROM pg_table_def WHERE schemaname = 'sh';
```

Now you have successfully completed a database migration from an Amazon RDS for Oracle DB instance to Amazon Redshift.

**Step 11: Delete Walkthrough Resources**

After you have completed this walkthrough, perform the following steps to avoid being charged further for AWS resources used in the walkthrough. It's necessary that you do the steps in order, because some resources cannot be deleted if they have a dependency upon another resource.

To delete AWS DMS resources, do the following:
1. On the navigation pane, choose **Tasks**, choose your migration task (**migratehrschema**), and then choose **Delete**.
2. On the navigation pane, choose **Endpoints**, choose the Oracle source endpoint (**orarsource**), and then choose **Delete**.
3. Choose the Amazon Redshift target endpoint (**redshifttarget**), and then choose **Delete**.
4. On the navigation pane, choose **Replication instances**, choose the replication instance (**DMSdemo-repserver**), and then choose **Delete**.

Next, you must delete your AWS CloudFormation stack, **DMSdemo**. Do the following:

   
   If you are signed in as an IAM user, you must have the appropriate permissions to access AWS CloudFormation.
2. Choose your AWS CloudFormation stack, **OracletoRedshiftDWusingDMS**.
3. For **Actions**, choose **Delete stack**.

The status of the stack changes to **DELETE_IN_PROGRESS** while AWS CloudFormation cleans up the resources associated with the **OracletoRedshiftDWusingDMS** stack. When AWS CloudFormation is finished cleaning up resources, it removes the stack from the list.

**Next Steps**

You can explore several other features of AWS DMS that were not included in this walkthrough, including the following:

- The AWS DMS change data capture (CDC) feature, for ongoing replication of data.
- Transformation actions that let you specify and apply transformations to the selected schema or table as part of the migration process.

For more information, see [https://docs.aws.amazon.com/dms/latest/userguide/CHAP_GettingStarted.html](https://docs.aws.amazon.com/dms/latest/userguide/CHAP_GettingStarted.html).

**Migrating a BigQuery Project to Amazon Redshift**

This walkthrough gets you started with heterogeneous database migration from BigQuery to Amazon Redshift. To automate the migration, we use the AWS Schema Conversion Tool (AWS SCT) that runs on Windows. This introductory exercise provides you with a good understanding of the steps involved in such a migration.

At a high level, the steps involved in this migration are the following:

- Use the Google Cloud management console to do the following:
  - Create a service account, which AWS SCT can use to connect to your source BigQuery project.
  - Create a Cloud Storage bucket to store your source data during migration.
- Use the AWS Management Console to do the following:
  - Create an Amazon Redshift cluster.
  - Create an Amazon Simple Storage Service (Amazon S3) bucket.
  - Use AWS SCT to convert source database schemas and apply converted code to your target database.
  - Use data extraction agents to migrate data.
To see all the steps of the migration process, watch this video.

This walkthrough takes approximately three hours to complete. Make sure that you delete resources at the end of this walkthrough to avoid additional charges.

Topics
- Prerequisites (p. 242)
- Migration Overview (p. 242)
- Step-by-Step Migration (p. 245)
- Next Steps (p. 252)

Prerequisites

The following prerequisites are also required to complete this walkthrough:

- Familiarity with the AWS Management Console, AWS SCT, Amazon Redshift, Google Cloud management console, and SQL.
- An AWS account with AWS Identity and Access Management (IAM) credentials. Make sure that you can use these credentials to launch Amazon Redshift clusters and create an Amazon S3 bucket in your AWS Region. For more information, see Creating an IAM role for your Amazon Redshift cluster and Create an IAM user for your Amazon S3 bucket.
- Basic knowledge of the Amazon Virtual Private Cloud (Amazon VPC) service and of security groups. For information about using Amazon Redshift in a VPC, see Managing clusters in a VPC.
- An understanding of the supported features and limitations on using BigQuery as a source for AWS SCT. For more information, see Using BigQuery as a source in AWS SCT User Guide.
- An AWS service profile in AWS SCT with access to the S3 bucket. For more information, see Storing service profiles in AWS SCT User Guide.

For more information about the AWS Schema Conversion Tool, see https://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_Welcome.html.

For this migration walkthrough, we expect that you are familiar with BigQuery. You can use your BigQuery project for this migration, or create a new one. For example, you can use one of the public BigQuery datasets that are available in Google Cloud Marketplace.

We recommend that you don’t use your production workloads for the migration in this walkthrough. After you get familiar with migration tools and AWS services, you can migrate your production workloads.

Migration Overview

This section provides high-level guidance for customers looking for a way to migrate from BigQuery to Amazon Redshift. After you complete this introductory exercise, understand the migration process, and become familiar with migration automation tools, plan the migration of your production workloads.

The following illustration demonstrates the migration architecture for this walkthrough.
First, you create a service account to connect to your BigQuery project. Then you create an Amazon Redshift database, as well as the buckets in Cloud Storage and Amazon S3. After this setup, you use AWS SCT to convert source database schemas and apply them to your target database. Finally, you install and configure a data extraction agent to migrate data, upload it to your S3 bucket, and then copy to Amazon Redshift. For big datasets, you can use several data extraction agents to increase the speed of data migration.

To connect to BigQuery, AWS SCT uses the Application Layer Transport Security (ALTS) authentication and encryption system. To connect to Amazon S3 and Amazon Redshift, AWS SCT uses the HTTPS and SSL protocols.

**Migration Strategy**

For BigQuery to Amazon Redshift migrations, you can use the following typical migration approach.

1. **Future State Architecture Design**
   This step defines the architecture of your new system in the target environment. This architecture includes databases, applications, scripts, and so on.

2. **Database Schema Conversion**
   You can use AWS SCT to automate the conversion of your source database to Amazon Redshift. For more information, see Convert Database Schemas (p. 250).

3. **Application Conversion or Remediation**
After you migrate your data storage, make sure that you update your applications. You can use AWS SCT to convert SQL queries in your application code. For more information, see Converting SQL code in your applications.

4. Scripts, ETL, Reports Conversion

In addition to applications, make sure that you update all other components of your source system. These include business intelligence reports, extract, transform, and load (ETL) processes, and other scripts.

5. Integration with Third-Party Applications

Your applications usually connect to other applications or monitoring tools. Your migration from BigQuery to Amazon Redshift affects these dependencies.

6. Data Migration

You can use AWS SCT to manage a data extraction agent that migrates data from BigQuery to Amazon Redshift. For more information, see Data Extraction Agents (p. 250).

7. Testing and Bug Fixing

Migration touches all the stored procedures and functions and affects substantial parts of the application code. For this reason, good testing is required both at the unit and system functional level.

8. Performance Tuning

Because of database platform differences and syntax, certain constructs or combinations of data objects can perform differently on the new platform. The performance tuning part of the migration resolves any bottlenecks.

9. Setup, DevOps, Integration, Deployment, and Security

Take the opportunity to embrace infrastructure as code for the migration. Make sure that you also focus on the application security. Finally, plan the cutover.

Security in the AWS Cloud

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. For more information, see Shared Responsibility Model.

Amazon Redshift protects data with AWS encryption solutions, along with all default security controls within AWS services. Your data is encrypted at rest and in transit. Amazon Redshift automatically integrates with AWS Key Management Service (AWS KMS) for key management. AWS KMS uses envelope encryption. For more information, see Data protection in Amazon Redshift.

Access to Amazon Redshift requires credentials that AWS can use to authenticate your requests. Those credentials must have permissions to access AWS resources, such as an Amazon Redshift cluster. You can use AWS Identity and Access Management (IAM) to secure your data by controlling who can access your Amazon Redshift cluster. For more information, see Identity and access management in Amazon Redshift.

Data Types Mapping

Amazon Redshift supports all BigQuery data types. The following table shows the data type mappings that AWS SCT uses by default. Users can set up migration rules in AWS SCT to change the data type of columns. For more information, see Creating migration rules.
### BigQuery data type | Amazon Redshift data type
--- | ---
BOOLEAN | BOOLEAN
BYTES(L) | BINARY VARYING(L)
BYTES | BINARY VARYING(1024000)
DATE | DATE
DATETIME | TIMESTAMP WITHOUT TIME ZONE
GEOGRAPHY | GEOGRAPHY
INTERVAL | CHARACTER VARYING(256)
JSON | SUPER
INTEGER | BIGINT
NUMERIC(p,s) | NUMERIC(p,s)
NUMERIC | NUMERIC(38,9)
BIGNUMERIC | NUMERIC(38,9)
BIGNUMERIC(p,s) | NUMERIC(p,s) if p is less than or equal to 38 or s is less than or equal to 37.
BIGNUMERIC(p,s) | CHARACTER VARYING(256) if p is more than 38 or s is more than 37.
FLOAT | DOUBLE PRECISION
STRING(L) | CHARACTER VARYING(L) if L is less than 65,535.
STRING | CHARACTER VARYING(65535)
STRUCT | SUPER
TIME | TIME WITHOUT TIME ZONE
TIMESTAMP | TIMESTAMP WITHOUT TIME ZONE

### Limitations

You can use AWS SCT to automatically convert a majority of your BigQuery code and storage objects. These objects include datasets, tables, views, stored procedures, functions, data types, and so on. However, AWS SCT has some limitations when using BigQuery as a source.

For example, AWS SCT can’t convert subqueries in analytic functions, as well as geography, statistical aggregate, or some of the string functions. You can find the full list of limitations in the AWS SCT user guide. For more information, see Limitations on using BigQuery as a source.

### Step-by-Step Migration

In the following sections, you can find step-by-step instructions for migrating your BigQuery project to Amazon Redshift. These steps assume that you have already prepared your source and target databases as described in preceding sections.
Topics

- Step 1: Create a BigQuery Service Account Key File (p. 246)
- Step 2: Create an Amazon Redshift Cluster (p. 246)
- Step 3: Create Buckets to Store Your Temporary Data (p. 247)
- Step 4: Install AWS SCT on Your Local Computer (p. 248)
- Step 5: Create an AWS SCT Project (p. 249)
- Step 6: Convert Database Schemas (p. 250)
- Step 7: Install and Configure Data Extraction Agents (p. 250)
- Step 8: Run Your Migration Task (p. 251)
- Step 9: Delete Walkthrough Resources (p. 252)

Step 1: Create a BigQuery Service Account Key File

You can connect to BigQuery with a user account or a service account. A service account is a special kind of account designed to be used by applications or compute workloads, rather than a person.

Service accounts don’t have passwords and use a unique email address for identification. You can associate each service account with a service account key, which is a public or private RSA key pair. In this walkthrough, we use a service account key in AWS SCT to access your BigQuery project.

To create a BigQuery service account key

1. Sign in to the Google Cloud management console.
2. Make sure that you have API enabled on your BigQuery API page. If you don’t see API Enabled, choose Enable.
3. On the Service accounts page, choose your BigQuery project, and then choose Create service account.
4. On the Service account details page, enter a descriptive value for Service account name. Choose Create and continue. The Grant this service account access to the project page opens.
5. For Select a role, choose BigQuery, and then choose BigQuery Admin. AWS SCT uses permissions to manage all resources within the project to load your BigQuery metadata in the migration project.
6. Choose Add another role. For Select a role, choose Cloud Storage, and then choose Storage Admin. AWS SCT uses full control of data objects and buckets to extract your data from BigQuery and then load it into Amazon Redshift.
7. Choose Continue, and then choose Done.
8. On the Service accounts page, choose the service account that you created.
9. Choose Keys, Add key, Create new key.
10. Choose JSON, and then choose Create. Choose the folder to save your private key or check the default folder for downloads in your browser.

Step 2: Create an Amazon Redshift Cluster

To store your data in the AWS cloud, you can use your existing Amazon Redshift cluster or create a new one. You don’t need to create any tables because AWS SCT automates this process.

If you don’t plan to migrate data as part of this walkthrough, you can skip this step. To see how AWS SCT converts your database code objects, use a virtual Amazon Redshift target in your project. For more information, see Using virtual targets.

To create an Amazon Redshift cluster
1. Sign in to the AWS Management Console and open the Amazon Redshift console at https://console.aws.amazon.com/redshift/.
2. On the navigation menu, choose **Clusters**.
3. Choose **Create cluster**.
4. For **Cluster identifier**, enter the unique name of your Amazon Redshift cluster.
5. Choose **Free trial**.
6. For **Admin user name**, enter the login for the admin user of your Amazon Redshift cluster.
7. For **Admin user password**, enter the password for the admin user.
8. Choose **Create cluster**.

After you create your Amazon Redshift database, configure a new database user. Then, use the credentials of this user in AWS SCT to access your Amazon Redshift cluster. We don't recommend you to use the admin user for the migration.

Make sure that you grant the following privileges to this new user to complete the migration:

- CREATE ON DATABASE — allows to create new schemas in the database.
- GRANT USAGE ON LANGUAGE — allows to create new functions and procedures in the database.
- GRANT SELECT ON ALL TABLES IN SCHEMA pg_catalog — provides the user with system information about the Amazon Redshift cluster.
- GRANT SELECT ON pg_class_info — provides the user with information about tables distribution style.

You can use the following code example to create a database user and grant the privileges.

```sql
CREATE USER user_name PASSWORD your_password;
GRANT CREATE ON DATABASE db_name TO user_name;
GRANT USAGE ON LANGUAGE plpythonu TO user_name;
GRANT USAGE ON LANGUAGE plpgsql TO user_name;
GRANT SELECT ON ALL TABLES IN SCHEMA pg_catalog TO user_name;
GRANT SELECT ON pg_class_info TO user_name;
GRANT SELECT ON sys_serverless_usage TO user_name;
GRANT SELECT ON pg_database_info TO user_name;
GRANT SELECT ON pg_statistic TO user_name;
```

In the preceding example, replace `user_name` with the name of your user. Then, replace `db_name` with the name of your target Amazon Redshift database. Finally, replace `your_password` with a secure password.

**Step 3: Create Buckets to Store Your Temporary Data**

Data migration from BigQuery to Amazon Redshift includes the following steps:

1. Export data from BigQuery to a Cloud Storage bucket.
2. Extract data from a Cloud Storage bucket.
3. Upload data to an Amazon Simple Storage Service (Amazon S3) bucket.
4. Copy data from an S3 bucket to Amazon Redshift.

You need all four steps because you can't access data directly in BigQuery and you can't upload data directly to Amazon Redshift. Because of these limitations, you need to create buckets to store your data during migration.
To create a Cloud Storage bucket

1. Sign in to the Google Cloud management console.
2. Open the Cloud Storage Browser page.
3. Choose Create bucket.
4. For Name your bucket, enter a name for your Cloud Storage bucket.
5. On the Choose where to store your data page, choose Region for Location type and then choose your region for Location.
6. Leave the default values for other options, and choose Create.

To create an Amazon S3 bucket

1. Sign in to the AWS Management Console and open the Amazon Redshift console at https://console.aws.amazon.com/s3/.
2. Choose Create bucket.
3. For Bucket name, enter the globally unique name of your Amazon S3 bucket.
4. For AWS Region, choose the AWS Region where you want the bucket to reside. Choose a Region close to you to minimize latency and costs.
5. Leave the default values for other options, and choose Create bucket.

Step 4: Install AWS SCT on Your Local Computer

In this step, you install and configure the AWS Schema Conversion Tool. In this walkthrough, we run AWS SCT and the data extraction agent on Windows. However, you can use AWS SCT and data extraction agents on other supported operating systems. For more information, see Installing the schema conversion tool and Installing extraction agents.

To install AWS SCT

2. Extract AWS SCT installer file.
3. Run AWS SCT installer file that you extracted in the previous step.
4. Choose Next, accept the terms of the License Agreement, and choose Next again.
5. Enter the path to the folder where you want to install AWS SCT, and choose Next.
6. Choose Install.
7. Choose Finish to close the installation wizard.

Now you can run AWS SCT. Before you create a new project, make sure that you add the path to an Amazon Redshift JDBC driver in the application settings. You don’t need a JDBC driver to connect to your BigQuery project.

To configure driver settings in AWS SCT

1. Download an Amazon Redshift JDBC driver version 2.1.0.9 or later from https://docs.aws.amazon.com/redshift/latest/mgmt/jdbc20-download-driver.html.
2. Extract the JDBC driver from the compressed file that you downloaded.
3. Open AWS SCT, and choose Global settings from Settings.
4. Choose Drivers.
5. For Amazon Redshift driver path, choose Browse and choose the `redshift-jdbc42-2.1.0.9.jar` file that you extracted.

6. Choose Apply, and then choose OK to close the settings window.

To access AWS services such as Amazon S3 from AWS SCT, you configure an AWS service profile. An AWS service profile is a set of AWS credentials that includes your AWS access key, AWS secret access key, AWS Region, and Amazon S3 bucket.

To create an AWS service profile in AWS SCT

1. Open AWS SCT, and choose Global settings from Settings.
2. Choose AWS service profiles.
3. Choose Add a new AWS service profile.
4. For Profile name, enter a descriptive name for your profile.
5. For AWS access key, enter your AWS access key.
6. For AWS secret key, enter your AWS secret access key. For more information about AWS access keys, see Programmatic access.
7. For Region, choose the AWS Region where you created your Amazon S3 bucket in the previous step.
8. For Amazon S3 bucket folder, choose the Amazon S3 bucket that you created in the previous step.
9. Choose Apply, and then choose OK to close the settings window.

Step 5: Create an AWS SCT Project

After you configure AWS SCT, create a new migration project.

1. In AWS SCT, choose File, then choose New Project.
2. For Project name, enter a descriptive name of your project, and then choose OK.
3. Choose Add source to add a source BigQuery data warehouse to your project, then choose BigQuery, and choose Next.
4. For Connection name, enter a name for your source data warehouse. AWS SCT displays this name in the tree in the left panel.
5. For Key path, choose Browse and then choose the BigQuery service account key file that you created in step 1.
6. Choose Connect to close the dialog box and to connect to your BigQuery data warehouse.
7. Choose Add target to add a target Amazon Redshift database to your project, then choose Amazon Redshift, and choose Next.
8. If you store your database credentials in AWS Secrets Manager, choose your secret and then choose Populate. For more information, see Using Secrets Manager.

If you don’t use Secrets Manager, enter your database credentials manually.

- For Connection name, enter a name for your target data warehouse. AWS SCT displays this name in the tree in the right panel.
- For Server name, enter the server name of the Amazon Redshift cluster that you created in step 2. You can copy the server name as JDBC URL in the General information for your Amazon Redshift cluster. Remove `jdbc:redshift://` from the URL that you copied.
- For Server port, enter 5439.
- For User name, enter the name of the user that you created in step 2.
- For Password, enter the password for the user that you created in step 2.
9. Turn off Use AWS Glue and choose Connect.
10 In the tree in the left panel, choose your BigQuery dataset. In the tree in the right panel, choose your target Amazon Redshift database. Choose Create mapping. You can add multiple mapping rules a single AWS SCT project. For more information about mapping rules, see Creating mapping rules.

11 Choose Main view.

**Step 6: Convert Database Schemas**

After you create a new AWS SCT project, convert your source database schemas and apply converted code to your target database.

1. In the tree in the left panel, choose your source dataset. Open the context (right-click) menu, and choose Convert schema.
2. Choose Yes for the confirmation message. AWS SCT then converts your schema to the target database format.
3. AWS SCT also generates the assessment report. This report includes database objects that require manual conversion. To view this report, choose View, and then choose Assessment report view.
4. On the Action items tab, AWS SCT provides you with the recommended actions for each conversion issue.
5. Check the report and make changes in your source or converted code where necessary. You can optionally save the report as a .CSV or .PDF file for later analysis.
6. Choose Action Items, and review any recommendations that you see.
7. In the tree in the right panel, choose the converted schema. Open the context (right-click) menu, and choose Apply to database to apply the schema scripts to the target Amazon Redshift cluster.

**Step 7: Install and Configure Data Extraction Agents**

AWS SCT uses a data extraction agent to migrate data from BigQuery to Amazon Redshift. The .zip file that you downloaded to install AWS SCT, includes the extraction agent installer file. In this walkthrough, we install the data extraction agent on Windows. However, you can install data extraction agents on Red Hat Enterprise Linux or Ubuntu. For more information, see Installing extraction agents.

To install and configure a data extraction agent

1. Find the aws-schema-conversion-tool-extractor-2.0.1.<version>.msi file in the agents folder. The number of the <version> in the file name depends on the version of AWS SCT that you use. To migrate data from BigQuery to Amazon Redshift, make sure that you use an extraction agent version 665 or higher.
2. Run the file.
3. Choose Next, accept the terms of the License Agreement, and choose Next again.
4. Enter the path to the folder where you want to install the data extraction agent, and choose Next.
5. Choose Install.
6. On Windows, the data extraction agent installer launches the configuration wizard in the command prompt window. On Linux, run the sct-extractor-setup.sh file from the location where you installed the agent.
7. For Listening port, enter 8192. This is the default value. You can choose another port.
8. For Add a source vendor, enter no. You don’t need to configure the data extraction agent to work with your BigQuery data warehouse because you don’t need a driver to connect to BigQuery.
9. For Add the Amazon Redshift driver, enter yes and then enter the path to the Amazon Redshift JDBC driver that you downloaded in Step 4 (p. 248).
10 For Working folder, enter the folder where the data extraction agent can store its data. Choose the project folder and make sure that you don’t need admin rights to write data to this folder.
11. For **Enable SSL communication**, enter no. Then enter yes to confirm your choice. In this walkthrough, we don’t use SSL to connect to databases. If you use SSL, configure the agent.

## Step 8: Run Your Migration Task

After you install and configure the data extraction agent, register it in AWS SCT.

**To register a data extraction agent**

1. In AWS SCT, for **View** choose **Data migration view (other)**, and then choose **Register**.
2. For **Description**, enter a name for your data extraction agent.
3. For **Host name**, enter 0.0.0.0 because you run the data extraction agent on the same computer as AWS SCT. If you install the data extraction agent on another computer, enter the IP address of this computer.
4. For **Port**, enter 8192. If you configured another listening port in the previous step, use the value that you configured.
5. Choose **Register**.

AWS SCT now can use the data extraction agent for data migration tasks.

When you migrate big tables, you can split data into virtual partitions in AWS SCT. Then AWS SCT creates subtasks for each virtual partition.

**To create virtual partitions for your table in AWS SCT**

1. In the tree in the left panel, choose your source table. Open the context (right-click) menu, and choose **Add virtual partitioning**.
2. For **Partition type**, choose **Range**.
3. For **Column name**, choose the column of your table. AWS SCT partitions data based on a range of column values. This partition type creates a WHERE clause, and you provide the range of values for each partition.
4. For **Values**, enter a list of values for the partitioned column.
5. Choose **OK** to create virtual partitions for your table.

Now, you can start the data migration.

**To create and run a migration task**

1. In the tree in the left panel, choose your source table. Open the context (right-click) menu, and choose **Create local task**.
2. For **Task name**, enter a descriptive name for your data migration task.
3. For **Migration mode**, choose **Extract, upload, and copy**.
4. Choose **Advanced**. For **Google CS bucket folder**, enter the name for your Cloud Storage bucket that you created in Step 3 (p. 247).
5. Choose **Amazon S3 settings**. For **Amazon S3 bucket folder**, enter the name of your Amazon S3 bucket that you created in Step 3 (p. 247).
6. Choose **Create** and then choose **Start**.

The AWS SCT data extraction agents migrates data from your BigQuery dataset to Amazon Redshift. You can manage the migration process in AWS SCT. After the data extraction agent completes the migration, check your data in Amazon Redshift. Make sure that all your source data migrated to the new target database.
Step 9: Delete Walkthrough Resources

After you complete this step-by-step guide, make sure that you delete your Amazon Redshift cluster to avoid additional charges.

To delete an Amazon Redshift cluster

1. Sign in to the AWS Management Console and open the Amazon Redshift console at https://console.aws.amazon.com/redshift/.
2. On the navigation menu, choose Clusters.
3. Choose the cluster to delete.
4. For Actions, choose Delete. The Delete cluster page appears.
5. Choose Delete cluster.

Next Steps

After you migrate your BigQuery project to Amazon Redshift, you can explore several other resources:

- Get started with Amazon Redshift. For more information, see Amazon Redshift Getting Started Guide.
- Consider Amazon Redshift Serverless as a migration target. For more information, see Amazon Redshift Serverless.
- Learn more about Amazon Redshift performance optimization.
- You can use AWS SCT command line interface (CLI) to automate database migrations with scripts. For more information, see CLI Reference.

Migrating MySQL-Compatible Databases to AWS

Amazon Web Services (AWS) has several services that allow you to run a MySQL-compatible database on AWS. Amazon Relational Database Service (Amazon RDS) supports MySQL-compatible databases including MySQL, MariaDB, and Amazon Aurora MySQL. Amazon Elastic Compute Cloud (Amazon EC2) provides platforms for running MySQL-compatible databases.

<table>
<thead>
<tr>
<th>Migrating From</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>An RDS MySQL DB instance</td>
<td>You can migrate data directly from an Amazon RDS for MySQL DB snapshot to an Amazon Aurora MySQL DB cluster. For details, see Migrating Data from an Amazon RDS MySQL DB Instance to an Amazon Aurora MySQL DB Cluster (p. 262).</td>
</tr>
<tr>
<td>A MySQL database external to Amazon RDS</td>
<td>If your database supports the InnoDB or MyISAM tablespaces, you have these options for migrating your data to an Amazon Aurora MySQL DB cluster:</td>
</tr>
<tr>
<td></td>
<td>• You can create a dump of your data using the mysqldump utility, and then import that data into an existing Amazon Aurora MySQL DB cluster.</td>
</tr>
<tr>
<td></td>
<td>• You can copy the source files from your database to an Amazon Simple Storage Service (Amazon S3) bucket, and then restore an Amazon Aurora MySQL DB cluster from those</td>
</tr>
</tbody>
</table>
Migrating a MySQL-Compatible Database to Amazon Aurora MySQL

If your database supports the InnoDB or MyISAM tablespaces, you have these options for migrating your data to an Amazon Aurora MySQL DB cluster:

- You can create a dump of your data using the mysqldump utility, and then import that data into an existing Amazon Aurora MySQL DB cluster. For more information, see Migrating MySQL to Amazon Aurora MySQL by Using mysqldump (p. 262).
- You can copy the source files from your database to an Amazon S3 bucket, and then restore an Amazon Aurora MySQL DB cluster from those files. This option can be considerably faster than migrating data using mysqldump. For more information, see Migrating Data from an External MySQL Database to an Amazon Aurora MySQL Using Amazon S3 (p. 253).

Migrating Data from an External MySQL Database to an Amazon Aurora MySQL Using Amazon S3

You can copy the source files from your source MySQL version 5.5, 5.6, or 5.7 database to an Amazon S3 bucket, and then restore an Amazon Aurora MySQL DB cluster from those files.

This option can be considerably faster than migrating data using mysqldump, because using mysqldump replays all of the commands to recreate the schema and data from your source database in your new Amazon Aurora MySQL DB cluster. By copying your source MySQL data files, Amazon Aurora MySQL can immediately use those files as the data for DB cluster.

Note
Restoring an Amazon Aurora MySQL DB cluster from backup files in an Amazon S3 bucket is not supported for the Asia Pacific (Mumbai) region.
Amazon Aurora MySQL does not restore everything from your database. You should save the database schema and values for the following items from your source MySQL or MariaDB database and add them to your restored Amazon Aurora MySQL DB cluster after it has been created.

- User accounts
- Functions
- Stored procedures
- Time zone information. Time zone information is loaded from the local operating system of your Amazon Aurora MySQL DB cluster.

**Prerequisites**

Before you can copy your data to an Amazon S3 bucket and restore a DB cluster from those files, you must do the following:

- Install Percona XtraBackup on your local server.
- Permit Amazon Aurora MySQL to access your Amazon S3 bucket on your behalf.

**Installing Percona XtraBackup**

Amazon Aurora MySQL can restore a DB cluster from files that were created using Percona XtraBackup. You can install Percona XtraBackup from the Percona website at [https://www.percona.com/software/mysql-database/percona-xtrabackup](https://www.percona.com/software/mysql-database/percona-xtrabackup).

**Required Permissions**

To migrate your MySQL data to an Amazon Aurora MySQL DB cluster, several permissions are required:

- The user that is requesting that Amazon RDS create a new cluster from an Amazon S3 bucket must have permission to list the buckets for your AWS account. You grant the user this permission using an AWS Identity and Access Management (IAM) policy.
- Amazon RDS requires permission to act on your behalf to access the Amazon S3 bucket where you store the files used to create your Amazon Aurora MySQL DB cluster. You grant Amazon RDS the required permissions using an IAM service role.
- The user making the request must also have permission to list the IAM roles for your AWS account.
- If the user making the request will create the IAM service role, or will request that Amazon RDS create the IAM service role (by using the console), then the user must have permission to create an IAM role for your AWS account.

For example, the following IAM policy grants a user the minimum required permissions to use the console to both list IAM roles, create an IAM role, and list the S3 buckets for your account.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "s3:ListBucket",
                "s3:ListObjects"
            ]
        }
    ]
}
```
Additionally, for a user to associate an IAM role with an S3 bucket, the IAM user must have the `iam:PassRole` permission for that IAM role. This permission allows an administrator to restrict which IAM roles a user can associate with S3 buckets.

For example, the following IAM policy allows a user to associate the role named `S3Access` with an S3 bucket.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowS3AccessRole",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::123456789012:role/S3Access"
        }
    ]
}
```

### Creating the IAM Service Role

You can have the Amazon RDS Management Console create a role for you by choosing the **Create a New Role** option (shown later in this topic). If you select this option and specify a name for the new role, then Amazon RDS will create the IAM service role required for Amazon RDS to access your Amazon S3 bucket with the name that you supply.

As an alternative, you can manually create the role using the following procedure.

1. Sign in to the AWS Management Console and open the IAM console at [https://console.aws.amazon.com/iam/](https://console.aws.amazon.com/iam/).
2. In the left navigation pane, choose **Roles**.
3. Choose **Create New Role**, specify a value for **Role Name** for the new role, and then choose **Next Step**.
4. Under **AWS Service Roles**, find **Amazon RDS** and choose **Select**.
5. Do not select a policy to attach in the **Attach Policy** step. Instead, choose **Next Step**.
6. Review your role information, and then choose **Create Role**.
7. In the list of roles, choose the name of your newly created role. Choose the **Permissions** tab.
8. Choose **Inline Policies**. Because your new role has no policy attached, you will be prompted to create one. Click the link to create a new policy.
9. On the **Set Permissions** page, choose **Custom Policy** and then choose **Select**.
10. Enter a **Policy Name** such as `S3-bucket-policy`. Add the following code for **Policy Document**, replacing `<bucket name>` with the name of the S3 bucket that you are allowing access to.

    ```json
    {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Sid": "AllowS3AccessRole",
                "Effect": "Allow",
                "Action": "iam:PassRole",
                "Resource": "arn:aws:iam::123456789012:role/S3Access"
            }
        ]
    }
    ```

    As part of the policy document, you can also include a file name prefix. If you specify a prefix, then Amazon Aurora MySQL will create the DB cluster using the files in the S3 bucket that begin with the specified prefix. If you don't specify a prefix, then Amazon Aurora MySQL will create the DB cluster using all of the files in the S3 bucket.

    To specify a prefix, replace `<prefix>` following with the prefix of your file names. Include the asterisk (*) after the prefix. If you don't want to specify a prefix, specify only an asterisk.
11. Choose **Apply Policy**.

### Step 1: Backing Up Files to be Restored as a DB Cluster

To create a backup of your MySQL database files that can be restored from S3 to create an Amazon Aurora MySQL DB cluster, use the Percona Xtrabackup utility (`innobackupex`) to back up your database.

For example, the following command creates a backup of a MySQL database and stores the files in the `/s3-restore/backup` folder.

```
innobackupex --user=myuser --password=<password> --no-timestamp /s3-restore/backup
```

If you want to compress your backup into a single file (which can be split, if needed), you can use the `--stream` option to save your backup in one of the following formats:

- Gzip (.gz)
- tar (.tar)
- Percona xbstream (.xbstream)

For example, the following command creates a backup of your MySQL database split into multiple Gzip files. The parameter values shown are for a small test database; for your scenario, you should determine the parameter values needed.

```
innobackupex --user=myuser --password=<password> --stream=tar /mydata/s3-restore/backup | split -d --bytes=512000 - /mydata/s3-restore/backup3/backup.tar.gz
```

For example, the following command creates a backup of your MySQL database split into multiple tar files.

```
innobackupex --user=myuser --password=<password> --stream=tar \
```
For example, the following command creates a backup of your MySQL database split into multiple xbstream files.

```
innobackupex --stream=xbstream
    /mydata/s3-restore/backup | split -d --bytes=512000
    - /mydata/s3-restore/backup.xbstream
```

Amazon S3 limits the size of a file uploaded to a bucket to 5 terabytes (TB). If the backup data for your database exceeds 5 TB, then you must use the `split` command to split the backup files into multiple files that are each less than 5 TB.

Amazon Aurora MySQL does not support partial backups created using Percona Xtrabackup. You cannot use the `--include`, `--tables-file`, or `--databases` options to create a partial backup when you backup the source files for your database.

For more information, see The innobackupex Script.

Amazon Aurora MySQL consumes your backup files based on the file name. Be sure to name your backup files with the appropriate file extension based on the file format—for example, `.xbstream` for files stored using the Percona xbstream format.

Amazon Aurora MySQL consumes your backup files in alphabetical order as well as natural number order. Always use the `split` option when you issue the `innobackupex` command to ensure that your backup files are written and named in the proper order.

**Step 2: Copying Files to an Amazon S3 Bucket**

Once you have backed up your MySQL database using the Percona Xtrabackup utility, then you can copy your backup files to an Amazon S3 bucket.

For information on creating and uploading a file to an Amazon S3 bucket, see Getting Started with Amazon Simple Storage Service in the Amazon S3 Getting Started Guide.

**Step 3: Restoring an Aurora MySQL DB Cluster from an Amazon S3 Bucket**

You can restore your backup files from your Amazon S3 bucket to a create new Amazon Aurora MySQL DB cluster by using the Amazon RDS console.

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the RDS Dashboard, choose Restore Aurora MySQL DB Cluster from S3.
3. In the Create database by restoring from S3 page, specify the following settings in the following sections:
   a. In the S3 Destination section, specify the following:

<table>
<thead>
<tr>
<th>For This Option</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>S3 Bucket</td>
<td>Select the Amazon S3 bucket where your backup files are stored.</td>
</tr>
</tbody>
</table>
### S3 Prefix (Optional)

<table>
<thead>
<tr>
<th>For This Option</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S3 Bucket Prefix</strong></td>
<td>Specify a file path prefix for the files stored in your Amazon S3 bucket. The S3 Bucket Prefix is optional. If you don’t specify a prefix, then Amazon Aurora MySQL will create the DB cluster using all of the files in the root folder of the S3 bucket. If you specify a prefix, then Amazon Aurora MySQL will create the DB cluster using the files in the S3 bucket where the full path for the file begins with the specified prefix. Amazon Aurora MySQL does not traverse subfolders in your S3 bucket looking for backup files. Only the files from the folder identified by the S3 Bucket Prefix are used. If you store your backup files in a subfolder in your S3 bucket, then you must specify a prefix that identifies the full path to the folder where the files are stored. For example, if you store your backup files in a subfolder of your S3 bucket named backups, and you have multiple sets of backup files, each in its own directory (gzip_backup1, gzip_backup2, and so on), then you would specify a prefix of backups/gzip_backup1 to restore from the files in the gzip_backup1 folder.</td>
</tr>
</tbody>
</table>

b. In the **Engine Options** section, specify the following:

<table>
<thead>
<tr>
<th>For This Option</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Engine Type</strong></td>
<td>Leave Amazon Aurora selected.</td>
</tr>
<tr>
<td><strong>Edition</strong></td>
<td>Leave Amazon Aurora with MySQL compatibility selected.</td>
</tr>
<tr>
<td><strong>Version</strong></td>
<td>Specify the version of the MySQL database that the backup files were created from, for example 5.7. MySQL version 5.6 and 5.7 are supported.</td>
</tr>
</tbody>
</table>

c. In the **IAM role** section, specify the following:

<table>
<thead>
<tr>
<th>For This Option</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IAM Role</strong></td>
<td>Choose the IAM role that you created to authorize Amazon Aurora MySQL to access Amazon S3 on your behalf. If you have not created an IAM role, you can choose <strong>Create a New Role</strong> to create one.</td>
</tr>
</tbody>
</table>

d. In the **Settings** section, specify the following:
### For This Option | Do This
---|---
**DB cluster identifier** | Enter a name for your DB cluster. This identifier will be used in the endpoint address for the primary instance of your DB cluster. The DB instance identifier has the following constraints:
* It must contain from 1 to 63 alphanumeric characters or hyphens.
* Its first character must be a letter.
* It cannot end with a hyphen or contain two consecutive hyphens.
* It must be unique for all DB instances per AWS account, for each region.

**Master Username** | Enter a name using alphanumeric characters that you will use as the master user name to log on to your DB cluster. The default privileges granted to the master user name account include: create, drop, references, event, alter, delete, index, insert, select, update, create temporary tables, lock tables, trigger, create view, show view, alter routine, create routine, execute, create user, process, show databases, grant option.

**Auto generate a password** | Leave unchecked.

**Master Password** | Enter a password that contains from 8 to 41 printable ASCII characters (excluding /,, and @) for your master user password.

**Confirm Password** | Retype the Master Password.

e. In the **DB Instance Class** section, specify the following:

<table>
<thead>
<tr>
<th>For This Option</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DB Instance Class</strong></td>
<td>Select a DB instance class that defines the processing and memory requirements for each instance in the DB cluster. Aurora MySQL supports the db.r3.large, db.r3.xlarge, db.r3.2xlarge, db.r3.4xlarge, and db.r3.8xlarge DB instance classes. For more information about DB instance class options, see the Amazon RDS documentation.</td>
</tr>
</tbody>
</table>

f. In the **Availability & durability** section, specify the following:
## Multi-AZ Deployment

Determine if you want to create Aurora MySQL Replicas in other Availability Zones for failover support. For more information about multiple Availability Zones, see the Amazon RDS documentation.

<table>
<thead>
<tr>
<th>For This Option</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Virtual private cloud (VPC)</strong></td>
<td>Select the VPC that will host the DB cluster. Select <strong>Create a New VPC</strong> to have Amazon RDS create a VPC for you. For more information, see earlier in this topic.</td>
</tr>
<tr>
<td><strong>Subnet group</strong></td>
<td>Select the DB subnet group to use for the DB cluster. Select <strong>Create a New DB Subnet Group</strong> to have Amazon RDS create a DB subnet group for you. For more information, see earlier in this topic.</td>
</tr>
<tr>
<td><strong>Public access</strong></td>
<td>Select <strong>Yes</strong> to give the DB cluster a public IP address; otherwise, select <strong>No</strong>. The instances in your DB cluster can be a mix of both public and private DB instances. For more information about hiding instances from public access, see the Amazon RDS documentation.</td>
</tr>
<tr>
<td><strong>VPC Security Group(s)</strong></td>
<td>Select one or more VPC security groups to secure network access to the DB cluster. Select <strong>Create a New VPC Security Group</strong> to have Amazon RDS create a VPC security group for you. For more information, see earlier in this topic.</td>
</tr>
<tr>
<td><strong>Availability Zone</strong></td>
<td>Determine if you want to specify a particular Availability Zone. For more information about Availability Zones, see the Amazon RDS documentation.</td>
</tr>
<tr>
<td><strong>Database Port</strong></td>
<td>Specify the port that applications and utilities will use to access the database. Aurora MySQL DB clusters default to the default MySQL port, 3306. The firewalls at some companies block connections to the default MySQL port. If your company firewall blocks the default port, choose another port for the new DB cluster.</td>
</tr>
</tbody>
</table>

### h. Database authentication

In the **Database authentication** section, specify the following:

<table>
<thead>
<tr>
<th>For This Option</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Database Authentication</strong></td>
<td>Leave <strong>Password authentication</strong> selected.</td>
</tr>
</tbody>
</table>
i. In the **Additional configuration** section, specify the following:

<table>
<thead>
<tr>
<th>For This Option</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Database Name</td>
<td>Enter a name for your database of up to 8 alphanumeric characters. If you don’t provide a name, Amazon RDS will not create a database on the DB cluster you are creating.</td>
</tr>
<tr>
<td>DB cluster parameter Group</td>
<td>Select a parameter group for the cluster. Aurora MySQL has a default parameter group you can use, or you can create your own parameter group. For more information about parameter groups, see the Amazon RDS documentation.</td>
</tr>
<tr>
<td>DB parameter Group</td>
<td>Select a parameter group for the database.</td>
</tr>
<tr>
<td>Option Group</td>
<td>Select an option group. Aurora MySQL has a default option group you can use, or you can create your own option group. For more information about option groups, see the Amazon RDS documentation.</td>
</tr>
<tr>
<td>Failover Priority</td>
<td>Choose a failover priority for the instance. If you don’t select a value, the default is <strong>tier-1</strong>. This priority determines the order in which Aurora MySQL Replicas are promoted when recovering from a primary instance failure. For more information, see Amazon RDS documentation.</td>
</tr>
<tr>
<td>Backup Retention Period</td>
<td>Select the length of time, from 1 to 35 days, that Aurora MySQL will retain backup copies of the database. Backup copies can be used for point-in-time restores (PITR) of your database, timed down to the second.</td>
</tr>
<tr>
<td>Copy tags to snapshots</td>
<td>Leave checked.</td>
</tr>
<tr>
<td>Enable Encryption</td>
<td>Check the box to enable encryption at rest for this DB cluster. Leave <strong>AWS KMS Key</strong> set to <strong>(default) aws/rds</strong>. For more information, see Amazon RDS documentation.</td>
</tr>
<tr>
<td>Backtrack</td>
<td>Leave unchecked.</td>
</tr>
<tr>
<td>Enable Performance insights</td>
<td>Leave checked. Leave <strong>Retention Period</strong> and <strong>AWS KMS Key</strong> as they are.</td>
</tr>
<tr>
<td>Enable Enhanced Monitoring</td>
<td>Choose <strong>Yes</strong> to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see Amazon RDS documentation.</td>
</tr>
<tr>
<td>Granularity</td>
<td>This option is only available if Enable Enhanced Monitoring is set to <strong>Yes</strong>. Set the interval, in seconds, between times at which metrics are collected for your DB cluster.</td>
</tr>
</tbody>
</table>
For This Option | Do This
---|---
Monitoring role | Leave as default.
Log exports | Leave unchecked.
Enable auto Minor Version Upgrade | Check this box if you want to enable your Aurora MySQL DB cluster to receive minor MySQL DB engine version upgrades automatically when they become available.
The Auto Minor Version Upgrade option only applies to upgrades to MySQL minor engine versions for your Amazon Aurora MySQL DB cluster. It doesn’t apply to regular patches applied to maintain system stability.

Maintenance Window | Select the weekly time range during which system maintenance can occur.
Enable deletion protection | Leave unchecked.

4. Choose Launch DB Instance to launch your Aurora MySQL DB instance, and then choose Close to close the wizard.

On the Amazon RDS console, the new DB instance appears in the list of DB instances. The DB instance has a status of creating until the DB instance is created and ready for use. When the state changes to available, you can connect to the primary instance for your DB cluster. Depending on the DB instance class and store allocated, it can take several minutes for the new instance to be available.

To view the newly created cluster, choose the Clusters view in the Amazon RDS console. For more information, see Amazon RDS documentation.

Note the port and the endpoint of the cluster. Use the endpoint and port of the cluster in your JDBC and ODBC connection strings for any application that performs write or read operations.

### Migrating MySQL to Amazon Aurora MySQL by Using mysqldump

You can create a dump of your data using the mysqldump utility, and then import that data into an existing Amazon Aurora MySQL DB cluster.

Because Amazon Aurora MySQL is a MySQL-compatible database, you can use the mysqldump utility to copy data from your MySQL or MariaDB database to an existing Amazon Aurora MySQL DB cluster.

### Migrating Data from an Amazon RDS MySQL DB Instance to an Amazon Aurora MySQL DB Cluster

You can migrate (copy) data to an Amazon Aurora MySQL DB cluster from an Amazon RDS snapshot, as described following.

**Note**
Because Amazon Aurora MySQL is compatible with MySQL, you can migrate data from your MySQL database by setting up replication between your MySQL database, and an Amazon Aurora MySQL DB cluster. We recommend that your MySQL database run MySQL version 5.5 or later.
Migrating an Amazon RDS for MySQL Snapshot to Aurora MySQL

You can migrate a DB snapshot of an Amazon RDS MySQL DB instance to create an Aurora MySQL DB cluster. The new DB cluster is populated with the data from the original Amazon RDS MySQL DB instance. The DB snapshot must have been made from an Amazon RDS DB instance running MySQL 5.6.

You can migrate either a manual or automated DB snapshot. After the DB cluster is created, you can then create optional Aurora MySQL Replicas.

The general steps you must take are as follows:

1. Determine the amount of space to provision for your Amazon Aurora MySQL DB cluster. For more information, see Amazon RDS documentation.
2. Use the console to create the snapshot in the region where the Amazon RDS MySQL 5.6 instance is located.
3. If the DB snapshot is not in the region as your DB cluster, use the Amazon RDS console to copy the DB snapshot to that region. For information about copying a DB snapshot, see the Amazon RDS documentation.
4. Use the console to migrate the DB snapshot and create an Amazon Aurora MySQL DB cluster with the same databases as the original DB instance of MySQL 5.6.

**Warning**

Amazon RDS limits each AWS account to one snapshot copy into each region at a time.

**How Much Space Do I Need?**

When you migrate a snapshot of a MySQL DB instance into an Aurora MySQL DB cluster, Aurora MySQL uses an Amazon Elastic Block Store (Amazon EBS) volume to format the data from the snapshot before migrating it. In some cases, additional space is needed to format the data for migration. When migrating data into your DB cluster, observe the following guidelines and limitations:

- Although Amazon Aurora MySQL supports storage up to 64 TB in size, the process of migrating a snapshot into an Aurora MySQL DB cluster is limited by the size of the EBS volume of the snapshot. Thus, the maximum size for a snapshot that you can migrate is 6 TB.
- Tables that are not MyISAM tables and are not compressed can be up to 6 TB in size. If you have MyISAM tables, then Aurora MySQL must use additional space in the volume to convert the tables to be compatible with Aurora MySQL. If you have compressed tables, then Aurora MySQL must use additional space in the volume to expand these tables before storing them on the Aurora MySQL cluster volume. Because of this additional space requirement, you should ensure that none of the MyISAM and compressed tables being migrated from your MySQL DB instance exceeds 3 TB in size.

**Reducing the Amount of Space Required to Migrate Data into Amazon Aurora MySQL**

You might want to modify your database schema prior to migrating it into Amazon Aurora MySQL. Such modification can be helpful in the following cases:

- You want to speed up the migration process.
- You are unsure of how much space you need to provision.
- You have attempted to migrate your data and the migration has failed due to a lack of provisioned space.
You can make the following changes to improve the process of migrating a database into Amazon Aurora MySQL.

**Important**
Be sure to perform these updates on a new DB instance restored from a snapshot of a production database, rather than on a production instance. You can then migrate the data from the snapshot of your new DB instance into your Amazon Aurora MySQL DB cluster to avoid any service interruptions on your production database.

<table>
<thead>
<tr>
<th>Table Type</th>
<th>Limitation or Guideline</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyISAM tables</td>
<td>Amazon Aurora MySQL supports InnoDB tables only. If you have MyISAM tables in your database, then those tables must be converted before being migrated into Amazon Aurora MySQL. The conversion process requires additional space for the MyISAM to InnoDB conversion during the migration procedure. To reduce your chances of running out of space or to speed up the migration process, convert all of your MyISAM tables to InnoDB tables before migrating them. The size of the resulting InnoDB table is equivalent to the size required by Amazon Aurora MySQL for that table. To convert a MyISAM table to InnoDB, run the following command: alter table &lt;schema&gt;.&lt;table_name&gt; engine=innodb, algorithm=copy;</td>
</tr>
<tr>
<td>Compressed tables</td>
<td>Amazon Aurora MySQL does not support compressed tables (that is, tables created with ROW_FORMAT=COMPRESSED). To reduce your chances of running out of space or to speed up the migration process, expand your compressed tables by setting ROW_FORMAT to DEFAULT, COMPACT, DYNAMIC, or REDUNDANT. For more information, see <a href="https://dev.mysql.com/doc/refman/5.6/en/innodb-row-format.html">https://dev.mysql.com/doc/refman/5.6/en/innodb-row-format.html</a>.</td>
</tr>
</tbody>
</table>

You can use the following SQL script on your existing MySQL DB instance to list the tables in your database that are MyISAM tables or compressed tables.

```sql
-- This script examines a MySQL database for conditions that will block migrating the database into an Amazon Aurora MySQL DB.
-- It needs to be run from an account that has read permission for the INFORMATION_SCHEMA database.
-- Verify that this is a supported version of MySQL.

select msg as '==> Checking current version of MySQL.'
from
( select
    'This script should be run on MySQL version 5.6.' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
    cast(substring_index(substring_index(version(), '.', 2), '.', -1) as unsigned) as version
) as supported_version
where
    supported_version < 50600
```
Database Migration Guide Step-by-Step Walkthroughs

Migrating Data from an Amazon RDS MySQL DB Instance to an Amazon Aurora MySQL DB Cluster

```sql

as unsigned)
as major_minor
) as T
where major_minor <> 506;

-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
    round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
    ENGINE <> 'InnoDB'
and
    (  -- User tables
        TABLE_SCHEMA not in ('mysql', 'performance_schema', 'information_schema')
    or
    -- Non-standard system tables
    (        TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
        (            'columns_priv', 'db', 'event', 'func', 'general_log',
            'help_category', 'help_keyword', 'help_relation',
            'help_topic', 'host', 'ndb_binlog_index', 'plugin',
            'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
            'tables_priv', 'time_zone', 'time_zone_leap_second',
            'time_zone_name', 'time_zone_transition',
            'time_zone_transition_type', 'user',
            'general_log_backup','slow_log_backup'
        )
    )
)  
or
    (  -- Compressed tables
        ROW_FORMAT = 'Compressed'
    )
);

The script produces output similar to the output in the following example. The example shows two tables that must be converted from MyISAM to InnoDB. The output also includes the approximate size of each table in megabytes (MB).

<table>
<thead>
<tr>
<th><code>==&gt; MyISAM or Compressed Tables</code></th>
<th>Approx size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>test.name_table</td>
<td>2102.25</td>
</tr>
<tr>
<td>test.my_table</td>
<td>65.25</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------</td>
</tr>
</tbody>
</table>

2 rows in set (0.01 sec)

Migrating a DB Snapshot by Using the Console

You can migrate a DB snapshot of an Amazon RDS for MySQL DB instance to create an Aurora MySQL DB cluster. The new DB cluster will be populated with the data from the original Amazon RDS for MySQL DB instance. The DB snapshot must have been made from an Amazon RDS DB instance running MySQL 5.6 and must not be encrypted. For information about creating a DB snapshot, see the Amazon RDS documentation.

If the DB snapshot is not in the AWS Region where you want to locate your data, use the Amazon RDS console to copy the DB snapshot to that region. For information about copying a DB snapshot, see the Amazon RDS documentation.
When you migrate the DB snapshot by using the console, the console takes the actions necessary to create both the DB cluster and the primary instance.

You can also choose for your new Aurora MySQL DB cluster to be encrypted "at rest" using an AWS Key Management Service (AWS KMS) encryption key. This option is available only for unencrypted DB snapshots.

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose Snapshots.
3. On the Snapshots page, choose the snapshot that you want to migrate into an Aurora MySQL DB cluster.
4. Choose Migrate Database.
5. Set the following values on the Migrate Database page:
   - **DB Instance Class**: Select a DB instance class that has the required storage and capacity for your database, for example `db.r3.large`. Aurora MySQL cluster volumes automatically grow as the amount of data in your database increases, up to a maximum size of 64 terabytes (TB). So you only need to select a DB instance class that meets your current storage requirements.
   - **DB Instance Identifier**: Enter a name for the DB cluster that is unique for your account in the region you selected. This identifier is used in the endpoint addresses for the instances in your DB cluster. You might choose to add some intelligence to the name, such as including the region and DB engine you selected, for example `aurora-cluster1`.

   The DB instance identifier has the following constraints:
   - It must contain from 1 to 63 alphanumeric characters or hyphens.
   - Its first character must be a letter.
- It cannot end with a hyphen or contain two consecutive hyphens.
- It must be unique for all DB instances per AWS account, for each AWS Region.

- **VPC:** If you have an existing VPC, then you can use that VPC with your Amazon Aurora MySQL DB cluster by selecting your VPC identifier, for example vpc-a464d1c1. For information on using an existing VPC, see the Amazon RDS documentation.

  Otherwise, you can choose to have Amazon RDS create a VPC for you by selecting **Create a new VPC.**

- **Subnet Group:** If you have an existing subnet group, then you can use that subnet group with your Amazon Aurora MySQL DB cluster by selecting your subnet group identifier, for example gs-subnet-group1.

  Otherwise, you can choose to have Amazon RDS create a subnet group for you by selecting **Create a new subnet group.**

- **Publicly Accessible:** Select **No** to specify that instances in your DB cluster can only be accessed by resources inside of your VPC. Select **Yes** to specify that instances in your DB cluster can be accessed by resources on the public network. The default is **Yes.**

  **Note**
  Your production DB cluster might not need to be in a public subnet, because only your application servers will require access to your DB cluster. If your DB cluster doesn't need to be in a public subnet, set **Publicly Accessible** to **No.**

- **Availability Zone:** Select the Availability Zone to host the primary instance for your Aurora MySQL DB cluster. To have Amazon RDS select an Availability Zone for you, select **No Preference.**

- **Database Port:** Enter the default port to be used when connecting to instances in the DB cluster. The default is 3306.

  **Note**
  You might be behind a corporate firewall that doesn't allow access to default ports such as the MySQL default port, 3306. In this case, provide a port value that your corporate firewall allows. Remember that port value later when you connect to the Aurora MySQL DB cluster.

- **Enable Encryption:** Choose **Yes** for your new Aurora MySQL DB cluster to be encrypted "at rest." If you choose **Yes,** you will be required to choose an AWS KMS encryption key as the KMS key value.

- **Auto Minor Version Upgrade:** Select **Yes** if you want to enable your Aurora MySQL DB cluster to receive minor MySQL DB engine version upgrades automatically when they become available.

  The **Auto Minor Version Upgrade** option only applies to upgrades to MySQL minor engine versions for your Amazon Aurora MySQL DB cluster. It doesn't apply to regular patches applied to maintain system stability.
Database Migration Guide Step-by-Step Walkthroughs
Migrating Data from an Amazon RDS MySQL DB Instance to an Amazon Aurora MySQL DB Cluster

### Instance Specifications

<table>
<thead>
<tr>
<th>Migrate to DB Engine</th>
<th>aurora</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB Instance Class</td>
<td>- Select One -</td>
</tr>
</tbody>
</table>

### Settings

<table>
<thead>
<tr>
<th>DB Snapshot ID</th>
<th>rds-&lt;instance-id&gt;-2016-02-22-07-42</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB Instance Identifier*</td>
<td>migrated-instance-name</td>
</tr>
</tbody>
</table>

### Network & Security

This instance will be created with the new Certificate Authority rds-ca-2015. If you are using SSL to connect to this instance, you should use the new certificate bundle. Learn more [here](#).

<table>
<thead>
<tr>
<th>VPC*</th>
<th>Default VPC (vpc-&lt;instance-id&gt;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subnet Group</td>
<td>default</td>
</tr>
<tr>
<td>Publicly Accessible</td>
<td>Yes</td>
</tr>
<tr>
<td>Availability Zone</td>
<td>No Preference</td>
</tr>
</tbody>
</table>

### Database Options

<table>
<thead>
<tr>
<th>Database Port</th>
<th>3306</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Encryption</td>
<td>No</td>
</tr>
</tbody>
</table>

### Maintenance

<table>
<thead>
<tr>
<th>Auto Minor Version Upgrade</th>
<th>Yes</th>
</tr>
</thead>
</table>
6. Choose **Migrate** to migrate your DB snapshot.

7. Choose **Instances**, and then choose the arrow icon to show the DB cluster details and monitor the progress of the migration. On the details page, you will find the cluster endpoint used to connect to the primary instance of the DB cluster. For more information on connecting to an Amazon Aurora MySQL DB cluster, see the [Amazon RDS documentation](#).

---

## Migrating a MariaDB Database to Amazon RDS for MySQL or Amazon Aurora MySQL

You can migrate data from existing on-premises MariaDB or Amazon RDS for MariaDB to Amazon Aurora MySQL using Database Migration Service. Amazon Aurora is a MySQL and PostgreSQL-compatible relational database built for the cloud. Amazon Aurora features a distributed, fault-tolerant, self-healing storage system that auto-scales up to 64 TB per database instance. It delivers high performance and availability with up to 15 low-latency read replicas, point-in-time recovery, and continuous backup to Amazon S3, and replication across three Availability Zones (AZs).

Some key features offered by Aurora MySQL are the following:

- High throughput with low latency
- Push-button compute scaling
- Storage autoscaling
- Custom database endpoints
- Parallel queries for faster analytics

In the following sections, we demonstrate migration from MariaDB as a source database to an Aurora MySQL database as a target using AWS DMS. At a high level, the steps involved in this migration are:

- Provision MariaDB as a source DB instance and load the data
- Provision Aurora Mysql as target DB instance
- Provision DMS replication instance and create DMS endpoints
- Create DMS task, migrate data and perform validation
Set up MariaDB as a source database

To provision MariaDB as a source database, download Mariadb_CF.zip with the YAML template. This AWS CloudFormation template creates an Amazon RDS for MariaDB instance with the required parameters.

1. On the AWS Management Console, under Services, choose CloudFormation.
2. Choose Create stack, and then choose With new resources (standard).
3. For Specify template, choose Upload a template file.
4. Select Choose file.
5. Choose the Mariadb_CF.yaml file, and then choose Next.
6. On the Specify stack details page, edit the predefined values as needed, and then choose Next:
   - Stack name — Enter a name for the stack.
   - CIDR — Enter the CIDR IP range to access the instance.
   - DBAllocatedStorage — Enter the database storage size in GB. The default is 20 GB.
   - DBBackupRetentionPeriod — The number of days to retain backups.
   - DBInstanceClass — Enter the instance type of the database server.
   - DBMonitoringInterval — Interval to publish database logs to Amazon CloudWatch.
   - DBSubnetGroup — Enter the DB subnet group name. For more information, see Create a DB subnet group in the Amazon RDS User Guide.
   - MariaDBEngine — Enter the MariaDB engine version.
   - DBMasterPassword — Enter the master password for the DB instance.
   - DBMasterUsername — Enter the master user name for the DB instance.
   - PreferredBackupWindow — Enter the daily time range in UTC during which you want to create automated backups.
   - PreferredMaintenanceWindow — Enter the weekly time range in UTC during which system maintenance can occur.
   - RDSDBName — Enter the name of the database.
   - RDSMultiAZ — Choose true to use Amazon RDS Multi-AZ for this instance. The default value for this option is false. For more information, see Multi-AZ deployments for high availability in the Amazon Relational Database Service User Guide.
   - VPCID — Enter the VPC to launch your DB instance. For more information, see Working with a DB instance in a VPC in the Amazon RDS User Guide.

Make sure that you entered the stack name, DB subnet group name, user name, password, database name, and VPC ID.

7. On the Configure stack options page, for Tags, specify any optional tags, and then choose Next.
8. On the Review page, select I acknowledge that AWS CloudFormation might create IAM resources, and then choose Next.

9. Choose Create stack.

After the Amazon RDS for MariaDB instance is created, log in to MariaDB and run the following statements to create webdb_user, a superuser that connects to a DMS instance for migration, and grant necessary privileges.

```
CREATE USER 'webdb_user'@'%' IDENTIFIED BY '******';
GRANT ALL ON migrate.* TO 'webdb_user'@'%' with grant option;
grant REPLICATION SLAVE ON *.* TO webdb_user;
grant REPLICATION CLIENT ON *.* TO webdb_user;
```

In this walkthrough, we created a database called migration and few sample tables, along with stored procedures, triggers, functions, and so on. The following query provides the list of tables in migration database:

```
MariaDB [(none)]> use migration
Database changed
MariaDB [migration]> show tables;
+---------------------+
| Tables_in_migration |
+---------------------+
| animal_count        |
| animals             |
| contacts            |
| seat_type           |
| sport_location      |
| sport_team          |
| sport_type          |
+---------------------+
7 rows in set (0.000 sec)
```

The following query returns a list of secondary indexes.

```
MariaDB [migration]> SELECT DISTINCT TABLE_NAME, INDEX_NAME,NON_UNIQUE
FROM INFORMATION_SCHEMA.STATISTICS
WHERE TABLE_SCHEMA = 'migration' and INDEX_NAME <> 'PRIMARY';
+------------+-------------------+------------+
| TABLE_NAME | INDEX_NAME        | NON_UNIQUE |
+------------+-------------------+------------+
| sport_location | city_id_sport_loc |          1 |
| sport_team | sport_team_u      |          0 |
| sport_team | home_field_fk     |          1 |
+------------+-------------------+------------+
3 rows in set (0.000 sec)
```

The following query returns a list of triggers.

```
MariaDB [migration]> select TRIGGER_SCHEMA,TRIGGER_NAME
FROM information_schema.triggers
where TRIGGER_SCHEMA='migration';
+------------+-----------------------+
| TRIGGER_SCHEMA | TRIGGER_NAME          |
+------------+-----------------------+
| migration  | increment_animal      |
| migration  | contacts_after_update |
+------------+-----------------------+
```

271
The following query returns a list of procedures and functions.

```
MariaDB [(none)]> select routine_schema as database_name,
            | routine_name,            |
            | routine_type as type,    |
            | data_type as return_type |
            | from information_schema.routines |
            | where routine_schema not in ('sys', 'information_schema', |
            | 'mysql', 'performance_schema');
```

<table>
<thead>
<tr>
<th>database_name</th>
<th>routine_name</th>
<th>type</th>
<th>return_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>migration</td>
<td>CalcValue</td>
<td>FUNCTION</td>
<td>int</td>
</tr>
<tr>
<td>migration</td>
<td>loadMLBPlayers</td>
<td>PROCEDURE</td>
<td></td>
</tr>
<tr>
<td>migration</td>
<td>loadNFLPlayers</td>
<td>PROCEDURE</td>
<td></td>
</tr>
</tbody>
</table>

3 rows in set (0.000 sec)

After all the data is loaded, use `mysqldump` to back up the database metadata. The `mysqldump` utility
to dump one or more databases for backup or transfer to another database server. The dump typically
contains SQL statements to create the table, populate it, or both. You can also use `mysqldump` to
generate files in comma-separated value (CSV), other delimited text, or XML format.

Use the following command exports tables and index definitions:

```
$ mysqldump --no-data --no-create-db --single_transaction -u root -p migration --skip-triggers > mysql_tables_indexes.sql
```

Use following command to exports routines (stored procedures, functions, and triggers) into the
routines.sql file:

```
$ mysqldump -u root --routines --no-create-info --no-data --no-create-db --skip-opt -p migration > routines.sql
```

The `mysqldump` utility doesn’t provide the option to remove a DEFINER statement. Some MySQL
clients provide the option to ignore the definer when creating a logical backup, but this isn’t the default
behavior. Use the following command in a UNIX or Linux environment to remove the DEFINER from
routines.sql:

```
$ sed -i -e 's/DEFINER=`root`@`localhost`/DEFINER=`master`@`%`/g' routines.sql
```

We now have a backup of MariaDB, in two .sql files (mysql_tables_indexes.sql and
routines.sql). We will use these files to load the table definition into an Aurora MySQL database.

After backups are completed into two .sql files (mysql_tables_indexes.sql, routines.sql), use
these files to load the table definition into the Aurora MySQL database.

**Set up Aurora MySQL as a target database**

To provision Aurora MySQL as a target database, download the AuroraMysql_CF.yaml template. This
template creates an Aurora MySQL database with required parameters.

1. On the AWS Management Console, under Services, choose CloudFormation.
2. Choose Create stack, and then choose With new resources (standard).
3. For **Specify template**, choose **Upload a template file**.
4. Select **Choose file**.
5. Choose the AuroraMySQL.yaml file.
6. Choose **Next**.
7. On the **Specify stack details** page, edit the predefined values as needed, and then choose **Next**:
   - **Stack name** — Enter a name for the stack.
   - **CIDR** — Enter the CIDR IP range to access the instance.
   - **DBBackupRetentionPeriod** — The number of days for backup retention.
   - **DBInstanceClass** — Enter the instance type of the database server.
   - **DBMasterPassword** — Enter the master password for the DB instance.
   - **DBMasterUsername** — Enter the master user name for the DB instance.
   - **DBName** — Enter the name of the database.
   - **DBSubnetGroup** — Enter the DB subnet group.
   - **Engine** — Enter the Aurora engine version; the default is 5.7.mysql-aurora.2.03.4.
   - **PreferredBackupWindow** — Enter the daily time range in UTC during which you want to create automated backups.
   - **PreferredMaintenanceWindow** — Enter the weekly time range in UTC during which system maintenance can occur.
   - **VPCID** — Enter the ID for the VPC to launch your DB instance in.
8. On the **Configure stack options** page, for **Tags**, specify any optional tags, and then choose **Next**.
9. On the **Review** page, choose **Next**.
10. Choose **Create stack**.

After the Aurora MySQL database is created, log in to the Aurora MySQL instance:

```
$ mysql -h mysqltrg-instance-1.xxxxxxxxx.us-east-1.rds.amazonaws.com -u master -p migration -P 3306
MySQL [(none)]> show databases;
+-------------------------------+
| Database                       |
+-------------------------------+
| information_schema            |
| awsdms_control                |
| mysql                         |
| performance_schema            |
| source                        |
| tmp                           |
| webdb                         |
+-------------------------------+
7 rows in set (0.001 sec)
MySQL [(none)]> create database migration;
Query OK, 1 row affected (0.016 sec)
MySQL [(none)]> use migration;
Database changed
MySQL [migration]> show tables;
Empty set (0.001 sec)
```

Use mysql_tables_indexes.sql to create table and index structures in Aurora MySQL.

```
$ mysql -h mysqltrg-instance-1.xxxxxxxxx.us-east-1.rds.amazonaws.com -u master -p migration -P 3306 < mysql_tables_indexes.sql
```
Set up an AWS DMS replication instance

To provision an AWS DMS replication instance, download the DMS_CF.yaml template.

1. On the AWS Management Console, under Services, choose CloudFormation.
2. Choose Create stack.
3. For Specify template, choose Upload a template file.
4. Select Choose File.
5. Choose the DMS_CF.yaml file.
6. Choose Next.
7. On the Specify Stack Details page, edit the predefined values as needed, and then choose Next:
   - Stack name — Enter a name for the stack.
   - AllocatedStorageSize — Enter the storage size in GB. The default is 200 GB.
   - DMSReplicationSubnetGroup — Enter the subnet group for DMS replication.
   - DMSSecurityGroup — Enter the security group for DMS replication.
   - InstanceType — Enter the instance type.
   - SourceDBPort — Enter the source database port.
   - SourceDatabaseName — Enter the source database name.
   - SourceServerName — Enter the IP address of the source database server.
   - SourceUsername — Enter the source database user name.
   - SourcePassword — Enter the source database password.
   - TargetDBPort — Enter the target database port.
   - TargetDatabaseName — Enter the target database name.
   - TargetServerName — Enter the IP address of the target database server.
   - TargetUsername — Enter the target database user name.
   - TargetPassword — Enter the target database password.
8. On the Configure stack options page, for Tags, specify any optional tags, and then choose Next.
9. On the Review page, choose I acknowledge that AWS CloudFormation might create IAM resources.
10. Choose Create Stack.

This AWS CloudFormation template creates a replication instance named mariadb-mysql. This replication instance has a source endpoint named maria-on-prem and a target endpoint named mysqltrg-rds. This target endpoint has extra connection attributes to disable foreign key constraint checks during the AWS DMS replication, as shown following.

```
ExtraConnectionAttributes : "initstmt=SET FOREIGN_KEY_CHECKS=0;parallelLoadThreads=1"
```

Test the endpoints

1. On the navigation pane, choose Endpoints.
2. Choose the source endpoint name (maria-on-prem) and do the following:
   a. Choose Test connections.
b. Choose the replication instance to test (mariadb-mysql).
c. Choose Run Test and wait for the status to be successful.

3. On the navigation pane, choose Endpoints.

4. Choose the target endpoint name (mysqltrg-rds) and do the following:
   a. Choose Test Connections.
   b. Choose the replication instance to test (mariadb-mysql).
   c. Choose Run Test and wait for the status to be successful.

Note
If Run Test returns a status other than successful, the reason for the failure is displayed. Make sure that you resolve the issue before proceeding further.

Create a migration task

We've now verified that the replication instance can connect to both the source and target endpoints. The next step is to create a database migration task.

1. On the navigation pane, choose Database Migration Tasks.
2. Choose Create Task. Provide the specified values for the following, and then choose Next:
   • Task identifier – maria-mysql
   • Replication instance – Choose the replication instance, mariadb-mysql.
   • Source database endpoint – Choose the source database, maria-on-prem.
   • Target database endpoint – Choose the target database, mysqltrg-rds.
   • Migration Type – Choose Migrate existing data and replicate ongoing changes for CDC, or Migrate existing data for full load.
3. For Task settings, choose the following settings:
   • Target table preparation mode – Do nothing
   • Stop task after full load completes – Don't stop
   • Include LOB columns in replication – Limited LOB mode
   • Maximum LOB size (KB) – 32
   • Enable validation
   • Enable CloudWatch logs
4. For Table mappings, choose the following settings:
   • Schema – Choose migration (assuming the schema and database to be migrated appear correctly).
   • Table name – Enter the table name, or % to specify all the tables in the database.
   • Action – Enter Include to include specific tables, or Exclude to exclude specific tables.
5. Choose Create Task.

Your new AWS DMS migration task reads the data from the tables in the MariaDB source and migrates your data to the Aurora MySQL target.

Validate the migration

AWS DMS performs data validation to confirm that your data successfully migrated the source database to the target. You can check the Table statistics page to determine the DML changes that occurred after the AWS DMS task started. During data validation, AWS DMS compares each row in the source with its corresponding row at the target, and verifies that those rows contain the same data. To accomplish this, AWS DMS issues the appropriate queries to retrieve the data.
After your data is loaded successfully, you can select your task on the AWS DMS page and choose Table statistics to show statistics about your migration. The following screenshot shows the Table statistics page and its relevant entries.

The following screenshot shows the table statics page and its relevant entries.

AWS DMS can validate the data between source and target engines. The Validation state column helps us to validate the data migration. This ensures that your data was migrated accurately from the source to the target.

**Cut over**

After the data validation is complete and any problems resolved, you can load the database triggers, functions, and procedures.

To do this, use the routines.sql file generated from MariaDB to create the necessary routines in Aurora MySQL. The following statement loads all procedures, functions, and triggers into the Aurora MySQL database.

```
$ mysql -h mysqltrg-instance-1.xxxxxxxxx.us-east-1.rds.amazonaws.com  -u master -p migration -P 3306 < routines.sql
```

After the routines are loaded, connect to the Aurora MySQL database to validate as shown following.

```
$ mysql -h mysqltrg-instance-1.xxxxxxxxx.us-east-1.rds.amazonaws.com  -u master -p migration -P 3306
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MySQL connection id is 957
Server version: 5.6.10 MySQL Community Server (GPL)
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
Enter 'help;' or '\h' for help. Enter '\c' to clear the current input statement.

MySQL [migration]> select routine_schema as database_name,             
->            routine_name,                                        
->            routine_type as type,                               
->            data_type as return_type                           
->            from information_schema.routines                    
->            where routine_schema not in ('sys', 'information_schema', 'mysql', 'performance_schema');
+---------------+----------------+-----------+-------------+                     
276
```
Migrating from MongoDB to Amazon DocumentDB

Use the following tutorial to guide you through the process of migrating from MongoDB to Amazon DocumentDB (with MongoDB compatibility). In this tutorial, you do the following:

- Install MongoDB on an Amazon EC2 instance.
- Populate MongoDB with sample data.
- Create an AWS DMS replication instance, a source endpoint (for MongoDB), and a target endpoint (for Amazon DocumentDB).
- Run an AWS DMS task to migrate the data from the source endpoint to the target endpoint.

**Important**
Before you begin, make sure to launch an Amazon DocumentDB cluster in your default virtual private cloud (VPC). For more information, see Getting started in the Amazon DocumentDB Developer Guide.

Topics
- Launch an Amazon EC2 instance (p. 277)
- Install and configure MongoDB community edition (p. 278)
- Create an AWS DMS replication instance (p. 279)
- Create source and target endpoints (p. 280)
- Create and run a migration task (p. 282)

Launch an Amazon EC2 instance

For this tutorial, you launch an Amazon EC2 instance into your default VPC.

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose Launch Instance, and do the following:
   a. On the Choose an Amazon Machine Image (AMI) page, at the top of the list of AMIs, go to Amazon Linux AMI and choose Select.
b. On the **Choose an Instance Type** page, at the top of the list of instance types, choose **t2.micro**. Then choose **Next: Configure Instance Details**.

c. On the **Configure Instance Details** page, for **Network**, choose your default VPC. Then choose **Next: Add Storage**.

d. On the **Add Storage** page, skip this step by choosing **Next: Add Tags**.

e. On the **Add Tags** page, skip this step by choosing **Next: Configure Security Group**.

f. On the **Configure Security Group** page, do the following:

   i. Choose **Select an existing security group**.

   ii. In the list of security groups, choose **default**. Doing this chooses the default security group for your VPC. By default, the security group accepts inbound Secure Shell (SSH) connections on TPC port 22. If this isn’t the case for your VPC, add this rule; for more information, see **What is Amazon VPC?** in the Amazon **VPC User Guide**.

   iii. Choose **Next: Review and Launch**.

   g. Review the information, and choose **Launch**.

3. In the **Select an existing key pair or create a new key pair** window, do one of the following:

   - If you don’t have an Amazon EC2 key pair, choose **Create a new key pair** and follow the instructions. You are asked to download a private key file (.pem file). You need this file later when you log in to your Amazon EC2 instance.

   - If you already have an Amazon EC2 key pair, for **Select a key pair** choose your key pair from the list. You must already have the private key file (.pem file) available in order to log in to your Amazon EC2 instance.

4. After you configure your key pair, choose **Launch Instances**.

   In the console navigation pane, choose **EC2 Dashboard**, and then choose the instance that you launched. In the lower pane, on the **Description** tab, find the **Public DNS** location for your instance, for example: `ec2-11-22-33-44.us-west-2.compute.amazonaws.com`.

   It takes a few minutes for your Amazon EC2 instance to become available.

5. Use the `ssh` command to log in to your Amazon EC2 instance, as in the following example.

   ```
   chmod 400 my-keypair.pem
   ssh -i my-keypair.pem ec2-user@public-dns-name
   ```

   Specify your private key file (.pem file) and the public DNS name of your EC2 instance. The login ID is `ec2-user`. No password is required.

   For further details about connecting to your EC instance, see **Connecting to your Linux instance using SSH** in the Amazon **EC2 User Guide for Linux Instances**.

**Install and configure MongoDB community edition**

Perform these steps on the Amazon EC2 instance that you launched in **Launch an Amazon EC2 instance** (p. 277).

1. Go to **Install MongoDB community edition on Amazon Linux** in the MongoDB documentation and follow the instructions there.

2. By default, the MongoDB server (`mongod`) only allows loopback connections from IP address 127.0.0.1 (localhost). To allow connections from elsewhere in your Amazon VPC, do the following:

   a. Edit the `/etc/mongod.conf` file and look for the following lines.

   ```
   # network interfaces
   net:
   ```

278
Create an AWS DMS replication instance

To perform replication in AWS DMS, you need a replication instance.

2. In the navigation pane, choose Replication instances.
3. Choose **Create replication instance** and enter the following information:
   - For **Name**, enter `mongodb2docdb`.
   - For **Description**, enter `MongoDB to Amazon DocumentDB replication instance`.
   - For **Instance class**, keep the default value.
   - For **Engine version**, keep the default value.
   - For **VPC**, choose your default VPC.
   - For **Multi-AZ**, choose **No**.
   - For **Publicly accessible**, enable this option.

   When the settings are as you want them, choose **Create replication instance**.

   **Note**
   You can begin using your replication instance when its status becomes **available**. This can take several minutes.

### Create source and target endpoints

The source endpoint is the endpoint for your MongoDB installation running on your Amazon EC2 instance.

2. In the navigation pane, choose **Endpoints**.
3. Choose **Create endpoint** and enter the following information:
   - For **Endpoint type**, choose **Source**.
   - For **Endpoint identifier**, enter a name that’s easy to remember, for example `mongodb-source`.
   - For **Source engine**, choose `mongodb`.
   - For **Server name**, enter the public DNS name of your Amazon EC2 instance, for example `ec2-11-22-33-44.us-west-2.compute.amazonaws.com`.
   - For **Port**, enter `27017`.
   - For **SSL mode**, choose **none**.
   - For **Authentication mode**, choose **none**.
   - For **Database name**, enter `zips-db`.
   - For **Metadata mode**, choose **document**.

   When the settings are as you want them, choose **Create endpoint**.

Next, you create a target endpoint. This endpoint is for your Amazon DocumentDB cluster, which should already be running. For more information on launching your Amazon DocumentDB cluster, see [Getting started](#) in the *Amazon DocumentDB Developer Guide*.

**Important**
Before you proceed, do the following:

- Create indexes on your Amazon DocumentDB cluster before you begin migration because it can reduce the overall time and increase the speed of the migration. To extract indexes from a running MongoDB instance, you can use the Amazon DocumentDB Index Tool.
- Get the master user name and password for your Amazon DocumentDB cluster.
- Get the DNS name and port number of your Amazon DocumentDB cluster, so that AWS DMS can connect to it. To determine this information, use the following AWS CLI command, replacing `cluster-id` with the name of your Amazon DocumentDB cluster.
Create source and target endpoints

aws docdb describe-db-clusters
    --db-cluster-identifier cluster-id
    --query "DBClusters[*].[Endpoint,Port]"

• Download a certificate bundle that Amazon DocumentDB can use to verify SSL connections. To do this, enter the following command. Here, `aws-api-domain` completes the Amazon S3 domain in your AWS Region required to access the specified S3 bucket and the `rds-combined-ca-bundle.pem` file that it provides.


To create a target endpoint, do the following:

1. In the navigation pane, choose **Endpoints**.
2. Choose **Create endpoint** and enter the following information:
   - For **Endpoint type**, choose **Target**.
   - For **Endpoint identifier**, enter a name that's easy to remember, for example `docdb-target`.
   - For **Target engine**, choose `docdb`.
   - For **Server name**, enter the DNS name of your Amazon DocumentDB cluster.
   - For **Port**, enter the port number of your Amazon DocumentDB cluster.
   - For **SSL mode**, choose **verify-full**.
   - For **CA certificate**, do one of the following to attach the SSL certificate to your endpoint:
     - If available, choose the existing `rds-combined-ca-bundle` certificate from the **Choose a certificate** drop down.
     - Choose **Add new CA certificate**. Then, for **Certificate identifier**, enter `rds-combined-ca-bundle`. For **Import certificate file**, choose **Choose file** and navigate to the `rds-combined-ca-bundle.pem` file that you previously downloaded. Select and open the file. Choose **Import certificate**, then choose `rds-combined-ca-bundle` from the **Choose a certificate** drop down.
   - For **User name**, enter the master user name of your Amazon DocumentDB cluster.
   - For **Password**, enter the master password of your Amazon DocumentDB cluster.
   - For **Database name**, enter `zips-db`.

   When the settings are as you want them, choose **Create endpoint**.

Now that you've created the source and target endpoints, test them to ensure that they work correctly. Also, to ensure that AWS DMS can access the database objects at each endpoint, refresh the endpoints' schemas.

To test an endpoint, do the following:

1. In the navigation pane, choose **Endpoints**.
2. Choose the source endpoint (mongodb-source), and then choose **Test connection**.
3. Choose your replication instance (mongodb2docdb), and then choose **Run test**. It takes a few minutes for the test to complete, and for the **Status** to change to **successful**.

   If the **Status** changes to **failed** instead, review the failure message. Correct any errors that might be present, and test the endpoint again.

**Note**
Repeat this procedure for the target endpoint (docdb-target).
To refresh schemas, do the following:

1. In the navigation pane, choose **Endpoints**.
2. Choose the source endpoint (mongodb-source), and then choose **Refresh schemas**.
3. Choose your replication instance (mongodb2docdb), and then choose **Refresh schemas**.

**Note**
Repeat this procedure for the target endpoint (docdb-target).

### Create and run a migration task

You are now ready to launch an AWS DMS migration task, to migrate the *zips* data from MongoDB to Amazon DocumentDB.

2. In the navigation pane, choose **Database migration tasks**.
3. Choose **Create task** and enter the following information:
   - For **Task configuration**, choose the following settings:
     - **Task identifier** — enter a name that's easy to remember, for example *my-dms-task*.
     - **Replication instance** — choose the replication instance that you created in Create a replication instance (p. 279).
     - **Source database endpoint** — choose the source endpoint that you created in Create source and target endpoints (p. 280).
     - **Target database endpoint** — choose the target endpoint that you created in Create source and target endpoints (p. 280).
     - **Migration type** — choose **Migrate existing data**.
   - For **Task settings**, choose the following settings:
     - **Target table preparation mode** — Do nothing
     - **Include LOB columns in replication** — Limited LOB mode
     - **Maximum LOB size (KB)** — 32
     - **Enable validation**
     - **Enable CloudWatch logs**
       **Note**
       CloudWatch logs usage will be charged at standard rates. See [here](#) for more details.
   - For **Advanced task settings**, keep all of the options at their default values.
   - For **Premigration assessment**, keep the option at its default value.
   - For **Start migration task in Migration task startup configuration**, choose **Automatically on create**.
   - For **Tags**, keep all of the options at their default values.

When the settings are as you want them, choose **Create task**.

AWS DMS now begins migrating data from MongoDB to Amazon DocumentDB. The task status changes from **Starting** to **Running**. You can monitor the progress by choosing **Tasks** in the AWS DMS console. After several minutes, the status changes to **Load complete**.

**Note**
After the migration is complete, you can use the mongo shell to connect to your Amazon DocumentDB cluster and view the *zips* data. For more information, see Access your Amazon DocumentDB cluster using the mongo shell in the *Amazon DocumentDB Developer Guide*.