
AWS Database Migration Service

User Guide

API Version API Version 2016-01-01



AWS Database Migration Service: User Guide

Copyright © 2019 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What Is AWS Database Migration Service?	1
Migration Tasks That AWS DMS Performs	1
How AWS DMS Works at the Basic Level	2
How AWS DMS Works	4
High-Level View of AWS DMS	4
Components	5
Sources	9
Targets	10
With Other AWS Services	10
Support for AWS CloudFormation	11
Constructing an ARN	11
Setting Up	14
Sign Up for AWS	14
Create an IAM User	14
Migration Planning for AWS Database Migration Service	16
Getting Started	17
Start a Database Migration	17
Step 1: Welcome	17
Step 2: Create a Replication Instance	18
Step 3: Specify Source and Target Endpoints	22
Step 4: Create a Task	25
Monitor Your Task	29
Security	31
IAM Permissions Required	31
IAM Roles for the CLI and API	34
Fine-Grained Access Control	38
Using Resource Names to Control Access	38
Using Tags to Control Access	40
Setting an Encryption Key	44
Network Security	46
Using SSL	47
Limitations on Using SSL with AWS Database Migration Service	48
Managing Certificates	48
Enabling SSL for a MySQL-compatible, PostgreSQL, or SQL Server Endpoint	49
SSL Support for an Oracle Endpoint	50
Changing the Database Password	54
Limits	56
Limits for AWS Database Migration Service	56
Replication Instance	57
Replication Instances in Depth	58
Public and Private Replication Instances	60
AWS DMS Maintenance	60
AWS DMS Maintenance Window	60
Replication Engine Versions	63
Deprecating a Replication Instance Version	63
Upgrading the Engine Version of a Replication Instance	63
Setting Up a Network for a Replication Instance	65
Network Configurations for Database Migration	65
Creating a Replication Subnet Group	70
Setting an Encryption Key	71
Creating a Replication Instance	72
Modifying a Replication Instance	76
Rebooting a Replication Instance	78
Deleting a Replication Instance	80

Supported DDL Statements	81
Endpoints	83
Sources for Data Migration	83
Using Oracle as a Source	84
Using SQL Server as a Source	100
Using Azure SQL Database as a Source	109
Using PostgreSQL as a Source	110
Using MySQL as a Source	122
Using SAP ASE as a Source	129
Using MongoDB as a Source	132
Using Amazon Simple Storage Service as a Source	138
Using IBM Db2 as a Source	144
Targets for Data Migration	147
Using Oracle as a Target	148
Using SQL Server as a Target	152
Using PostgreSQL as a Target	156
Using MySQL as a Target	159
Using Amazon Redshift as a Target	163
Using SAP ASE as a Target	170
Using Amazon Simple Storage Service as a Target	171
Using Amazon DynamoDB as a Target	175
Using Amazon Kinesis Data Streams as a Target	189
Using Amazon Elasticsearch Service as a Target	195
Using Amazon DocumentDB as a Target	198
Creating Source and Target Endpoints	210
Tasks	214
Creating a Task Assessment Report	215
Creating a Task	218
Task Settings	224
Setting LOB Support	238
Creating Multiple Tasks	239
Continuous Replication Tasks	239
Replication Starting from a CDC Start Point	240
Modifying a Task	242
Reloading Tables During a Task	242
AWS Management Console	242
Table Mapping	245
Specifying Table Selection and Transformations by Table Mapping from the Console	245
Specifying Table Selection and Transformations by Table Mapping Using JSON	250
Using Source Filters	257
Monitoring Tasks	261
Task Status	261
Table State During Tasks	262
Monitoring Replication Tasks Using Amazon CloudWatch	263
Data Migration Service Metrics	265
Replication Instance Metrics	265
Replication Task Metrics	266
Managing AWS DMS Logs	267
Logging AWS DMS API Calls with AWS CloudTrail	268
AWS DMS Information in CloudTrail	269
Understanding AWS DMS Log File Entries	269
Validating Tasks	272
Replication Task Statistics	273
Revalidating Tables During a Task	274
AWS Management Console	275
Troubleshooting	275
Limitations	276

Tagging Resources	277
API	278
Working with Events and Notifications	280
AWS DMS Event Categories and Event Messages	281
Subscribing to AWS DMS Event Notification	282
AWS Management Console	283
AWS DMS API	284
Migrating Large Data Stores Using AWS DMS and Snowball	285
Process Overview	285
Step-by-Step Procedures for Migrating Data using AWS DMS and AWS Snowball	287
Step 1: Create an AWS Snowball Job	287
Step 2: Install SCT	287
Step 3: Install and Configure the SCT DMS Agent	287
Step 4: Unlock the AWS Snowball Edge Device	288
Step 5: Create a New AWS SCT Project	288
Step 6: Configure the AWS SCT Profile	288
Step 7: Register the DMS Agent	290
Step 8: Install the Source Database Driver	291
Step 9: Configure AWS SCT to Access the Amazon S3 Bucket	293
Step 10: Creating a Local & DMS Task	293
Step 11: Running and monitoring the Local & DMS Task	295
Step 11: Manage the AWS Snowball Appliance	295
Snowball to Amazon S3	296
Troubleshooting	297
Slow Running Migration Tasks	297
Task Status Bar Not Moving	298
Missing Foreign Keys and Secondary Indexes	298
Amazon RDS Connection Issues	298
Error Message: Incorrect thread connection string: incorrect thread value 0	298
Networking Issues	298
CDC Stuck After Full Load	299
Primary Key Violation Errors When Restarting a Task	299
Initial Load of Schema Fails	299
Tasks Failing With Unknown Error	299
Task Restart Loads Tables From the Beginning	300
Number of Tables Per Task	300
Troubleshooting Oracle Specific Issues	300
Pulling Data from Views	300
Migrating LOBs from Oracle 12c	300
Switching Between Oracle LogMiner and Binary Reader	301
Error: Oracle CDC stopped 122301 Oracle CDC maximum retry counter exceeded.	301
Automatically Add Supplemental Logging to an Oracle Source Endpoint	301
LOB Changes not being Captured	302
Error: ORA-12899: value too large for column <column-name>	302
NUMBER data type being misinterpreted	302
Troubleshooting MySQL Specific Issues	302
CDC Task Failing for Amazon RDS DB Instance Endpoint Because Binary Logging Disabled	303
Connections to a target MySQL instance are disconnected during a task	303
Adding Autocommit to a MySQL-compatible Endpoint	303
Disable Foreign Keys on a Target MySQL-compatible Endpoint	304
Characters Replaced with Question Mark	304
"Bad event" Log Entries	304
Change Data Capture with MySQL 5.5	304
Increasing Binary Log Retention for Amazon RDS DB Instances	305
Log Message: Some changes from the source database had no impact when applied to the target database.	305
Error: Identifier too long	305

Error: Unsupported Character Set Causes Field Data Conversion to Fail	305
Error: Codepage 1252 to UTF8 [120112] A field data conversion failed	306
Troubleshooting PostgreSQL Specific Issues	306
JSON data types being truncated	306
Columns of a user defined data type not being migrated correctly	307
Error: No schema has been selected to create in	307
Deletes and updates to a table are not being replicated using CDC	307
Truncate statements are not being propagated	307
Preventing PostgreSQL from capturing DDL	307
Selecting the schema where database objects for capturing DDL are created	308
Oracle tables missing after migrating to PostgreSQL	308
Task Using View as a Source Has No Rows Copied	308
Troubleshooting Microsoft SQL Server Specific Issues	308
Special Permissions for AWS DMS user account to use CDC	308
Errors Capturing Changes for SQL Server Database	309
Missing Identity Columns	309
Error: SQL Server Does Not Support Publications	309
Changes Not Appearing in Target	309
Troubleshooting Amazon Redshift Specific Issues	309
Loading into a Amazon Redshift Cluster in a Different Region Than the AWS DMS Replication Instance	310
Error: Relation "awsdms_apply_exceptions" already exists	310
Errors with Tables Whose Name Begins with "awsdms_changes"	310
Seeing Tables in Cluster with Names Like dms.awsdms_changes000000000XXXX	310
Permissions Required to Work with Amazon Redshift	310
Troubleshooting Amazon Aurora MySQL Specific Issues	310
Error: CHARACTER SET UTF8 fields terminated by ',' enclosed by '"' lines terminated by '\n'	311
Best Practices	312
Improving Performance	312
Sizing a replication instance	314
Reducing Load on Your Source Database	315
Using the Task Log	315
Schema conversion	315
Migrating Large Binary Objects (LOBs)	315
Using Limited LOB Mode	316
Ongoing Replication	316
Changing the User and Schema for an Oracle Target	317
Improving Performance When Migrating Large Tables	317
Reference	319
AWS DMS Data Types	319
Release Notes	321
AWS DMS 3.1.2 Release Notes	321
AWS DMS 3.1.1 Release Notes	321
AWS DMS 2.4.4 Release Notes	323
AWS DMS 2.4.3 Release Notes	324
AWS DMS 2.4.2 Release Notes	324
AWS DMS 2.4.1 Release Notes	327
AWS DMS 2.4.0 Release Notes	328
AWS DMS 2.3.0 Release Notes	329
Document History	332
Earlier Updates	332
AWS Glossary	335

What Is AWS Database Migration Service?

AWS Database Migration Service (AWS DMS) is a cloud service that makes it easy to migrate relational databases, data warehouses, NoSQL databases, and other types of data stores. You can use AWS DMS to migrate your data into the AWS Cloud, between on-premises instances (through an AWS Cloud setup), or between combinations of cloud and on-premises setups.

With AWS DMS, you can perform one-time migrations, and you can replicate ongoing changes to keep sources and targets in sync. If you want to change database engines, you can use the AWS Schema Conversion Tool (AWS SCT) to translate your database schema to the new platform. You then use AWS DMS to migrate the data. Because AWS DMS is a part of the AWS Cloud, you get the cost efficiency, speed to market, security, and flexibility that AWS services offer.

For information about what AWS Regions support AWS DMS, see [Working with an AWS DMS Replication Instance \(p. 57\)](#). For information on the cost of database migration, see the [AWS Database Migration Service pricing page](#).

Migration Tasks That AWS DMS Performs

AWS DMS takes over many of the difficult or tedious tasks involved in a migration project:

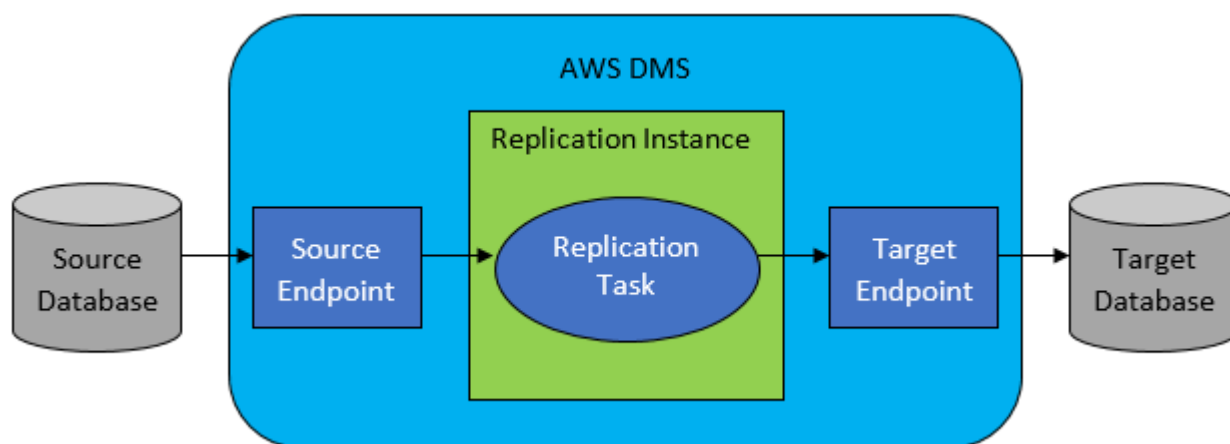
- In a traditional solution, you need to perform capacity analysis, procure hardware and software, install and administer systems, and test and debug the installation. AWS DMS automatically manages the deployment, management, and monitoring of all hardware and software needed for your migration. Your migration can be up and running within minutes of starting the AWS DMS configuration process.
- With AWS DMS, you can scale up (or scale down) your migration resources as needed to match your actual workload. For example, if you determine that you need additional storage, you can easily increase your allocated storage and restart your migration, usually within minutes. On the other hand, if you discover that you aren't using all of the resource capacity you configured, you can easily downsize to meet your actual workload.
- AWS DMS uses a pay-as-you-go model. You only pay for AWS DMS resources while you use them, as opposed to traditional licensing models with up-front purchase costs and ongoing maintenance charges.
- AWS DMS automatically manages all of the infrastructure that supports your migration server, including hardware and software, software patching, and error reporting.
- AWS DMS provides automatic failover. If your primary replication server fails for any reason, a backup replication server can take over with little or no interruption of service.
- AWS DMS can help you switch to a modern, perhaps more cost-effective, database engine than the one you are running now. For example, AWS DMS can help you take advantage of the managed database services provided by Amazon RDS or Amazon Aurora. Or it can help you move to the managed data warehouse service provided by Amazon Redshift, NoSQL platforms like Amazon DynamoDB, or low-cost storage platforms like Amazon Simple Storage Service. Conversely, if you want to migrate away from old infrastructure but continue to use the same database engine, AWS DMS also supports that process.
- AWS DMS supports nearly all of today's most popular DBMS engines as data sources, including Oracle, Microsoft SQL Server, MySQL, MariaDB, PostgreSQL, Db2 LUW, SAP, MongoDB, and Amazon Aurora.
- AWS DMS provides a broad coverage of available target engines including Oracle, Microsoft SQL Server, PostgreSQL, MySQL, Amazon Redshift, SAP ASE, Amazon S3, and Amazon DynamoDB.

- You can migrate from any of the supported data sources to any of the supported data targets. AWS DMS supports fully heterogeneous data migrations between the supported engines.
- AWS DMS ensures that your data migration is secure. Data at rest is encrypted with AWS Key Management Service (AWS KMS) encryption. During migration, you can use Secure Socket Layers (SSL) to encrypt your in-flight data as it travels from source to target.

How AWS DMS Works at the Basic Level

At its most basic level, AWS DMS is a server in the AWS Cloud that runs replication software. You create a source and target connection to tell AWS DMS where to extract from and load to. Then you schedule a task that runs on this server to move your data. AWS DMS creates the tables and associated primary keys if they don't exist on the target. You can precreate the target tables manually, if you prefer. Or you can use AWS SCT to create some or all of the target tables, indexes, views, triggers, and so on.

The following diagram illustrates the AWS DMS process.



To run the AWS DMS process, start to finish

1. To start a migration project, identify your source and target data stores. These data stores can reside on any of the data engines mentioned preceding.
2. For both the source and target, configure endpoints within AWS DMS that specify the connection information to the databases. The endpoints use the appropriate ODBC drivers to communicate with your source and target.
3. Provision a *replication instance*, which is a server that AWS DMS automatically configures with replication software.
4. Create a *replication task*, which specifies the actual data tables to migrate and data transformation rules to apply. AWS DMS manages running the replication task and provides you status on the migration process.

To learn more, see the following:

- If you are new to AWS DMS but familiar with other AWS services, start with [How AWS Database Migration Service Works \(p. 4\)](#). This section dives into the key components of AWS DMS and the overall process of setting up and running a migration.
- If you want to switch database engines, the [AWS Schema Conversion Tool](#) can convert your existing database schema, including tables, indexes, and most application code, to the target platform.
- For information on related AWS services that you might need to design your migration strategy, see [AWS Cloud Products](#).

- Amazon Web Services provides a number of database services. For guidance on which service is best for your environment, see [Running Databases on AWS](#).
- For an overview of all AWS products, see [What is Cloud Computing?](#)

How AWS Database Migration Service Works

AWS Database Migration Service (AWS DMS) is a web service that you can use to migrate data from a source data store to a target data store. These two data stores are called endpoints. You can migrate between source and target endpoints that use the same database engine, such as from an Oracle database to an Oracle database. You can also migrate between source and target endpoints that use different database engines, such as from an Oracle database to a PostgreSQL database. The only requirement to use AWS DMS is that one of your endpoints must be on an AWS service. You can't use AWS DMS to migrate from an on-premises database to another on-premises database.

For information on the cost of database migration, see the [AWS Database Migration Service pricing page](#).

Use the following topics to better understand AWS DMS.

Topics

- [High-Level View of AWS DMS \(p. 4\)](#)
- [Components of AWS Database Migration Service \(p. 5\)](#)
- [Sources for AWS Database Migration Service \(p. 9\)](#)
- [Targets for AWS Database Migration Service \(p. 10\)](#)
- [Using AWS DMS with Other AWS Services \(p. 10\)](#)

High-Level View of AWS DMS

To perform a database migration, AWS DMS connects to the source data store, reads the source data, and formats the data for consumption by the target data store. It then loads the data into the target data store. Most of this processing happens in memory, though large transactions might require some buffering to disk. Cached transactions and log files are also written to disk.

At a high level, when using AWS DMS you do the following:

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

A task can consist of three major phases:

- The full load of existing data
- The application of cached changes
- Ongoing replication

During a full load migration, where existing data from the source is moved to the target, AWS DMS loads data from tables on the source data store to tables on the target data store. While the full load is in

progress, any changes made to the tables being loaded are cached on the replication server; these are the cached changes. It's important to note that AWS DMS doesn't capture changes for a given table until the full load for that table is started. In other words, the point when change capture starts is different for each individual table.

When the full load for a given table is complete, AWS DMS immediately begins to apply the cached changes for that table. When all tables have been loaded, AWS DMS begins to collect changes as transactions for the ongoing replication phase. After AWS DMS applies all cached changes, tables are transactionally consistent. At this point, AWS DMS moves to the ongoing replication phase, applying changes as transactions.

At the start of the ongoing replication phase, a backlog of transactions generally causes some lag between the source and target databases. The migration eventually reaches a steady state after working through this backlog of transactions. At this point, you can shut down your applications, allow any remaining transactions to be applied to the target, and bring your applications up, now pointing at the target database.

AWS DMS creates the target schema objects necessary to perform the migration. However, AWS DMS takes a minimalist approach and creates only those objects required to efficiently migrate the data. In other words, AWS DMS creates tables, primary keys, and in some cases unique indexes, but doesn't create any other objects that are not required to efficiently migrate the data from the source. For example, it doesn't create secondary indexes, nonprimary key constraints, or data defaults.

In most cases, when performing a migration, you also migrate most or all of the source schema. If you are performing a homogeneous migration (between two databases of the same engine type), you migrate the schema by using your engine's native tools to export and import the schema itself, without any data.

If your migration is heterogeneous (between two databases that use different engine types), you can use the AWS Schema Conversion Tool (AWS SCT) to generate a complete target schema for you. If you use the tool, any dependencies between tables such as foreign key constraints need to be disabled during the migration's "full load" and "cached change apply" phases. If performance is an issue, removing or disabling secondary indexes during the migration process helps. For more information on the AWS SCT, see [AWS Schema Conversion Tool](#) in the AWS SCT documentation.

Components of AWS Database Migration Service

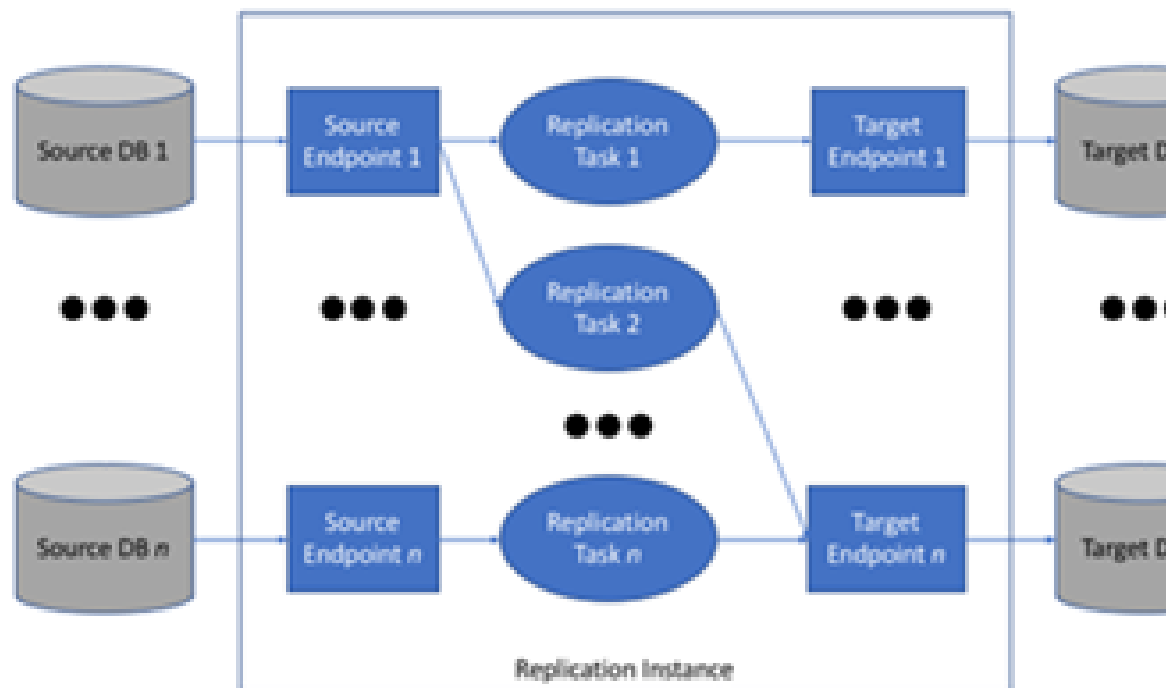
This section describes the internal components of AWS DMS and how they function together to accomplish your data migration. Understanding the underlying components of AWS DMS can help you migrate data more efficiently and provide better insight when troubleshooting or investigating issues.

An AWS DMS migration consists of three components: a replication instance, source and target endpoints, and a replication task. You create an AWS DMS migration by creating the necessary replication instance, endpoints, and tasks in an AWS Region.

Replication instance

At a high level, an AWS DMS replication instance is simply a managed Amazon Elastic Compute Cloud (Amazon EC2) instance that hosts one or more replication tasks.

The figure following shows an example replication instance running several associated replication tasks.



A single replication instance can host one or more replication tasks, depending on the characteristics of your migration and the capacity of the replication server. AWS DMS provides a variety of replication instances so you can choose the optimal configuration for your use case. For more information about the various classes of replication instances, see [Selecting the Right AWS DMS Replication Instance for Your Migration](#) (p. 58).

AWS DMS creates the replication instance on an Amazon Elastic Compute Cloud (Amazon EC2) instance. Some of the smaller instance classes are sufficient for testing the service or for small migrations. If your migration involves a large number of tables, or if you intend to run multiple concurrent replication tasks, you should consider using one of the larger instances. We recommend this approach because AWS DMS can consume a significant amount of memory and CPU.

Depending on the Amazon EC2 instance class you select, your replication instance comes with either 50 GB or 100 GB of data storage. This amount is usually sufficient for most customers. However, if your migration involves large transactions or a high-volume of data changes then you may want to increase the base storage allocation. Change data capture (CDC) may cause data to be written to disk, depending on how fast the target can write the changes.

AWS DMS can provide high availability and failover support using a Multi-AZ deployment. In a Multi-AZ deployment, AWS DMS automatically provisions and maintains a standby replica of the replication instance in a different Availability Zone. The primary replication instance is synchronously replicated to the standby replica. If the primary replication instance fails or becomes unresponsive, the standby resumes any running tasks with minimal interruption. Because the primary is constantly replicating its state to the standby, Multi-AZ deployment does incur some performance overhead.

For more detailed information about the AWS DMS replication instance, see [Working with an AWS DMS Replication Instance](#) (p. 57).

Endpoints

AWS DMS uses an endpoint to access your source or target data store. The specific connection information is different, depending on your data store, but in general you supply the following information when you create an endpoint.

- Endpoint type — Source or target.
- Engine type — Type of database engine, such as Oracle, Postgres, or Amazon S3.
- Server name — Server name or IP address, reachable by AWS DMS
- Port — Port number used for database server connections.
- Encryption — SSL mode, if used to encrypt the connection.
- Credentials — User name and password for an account with the required access rights.

When you create an endpoint using the AWS DMS console, the console requires that you test the endpoint connection. The test must be successful before using the endpoint in a DMS task. Like the connection information, the specific test criteria are different for different engine types. In general, AWS DMS verifies that the database exists at the given server name and port, and that the supplied credentials can be used to connect to the database with the necessary privileges to perform a migration. If the connection test is successful, AWS DMS downloads and stores schema information, including table definitions and primary/unique key definitions, that can be used later during task configuration.

A single endpoint can be used by more than one replication task. For example, you may have two logically distinct applications hosted on the same source database that you want to migrate separately. You would create two replication tasks, one for each set of application tables, but you can use the same AWS DMS endpoint in both tasks.

You can customize the behavior of an endpoint by using *extra connection attributes*. These attributes can control various behavior such as logging detail, file size, and other parameters. Each data store engine type has different extra connection attributes available. You can find the specific extra connection attributes for each data store in the source or target section for that data store. For a list of supported source and target data stores, see [Sources for AWS Database Migration Service \(p. 9\)](#) and [Targets for AWS Database Migration Service \(p. 10\)](#).

For more detailed information about AWS DMS endpoints, see [Working with AWS DMS Endpoints \(p. 83\)](#).

Replication Tasks

You use an AWS DMS replication task to move a set of data from the source endpoint to the target endpoint. Creating a replication task is the last step you need to take before you start a migration.

When you create a replication task, you specify the following task settings:

- Replication instance – the instance that will host and run the task
- Source endpoint
- Target endpoint
- Migration type options – a migration type can be one of the following:
 - Full load (Migrate existing data) – If you can afford an outage long enough to copy your existing data, this option is a good one to choose. This option simply migrates the data from your source database to your target database, creating tables when necessary.
 - Full load + CDC (Migrate existing data and replicate ongoing changes) – This option performs a full data load while capturing changes on the source. Once the full load is complete, captured changes are applied to the target. Eventually the application of changes reaches a steady state. At this point you can shut down your applications, let the remaining changes flow through to the target, and then restart your applications pointing at the target.
 - CDC only (Replicate data changes only) – In some situations it might be more efficient to copy existing data using a method other than AWS DMS. For example, in a homogeneous migration, using native export/import tools might be more efficient at loading the bulk data. In this

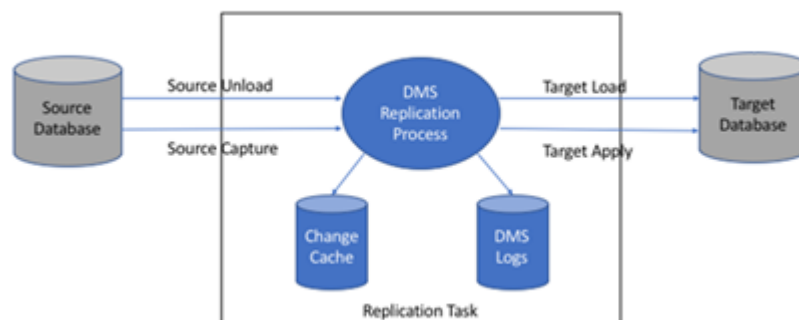
situation, you can use AWS DMS to replicate changes starting when you start your bulk load to bring and keep your source and target databases in sync.

For a full explanation of the migration type options, see [Creating a Task \(p. 218\)](#).

- Target table preparation mode options. For a full explanation of target table modes, see [Creating a Task \(p. 218\)](#).
 - Do nothing – AWS DMS assumes that the target tables are pre-created on the target.
 - Drop tables on target – AWS DMS drops and recreates the target tables.
 - Truncate – If you created tables on the target, AWS DMS truncates them before the migration starts. If no tables exist and you select this option, AWS DMS creates any missing tables.
- LOB mode options. For a full explanation of LOB modes, see [Setting LOB Support for Source Databases in a AWS DMS Task \(p. 238\)](#).
 - Don't include LOB columns – LOB columns are excluded from the migration.
 - Full LOB mode – Migrate complete LOBs regardless of size. AWS DMS migrates LOBs piecewise in chunks controlled by the **Max LOB Size** parameter. This mode is slower than using Limited LOB mode.
 - Limited LOB mode – Truncate LOBs to the value specified by the **Max LOB Size** parameter. This mode is faster than using Full LOB mode.
- Table mappings – indicates the tables to migrate
- Data transformations – changing schema, table, and column names
- data validation
- CloudWatch logging

You use the task to migrate data from the source endpoint to the target endpoint, and the task processing is done on the replication instance. You specify what tables and schemas to migrate and any special processing, such as logging requirements, control table data, and error handling.

Conceptually, an AWS DMS replication task performs two distinct functions as shown in the diagram following:



The full load process is straight-forward to understand. Data is extracted from the source in a bulk extract manner and loaded directly into the target. You can specify the number of tables to extract and load in parallel on the AWS DMS console under **Advanced Settings**.

For more information about AWS DMS tasks, see [Working with AWS DMS Tasks \(p. 214\)](#).

Ongoing replication, or change data capture (CDC)

You can also use an AWS DMS task to capture ongoing changes to the source data store while you are migrating your data to a target. The change capture process that AWS DMS uses when replicating ongoing changes from a source endpoint collects changes to the database logs by using the database engine's native API.

In the CDC process, the replication task is designed to stream changes from the source to the target, using in-memory buffers to hold data in-transit. If the in-memory buffers become exhausted for any reason, the replication task will spill pending changes to the Change Cache on disk. This could occur, for example, if AWS DMS is capturing changes from the source faster than they can be applied on the target. In this case, you will see the task's *target latency* exceed the task's *source latency*.

You can check this by navigating to your task on the AWS DMS console, and opening the Task Monitoring tab. The CDCLatencyTarget and CDCLatencySource graphs are shown at the bottom of the page. If you have a task that is showing target latency then there is likely some tuning on the target endpoint needed to increase the application rate.

The replication task also uses storage for task logs as discussed above. The disk space that comes pre-configured with your replication instance is usually sufficient for logging and spilled changes. If you need additional disk space, for example, when using detailed debugging to investigate a migration issue, you can modify the replication instance to allocate more space.

Schema and code migration

AWS DMS doesn't perform schema or code conversion. You can use tools such as Oracle SQL Developer, MySQL Workbench, or pgAdmin III to move your schema if your source and target are the same database engine. If you want to convert an existing schema to a different database engine, you can use AWS SCT. It can create a target schema and also can generate and create an entire schema: tables, indexes, views, and so on. You can also use AWS SCT to convert PL/SQL or TSQL to PostgreSQL and other formats. For more information on AWS SCT, see [AWS Schema Conversion Tool](#).

Whenever possible, AWS DMS attempts to create the target schema for you. Sometimes, AWS DMS can't create the schema—for example, AWS DMS doesn't create a target Oracle schema for security reasons. For MySQL database targets, you can use extra connection attributes to have AWS DMS migrate all objects to the specified database and schema or create each database and schema for you as it finds the schema on the source.

Sources for AWS Database Migration Service

You can use the following data stores as source endpoints for data migration using AWS Database Migration Service.

On-premises and EC2 instance databases

- Oracle versions 10.2 and later, 11g, and up to 12.1, for the Enterprise, Standard, Standard One, and Standard Two editions
- Microsoft SQL Server versions 2005, 2008, 2008R2, 2012, 2014, and 2016, for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.
- MySQL versions 5.5, 5.6, and 5.7.
- MariaDB (supported as a MySQL-compatible data source).
- PostgreSQL version 9.4 and later.
- MongoDB versions 2.6.x and 3.x and later.
- SAP Adaptive Server Enterprise (ASE) versions 12.5, 15, 15.5, 15.7, 16 and later.
- Db2 LUW versions:
 - Version 9.7, all Fix Packs are supported.
 - Version 10.1, all Fix Packs are supported.
 - Version 10.5, all Fix Packs except for Fix Pack 5 are supported.

Microsoft Azure

- Azure SQL Database.

Amazon RDS instance databases, and Amazon Simple Storage Service

- Oracle versions 11g (versions 11.2.0.3.v1 and later) and 12c, for the Enterprise, Standard, Standard One, and Standard Two editions.
- Microsoft SQL Server versions 2008R2, 2012, 2014, and 2016 for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.
- MySQL versions 5.5, 5.6, and 5.7.
- MariaDB (supported as a MySQL-compatible data source).
- PostgreSQL 9.4 and later. Change data capture (CDC) is only supported for versions 9.4.9 and higher and 9.5.4 and higher. The `rds.logical_replication` parameter, which is required for CDC, is supported only in these versions and later.
- Amazon Aurora (supported as a MySQL-compatible data source).
- Amazon Simple Storage Service.

Targets for AWS Database Migration Service

You can use the following data stores as target endpoints for data migration using AWS Database Migration Service.

On-premises and Amazon EC2 instance databases

- Oracle versions 10g, 11g, 12c, for the Enterprise, Standard, Standard One, and Standard Two editions
- Microsoft SQL Server versions 2005, 2008, 2008R2, 2012, 2014, and 2016, for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.
- MySQL, versions 5.5, 5.6, and 5.7
- MariaDB (supported as a MySQL-compatible data target)
- PostgreSQL, versions 9.4 and later
- SAP Adaptive Server Enterprise (ASE) versions 15, 15.5, 15.7, 16 and later

Amazon RDS instance databases, Amazon Redshift, Amazon DynamoDB, and Amazon S3

- Oracle versions 11g (versions 11.2.0.3.v1 and later) and 12c, for the Enterprise, Standard, Standard One, and Standard Two editions
- Microsoft SQL Server versions 2008R2, 2012, and 2014, for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.
- MySQL, versions 5.5, 5.6, and 5.7
- MariaDB (supported as a MySQL-compatible data target)
- PostgreSQL, versions 9.4 and later
- Amazon Aurora with MySQL compatibility
- Amazon Aurora with PostgreSQL compatibility
- Amazon Redshift
- Amazon S3
- Amazon DynamoDB

Using AWS DMS with Other AWS Services

You can use AWS DMS with several other AWS services :

- You can use an Amazon EC2 instance or Amazon RDS DB instance as a target for a data migration.
- You can use the AWS Schema Conversion Tool (AWS SCT) to convert your source schema and SQL code into an equivalent target schema and SQL code.
- You can use Amazon S3 as a storage site for your data or you can use it as an intermediate step when migrating large amounts of data.
- You can use AWS CloudFormation to set up your AWS resources for infrastructure management or deployment. For example, you can provision AWS DMS resources such as replication instances, tasks, certificates, and endpoints. You create a template that describes all the AWS resources that you want, and AWS CloudFormation provisions and configures those resources for you.

AWS DMS Support for AWS CloudFormation

You can provision AWS Database Migration Service resources using AWS CloudFormation. AWS CloudFormation is a service that helps you model and set up your AWS resources for infrastructure management or deployment. For example, you can provision AWS DMS resources such as replication instances, tasks, certificates, and endpoints. You create a template that describes all the AWS resources that you want and AWS CloudFormation provisions and configures those resources for you.

As a developer or system administrator, you can create and manage collections of these resources that you can then use for repetitive migration tasks or deploying resources to your organization. For more information about AWS CloudFormation, see [AWS CloudFormation Concepts](#) in the *AWS CloudFormation User Guide*.

AWS DMS supports creating the following AWS DMS resources using AWS CloudFormation:

- [AWS::DMS::Certificate](#)
- [AWS::DMS::Endpoint](#)
- [AWS::DMS::EventSubscription](#)
- [AWS::DMS::ReplicationInstance](#)
- [AWS::DMS::ReplicationSubnetGroup](#)
- [AWS::DMS::ReplicationTask](#)

Constructing an Amazon Resource Name (ARN) for AWS DMS

If you use the AWS CLI or AWS Database Migration Service API to automate your database migration, then you need to know about working with an Amazon Resource Name (ARN). Resources that are created in Amazon Web Services are identified by an ARN, which is a unique identifier. If you use the AWS CLI or AWS DMS API to set up your database migration, you must supply the ARN of the resource you want to work with.

An ARN for an AWS DMS resource uses the following syntax:

```
arn:aws:dms:<region>:<account number>:<resourcetype>:<resourcename>
```

In this syntax:

- **<region>** is the ID of the AWS Region where the AWS DMS resource was created, such as `us-west-2`.

The following table shows AWS Region names and the values you should use when constructing an ARN.

Region	Name
Asia Pacific (Tokyo) Region	ap-northeast-1
Asia Pacific (Seoul) Region	ap-northeast-2
Asia Pacific (Mumbai) Region	ap-south-1
Asia Pacific (Singapore) Region	ap-southeast-1
Asia Pacific (Sydney) Region	ap-southeast-2
Canada (Central) Region	ca-central-1
EU (Frankfurt) Region	eu-central-1
EU (Ireland) Region	eu-west-1
EU (London) Region	eu-west-2
South America (São Paulo) Region	sa-east-1
US East (N. Virginia) Region	us-east-1
US East (Ohio) Region	us-east-2
US West (N. California) Region	us-west-1
US West (Oregon) Region	us-west-2

- *<account number>* is your account number with dashes omitted. To find your account number, log in to your AWS account at <http://aws.amazon.com>, choose **My Account/Console**, and then choose **My Account**.
- *<resourcetype>* is the type of AWS DMS resource.

The following table shows the resource types that you should use when constructing an ARN for a particular AWS DMS resource.

AWS DMS Resource Type	ARN Format
Replication instance	arn:aws:dms:<region>: <account>:rep: <resourcename>
Endpoint	arn:aws:dms:<region>:<account>:endpoint: <resourcename>
Replication task	arn:aws:dms:<region>:<account>:task:<resourcename>
Subnet group	arn:aws:dms:<region>:<account>:subgrp:<resourcename>

- *<resourcename>* is the resource name assigned to the AWS DMS resource. This is a generated arbitrary string.

The following table shows examples of ARNs for AWS DMS resources with an AWS account of 123456789012, which were created in the US East (N. Virginia) region, and has a resource name:

Resource Type	Sample ARN
Replication instance	<code>arn:aws:dms:us-east-1:123456789012:rep:QLXQZ64MH7CXF4QCQMGRVYVXAI</code>
Endpoint	<code>arn:aws:dms:us-east-1:123456789012:endpoint:D3HMZ2IGUCGFF3NTAXUXGF6S5A</code>
Replication task	<code>arn:aws:dms:us-east-1:123456789012:task:2PVREMWNPGYJCVU2IBPTOYTIV4</code>
Subnet group	<code>arn:aws:dms:us-east-1:123456789012:subgrp:test-tag-grp</code>

Setting Up for AWS Database Migration Service

Before you use AWS Database Migration Service (AWS DMS) for the first time, complete the following tasks:

1. [Sign Up for AWS](#) (p. 14)
2. [Create an IAM User](#) (p. 14)
3. [Migration Planning for AWS Database Migration Service](#) (p. 16)

Sign Up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all services in AWS, including AWS DMS. You are charged only for the services that you use.

With AWS DMS, you pay only for the resources you use. The AWS DMS replication instance that you create will be live (not running in a sandbox). You will incur the standard AWS DMS usage fees for the instance until you terminate it. For more information about AWS DMS usage rates, see the [AWS DMS product page](#). If you are a new AWS customer, you can get started with AWS DMS for free; for more information, see [AWS Free Usage Tier](#).

If you close your AWS account, all AWS DMS resources and configurations associated with your account are deleted after two days. These resources include all replication instances, source and target endpoint configuration, replication tasks, and SSL certificates. If after two days you decide to use AWS DMS again, you recreate the resources you need.

If you have an AWS account already, skip to the next task.

If you do not have an AWS account, use the following procedure to create one.

To sign up for AWS

1. Open <https://aws.amazon.com/>, and then choose **Create an AWS Account**.

Note

If you previously signed in to the AWS Management Console using AWS account root user credentials, choose **Sign in to a different account**. If you previously signed in to the console using IAM credentials, choose **Sign-in using root account credentials**. Then choose **Create a new AWS account**.

2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code using the phone keypad.

Note your AWS account number, because you'll need it for the next task.

Create an IAM User

Services in AWS, such as AWS DMS, require that you provide credentials when you access them, so that the service can determine whether you have permission to access its resources. The console requires

your password. You can create access keys for your AWS account to access the command line interface or API. However, we don't recommend that you access AWS using the credentials for your AWS account; we recommend that you use AWS Identity and Access Management (IAM) instead. Create an IAM user, and then add the user to an IAM group with administrative permissions or and grant this user administrative permissions. You can then access AWS using a special URL and the credentials for the IAM user.

If you signed up for AWS but have not created an IAM user for yourself, you can create one using the IAM console.

To create an IAM user for yourself and add the user to an Administrators group

1. Use your AWS account email address and password to sign in as the *AWS account root user* to the IAM console at <https://console.aws.amazon.com/iam/>.

Note

We strongly recommend that you adhere to the best practice of using the **Administrator** IAM user below and securely lock away the root user credentials. Sign in as the root user only to perform a few [account and service management tasks](#).

2. In the navigation pane of the console, choose **Users**, and then choose **Add user**.
3. For **User name**, type **Administrator**.
4. Select the check box next to **AWS Management Console access**, select **Custom password**, and then type the new user's password in the text box. You can optionally select **Require password reset** to force the user to create a new password the next time the user signs in.
5. Choose **Next: Permissions**.
6. On the **Set permissions** page, choose **Add user to group**.
7. Choose **Create group**.
8. In the **Create group** dialog box, for **Group name** type **Administrators**.
9. For **Filter policies**, select the check box for **AWS managed - job function**.
10. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.
11. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
12. Choose **Next: Tags** to add metadata to the user by attaching tags as key-value pairs.
13. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

You can use this same process to create more groups and users, and to give your users access to your AWS account resources. To learn about using policies to restrict users' permissions to specific AWS resources, go to [Access Management](#) and [Example Policies](#).

To sign in as this new IAM user, sign out of the AWS console, then use the following URL, where *your_aws_account_id* is your AWS account number without the hyphens (for example, if your AWS account number is 1234-5678-9012, your AWS account ID is 123456789012):

```
https://your_aws_account_id.signin.aws.amazon.com/console/
```

Enter the IAM user name and password that you just created. When you're signed in, the navigation bar displays "*your_user_name @ your_aws_account_id*".

If you don't want the URL for your sign-in page to contain your AWS account ID, you can create an account alias. On the IAM dashboard, choose **Customize** and type an alias, such as your company name. To sign in after you create an account alias, use the following URL.

```
https://your_account_alias.signin.aws.amazon.com/console/
```

To verify the sign-in link for IAM users for your account, open the IAM console and check under **AWS Account Alias** on the dashboard.

Migration Planning for AWS Database Migration Service

When planning a database migration using AWS Database Migration Service, consider the following:

- You will need to configure a network that connects your source and target databases to a AWS DMS replication instance. This can be as simple as connecting two AWS resources in the same VPC as the replication instance to more complex configurations such as connecting an on-premises database to an Amazon RDS DB instance over VPN. For more information, see [Network Configurations for Database Migration \(p. 65\)](#)
- **Source and Target Endpoints** – You will need to know what information and tables in the source database need to be migrated to the target database. AWS DMS supports basic schema migration, including the creation of tables and primary keys. However, AWS DMS doesn't automatically create secondary indexes, foreign keys, user accounts, and so on in the target database. Note that, depending on your source and target database engine, you may need to set up supplemental logging or modify other settings for a source or target database. See the [Sources for Data Migration \(p. 83\)](#) and [Targets for Data Migration \(p. 147\)](#) sections for more information.
- **Schema/Code Migration** – AWS DMS doesn't perform schema or code conversion. You can use tools such as Oracle SQL Developer, MySQL Workbench, or pgAdmin III to convert your schema. If you want to convert an existing schema to a different database engine, you can use the AWS Schema Conversion Tool. It can create a target schema and also can generate and create an entire schema: tables, indexes, views, and so on. You can also use the tool to convert PL/SQL or TSQL to PostgreSQL and other formats. For more information on the AWS Schema Conversion Tool, see [AWS Schema Conversion Tool](#) .
- **Unsupported Data Types** – Some source data types need to be converted into the equivalent data types for the target database. See the source or target section for your data store to find more information on supported data types.

Getting Started with AWS Database Migration Service

AWS Database Migration Service (AWS DMS) helps you migrate databases to AWS easily and securely. You can migrate your data to and from most widely used commercial and open-source databases, such as Oracle, MySQL, and PostgreSQL. The service supports homogeneous migrations such as Oracle to Oracle, and also heterogeneous migrations between different database platforms, such as Oracle to PostgreSQL or MySQL to Oracle.

For information on the cost of database migration using AWS Database Migration Service, see the [AWS Database Migration Service pricing page](#).

Topics

- [Start a Database Migration with AWS Database Migration Service \(p. 17\)](#)
- [Step 1: Welcome \(p. 17\)](#)
- [Step 2: Create a Replication Instance \(p. 18\)](#)
- [Step 3: Specify Source and Target Endpoints \(p. 22\)](#)
- [Step 4: Create a Task \(p. 25\)](#)
- [Monitor Your Task \(p. 29\)](#)

Start a Database Migration with AWS Database Migration Service

You can begin a database migration in several ways. You can select the AWS DMS console wizard that will walk you through each step of the process, or you can do each step by selecting the appropriate task from the navigation pane. You can also use the AWS CLI; for information on using the CLI with AWS DMS, see [AWS CLI for AWS DMS](#).

To use the wizard, select **Getting started** for from the navigation pane on the AWS DMS console. You can use the wizard to help create your first data migration. Following the wizard process, you allocate a replication instance that performs all the processes for the migration, specify a source and a target database, and then create a task or set of tasks to define what tables and replication processes you want to use. AWS DMS then creates your replication instance and performs the tasks on the data being migrated.

Alternatively, you can create each of the components of an AWS DMS database migration by selecting the items from the navigation pane. For a database migration, you must do the following:

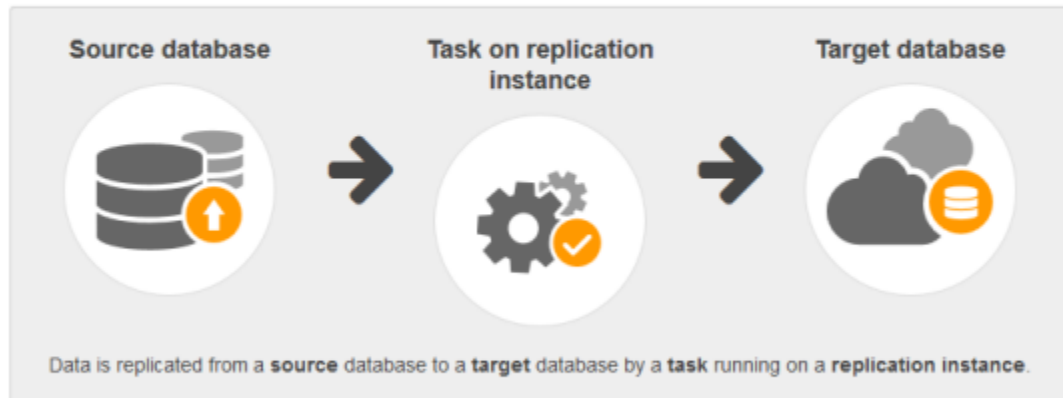
- Complete the tasks outlined in [Setting Up for AWS Database Migration Service \(p. 14\)](#)
- Allocate a replication instance that performs all the processes for the migration
- Specify a source and a target database endpoint
- Create a task or set of tasks to define what tables and replication processes you want to use

Step 1: Welcome

If you start your database migration using the AWS DMS console wizard, you will see the Welcome page, which explains the process of database migration using AWS DMS.

Welcome to AWS Database Migration Service

AWS Database Migration Service tasks require at least a source, a target, and a replication instance. Your source is the database you wish to move data from and the target is the database you're moving data to. The replication instance processes the migration tasks and requires access to your source and target endpoints inside your VPC. Replication instances come in different sizes depending on your performance needs. If you're migrating to a different database engine, AWS Schema Conversion Tool can generate the new schema for you. [Download AWS Schema Conversion Tool](#)



Cancel Back Next

To start a database migration from the console's Welcome page

- Choose **Next**.

Step 2: Create a Replication Instance

Your first task in migrating a database is to create a replication instance that has sufficient storage and processing power to perform the tasks you assign and migrate data from your source database to the target database. The required size of this instance varies depending on the amount of data you need to migrate and the tasks that you need the instance to perform. For more information about replication instances, see [Working with an AWS DMS Replication Instance \(p. 57\)](#).

The procedure following assumes that you have chosen the AWS DMS console wizard. Note that you can also do this step by selecting **Replication instances** from the AWS DMS console's navigation pane and then selecting **Create replication instance**.

To create a replication instance by using the AWS console

1. In the navigation pane, click **Replication instances**.
2. Select **Create Replication Instance**.
3. On the **Create replication instance** page, specify your replication instance information. The following table describes the settings.

Create replication instance

A replication instance initiates the connection between the source and target databases, transfers the data, and caches any changes that occur on the source database during the initial data load. Use the fields below to configure the parameters of your new replication instance including network and security information, encryption details, and performance characteristics. We suggest you shut down the replication instance once your migration is complete to prevent further usage charges.

Name* ⓘ

Description* ⓘ

Instance class* ⓘ

Replication engine version* ⓘ

VPC* ⓘ

Multi-AZ ⓘ

Publicly accessible ⓘ

For This Option	Do This
Name	Type a name for the replication instance that contains from 8 to 16 printable ASCII characters (excluding /, ", and @). The name should be unique for your account for the region you selected. You can choose to add some intelligence to the name, such as including the region and task you are performing, for example west2-mysql2mysql-instance1 .
Description	Type a brief description of the replication instance.
Instance class	Choose an instance class with the configuration you need for your migration. Keep in mind that the instance must have enough storage, network, and processing power to successfully complete your migration. For more information on how to determine which instance class is best for your migration, see Working with an AWS DMS Replication Instance (p. 57).
Replication engine version	By default, the replication instance runs the latest version of the AWS DMS replication engine software. We recommend that you accept this default; however, you can choose a previous engine version if necessary.

For This Option	Do This
VPC	Choose the Amazon Virtual Private Cloud (Amazon VPC) you want to use. If your source or your target database is in an VPC, choose that VPC. If your source and your target databases are in different VPCs, ensure that they are both in public subnets and are publicly accessible, and then choose the VPC where the replication instance is to be located. The replication instance must be able to access the data in the source VPC. If neither your source nor your target database is in a VPC, select a VPC where the replication instance is to be located.
Multi-AZ	Use this optional parameter to create a standby replica of your replication instance in another Availability Zone for failover support. If you intend to use change data capture (CDC) or ongoing replication, you should enable this option.
Publicly accessible	Choose this option if you want the replication instance to be accessible from the Internet.

4. Choose the **Advanced** tab, shown following, to set values for network and encryption settings if you need them. The following table describes the settings.

▼ **Advanced**

Allocated storage (GB)*

Replication Subnet Group*

Availability zone*

VPC Security Group(s)

KMS master key

Description Default master key that protects my DMS replication instance volumes when no other key is defined

Account

Key ARN arn:aws:kms:us-west-2:...:key/...

For This Option	Do This
Allocated storage (GB)	<p>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:</p> <ul style="list-style-type: none"> • 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 Amazon Aurora is the target. <p>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.</p>
Replication Subnet Group	<p>Choose the replication subnet group in your selected VPC where you want the replication instance to be created. If your source database is in a VPC, choose the subnet group that contains the source database as the location for your replication instance. For more information about replication subnet groups, see Creating a Replication Subnet Group (p. 70).</p>
Availability zone	<p>Choose the Availability Zone where your source database is located.</p>
VPC Security group(s)	<p>The replication instance is created in a VPC. If your source database is in a VPC, select the VPC security group that provides access to the DB instance where the database resides.</p>
KMS master key	<p>Choose the encryption key to use to encrypt replication storage and connection information. If you choose (Default) aws/dms, the default AWS Key Management Service (AWS KMS) key associated with your account and region is used. A description and your account number are shown, along with the key's ARN. For more information on using the encryption key, see Setting an Encryption Key and Specifying KMS Permissions (p. 44).</p>

5. Specify the **Maintenance** settings. The following table describes the settings. For more information about maintenance settings, see [AWS DMS Maintenance Window \(p. 60\)](#)

▼ Maintenance

Auto minor version upgrade ⓘ

Maintenance window* for ⓘ
 : (UTC-7)
 hrs.

[Cancel](#) [Create replication instance](#)

For This Option	Do This
Auto minor version upgrade	Select to have minor engine upgrades applied automatically to the replication instance during the maintenance window.
Maintenance window	Choose a weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC). Default: A 30-minute window selected at random from an 8-hour block of time per region, occurring on a random day of the week.

6. Choose **Create replication instance**.

Step 3: Specify Source and Target Endpoints


While your replication instance is being created, you can specify the source and target data stores. The source and target data stores can be on an Amazon Elastic Compute Cloud (Amazon EC2) instance, an Amazon Relational Database Service (Amazon RDS) DB instance, or an on-premises database.

The procedure following assumes that you have chosen the AWS DMS console wizard. Note that you can also do this step by selecting **Endpoints** from the AWS DMS console's navigation pane and then selecting **Create endpoint**. When using the console wizard, you create both the source and target endpoints on the same page. When not using the console wizard, you create each endpoint separately.

To specify source or target database endpoints using the AWS console

1. On the **Connect source and target database endpoints** page, specify your connection information for the source or target database. The following table describes the settings.

Connect source and target database endpoints

 Your replication instance is being created. You can start entering your endpoint connection details now. Once created, you can test your endpoint connections and move on to task creation.

Your database endpoint can be on-premise, in EC2, RDS or in the cloud. Define the connection details below. It is recommended that you test your endpoint connections here to avoid errors later.

Source database connection details	Target database connection details
<p>Endpoint identifier* <input style="width: 100%;" type="text" value="ProdEndpoint"/> ⓘ</p> <p>Source engine* <input style="width: 100%;" type="text" value="- Select One -"/> ⓘ</p> <p>Server name* <input style="width: 100%;" type="text"/></p> <p>Port* <input style="width: 100%;" type="text"/></p> <p>SSL mode* <input style="width: 100%;" type="text" value="- Select One -"/> ⓘ</p> <p>User name* <input style="width: 100%;" type="text"/></p> <p>Password* <input style="width: 100%;" type="password"/></p> <p>▶ Advanced</p> <p style="text-align: center;"><input type="button" value="Run test"/></p>	<p>Endpoint identifier* <input style="width: 100%;" type="text" value="TestEndpoint"/> ⓘ</p> <p>Target engine* <input style="width: 100%;" type="text" value="- Select One -"/> ⓘ</p> <p>Server name* <input style="width: 100%;" type="text"/></p> <p>Port* <input style="width: 100%;" type="text"/></p> <p>SSL mode* <input style="width: 100%;" type="text" value="- Select One -"/> ⓘ</p> <p>User name* <input style="width: 100%;" type="text"/></p> <p>Password* <input style="width: 100%;" type="password"/></p> <p>▶ Advanced</p> <p style="text-align: center;"><input type="button" value="Run test"/></p>
<p><input type="button" value="Cancel"/> <input type="button" value="Previous"/> <input type="button" value="Next"/></p>	

For This Option	Do This
Endpoint identifier	Type the name you want to use to identify the endpoint. You might want to include in the name the type of endpoint, such as oracle-source or PostgreSQL-target . The name must be unique for all replication instances.

For This Option	Do This
Source engine and Target engine	Choose the type of database engine that is the endpoint.
Server name	Type the server name. For an on-premises database, this can be the IP address or the public hostname. For an Amazon RDS DB instance, this can be the endpoint (also called the DNS name) for the DB instance, such as <code>mysqlsrvinst.abcd12345678.us-west-2.rds.amazonaws.com</code> .
Port	Type the port used by the database.
SSL mode	Choose an SSL mode if you want to enable connection encryption for this endpoint. Depending on the mode you select, you might be asked to provide certificate and server certificate information.
User name	Type the user name with the permissions required to allow data migration. For information on the permissions required, see the security section for the source or target database engine in this user guide.
Password	Type the password for the account with the required permissions. If you want to use special characters in your password, such as "+" or "&", enclose the entire password in curly braces "{}".

2. Choose the **Advanced** tab, shown following, to set values for connection string and encryption key if you need them. You can test the endpoint connection by choosing **Run test**.

▼ Advanced

Extra connection attributes

KMS master key (Default) aws/... ⓘ

Description Default master key that protects my DMS replication instance volumes when no other key is defined

Account [REDACTED]

Key ARN arn:aws:kms:us-west-2:[REDACTED]:key/7944f0ec-[REDACTED]

▼ Advanced

Extra connection attributes

KMS master key (Default) aws/... ⓘ

Description Default master key that protects my DMS replication instance volumes when no other key is defined

Account [REDACTED]

Key ARN arn:aws:kms:us-west-2:[REDACTED]:key/7944f0ec-[REDACTED]

For This Option	Do This
Extra connection attributes	Type any additional connection parameters here. For more information about extra connection attributes, see the documentation section for your data store.
KMS master key	Choose the encryption key to use to encrypt replication storage and connection information. If you choose (Default) aws/dms , the default AWS Key Management Service (AWS KMS) key associated with your account and region is used. For more information on using the encryption key, see Setting an Encryption Key and Specifying KMS Permissions (p. 44).

Step 4: Create a Task

Create a task to specify what tables to migrate, to map data using a target schema, and to create new tables on the target database. As part of creating a task, you can choose the type of migration: to

migrate existing data, migrate existing data and replicate ongoing changes, or replicate data changes only.

Using AWS DMS, you can specify precise mapping of your data between the source and the target database. Before you specify your mapping, make sure you review the documentation section on data type mapping for your source and your target database.

You can choose to start a task as soon as you finish specifying information for that task on the **Create task** page, or you can start the task from the Dashboard page once you finish specifying task information.

The procedure following assumes that you have chosen the AWS DMS console wizard and specified replication instance information and endpoints using the console wizard. Note that you can also do this step by selecting **Tasks** from the AWS DMS console's navigation pane and then selecting **Create task**.

To create a migration task

1. On the **Create Task** page, specify the task options. The following table describes the settings.

Create Task

A task can contain one or more table mappings which define what data is moved from the source to the target. If a table does not exist on the target, it can be created automatically.

Task name ProdEndpoint-TestEndpoint ⓘ

Task description e.g. migrate customer tables to RD ⓘ

Source endpoint rdsoracle-endpoint

Target endpoint rdspostgres-endpoint

Replication instance replinstance1-26

Migration type Migrate existing data ⓘ

Start task on create

▶ Task Settings

▶ Table mappings

For This Option	Do This
Task name	Type a name for the task.
Task description	Type a description for the task.
Source endpoint	Shows the source endpoint that will be used.
Target endpoint	Shows the target endpoint that will be used.

For This Option	Do This
Replication instance	Shows the replication instance that will be used.
Migration type	Choose the migration method you want to use. You can choose to have just the existing data migrated to the target database or have ongoing changes sent to the target database in addition to the migrated data.
Start task on create	When this option is selected, the task begins as soon as it is created.

2. Choose the **Task Settings** tab, shown following, and specify values for your target table, LOB support, and to enable logging. The task settings shown depend on the **Migration type** value you select. For example, when you select **Migrate existing data**, the following options are shown:

▼ **Task Settings**

Target table preparation mode Do nothing ⓘ
 Drop tables on target
 Truncate

Include LOB columns in replication Don't include LOB columns ⓘ
 Full LOB mode
 Limited LOB mode

Max LOB size (kb)* ⓘ

Enable logging

For This Option	Do This
Target table preparation mode	<p>Do nothing - Data and metadata of the target tables are not changed.</p> <p>Drop tables on target - The tables are dropped and new tables are created in their place.</p> <p>Truncate - Tables are truncated without affecting table metadata.</p>
Include LOB columns in replication	<p>Don't include LOB columns - LOB columns will be excluded from the migration.</p> <p>Full LOB mode - Migrate complete LOBs regardless of size. LOBs are migrated piecewise in chunks controlled by the LOB chunk size. This method is slower than using Limited LOB Mode.</p> <p>Limited LOB mode - Truncate LOBs to 'Max LOB Size' This method is faster than using Full LOB Mode.</p>

For This Option	Do This
Max LOB size (kb)	In Limited LOB Mode , LOB columns which exceed the setting of Max LOB Size will be truncated to the specified Max LOB Size.
Enable logging	Enables logging by Amazon CloudWatch.

When you select **Migrate existing data and replicate** for **Migration type**, the following options are shown:

▼ Task Settings

Target table preparation mode Do nothing ⓘ
 Drop tables on target
 Truncate

Stop task after full load completes Don't stop ⓘ
 Stop Before Applying Cached Changes
 Stop After Applying Cached Changes

Include LOB columns in replication Don't include LOB columns ⓘ
 Full LOB mode
 Limited LOB mode

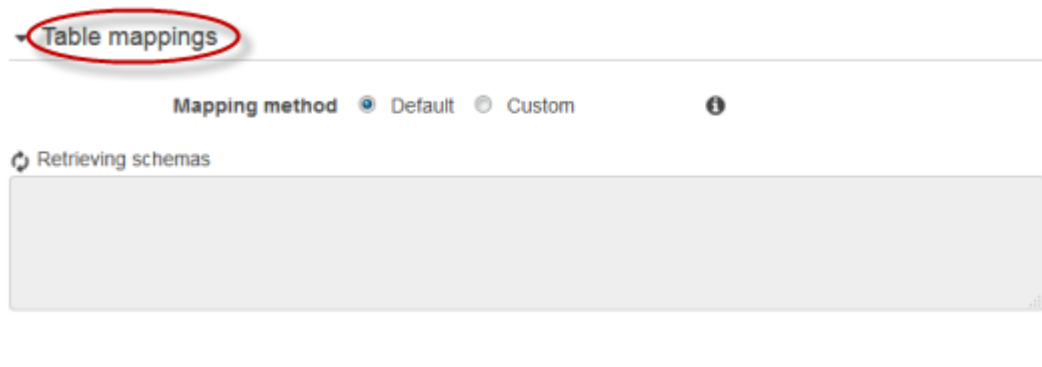
Max LOB size (kb)* ⓘ

Enable logging

For This Option	Do This
Target table preparation mode	<p>Do nothing - Data and metadata of the target tables are not changed.</p> <p>Drop tables on target - The tables are dropped and new tables are created in their place.</p> <p>Truncate - Tables are truncated without affecting table metadata.</p>

For This Option	Do This
Stop task after full load completes	<p>Don't stop - Do not stop the task, immediately apply cached changes and continue on.</p> <p>Stop before applying cached changes - Stop the task prior to the application of cached changes. This will allow you to add secondary indexes which may speed the application of changes.</p> <p>Stop after applying cached changes - Stop the task after cached changes have been applied. This will allow you to add foreign keys, triggers etc. if you are using Transactional Apply.</p>
Include LOB columns in replication	<p>Don't include LOB columns - LOB columns will be excluded from the migration.</p> <p>Full LOB mode - Migrate complete LOBs regardless of size. LOBs are migrated piecewise in chunks controlled by the LOB chunk size. This method is slower than using Limited LOB Mode.</p> <p>Limited LOB mode - Truncate LOBs to 'Max LOB Size' This method is faster than using Full LOB Mode.</p>
Max LOB size (kb)	In Limited LOB Mode , LOB columns which exceed the setting of Max LOB Size will be truncated to the specified Max LOB Size.
Enable logging	Enables logging by Amazon CloudWatch.

- Choose the **Table mappings** tab, shown following, to set values for schema mapping and the mapping method. If you choose **Custom**, you can specify the target schema and table values. For more information about table mapping, see [Using Table Mapping to Specify Task Settings \(p. 245\)](#).



- Once you have finished with the task settings, choose **Create task**.

Monitor Your Task

If you select **Start task on create** when you create a task, your task begins immediately to migrate your data when you choose **Create task**. You can view statistics and monitoring information for your task by choosing the running task from the AWS Management Console. The following screenshot shows the

table statistics of a database migration. For more information about monitoring, see [Monitoring AWS DMS Tasks \(p. 261\)](#)

task-fkvacppsulxnqyv

Overview Task monitoring **Table statistics** Logs

Filter: X

Schema	Table	State	Inserts	Deletes	Updates	DDLs	Full Load Rows	Total	La
aat	T20	Full load	0	0	0	0	16,080,394	16,080,394	3/1
aat	T21	Full load	0	0	0	0	16,079,437	16,079,437	3/1
aat	T22	Full load	0	0	0	0	15,804,000	15,804,000	3/1

Security for AWS Database Migration Service

AWS Database Migration Service (AWS DMS) uses several processes to secure your data during migration. The service encrypts the storage used by your replication instance and the endpoint connection information using an AWS Key Management Service (AWS KMS) key that is unique to your AWS account. Secure Sockets Layer (SSL) is supported. AWS Database Migration Service also requires that you have the appropriate permissions if you sign in as an AWS Identity and Access Management (IAM) user.

The VPC based on the Amazon Virtual Private Cloud (Amazon VPC) service that you use with your replication instance must be associated with a security group that has rules that allow all traffic on all ports to leave (egress) the VPC. This approach allows communication from the replication instance to your source and target database endpoints, as long as correct ingress is enabled on those endpoints.

If you want to view database migration logs, you need the appropriate Amazon CloudWatch Logs permissions for the IAM role you are using.

Topics

- [IAM Permissions Needed to Use AWS DMS \(p. 31\)](#)
- [Creating the IAM Roles to Use With the AWS CLI and AWS DMS API \(p. 34\)](#)
- [Fine-Grained Access Control Using Resource Names and Tags \(p. 38\)](#)
- [Setting an Encryption Key and Specifying KMS Permissions \(p. 44\)](#)
- [Network Security for AWS Database Migration Service \(p. 46\)](#)
- [Using SSL With AWS Database Migration Service \(p. 47\)](#)
- [Changing the Database Password \(p. 54\)](#)

IAM Permissions Needed to Use AWS DMS

You need to use certain IAM permissions and IAM roles to use AWS DMS. If you are signed in as an IAM user and want to use AWS DMS, your account administrator must attach the policy discussed in this section to the IAM user, group, or role that you use to run AWS DMS. For more information about IAM permissions, see the [IAM User Guide](#).

The following set of permissions gives you access to AWS DMS, and also permissions for certain actions needed from other Amazon services such as AWS KMS, IAM, Amazon Elastic Compute Cloud (Amazon EC2), and Amazon CloudWatch. CloudWatch monitors your AWS DMS migration in real time and collects and tracks metrics that indicate the progress of your migration. You can use CloudWatch Logs to debug problems with a task.

Note

You can further restrict access to AWS DMS resources using tagging. For more information about restricting access to AWS DMS resources using tagging, see [Fine-Grained Access Control Using Resource Names and Tags \(p. 38\)](#)

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
{
  "Effect": "Allow",
  "Action": "dms:*",
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": [
    "kms:ListAliases",
    "kms:DescribeKey"
  ],
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": [
    "iam:GetRole",
    "iam:PassRole",
    "iam:CreateRole",
    "iam:AttachRolePolicy"
  ],
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": [
    "ec2:DescribeVpcs",
    "ec2:DescribeInternetGateways",
    "ec2:DescribeAvailabilityZones",
    "ec2:DescribeSubnets",
    "ec2:DescribeSecurityGroups",
    "ec2:ModifyNetworkInterfaceAttribute",
    "ec2:CreateNetworkInterface",
    "ec2>DeleteNetworkInterface"
  ],
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": [
    "cloudwatch:Get*",
    "cloudwatch:List*"
  ],
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": [
    "logs:DescribeLogGroups",
    "logs:DescribeLogStreams",
    "logs:FilterLogEvents",
    "logs:GetLogEvents"
  ],
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": [
    "redshift:Describe*",
    "redshift:ModifyClusterIamRoles"
  ],
  "Resource": "*"
}
]
}
```

A breakdown of these permissions might help you better understand why each one is necessary.

This section is required to allow the user to call AWS DMS API operations.

```
{
    "Effect": "Allow",
    "Action": "dms:*",
    "Resource": "*"
}
```

This section is required to allow the user to list their available KMS Keys and alias for display in the console. This entry is not required if the KMSkey ARN is known and when using only the CLI.

```
{
    "Effect": "Allow",
    "Action": [
        "kms:ListAliases",
        "kms:DescribeKey"
    ],
    "Resource": "*"
}
```

This section is required for certain endpoint types that require a Role ARN to be passed in with the endpoint. In addition, if the required AWS DMS roles are not created ahead of time, the AWS DMS console has the ability to create the role. If all roles are configured ahead of time, all that is required in iam:GetRole and iam:PassRole. For more information about roles, see [Creating the IAM Roles to Use With the AWS CLI and AWS DMS API \(p. 34\)](#).

```
{
    "Effect": "Allow",
    "Action": [
        "iam:GetRole",
        "iam:PassRole",
        "iam:CreateRole",
        "iam:AttachRolePolicy"
    ],
    "Resource": "*"
}
```

This section is required since AWS DMS needs to create the EC2 instance and configure the network for the replication instance that is created. These resources exist in the customer's account, so the ability to perform these actions on behalf of the customer is required.

```
{
    "Effect": "Allow",
    "Action": [
        "ec2:DescribeVpcs",
        "ec2:DescribeInternetGateways",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:ModifyNetworkInterfaceAttribute",
        "ec2:CreateNetworkInterface",
        "ec2>DeleteNetworkInterface"
    ],
    "Resource": "*"
}
```

This section is required to allow the user to be able to view replication instance metrics.

```
{
  "Effect": "Allow",
  "Action": [
    "cloudwatch:Get*",
    "cloudwatch:List*"
  ],
  "Resource": "*"
}
```

This section is required to allow the user to view replication logs.

```
{
  "Effect": "Allow",
  "Action": [
    "logs:DescribeLogGroups",
    "logs:DescribeLogStreams",
    "logs:FilterLogEvents",
    "logs:GetLogEvents"
  ],
  "Resource": "*"
}
```

This section is required when using Redshift as a target. It allows AWS DMS to validate that the Redshift cluster is set up properly for AWS DMS.

```
{
  "Effect": "Allow",
  "Action": [
    "redshift:Describe*",
    "redshift:ModifyClusterIamRoles"
  ],
  "Resource": "*"
}
```

The AWS DMS console creates several roles that are automatically attached to your AWS account when you use the AWS DMS console. If you use the AWS Command Line Interface (AWS CLI) or the AWS DMS API for your migration, you need to add these roles to your account. For more information on adding these roles, see [Creating the IAM Roles to Use With the AWS CLI and AWS DMS API](#) (p. 34).

Creating the IAM Roles to Use With the AWS CLI and AWS DMS API

If you use the AWS CLI or the AWS DMS API for your database migration, you must add three IAM roles to your AWS account before you can use the features of AWS DMS. Two of these are `dms-vpc-role` and `dms-cloudwatch-logs-role`. If you use Amazon Redshift as a target database, you must also add the IAM role `dms-access-for-endpoint` to your AWS account.

Updates to managed policies are automatic. If you are using a custom policy with the IAM roles, be sure to periodically check for updates to the managed policy in this documentation. You can view the details of the managed policy by using a combination of the `get-policy` and `get-policy-version` commands.

For example, the following `get-policy` command retrieves information on the role.


```
aws iam get-policy --policy-arn arn:aws:iam::aws:policy/service-role/  
AmazonDMSVPCManagementRole
```

The information returned from the command is as follows.

```
{  
  "Policy": {  
    "PolicyName": "AmazonDMSVPCManagementRole",  
    "Description": "Provides access to manage VPC settings for AWS managed customer  
configurations",  
    "CreateDate": "2015-11-18T16:33:19Z",  
    "AttachmentCount": 1,  
    "IsAttachable": true,  
    "PolicyId": "ANPAJHKIGMBQI4AEFFSYO",  
    "DefaultVersionId": "v3",  
    "Path": "/service-role/",  
    "Arn": "arn:aws:iam::aws:policy/service-role/AmazonDMSVPCManagementRole",  
    "UpdateDate": "2016-05-23T16:29:57Z"  
  }  
}
```

The following `get-policy-version` command retrieves policy information.

```
aws iam get-policy-version --policy-arn arn:aws:iam::aws:policy/service-role/  
AmazonDMSVPCManagementRole --version-id v3
```

The information returned from the command is as follows.

```
{  
  "PolicyVersion": {  
    "CreateDate": "2016-05-23T16:29:57Z",  
    "VersionId": "v3",  
    "Document": {  
      "Version": "2012-10-17",  
      "Statement": [  
        {  
          "Action": [  
            "ec2:CreateNetworkInterface",  
            "ec2:DescribeAvailabilityZones",  
            "ec2:DescribeInternetGateways",  
            "ec2:DescribeSecurityGroups",  
            "ec2:DescribeSubnets",  
            "ec2:DescribeVpcs",  
            "ec2>DeleteNetworkInterface",  
            "ec2:ModifyNetworkInterfaceAttribute"  
          ],  
          "Resource": "*",  
          "Effect": "Allow"  
        }  
      ]  
    },  
    "IsDefaultVersion": true  
  }  
}
```

The same commands can be used to get information on the `AmazonDMSCloudWatchLogsRole` and the `AmazonDMSRedshiftS3Role` managed policy.

Note

If you use the AWS DMS console for your database migration, these roles are added to your AWS account automatically.

The following procedures create the `dms-vpc-role`, `dms-cloudwatch-logs-role`, and `dms-access-for-endpoint` IAM roles.

To create the `dms-vpc-role` IAM role for use with the AWS CLI or AWS DMS API

1. Create a JSON file with the IAM policy following. Name the JSON file `dmsAssumeRolePolicyDocument.json`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "dms.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Create the role using the AWS CLI using the following command.

```
aws iam create-role --role-name dms-vpc-role --assume-role-policy-document file://
dmsAssumeRolePolicyDocument.json
```

2. Attach the `AmazonDMSVPCManagementRole` policy to `dms-vpc-role` using the following command.

```
aws iam attach-role-policy --role-name dms-vpc-role --policy-arn
arn:aws:iam::aws:policy/service-role/AmazonDMSVPCManagementRole
```

To create the `dms-cloudwatch-logs-role` IAM role for use with the AWS CLI or AWS DMS API

1. Create a JSON file with the IAM policy following. Name the JSON file `dmsAssumeRolePolicyDocument2.json`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "dms.amazonaws.com"
      },
    },
  ]
}
```

```
"Action": "sts:AssumeRole"
}
]
}
```

Create the role using the AWS CLI using the following command.

```
aws iam create-role --role-name dms-cloudwatch-logs-role --assume-role-policy-document
file://dmsAssumeRolePolicyDocument2.json
```

2. Attach the `AmazonDMSCloudWatchLogsRole` policy to `dms-cloudwatch-logs-role` using the following command.

```
aws iam attach-role-policy --role-name dms-cloudwatch-logs-role --policy-arn
arn:aws:iam::aws:policy/service-role/AmazonDMSCloudWatchLogsRole
```

If you use Amazon Redshift as your target database, you must create the IAM role `dms-access-for-endpoint` to provide access to Amazon Simple Storage Service (Amazon S3).

To create the `dms-access-for-endpoint` IAM role for use with Amazon Redshift as a target database

1. Create a JSON file with the IAM policy following. Name the JSON file `dmsAssumeRolePolicyDocument3.json`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
        "Service": "dms.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    },
    {
      "Sid": "2",
      "Effect": "Allow",
      "Principal": {
        "Service": "redshift.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Create the role using the AWS CLI using the following command.

```
aws iam create-role --role-name dms-access-for-endpoint --assume-role-policy-document
file://dmsAssumeRolePolicyDocument3.json
```

3. Attach the `AmazonDMSRedshiftS3Role` policy to `dms-access-for-endpoint` role using the following command.

```
aws iam attach-role-policy --role-name dms-access-for-endpoint \  
--policy-arn arn:aws:iam::aws:policy/service-role/AmazonDMSRedshiftS3Role
```

You should now have the IAM policies in place to use the AWS CLI or AWS DMS API.

Fine-Grained Access Control Using Resource Names and Tags

You can use ARN-based resource names and resource tags to manage access to AWS DMS resources. You do this by defining permitted action or including conditional statements in IAM policies.

Using Resource Names to Control Access

You can create an IAM user account and assign a policy based on the AWS DMS resource's Amazon Resource Name (ARN).

The following policy denies access to the AWS DMS replication instance with the ARN `arn:aws:dms:us-east-1:152683116:rep:DOH67ZTOXGLIXMIHKITV`:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Action": [  
        "dms:*"  
      ],  
      "Effect": "Deny",  
      "Resource": "arn:aws:dms:us-east-1:152683116:rep:DOH67ZTOXGLIXMIHKITV"  
    }  
  ]  
}
```

For example, the following commands would fail when the policy is in effect:

```
$ aws dms delete-replication-instance  
--replication-instance-arn "arn:aws:dms:us-east-1:152683116:rep:DOH67ZTOXGLIXMIHKITV"  
  
A client error (AccessDeniedException) occurred when calling the DeleteReplicationInstance  
operation: User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform:  
dms:DeleteReplicationInstance on resource: arn:aws:dms:us-  
east-1:152683116:rep:DOH67ZTOXGLIXMIHKITV  
  
$ aws dms modify-replication-instance  
--replication-instance-arn "arn:aws:dms:us-east-1:152683116:rep:DOH67ZTOXGLIXMIHKITV"  
  
A client error (AccessDeniedException) occurred when calling the ModifyReplicationInstance  
operation: User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform:
```

```
dms:ModifyReplicationInstance on resource: arn:aws:dms:us-  
east-1:152683116:rep:DOH67ZTOXGLIXMIHKITV
```

You can also specify IAM policies that limit access to AWS DMS endpoints and replication tasks.

The following policy limits access to an AWS DMS endpoint using the endpoint's ARN:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Action": [  
        "dms:*"  
      ],  
      "Effect": "Deny",  
      "Resource": "arn:aws:dms:us-east-1:152683116:endpoint:D6E37YBXTNHOA6XRQSZCUGX"  
    }  
  ]  
}
```

For example, the following commands would fail when the policy using the endpoint's ARN is in effect:

```
$ aws dms delete-endpoint  
  --endpoint-arn "arn:aws:dms:us-east-1:152683116:endpoint:D6E37YBXTNHOA6XRQSZCUGX"  
  
A client error (AccessDeniedException) occurred when calling the DeleteEndpoint operation:  
User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform:  
  dms:DeleteEndpoint  
on resource: arn:aws:dms:us-east-1:152683116:endpoint:D6E37YBXTNHOA6XRQSZCUGX  
  
$ aws dms modify-endpoint  
  --endpoint-arn "arn:aws:dms:us-east-1:152683116:endpoint:D6E37YBXTNHOA6XRQSZCUGX"  
  
A client error (AccessDeniedException) occurred when calling the ModifyEndpoint operation:  
User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform:  
  dms:ModifyEndpoint  
on resource: arn:aws:dms:us-east-1:152683116:endpoint:D6E37YBXTNHOA6XRQSZCUGX
```

The following policy limits access to an AWS DMS task using the task's ARN:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Action": [  
        "dms:*"  
      ],  
      "Effect": "Deny",  
      "Resource": "arn:aws:dms:us-east-1:152683116:task:UO3YR4N47DXH3ATT4YMW0IT"  
    }  
  ]  
}
```

For example, the following commands would fail when the policy using the task's ARN is in effect:

```
$ aws dms delete-replication-task
  --replication-task-arn "arn:aws:dms:us-east-1:152683116:task:U03YR4N47DXH3ATT4YMW0IT"

A client error (AccessDeniedException) occurred when calling the DeleteReplicationTask
operation:
User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform:
dms:DeleteReplicationTask
on resource: arn:aws:dms:us-east-1:152683116:task:U03YR4N47DXH3ATT4YMW0IT
```

Using Tags to Control Access

AWS DMS defines a set of common key/value pairs that are available for use in customer defined policies without any additional tagging requirements. For more information about tagging AWS DMS resources, see [Tagging Resources in AWS Database Migration Service \(p. 277\)](#).

The following lists the standard tags available for use with AWS DMS:

- `aws:CurrentTime` – Represents the request date and time, allowing the restriction of access based on temporal criteria.
- `aws:EpochTime` – This tag is similar to the `aws:CurrentTime` tag above, except that the current time is represented as the number of seconds elapsed since the Unix Epoch.
- `aws:MultiFactorAuthPresent` – This is a boolean tag that indicates whether or not the request was signed via multi-factor authentication.
- `aws:MultiFactorAuthAge` – Provides access to the age of the multi-factor authentication token (in seconds).
- `aws:principaltype` - Provides access to the type of principal (user, account, federated user, etc.) for the current request.
- `aws:SourceIp` - Represents the source ip address for the user issuing the request.
- `aws:UserAgent` – Provides information about the client application requesting a resource.
- `aws:userid` – Provides access to the ID of the user issuing the request.
- `aws:username` – Provides access to the name of the user issuing the request.
- `dms:InstanceClass` – Provides access to the compute size of the replication instance host(s).
- `dms:StorageSize` - Provides access to the storage volume size (in GB).

You can also define your own tags. Customer-defined tags are simple key/value pairs that are persisted in the AWS Tagging service and can be added to AWS DMS resources, including replication instances, endpoints, and tasks. These tags are matched via IAM "Conditional" statements in policies, and are referenced using a specific conditional tag. The tag keys are prefixed with "dms", the resource type, and the "tag" prefix. The following shows the tag format:

```
dms:{resource type}-tag/{tag key}={tag value}
```

For example, suppose you want to define a policy that only allows an API call to succeed for a replication instance that contains the tag "stage=production". The following conditional statement would match a resource with the given tag:

```
"Condition":
{
  "streq":
  {
    "dms:rep-tag/stage": "production"
  }
}
```

```
}  
}
```

You would add the following tag to a replication instance that would match this policy condition:

```
stage production
```

In addition to tags already assigned to AWS DMS resources, policies can also be written to limit the tag keys and values that may be applied to a given resource. In this case, the tag prefix would be "req".

For example, the following policy statement would limit the tags that a user can assign to a given resource to a specific list of allowed values:

```
"Condition":  
{  
  "streq":  
  {  
    "dms:req-tag/stage": [ "production", "development", "testing" ]  
  }  
}
```

The following policy examples limit access to an AWS DMS resource based on resource tags.

The following policy limits access to a replication instance where the tag value is "Desktop" and the tag key is "Env":

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Action": [  
        "dms:*"  
      ],  
      "Effect": "Deny",  
      "Resource": "*",  
      "Condition": {  
        "StringEquals": {  
          "dms:rep-tag/Env": [  
            "Desktop"  
          ]  
        }  
      }  
    }  
  ]  
}
```

The following commands succeed or fail based on the IAM policy that restricts access when the tag value is "Desktop" and the tag key is "Env":

```
$ aws dms list-tags-for-resource  
--resource-name arn:aws:dms:us-east-1:152683116:rep:46DHOU7JOJYOJXWDOZNFEN  
--endpoint-url http://localhost:8000  
{  
  "TagList": [  
    {  
      "Value": "Desktop",  
      "Key": "Env"  
    }  
  ]  
}
```

```
]
}

$ aws dms delete-replication-instance
  --replication-instance-arn "arn:aws:dms:us-east-1:152683116:rep:46DHOU7JOJYOJXWDOZNFEN"
A client error (AccessDeniedException) occurred when calling the DeleteReplicationInstance
operation: User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform:
dms:DeleteReplicationInstance on resource: arn:aws:dms:us-
east-1:152683116:rep:46DHOU7JOJYOJXWDOZNFEN

$ aws dms modify-replication-instance
  --replication-instance-arn "arn:aws:dms:us-east-1:152683116:rep:46DHOU7JOJYOJXWDOZNFEN"

A client error (AccessDeniedException) occurred when calling the ModifyReplicationInstance
operation: User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform:
dms:ModifyReplicationInstance on resource: arn:aws:dms:us-
east-1:152683116:rep:46DHOU7JOJYOJXWDOZNFEN

$ aws dms add-tags-to-resource
  --resource-name arn:aws:dms:us-east-1:152683116:rep:46DHOU7JOJYOJXWDOZNFEN
  --tags Key=CostCenter,Value=1234

A client error (AccessDeniedException) occurred when calling the AddTagsToResource
operation: User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform:
dms:AddTagsToResource on resource: arn:aws:dms:us-
east-1:152683116:rep:46DHOU7JOJYOJXWDOZNFEN

$ aws dms remove-tags-from-resource
  --resource-name arn:aws:dms:us-east-1:152683116:rep:46DHOU7JOJYOJXWDOZNFEN
  --tag-keys Env

A client error (AccessDeniedException) occurred when calling the RemoveTagsFromResource
operation: User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform:
dms:RemoveTagsFromResource on resource: arn:aws:dms:us-
east-1:152683116:rep:46DHOU7JOJYOJXWDOZNFEN
```

The following policy limits access to a AWS DMS endpoint where the tag value is "Desktop" and the tag key is "Env":

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "dms:*"
      ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "dms:endpoint-tag/Env": [
            "Desktop"
          ]
        }
      }
    }
  ]
}
```

The following commands succeed or fail based on the IAM policy that restricts access when the tag value is "Desktop" and the tag key is "Env":


```
$ aws dms list-tags-for-resource
--resource-name arn:aws:dms:us-east-1:152683116:endpoint:J2YCZPNGOLFY52344IZWA6I
{
  "TagList": [
    {
      "Value": "Desktop",
      "Key": "Env"
    }
  ]
}

$ aws dms delete-endpoint
--endpoint-arn "arn:aws:dms:us-east-1:152683116:endpoint:J2YCZPNGOLFY52344IZWA6I"

A client error (AccessDeniedException) occurred when calling the DeleteEndpoint
operation: User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform:
dms:DeleteEndpoint on resource: arn:aws:dms:us-
east-1:152683116:endpoint:J2YCZPNGOLFY52344IZWA6I

$ aws dms modify-endpoint
--endpoint-arn "arn:aws:dms:us-east-1:152683116:endpoint:J2YCZPNGOLFY52344IZWA6I"

A client error (AccessDeniedException) occurred when calling the ModifyEndpoint
operation: User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform:
dms:ModifyEndpoint on resource: arn:aws:dms:us-
east-1:152683116:endpoint:J2YCZPNGOLFY52344IZWA6I

$ aws dms add-tags-to-resource
--resource-name arn:aws:dms:us-east-1:152683116:endpoint:J2YCZPNGOLFY52344IZWA6I
--tags Key=CostCenter,Value=1234

A client error (AccessDeniedException) occurred when calling the AddTagsToResource
operation: User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform:
dms:AddTagsToResource on resource: arn:aws:dms:us-
east-1:152683116:endpoint:J2YCZPNGOLFY52344IZWA6I

$ aws dms remove-tags-from-resource
--resource-name arn:aws:dms:us-east-1:152683116:endpoint:J2YCZPNGOLFY52344IZWA6I
--tag-keys Env

A client error (AccessDeniedException) occurred when calling the RemoveTagsFromResource
operation: User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform:
dms:RemoveTagsFromResource on resource: arn:aws:dms:us-
east-1:152683116:endpoint:J2YCZPNGOLFY52344IZWA6I
```

The following policy limits access to a replication task where the tag value is "Desktop" and the tag key is "Env":

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "dms:*"
      ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "dms:task-tag/Env": [
            "Desktop"
          ]
        }
      }
    }
  ]
}
```

```
}  
  }  
} ]
```

The following commands succeed or fail based on the IAM policy that restricts access when the tag value is "Desktop" and the tag key is "Env":

```
$ aws dms list-tags-for-resource  
--resource-name arn:aws:dms:us-east-1:152683116:task:RB7N24J2XBUPS3RFABZTG3  
{  
  "TagList": [  
    {  
      "Value": "Desktop",  
      "Key": "Env"  
    }  
  ]  
}  
  
$ aws dms delete-replication-task  
--replication-task-arn "arn:aws:dms:us-east-1:152683116:task:RB7N24J2XBUPS3RFABZTG3"
```

A client error (AccessDeniedException) occurred when calling the DeleteReplicationTask operation: User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform: dms:DeleteReplicationTask on resource: arn:aws:dms:us-east-1:152683116:task:RB7N24J2XBUPS3RFABZTG3

```
$ aws dms add-tags-to-resource  
--resource-name arn:aws:dms:us-east-1:152683116:task:RB7N24J2XBUPS3RFABZTG3  
--tags Key=CostCenter,Value=1234
```

A client error (AccessDeniedException) occurred when calling the AddTagsToResource operation: User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform: dms:AddTagsToResource on resource: arn:aws:dms:us-east-1:152683116:task:RB7N24J2XBUPS3RFABZTG3

```
$ aws dms remove-tags-from-resource  
--resource-name arn:aws:dms:us-east-1:152683116:task:RB7N24J2XBUPS3RFABZTG3  
--tag-keys Env
```

A client error (AccessDeniedException) occurred when calling the RemoveTagsFromResource operation: User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform: dms:RemoveTagsFromResource on resource: arn:aws:dms:us-east-1:152683116:task:RB7N24J2XBUPS3RFABZTG3

Setting an Encryption Key and Specifying KMS Permissions

AWS DMS encrypts the storage used by a replication instance and the endpoint connection information. To encrypt the storage used by a replication instance, AWS DMS uses an AWS Key Management Service (KMS) key that is unique to your AWS account. You can view and manage this key with KMS. You can use the default KMS key in your account (`aws/dms`) or you can create a custom KMS key. If you have an existing KMS key, you can also use that key for encryption.

The default KMS key (`aws/dms`) is created when you first launch a replication instance and you have not selected a custom KMS master key from the **Advanced** section of the **Create Replication Instance** page. If you use the default KMS key, the only permissions you need to grant to the IAM user account you are using for migration are `kms:ListAliases` and `kms:DescribeKey`. For more information about using the default KMS key, see [IAM Permissions Needed to Use AWS DMS \(p. 31\)](#).

To use a custom KMS key, assign permissions for the custom KMS key using one of the following options.

- Add the IAM user account used for the migration as a Key Administrator/Key User for the KMS custom key. This will ensure that necessary KMS permissions are granted to the IAM user account. Note that this action is in addition to the IAM permissions that you must grant to the IAM user account to use AWS DMS. For more information about granting permissions to a key user, see [Allows Key Users to Use the CMK](#).
- If you do not want to add the IAM user account as a Key Administrator/Key User for your custom KMS key, then add the following additional permissions to the IAM permissions that you must grant to the IAM user account to use AWS DMS.

```
{
    "Effect": "Allow",
    "Action": [
        "kms:ListAliases",
        "kms:DescribeKey",
        "kms:CreateGrant",
        "kms:Encrypt",
        "kms:ReEncrypt*"
    ],
    "Resource": "*"
},
```

AWS DMS does not work with KMS Key Aliases, but you can use the KMS key's Amazon Resource Number (ARN) when specifying the KMS key information. For more information on creating your own KMS keys and giving users access to a KMS key, see the [KMS Developer Guide](#).

If you don't specify a KMS key identifier, then AWS DMS uses your default encryption key. KMS creates the default encryption key for AWS DMS for your AWS account. Your AWS account has a different default encryption key for each AWS region.

To manage the KMS keys used for encrypting your AWS DMS resources, you use KMS. You can find KMS in the AWS Management Console by choosing **Identity & Access Management** on the console home page and then choosing **Encryption Keys** on the navigation pane. KMS combines secure, highly available hardware and software to provide a key management system scaled for the cloud. Using KMS, you can create encryption keys and define the policies that control how these keys can be used. KMS supports AWS CloudTrail, so you can audit key usage to verify that keys are being used appropriately. Your KMS keys can be used in combination with AWS DMS and supported AWS services such as Amazon RDS, Amazon Simple Storage Service (Amazon S3), Amazon Redshift, and Amazon Elastic Block Store (Amazon EBS).

Once you have created your AWS DMS resources with the KMS key, you cannot change the encryption key for those resources. Make sure to determine your encryption key requirements before you create your AWS DMS resources.

Network Security for AWS Database Migration Service

The security requirements for the network you create when using AWS Database Migration Service depend on how you configure the network. The general rules for network security for AWS DMS are as follows:

- The replication instance must have access to the source and target endpoints. The security group for the replication instance must have network ACLs or rules that allow egress from the instance out on the database port to the database endpoints.
- Database endpoints must include network ACLs and security group rules that allow incoming access from the replication instance. You can achieve this using the replication instance's security group, the private IP address, the public IP address, or the NAT gateway's public address, depending on your configuration.
- If your network uses a VPN Tunnel, the EC2 instance acting as the NAT Gateway must use a security group that has rules that allow the replication instance to send traffic through it.

By default, the VPC security group used by the AWS DMS replication instance has rules that allow egress to 0.0.0.0/0 on all ports. If you modify this security group or use your own security group, egress must, at a minimum, be permitted to the source and target endpoints on the respective database ports.

The network configurations you can use for database migration each require specific security considerations:

- [Configuration with All Database Migration Components in One VPC \(p. 66\)](#) — The security group used by the endpoints must allow ingress on the database port from the replication instance. Ensure that the security group used by the replication instance has ingress to the endpoints, or you can create a rule in the security group used by the endpoints that allows the private IP address of the replication instance access.
- [Configuration with Two VPCs \(p. 66\)](#) — The security group used by the replication instance must have a rule for the VPC range and the DB port on the database.
- [Configuration for a Network to a VPC Using AWS Direct Connect or a VPN \(p. 66\)](#) — a VPN tunnel allowing traffic to tunnel from the VPC into an on- premises VPN. In this configuration, the VPC includes a routing rule that sends traffic destined for a specific IP address or range to a host that can bridge traffic from the VPC into the on-premises VPN. If this case, the NAT host includes its own Security Group settings that must allow traffic from the Replication Instance's private IP address or security group into the NAT instance.
- [Configuration for a Network to a VPC Using the Internet \(p. 67\)](#) — The VPC security group must include routing rules that send traffic not destined for the VPC to the Internet gateway. In this configuration, the connection to the endpoint appears to come from the public IP address on the replication instance.
- [Configuration with an Amazon RDS DB instance not in a VPC to a DB instance in a VPC Using ClassicLink \(p. 67\)](#) — When the source or target Amazon RDS DB instance is not in a VPC and does not share a security group with the VPC where the replication instance is located, you can setup a proxy server and use ClassicLink to connect the source and target databases.
- **Source endpoint is outside the VPC used by the replication instance and uses a NAT gateway** — You can configure a network address translation (NAT) gateway using a single Elastic IP Address bound to a single Elastic Network Interface, which then receives a NAT identifier (nat-#####). If the VPC includes a default route to that NAT Gateway instead of the Internet Gateway, the replication instance will instead appear to contact the Database Endpoint using the public IP address of the Internet Gateway. In this case, the ingress to the Database Endpoint outside the VPC needs to allow ingress from the NAT address instead of the Replication Instance's public IP Address.

Using SSL With AWS Database Migration Service

You can encrypt connections for source and target endpoints by using Secure Sockets Layer (SSL). To do so, you can use the AWS DMS Management Console or AWS DMS API to assign a certificate to an endpoint. You can also use the AWS DMS console to manage your certificates.

Not all databases use SSL in the same way. Amazon Aurora with MySQL compatibility uses the server name, the endpoint of the primary instance in the cluster, as the endpoint for SSL. An Amazon Redshift endpoint already uses an SSL connection and does not require an SSL connection set up by AWS DMS. An Oracle endpoint requires additional steps; for more information, see [SSL Support for an Oracle Endpoint \(p. 50\)](#).

Topics

- [Limitations on Using SSL with AWS Database Migration Service \(p. 48\)](#)
- [Managing Certificates \(p. 48\)](#)
- [Enabling SSL for a MySQL-compatible, PostgreSQL, or SQL Server Endpoint \(p. 49\)](#)
- [SSL Support for an Oracle Endpoint \(p. 50\)](#)

To assign a certificate to an endpoint, you provide the root certificate or the chain of intermediate CA certificates leading up to the root (as a certificate bundle), that was used to sign the server SSL certificate that is deployed on your endpoint. Certificates are accepted as PEM formatted X509 files, only. When you import a certificate, you receive an Amazon Resource Name (ARN) that you can use to specify that certificate for an endpoint. If you use Amazon RDS, you can download the root CA and certificate bundle provided by Amazon RDS at <https://s3.amazonaws.com/rds-downloads/rds-combined-ca-bundle.pem>.

You can choose from several SSL modes to use for your SSL certificate verification.

- **none** – The connection is not encrypted. This option is not secure, but requires less overhead.
- **require** – The connection is encrypted using SSL (TLS) but no CA verification is made. This option is more secure, and requires more overhead.
- **verify-ca** – The connection is encrypted. This option is more secure, and requires more overhead. This option verifies the server certificate.
- **verify-full** – The connection is encrypted. This option is more secure, and requires more overhead. This option verifies the server certificate and verifies that the server hostname matches the hostname attribute for the certificate.

Not all SSL modes work with all database endpoints. The following table shows which SSL modes are supported for each database engine.

DB Engine	none	require	verify-ca	verify-full
MySQL/MariaDB/ Amazon Aurora MySQL	Default	Not supported	Supported	Supported
Microsoft SQL Server	Default	Supported	Not Supported	Supported
PostgreSQL	Default	Supported	Supported	Supported
Amazon Redshift	Default	SSL not enabled	SSL not enabled	SSL not enabled
Oracle	Default	Not supported	Supported	Not Supported
SAP ASE	Default	SSL not enabled	SSL not enabled	Supported

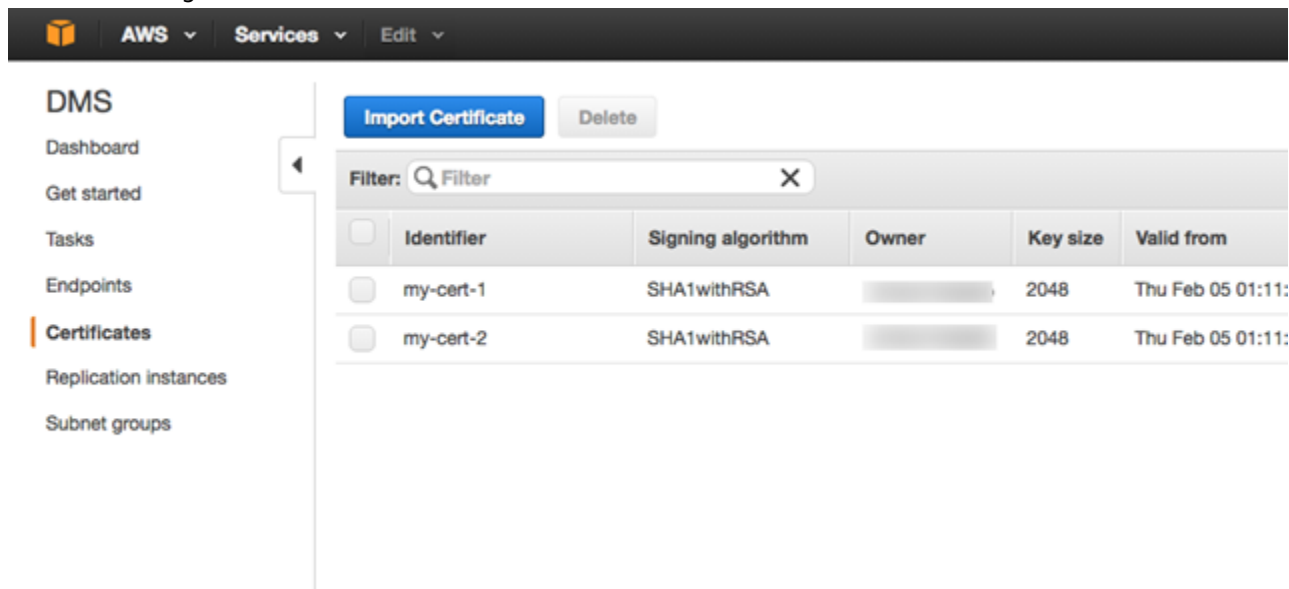
DB Engine	none	require	verify-ca	verify-full
MongoDB	Default	Supported	Not Supported	Supported
Db2 LUW	Default	Not Supported	Supported	Not Supported

Limitations on Using SSL with AWS Database Migration Service

- SSL connections to Amazon Redshift target endpoints are not supported. AWS DMS uses an Amazon S3 bucket to transfer data to the Redshift database. This transmission is encrypted by Amazon Redshift by default.
- SQL timeouts can occur when performing CDC tasks with SSL-enabled Oracle endpoints. If you have this issue, where CDC counters don't reflect the expected numbers, set the `MinimumTransactionSize` parameter from the `ChangeProcessingTuning` section of task settings to a lower value, starting with a value as low as 100. For more information about the `MinimumTransactionSize` parameter, see [Change Processing Tuning Settings \(p. 233\)](#).
- Certificates can only be imported in the .PEM and .SSO (Oracle wallet) formats.
- If your server SSL certificate is signed by an intermediate CA, make sure the entire certificate chain leading from the intermediate CA up to the root CA is imported as a single .PEM file.
- If you are using self-signed certificates on your server, choose **require** as your SSL mode. The **require** SSL mode implicitly trusts the server's SSL certificate and will not try to validate that the certificate was signed by a CA.

Managing Certificates

You can use the DMS console to view and manage your SSL certificates. You can also import your certificates using the DMS console.



Enabling SSL for a MySQL-compatible, PostgreSQL, or SQL Server Endpoint

You can add an SSL connection to a newly created endpoint or to an existing endpoint.

To create an AWS DMS endpoint with SSL

1. Sign in to the AWS Management Console and choose AWS Database Migration Service.

Note

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 for database migration, see [IAM Permissions Needed to Use AWS DMS \(p. 31\)](#).

2. In the navigation pane, choose **Certificates**.
3. Choose **Import Certificate**.
4. Upload the certificate you want to use for encrypting the connection to an endpoint.

Note

You can also upload a certificate using the AWS DMS console when you create or modify an endpoint by selecting **Add new CA certificate** on the **Create database endpoint** page.

5. Create an endpoint as described in [Step 3: Specify Source and Target Endpoints \(p. 22\)](#)

To modify an existing AWS DMS endpoint to use SSL:

1. Sign in to the AWS Management Console and choose AWS Database Migration Service.

Note

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 for database migration, see [IAM Permissions Needed to Use AWS DMS \(p. 31\)](#).

2. In the navigation pane, choose **Certificates**.
3. Choose **Import Certificate**.
4. Upload the certificate you want to use for encrypting the connection to an endpoint.

Note

You can also upload a certificate using the AWS DMS console when you create or modify an endpoint by selecting **Add new CA certificate** on the **Create database endpoint** page.

5. In the navigation pane, choose **Endpoints**, select the endpoint you want to modify, and choose **Modify**.
6. Choose an **SSL mode**.

If you select either the **verify-ca** or **verify-full** mode, you must specify the **CA certificate** that you want to use, as shown following.

Create database endpoint

A database endpoint is used by the replication server to connect to a database. The database specified in the endpoint can be on-prem. Details should be specified in the form below. It is recommended that you test your endpoint connections here to avoid errors during pr

Endpoint type* Source Target ⓘ

Endpoint identifier* ⓘ

Source engine* ⓘ

Server name*

Port*

SSL mode* ⓘ

CA certificate* ⓘ
[Add new CA certificate](#)

User name*

Password*

▶ Advanced

7. Choose **Modify**.
8. When the endpoint has been modified, select the endpoint and choose **Test connection** to determine if the SSL connection is working.

After you create your source and target endpoints, create a task that uses these endpoints. For more information on creating a task, see [Step 4: Create a Task \(p. 25\)](#).

SSL Support for an Oracle Endpoint

Oracle endpoints in AWS DMS support `none` and `verify-ca` SSL modes. To use SSL with an Oracle endpoint, you must upload the Oracle wallet for the endpoint instead of `.pem` certificate files.

Topics

- [Using an Existing Certificate for Oracle SSL \(p. 50\)](#)
- [Using a Self-Signed Certificate for Oracle SSL \(p. 51\)](#)

Using an Existing Certificate for Oracle SSL

To use an existing Oracle client installation to create the Oracle wallet file from the CA certificate file, do the following steps.

To use an existing Oracle client installation for Oracle SSL with AWS DMS

1. Set the ORACLE_HOME system variable to the location of your dbhome_1 directory by running the following command:

```
prompt>export ORACLE_HOME=/home/user/app/user/product/12.1.0/dbhome_1
```

2. Append \$ORACLE_HOME/lib to the LD_LIBRARY_PATH system variable.

```
prompt>export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib
```

3. Create a directory for the Oracle wallet at \$ORACLE_HOME/ssl_wallet.

```
prompt>mkdir $ORACLE_HOME/ssl_wallet
```

4. Put the CA certificate .pem file in the ssl_wallet directory. Amazon RDS customers can download the RDS CA certificates file from <https://s3.amazonaws.com/rds-downloads/rds-ca-2015-root.pem>.
5. Run the following commands to create the Oracle wallet:

```
prompt>orapki wallet create -wallet $ORACLE_HOME/ssl_wallet -auto_login_only  
prompt>orapki wallet add -wallet $ORACLE_HOME/ssl_wallet -trusted_cert -cert  
$ORACLE_HOME/ssl_wallet/ca-cert.pem -auto_login_only
```

When you have completed the steps previous, you can import the wallet file with the ImportCertificate API by specifying the certificate-wallet parameter. You can then use the imported wallet certificate when you select `verify-ca` as the SSL mode when creating or modifying your Oracle endpoint.

Note

Oracle wallets are binary files. AWS DMS accepts these files as-is.

Using a Self-Signed Certificate for Oracle SSL

To use a self-signed certificate for Oracle SSL, do the following.

To use a self-signed certificate for Oracle SSL with AWS DMS

1. Create a directory you will use to work with the self-signed certificate.

```
mkdir <SELF_SIGNED_CERT_DIRECTORY>
```

2. Change into the directory you created in the previous step.

```
cd <SELF_SIGNED_CERT_DIRECTORY>
```

3. Create a root key.

```
openssl genrsa -out self-rootCA.key 2048
```

- Self sign a root certificate using the root key you created in the previous step.

```
openssl req -x509 -new -nodes -key self-rootCA.key  
-sha256 -days 1024 -out self-rootCA.pem
```

- Create an Oracle wallet directory for the Oracle database.

```
mkdir $ORACLE_HOME/self_signed_ssl_wallet
```

- Create a new Oracle wallet.

```
orapki wallet create -wallet $ORACLE_HOME/self_signed_ssl_wallet  
-pwd <password> -auto_login_local
```

- Add the root certificate to the Oracle wallet.

```
orapki wallet add -wallet $ORACLE_HOME/self_signed_ssl_wallet  
-trusted_cert -cert self-rootCA.pem -pwd <password>
```

- List the contents of the Oracle wallet. The list should include the root certificate.

```
orapki wallet display -wallet $ORACLE_HOME/self_signed_ssl_wallet
```

- Generate the Certificate Signing Request (CSR) using the ORAPKI utility.

```
orapki wallet add -wallet $ORACLE_HOME/self_signed_ssl_wallet  
-dn "CN=dms" -keysize 2048 -sign_alg sha256 -pwd <password>
```

- Run the following command.

```
openssl pkcs12 -in ewallet.p12 -nodes -out nonoracle_wallet.pem
```

- Put 'dms' as the common name.

```
openssl req -new -key nonoracle_wallet.pem -out certrequest.csr
```

- Get the certificate signature.

```
openssl req -noout -text -in self-signed-oracle.csr | grep -i signature
```

- If the output from step 12 is sha256WithRSAEncryption, then run the following code.

```
openssl x509 -req -in self-signed-oracle.csr -CA self-rootCA.pem  
-CAkey self-rootCA.key -CAcreateserial  
-out self-signed-oracle.crt -days 365 -sha256
```

- If the output from step 12 is md5WithRSAEncryption, then run the following code.

```
openssl x509 -req -in certrequest.csr -CA self-rootCA.pem  
-CAkey self-rootCA.key -CAcreateserial  
-out certrequest.crt -days 365 -sha256
```

- Add the certificate to the wallet.

```
orapki wallet add -wallet $ORACLE_HOME/self_signed_ssl_wallet -user_cert  
-cert certrequest.crt -pwd <password>
```

16. Configure *sqlnet.ora* file ($\$ORACLE_HOME/network/admin/sqlnet.ora$).

```
WALLET_LOCATION =
  (SOURCE =
    (METHOD = FILE)
    (METHOD_DATA =
      (DIRECTORY = <ORACLE_HOME>/self_signed_ssl_wallet)
    )
  )

SQLNET.AUTHENTICATION_SERVICES = (NONE)
SSL_VERSION = 1.0
SSL_CLIENT_AUTHENTICATION = FALSE
SSL_CIPHER_SUITES = (SSL_RSA_WITH_AES_256_CBC_SHA)
```

17. Stop the Oracle listener.

```
lsnrctl stop
```

18. Add entries for SSL in the *listener.ora* file ($\$ORACLE_HOME/network/admin/listener.ora$).

```
SSL_CLIENT_AUTHENTICATION = FALSE
WALLET_LOCATION =
  (SOURCE =
    (METHOD = FILE)
    (METHOD_DATA =
      (DIRECTORY = <ORACLE_HOME>/self_signed_ssl_wallet)
    )
  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = <SID>)
      (ORACLE_HOME = <ORACLE_HOME>)
      (SID_NAME = <SID>)
    )
  )

LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = localhost.localdomain)(PORT = 1521))
      (ADDRESS = (PROTOCOL = TCPS)(HOST = localhost.localdomain)(PORT = 1522))
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
    )
  )
```

19. Configure the *tnsnames.ora* file ($\$ORACLE_HOME/network/admin/tnsnames.ora$).

```
<SID>=
(DESCRIPTION=
  (ADDRESS_LIST =
    (ADDRESS=(PROTOCOL = TCP)(HOST = localhost.localdomain)(PORT = 1521))
  )
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = <SID>)
  )
)

<SID>_ssl=
(DESCRIPTION=
```

```
(ADDRESS_LIST =
  (ADDRESS=(PROTOCOL = TCPS)(HOST = localhost.localdomain)(PORT = 1522))
)
(CONNECT_DATA =
  (SERVER = DEDICATED)
  (SERVICE_NAME = <SID>)
)
)
```

20. Restart the Oracle listener.

```
lsnrctl start
```

21. Show the Oracle listener status.

```
lsnrctl status
```

22. Test the SSL connection to the database from localhost using sqlplus and the SSL tnsnames entry.

```
sqlplus -L <ORACLE_USER>@<SID>_ssl
```

23. Verify that you successfully connected using SSL.

```
SELECT SYS_CONTEXT('USERENV', 'network_protocol') FROM DUAL;

SYS_CONTEXT('USERENV', 'NETWORK_PROTOCOL')
-----
tcps
```

24. Change directory to the directory with the self-signed certificate.

```
cd <SELF_SIGNED_CERT_DIRECTORY>
```

25. Create a new client Oracle wallet that AWS DMS will use.

```
orapki wallet create -wallet ./ -auto_login_only
```

26. Add the self-signed root certificate to the Oracle wallet.

```
orapki wallet add -wallet ./ -trusted_cert -cert rootCA.pem -auto_login_only
```

27. List the contents of the Oracle wallet that AWS DMS will use. The list should include the self-signed root certificate.

```
orapki wallet display -wallet ./
```

28. Upload the Oracle wallet you just created to AWS DMS.

Changing the Database Password

In most situations, changing the database password for your source or target endpoint is straightforward. If you need to change the database password for an endpoint that you are currently using in a migration or replication task, the process is slightly more complex. The procedure following shows how to do this.

To change the database password for an endpoint in a migration or replication task

1. Sign in to the AWS Management Console and choose AWS DMS. 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 DMS. For more information on the permissions required, see [IAM Permissions Needed to Use AWS DMS \(p. 31\)](#).
2. In the navigation pane, choose **Tasks**.
3. Choose the task that uses the endpoint you want to change the database password for, and then choose **Stop**.
4. While the task is stopped, you can change the password of the database for the endpoint using the native tools you use to work with the database.
5. Return to the DMS Management Console and choose **Endpoints** from the navigation pane.
6. Choose the endpoint for the database you changed the password for, and then choose **Modify**.
7. Type the new password in the **Password** box, and then choose **Modify**.
8. Choose **Tasks** from the navigation pane.
9. Choose the task that you stopped previously, and choose **Start/Resume**.
10. Choose either **Start** or **Resume**, depending on how you want to continue the task, and then choose **Start task**.

Limits for AWS Database Migration Service

Following, you can find the resource limits and naming constraints for AWS Database Migration Service (AWS DMS).

The maximum size of a database that AWS DMS can migrate depends on your source environment, the distribution of data in your source database, and how busy your source system is. The best way to determine whether your particular system is a candidate for AWS DMS is to test it out. Start slowly so you can get the configuration worked out, then add some complex objects, and finally, attempt a full load as a test.

Limits for AWS Database Migration Service

Each AWS account has limits, per region, on the number of AWS DMS resources that can be created. Once a limit for a resource has been reached, additional calls to create that resource will fail with an exception.

The 6 TB limit for storage applies to the DMS replication instance. This storage is used to cache changes if the target cannot keep up with the source and for storing log information. This limit does not apply to the target size; target endpoints can be larger than 6 TB.

The following table lists the AWS DMS resources and their limits per region.

Resource	Default Limit
Replication instances	20
Total amount of storage	6 TB
Event subscriptions	20
Replication subnet groups	20
Subnets per replication subnet group	20
Endpoints	100
Tasks	200
Endpoints per instance	20

Working with an AWS DMS Replication Instance

When you create an AWS DMS replication instance, AWS DMS creates the replication instance on an Amazon Elastic Compute Cloud (Amazon EC2) instance in a VPC based on the Amazon Virtual Private Cloud (Amazon VPC) service. You use this replication instance to perform your database migration. The replication instance provides high availability and failover support using a Multi-AZ deployment when you select the **Multi-AZ** option.

In a Multi-AZ deployment, AWS DMS automatically provisions and maintains a synchronous standby replica of the replication instance in a different Availability Zone. The primary replication instance is synchronously replicated across Availability Zones to a standby replica. This approach provides data redundancy, eliminates I/O freezes, and minimizes latency spikes.



AWS DMS uses a replication instance to connect to your source data store, read the source data, and format the data for consumption by the target data store. A replication instance also loads the data into the target data store. Most of this processing happens in memory. However, large transactions might require some buffering on disk. Cached transactions and log files are also written to disk.

You can create an AWS DMS replication instance in the following AWS Regions.

Region	Name
Asia Pacific (Tokyo) Region	ap-northeast-1
Asia Pacific (Seoul) Region	ap-northeast-2
Asia Pacific (Mumbai) Region	ap-south-1
Asia Pacific (Singapore) Region	ap-southeast-1
Asia Pacific (Sydney) Region	ap-southeast-2
Canada (Central) Region	ca-central-1
EU (Frankfurt) Region	eu-central-1
EU (Ireland) Region	eu-west-1
EU (London) Region	eu-west-2
South America (São Paulo) Region	sa-east-1
US East (N. Virginia) Region	us-east-1

Region	Name
US East (Ohio) Region	us-east-2
US West (N. California) Region	us-west-1
US West (Oregon) Region	us-west-2

AWS DMS supports a special AWS Region called AWS GovCloud (US) that is designed to allow US government agencies and customers to move more sensitive workloads into the cloud. AWS GovCloud (US) addresses the US government's specific regulatory and compliance requirements. For more information about AWS GovCloud (US), see [What Is AWS GovCloud \(US\)?](#)

Following, you can find out more details about replication instances.

Topics

- [Selecting the Right AWS DMS Replication Instance for Your Migration \(p. 58\)](#)
- [Public and Private Replication Instances \(p. 60\)](#)
- [AWS DMS Maintenance \(p. 60\)](#)
- [Working with Replication Engine Versions \(p. 63\)](#)
- [Setting Up a Network for a Replication Instance \(p. 65\)](#)
- [Setting an Encryption Key for a Replication Instance \(p. 71\)](#)
- [Creating a Replication Instance \(p. 72\)](#)
- [Modifying a Replication Instance \(p. 76\)](#)
- [Rebooting a Replication Instance \(p. 78\)](#)
- [Deleting a Replication Instance \(p. 80\)](#)
- [DDL Statements Supported by AWS DMS \(p. 81\)](#)

Selecting the Right AWS DMS Replication Instance for Your Migration

AWS DMS creates the replication instance on an Amazon Elastic Compute Cloud (Amazon EC2) instance. AWS DMS currently supports the T2, C4, and R4 Amazon EC2 instance classes for replication instances:

- The T2 instance classes are low-cost standard instances designed to provide a baseline level of CPU performance with the ability to burst above the baseline. They are suitable for developing, configuring, and testing your database migration process. They also work well for periodic data migration tasks that can benefit from the CPU burst capability.
- The C4 instance classes are designed to deliver the highest level of processor performance for computer-intensive workloads. They achieve significantly higher packet per second (PPS) performance, lower network jitter, and lower network latency. AWS DMS can be CPU-intensive, especially when performing heterogeneous migrations and replications such as migrating from Oracle to PostgreSQL. C4 instances can be a good choice for these situations.
- The R4 instance classes are memory optimized for memory-intensive workloads. Ongoing migrations or replications of high-throughput transaction systems using DMS can, at times, consume large amounts of CPU and memory. R4 instances include more memory per vCPU.

Each replication instance has a specific configuration of memory and vCPU. The following table shows the configuration for each replication instance type. For pricing information, see the [AWS Database Migration Service pricing page](#).

Replication Instance Type	vCPU	Memory (GB)
General Purpose		
dms.t2.micro	1	1
dms.t2.small	1	2
dms.t2.medium	2	4
dms.t2.large	2	8
Compute Optimized		
dms.c4.large	2	3.75
dms.c4.xlarge	4	7.5
dms.c4.2xlarge	8	15
dms.c4.4xlarge	16	30
Memory Optimized		
dms.r4.large	2	15.25
dms.r4.xlarge	4	30.5
dms.r4.2xlarge	8	61
dms.r4.4xlarge	16	122
dms.r4.8xlarge	32	244

To help you determine which replication instance class would work best for your migration, let's look at the change data capture (CDC) process that the AWS DMS replication instance uses.

Let's assume that you're running a full load plus CDC task (bulk load plus ongoing replication). In this case, the task has its own SQLite repository to store metadata and other information. Before AWS DMS starts a full load, these steps occur:

- AWS DMS starts capturing changes for the tables it's migrating from the source engine's transaction log (we call these *cached changes*). After full load is done, these cached changes are collected and applied on the target. Depending on the volume of cached changes, these changes can directly be applied from memory, where they are collected first, up to a set threshold. Alternatively, they can be applied from disk, where changes are written when they can't be held in memory.
- After cached changes are applied, by default AWS DMS starts a transactional apply on the target instance.

During the applied cached changes phase and ongoing replications phase, AWS DMS uses two stream buffers, one each for incoming and outgoing data. AWS DMS also uses an important component called a sorter, which is another memory buffer. Following are two important uses of the sorter component (which has others):

- It tracks all transactions and makes sure that it forwards only relevant transactions to the outgoing buffer.
- It makes sure that transactions are forwarded in the same commit order as on the source.

As you can see, we have three important memory buffers in this architecture for CDC in AWS DMS. If any of these buffers experience memory pressure, the migration can have performance issues that can potentially cause failures.

When you plug heavy workloads with a high number of transactions per second (TPS) into this architecture, you can find the extra memory provided by R4 instances useful. You can use R4 instances to hold a large number of transactions in memory and prevent memory-pressure issues during ongoing replications.

Public and Private Replication Instances

You can specify whether a replication instance has a public or private IP address that the instance uses to connect to the source and target databases.

A private replication instance has a private IP address that you can't access outside the replication network. A replication instance should have a private IP address when both source and target databases are in the same network that is connected to the replication instance's VPC by using a VPN, AWS Direct Connect, or VPC peering.

A *VPC peering* connection is a networking connection between two VPCs that enables routing using each VPC's private IP addresses as if they were in the same network. For more information about VPC peering, see [VPC Peering](#) in the *Amazon VPC User Guide*.

AWS DMS Maintenance

Periodically, AWS DMS performs maintenance on AWS DMS resources. Maintenance most often involves updates to the replication instance or the replication instance's operating system (OS). You can manage the time period for your maintenance window and see maintenance updates using the AWS CLI or AWS DMS API. The AWS DMS console is not currently supported for this work.

Maintenance items require that AWS DMS take your replication instance offline for a short time. Maintenance that requires a resource to be offline includes required operating system or instance patching. Required patching is automatically scheduled only for patches that are related to security and instance reliability. Such patching occurs infrequently (typically once or twice a year) and seldom requires more than a fraction of your maintenance window. You can have minor version updates applied automatically by choosing the **Auto minor version upgrade** console option.

AWS DMS Maintenance Window

Every AWS DMS replication instance has a weekly maintenance window during which any available system changes are applied. You can think of the maintenance window as an opportunity to control when modifications and software patching occurs.

If AWS DMS determines that maintenance is required during a given week, the maintenance occurs during the 30-minute maintenance window you chose when you created the replication instance. AWS DMS completes most maintenance during the 30-minute maintenance window. However, a longer time might be required for larger changes.

The 30-minute maintenance window that you selected when you created the replication instance is from an 8-hour block of time allocated for each AWS Region. If you don't specify a preferred maintenance window when you create your replication instance, AWS DMS assigns one on a randomly selected day of the week. For a replication instance that uses a Multi-AZ deployment, a failover might be required for maintenance to be completed.

The following table lists the maintenance window for each AWS Region that supports AWS DMS.

Region	Time Block
Asia Pacific (Sydney) Region	12:00–20:00 UTC
Asia Pacific (Tokyo) Region	13:00–21:00 UTC
Asia Pacific (Mumbai) Region	17:30–01:30 UTC
Asia Pacific (Seoul) Region	13:00–21:00 UTC
Asia Pacific (Singapore) Region	14:00–22:00 UTC
Canada (Central) Region	06:29–14:29 UTC
EU (Frankfurt) Region	23:00–07:00 UTC
EU (Ireland) Region	22:00–06:00 UTC
EU (London) Region	06:00–14:00 UTC
South America (São Paulo) Region	00:00–08:00 UTC
US East (N. Virginia) Region	03:00–11:00 UTC
US East (Ohio) Region	03:00–11:00 UTC
US West (N. California) Region	06:00–14:00 UTC
US West (Oregon) Region	06:00–14:00 UTC
AWS GovCloud (US-West)	06:00–14:00 UTC

Effect of Maintenance on Existing Migration Tasks

When an AWS DMS migration task is running on an instance, the following events occur when a patch is applied:

- If the tables in the migration task are in the replicating ongoing changes phase (CDC), AWS DMS pauses the task for a moment while the patch is applied. The migration then continues from where it was interrupted when the patch was applied.
- If AWS DMS is migrating a table when the patch is applied, AWS DMS restarts the migration for the table.

Changing the Maintenance Window Setting

You can change the maintenance window time frame using the AWS Management Console, the AWS CLI, or the AWS DMS API.

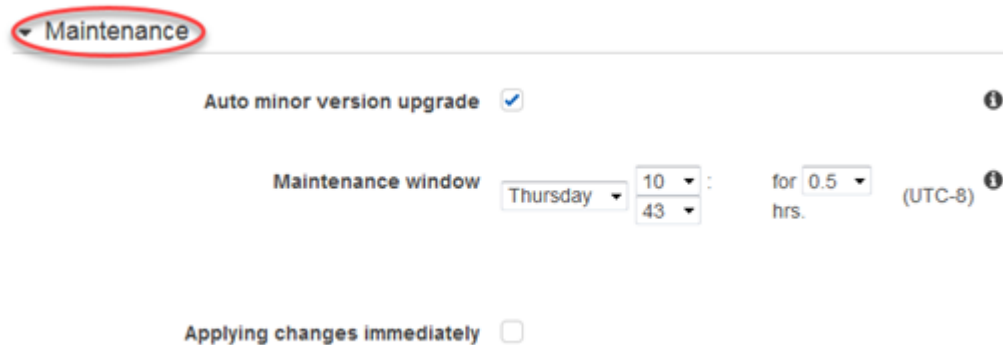
Changing the Maintenance Window Setting Using the AWS Console

You can change the maintenance window time frame using the AWS Management Console.

To change the preferred maintenance window using the AWS console

1. Sign in to the AWS Management Console and choose AWS DMS.

2. In the navigation pane, choose **Replication instances**.
3. Choose the replication instance you want to modify and choose **Modify**.
4. Expand the **Maintenance** section and choose a date and time for your maintenance window.



5. Choose **Apply changes immediately**.
6. Choose **Modify**.

Changing the Maintenance Window Setting Using the CLI

To adjust the preferred maintenance window, use the AWS CLI `modify-replication-instance` command with the following parameters.

- `--replication-instance-identifier`
- `--preferred-maintenance-window`

Example

The following AWS CLI example sets the maintenance window to Tuesdays from 4:00–4:30 a.m. UTC.

```
aws dms modify-replication-instance \  
--replication-instance-identifier myrepinstance \  
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

Changing the Maintenance Window Setting Using the API

To adjust the preferred maintenance window, use the AWS DMS API `ModifyReplicationInstance` action with the following parameters.

- `ReplicationInstanceIdentifier` = *myrepinstance*
- `PreferredMaintenanceWindow` = *Tue:04:00-Tue:04:30*

Example

The following code example sets the maintenance window to Tuesdays from 4:00–4:30 a.m. UTC.

```
https://dms.us-west-2.amazonaws.com/  
?Action=ModifyReplicationInstance  
&DBInstanceIdentifier=myrepinstance  
&PreferredMaintenanceWindow=Tue:04:00-Tue:04:30  
&SignatureMethod=HmacSHA256  
&SignatureVersion=4
```

```
&Version=2014-09-01
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140425/us-east-1/dms/aws4_request
&X-Amz-Date=20140425T192732Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=1dc9dd716f4855e9bdf188c70f1cf9f6251b070b68b81103b59ec70c3e7854b3
```

Working with Replication Engine Versions

The *replication engine* is the core AWS DMS software that runs on your replication instance and performs the migration tasks you specify. AWS periodically releases new versions of the AWS DMS replication engine software, with new features and performance improvements. Each version of the replication engine software has its own version number, to distinguish it from other versions.

When you launch a new replication instance, it runs the latest AWS DMS engine version unless you specify otherwise. For more information, see [Working with an AWS DMS Replication Instance \(p. 57\)](#).

If you have a replication instance that is currently running, you can upgrade it to a more recent engine version. (AWS DMS doesn't support engine version downgrades.) For more information, including a list of replication engine versions, see the following section.

Deprecating a Replication Instance Version

Occasionally AWS DMS deprecates older versions of the replication instance. Beginning April 2, 2018, AWS DMS will disable creation of any new replication instance version 1.9.0. This version was initially supported in AWS DMS on March 15, 2016, and has since been replaced by subsequent versions containing improvements to functionality, security, and reliability.

Beginning on August 5, 2018, at 0:00 UTC, all DMS replication instances running version 1.9.0 will be scheduled for automatic upgrade to the latest available version during the maintenance window specified for each instance. We recommend that you upgrade your instances before that time, at a time that is convenient for you.

You can initiate an upgrade of your replication instance by using the instructions in the following section, [Upgrading the Engine Version of a Replication Instance \(p. 63\)](#).

For migration tasks that are running when you choose to upgrade the replication instance, tables in the full load phase at the time of the upgrade are reloaded from the start once the upgrade is complete. Replication for all other tables should resume without interruption once the upgrade is complete. We recommend testing all current migration tasks on the latest available version of AWS DMS replication instance before upgrading the instances from version 1.9.0.

Upgrading the Engine Version of a Replication Instance

AWS periodically releases new versions of the AWS DMS replication engine software, with new features and performance improvements. The following is a summary of available AWS DMS engine versions.

Version	Summary
2.4.x	<ul style="list-style-type: none">• Support for replication of Oracle index tablespaces.• Support for canned access ACLs to support cross account access with Amazon S3 endpoints.

Version	Summary
2.3.x	<ul style="list-style-type: none">• Support for Amazon S3 as an AWS DMS source.• Support for replication of tablespaces for Oracle as an AWS DMS source only.• Support for Oracle active data guard standby as a source for Oracle as an AWS DMS source only.
2.2.x	<ul style="list-style-type: none">• Support for Microsoft SQL Server 2016, as either an AWS DMS source or an AWS DMS target.• Support for SAP ASE 16, as either an AWS DMS source or an AWS DMS target.• Support for Microsoft SQL Server running on Microsoft Azure, as an AWS DMS source only. You can perform a full migration of existing data; however, change data capture (CDC) is not available.
1.9.x	Cumulative release of AWS DMS replication engine software.

Upgrading the Engine Version Using the Console

You can upgrade an AWS DMS replication instance using the AWS Management Console.

To upgrade a replication instance using the console

1. Open the AWS DMS console at <https://console.aws.amazon.com/dms/>.
2. In the navigation pane, choose **Replication instances**.
3. Choose your replication engine, and then choose **Modify**.
4. For **Replication engine version**, choose the version number you want, and then choose **Modify**.

Note

Upgrading the replication instance takes several minutes. When the instance is ready, its status changes to **available**.

Upgrading the Engine Version Using the CLI

You can upgrade an AWS DMS replication instance using the AWS CLI, as follows.

To upgrade a replication instance using the AWS CLI

1. Determine the Amazon Resource Name (ARN) of your replication instance by using the following command.

```
aws dms describe-replication-instances \
--query "ReplicationInstances[*].
[ReplicationInstanceIdentifier,ReplicationInstanceArn,ReplicationInstanceClass]"
```

In the output, take note of the ARN for the replication instance you want to upgrade, for example:
arn:aws:dms:us-east-1:123456789012:rep:6EFQQO6U6EDPRCPKLNPL2SCEEY

2. Determine which replication instance versions are available by using the following command.

```
aws dms describe-orderable-replication-instances \
--query "OrderableReplicationInstances[*].[ReplicationInstanceClass,EngineVersion]"
```

In the output, take note of the engine version number or numbers that are available for your replication instance class. You should see this information in the output from step 1.

3. Upgrade the replication instance by using the following command.

```
aws dms modify-replication-instance \  
--replication-instance-arn arn \  
--engine-version n.n.n
```

Replace *arn* in the preceding with the actual replication instance ARN from the previous step.

Replace *n.n.n* with the engine version number that you want, for example: 2.2.1

Note

Upgrading the replication instance takes several minutes. You can view the replication instance status using the following command.

```
aws dms describe-replication-instances \  
--query "ReplicationInstances[*].  
[ReplicationInstanceIdentifier,ReplicationInstanceStatus]"
```

When the replication instance is ready, its status changes to **available**.

Setting Up a Network for a Replication Instance

AWS DMS always creates the replication instance in a VPC based on Amazon Virtual Private Cloud (Amazon VPC). You specify the VPC where your replication instance is located. You can use your default VPC for your account and AWS Region, or you can create a new VPC. The VPC must have two subnets in at least one Availability Zone.

The Elastic Network Interface (ENI) allocated for the replication instance in your VPC must be associated with a security group that has rules that allow all traffic on all ports to leave (egress) the VPC. This approach allows communication from the replication instance to your source and target database endpoints, as long as correct egress rules are enabled on the endpoints. We recommend that you use the default settings for the endpoints, which allows egress on all ports to all addresses.

The source and target endpoints access the replication instance that is inside the VPC either by connecting to the VPC or by being inside the VPC. The database endpoints must include network access control lists (ACLs) and security group rules (if applicable) that allow incoming access from the replication instance. Depending on the network configuration you are using, you can use the replication instance VPC security group, the replication instance's private or public IP address, or the NAT gateway's public IP address. These connections form a network that you use for data migration.

Network Configurations for Database Migration

You can use several different network configurations with AWS Database Migration Service. The following are common configurations for a network used for database migration.

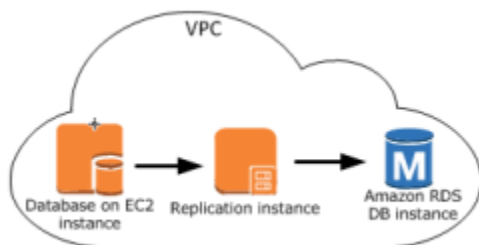
Topics

- [Configuration with All Database Migration Components in One VPC \(p. 66\)](#)
- [Configuration with Two VPCs \(p. 66\)](#)
- [Configuration for a Network to a VPC Using AWS Direct Connect or a VPN \(p. 66\)](#)
- [Configuration for a Network to a VPC Using the Internet \(p. 67\)](#)
- [Configuration with an Amazon RDS DB instance not in a VPC to a DB instance in a VPC Using ClassicLink \(p. 67\)](#)

Configuration with All Database Migration Components in One VPC

The simplest network for database migration is for the source endpoint, the replication instance, and the target endpoint to all be in the same VPC. This configuration is a good one if your source and target endpoints are on an Amazon RDS DB instance or an Amazon EC2 instance.

The following illustration shows a configuration where a database on an Amazon EC2 instance connects to the replication instance and data is migrated to an Amazon RDS DB instance.



The VPC security group used in this configuration must allow ingress on the database port from the replication instance. You can do this by either ensuring that the security group used by the replication instance has ingress to the endpoints, or by explicitly allowing the private IP address of the replication instance.

Configuration with Two VPCs

If your source endpoint and target endpoints are in different VPCs, you can create your replication instance in one of the VPCs and then link the two VPCs by using VPC peering.

A VPC peering connection is a networking connection between two VPCs that enables routing using each VPC's private IP addresses as if they were in the same network. We recommend this method for connecting VPCs within an AWS Region. You can create VPC peering connections between your own VPCs or with a VPC in another AWS account within the same AWS Region. For more information about VPC peering, see [VPC Peering](#) in the *Amazon VPC User Guide*.

The following illustration shows an example configuration using VPC peering. Here, the source database on an Amazon EC2 instance in a VPC connects by VPC peering to a VPC. This VPC contains the replication instance and the target database on an Amazon RDS DB instance.



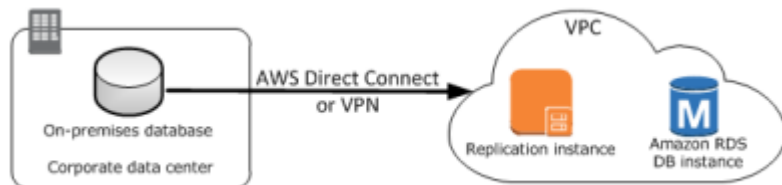
The VPC security groups used in this configuration must allow ingress on the database port from the replication instance.

Configuration for a Network to a VPC Using AWS Direct Connect or a VPN

Remote networks can connect to a VPC using several options such as AWS Direct Connect or a software or hardware VPN connection. These options are often used to integrate existing on-site services, such

as monitoring, authentication, security, data, or other systems, by extending an internal network into the AWS cloud. By using this type of network extension, you can seamlessly connect to AWS-hosted resources such as a VPC.

The following illustration shows a configuration where the source endpoint is an on-premises database in a corporate data center. It is connected by using AWS Direct Connect or a VPN to a VPC that contains the replication instance and a target database on an Amazon RDS DB instance.



In this configuration, the VPC security group must include a routing rule that sends traffic destined for a specific IP address or range to a host. This host must be able to bridge traffic from the VPC into the on-premises VPN. In this case, the NAT host includes its own security group settings that must allow traffic from the replication instance's private IP address or security group into the NAT instance.

Configuration for a Network to a VPC Using the Internet

If you don't use a VPN or AWS Direct Connect to connect to AWS resources, you can use the Internet to migrate a database to an Amazon EC2 instance or Amazon RDS DB instance. This configuration involves a public replication instance in a VPC with an internet gateway that contains the target endpoint and the replication instance.



To add an Internet gateway to your VPC, see [Attaching an Internet Gateway](#) in the *Amazon VPC User Guide*.

The VPC security group must include routing rules that send traffic not destined for the VPC by default to the Internet gateway. In this configuration, the connection to the endpoint appears to come from the public IP address of the replication instance, not the private IP address.

Configuration with an Amazon RDS DB instance not in a VPC to a DB instance in a VPC Using ClassicLink

You can use ClassicLink with a proxy server to connect an Amazon RDS DB instance that is not in a VPC to an AWS DMS replication server and DB instance that reside in a VPC.

ClassicLink allows you to link an EC2-Classic DB instance to a VPC in your account, within the same AWS Region. After you've created the link, the source DB instance can communicate with the replication instance inside the VPC using their private IP addresses.

Because the replication instance in the VPC cannot directly access the source DB instance on the EC2-Classic platform using ClassicLink, you must use a proxy server. The proxy server connects the source DB instance to the VPC containing the replication instance and target DB instance. The proxy server uses ClassicLink to connect to the VPC. Port forwarding on the proxy server allows communication between the source DB instance and the target DB instance in the VPC.



Using ClassicLink with AWS Database Migration Service

You can use ClassicLink, in conjunction with a proxy server, to connect an Amazon RDS DB instance that is not in a VPC to a AWS DMS replication server and DB instance that reside in a VPC.

The following procedure shows how to use ClassicLink to connect an Amazon RDS source DB instance that is not in a VPC to a VPC containing an AWS DMS replication instance and a target DB instance.

- Create an AWS DMS replication instance in a VPC. (All replication instances are created in a VPC).
- Associate a VPC security group to the replication instance and the target DB instance. When two instances share a VPC security group, they can communicate with each other by default.
- Set up a proxy server on an EC2 Classic instance.
- Create a connection using ClassicLink between the proxy server and the VPC.
- Create AWS DMS endpoints for the source and target databases.
- Create an AWS DMS task.

To use ClassicLink to migrate a database on a DB instance not in a VPC to a database on a DB instance in a VPC

1. Step 1: Create an AWS DMS replication instance.

To create a AWS DMS replication instance and assign a VPC security group:

- a. Sign in to the AWS Management Console and choose AWS Database Migration Service. 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 DMS. For more information on the permissions required for database migration, see [IAM Permissions Needed to Use AWS DMS \(p. 31\)](#).
 - b. On the **Dashboard** page, choose **Replication Instance**. Follow the instructions at [Step 2: Create a Replication Instance \(p. 18\)](#) to create a replication instance.
 - c. After you have created the AWS DMS replication instance, open the EC2 service console. Select **Network Interfaces** from the navigation pane.
 - d. Select the *DMSNetworkInterface*, and then choose **Change Security Groups** from the **Actions** menu.
 - e. Select the security group you want to use for the replication instance and the target DB instance.
2. Step 2: Associate the security group from the last step with the target DB instance.

To associate a security group with a DB instance

- a. Open the Amazon RDS service console. Select **Instances** from the navigation pane.
- b. Select the target DB instance. From **Instance Actions**, select **Modify**.
- c. For the **Security Group** parameter, select the security group you used in the previous step.
- d. Select **Continue**, and then **Modify DB Instance**.

3. Step 3: Set up a proxy server on an EC2 Classic instance using NGINX. Use an AMI of your choice to launch an EC2 Classic instance. The example below is based on the AMI Ubuntu Server 14.04 LTS (HVM).

To set up a proxy server on an EC2 Classic instance

- a. Connect to the EC2 Classic instance and install NGINX using the following commands:

```
Prompt> sudo apt-get update
Prompt> sudo wget http://nginx.org/download/nginx-1.9.12.tar.gz
Prompt> sudo tar -xvzf nginx-1.9.12.tar.gz
Prompt> cd nginx-1.9.12
Prompt> sudo apt-get install build-essential
Prompt> sudo apt-get install libpcre3 libpcre3-dev
Prompt> sudo apt-get install zlib1g-dev
Prompt> sudo ./configure --with-stream
Prompt> sudo make
Prompt> sudo make install
```

- b. Edit the NGINX daemon file, `/etc/init/nginx.conf`, using the following code:

```
# /etc/init/nginx.conf - Upstart file

description "nginx http daemon"
author "email"

start on (filesystem and net-device-up IFACE=lo)
stop on runlevel [!2345]

env DAEMON=/usr/local/nginx/sbin/nginx
env PID=/usr/local/nginx/logs/nginx.pid

expect fork
respawn
respawn limit 10 5

pre-start script
    $DAEMON -t
    if [ $? -ne 0 ]
        then exit $?
    fi
end script

exec $DAEMON
```

- c. Create an NGINX configuration file at `/usr/local/nginx/conf/nginx.conf`. In the configuration file, add the following:

```
# /usr/local/nginx/conf/nginx.conf - NGINX configuration file

worker_processes 1;

events {
    worker_connections 1024;
}

stream {
    server {
```

```
listen <DB instance port number>;
proxy_pass <DB instance identifier>:<DB instance port number>;
}
}
```

- d. From the command line, start NGINX using the following commands:

```
Prompt> sudo initctl reload-configuration
Prompt> sudo initctl list | grep nginx
Prompt> sudo initctl start nginx
```

4. Step 4: Create a ClassicLink connection between the proxy server and the target VPC that contains the target DB instance and the replication instance

Use ClassicLink to connect the proxy server with the target VPC

- a. Open the EC2 console and select the EC2 Classic instance that is running the proxy server.
 - b. Select **ClassicLink** under **Actions**, then select **Link to VPC**.
 - c. Select the security group you used earlier in this procedure.
 - d. Select **Link to VPC**.
5. Step 5: Create AWS DMS endpoints using the procedure at [Step 3: Specify Source and Target Endpoints \(p. 22\)](#). You must use the internal EC2 DNS hostname of the proxy as the server name when specifying the source endpoint.
6. Step 6: Create a AWS DMS task using the procedure at [Step 4: Create a Task \(p. 25\)](#).

Creating a Replication Subnet Group

As part of the network to use for database migration, you need to specify what subnets in your Amazon Virtual Private Cloud (Amazon VPC) you plan to use. A *subnet* is a range of IP addresses in your VPC in a given Availability Zone. These subnets can be distributed among the Availability Zones for the AWS Region where your VPC is located.

You create a replication instance in a subnet that you select, and you can manage what subnet a source or target endpoint uses by using the AWS DMS console.

You create a replication subnet group to define which subnets to use. You must specify at least one subnet in two different Availability Zones.

To create a replication subnet group

1. Sign in to the AWS Management Console and choose AWS Database Migration Service. 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 for database migration, see [IAM Permissions Needed to Use AWS DMS \(p. 31\)](#).
2. In the navigation pane, choose **Subnet Groups**.
3. Choose **Create Subnet Group**.
4. On the **Edit Replication Subnet Group** page, shown following, specify your replication subnet group information. The following table describes the settings.

Edit Replication Subnet Group

Identifier ⓘ

Description ⓘ

VPC

Add Subnet(s) to this Subnet Group. You may add subnets one at a time or add all the subnets related to this VPC. You may make additions/edits after this group is created.

Available Subnets				Subnet Group		
AZ	Subnet	CIDR		AZ	Subnet	CIDR
<input type="checkbox"/>	us-east-1a	subnet-4d902a3b	172.30.5.0/28	<input type="button" value="Add"/> <input type="button" value="Remove"/> Reset	No records found.	
<input checked="" type="checkbox"/>	us-east-1a	subnet-6bec2f1d	172.30.0.0/24			
<input type="checkbox"/>	us-east-1b	subnet-37b3566f	172.30.1.0/24			
<input checked="" type="checkbox"/>	us-east-1b	subnet-3b945863	172.30.6.0/28			
<input checked="" type="checkbox"/>	us-east-1c	subnet-d6bc43b3	172.30.2.0/24			
<input type="checkbox"/>	us-east-1d	subnet-68ce7755	172.30.3.0/24			
<input type="checkbox"/>	us-east-1e	subnet-b38b9f98	172.30.4.0/24			

For This Option	Do This
Identifier	Type a name for the replication subnet group that contains from 8 to 16 printable ASCII characters (excluding /, ", and @). The name should be unique for your account for the AWS Region you selected. You can choose to add some intelligence to the name such as including the AWS Region and task you are performing, for example DMS-default-VPC .
Description	Type a brief description of the replication subnet group.
VPC	Choose the VPC you want to use for database migration. Keep in mind that the VPC must have at least one subnet in at least two Availability Zones.
Available Subnets	Choose the subnets you want to include in the replication subnet group. You must select subnets in at least two Availability Zones.

- Choose **Add** to add the subnets to the replication subnet group.
- Choose **Create**.

Setting an Encryption Key for a Replication Instance

AWS DMS encrypts the storage used by a replication instance and the endpoint connection information. To encrypt the storage used by a replication instance, AWS DMS uses a master key that is unique to your

AWS account. You can view and manage this master key with AWS Key Management Service (AWS KMS). You can use the default master key in your account (`aws/dms`) or a custom master key that you create. If you have an existing AWS KMS encryption key, you can also use that key for encryption.

You can specify your own encryption key by supplying a KMS key identifier to encrypt your AWS DMS resources. When you specify your own encryption key, the user account used to perform the database migration must have access to that key. For more information on creating your own encryption keys and giving users access to an encryption key, see the [AWS KMS Developer Guide](#).

If you don't specify a KMS key identifier, then AWS DMS uses your default encryption key. KMS creates the default encryption key for AWS DMS for your AWS account. Your AWS account has a different default encryption key for each AWS Region.

To manage the keys used for encrypting your AWS DMS resources, you use KMS. You can find KMS in the AWS Management Console by choosing **Identity & Access Management** on the console home page and then choosing **Encryption Keys** on the navigation pane.

KMS combines secure, highly available hardware and software to provide a key management system scaled for the cloud. Using KMS, you can create encryption keys and define the policies that control how these keys can be used. KMS supports AWS CloudTrail, so you can audit key usage to verify that keys are being used appropriately. Your KMS keys can be used in combination with AWS DMS and supported AWS services such as Amazon RDS, Amazon S3, Amazon Elastic Block Store (Amazon EBS), and Amazon Redshift.

When you have created your AWS DMS resources with a specific encryption key, you can't change the encryption key for those resources. Make sure to determine your encryption key requirements before you create your AWS DMS resources.

Creating a Replication Instance

Your first task in migrating a database is to create a replication instance that has sufficient storage and processing power to perform the tasks you assign and migrate data from your source database to the target database. The required size of this instance varies depending on the amount of data you need to migrate and the tasks that you need the instance to perform. For more information about replication instances, see [Working with an AWS DMS Replication Instance \(p. 57\)](#).

The procedure following assumes that you have chosen the AWS DMS console wizard. You can also do this step by selecting **Replication instances** from the AWS DMS console's navigation pane and then selecting **Create replication instance**.

To create a replication instance by using the AWS console

1. On the **Create replication instance** page, specify your replication instance information. The following table describes the settings.

Create replication instance

A replication instance initiates the connection between the source and target databases, transfers the data, and caches any changes that occur on the source database during the initial data load. Use the fields below to configure the parameters of your new replication instance including network and security information, encryption details, and performance characteristics. We suggest you shut down the replication instance once your migration is complete to prevent further usage charges.

Name* ⓘ

Description* ⓘ

Instance class* ⓘ

Replication engine version* ⓘ

VPC* ⓘ

Multi-AZ ⓘ

Publicly accessible ⓘ

For This Option	Do This
Name	Type a name for the replication instance that contains from 8 to 16 printable ASCII characters (excluding /, ", and @). The name should be unique for your account for the AWS Region you selected. You can choose to add some intelligence to the name, such as including the AWS Region and task you are performing, for example west2-mysql2mysql-instance1 .
Description	Type a brief description of the replication instance.
Instance class	Choose an instance class with the configuration you need for your migration. Keep in mind that the instance must have enough storage, network, and processing power to successfully complete your migration. For more information on how to determine which instance class is best for your migration, see Working with an AWS DMS Replication Instance (p. 57).
Replication engine version	By default, the replication instance runs the latest version of the AWS DMS replication engine software. We recommend that you accept this default; however, you can choose a previous engine version if necessary.

For This Option	Do This
VPC	Choose the Amazon Virtual Private Cloud (Amazon VPC) you want to use. If your source or your target database is in a VPC, choose that VPC. If your source and your target databases are in different VPCs, ensure that they are both in public subnets and are publicly accessible, and then choose the VPC where the replication instance is to be located. The replication instance must be able to access the data in the source VPC. If neither your source nor your target database is in a VPC, select a VPC where the replication instance is to be located.
Multi-AZ	Use this optional parameter to create a standby replica of your replication instance in another Availability Zone for failover support. If you intend to use change data capture (CDC) or ongoing replication, you should enable this option.
Publicly accessible	Choose this option if you want the replication instance to be accessible from the Internet.

- Choose the **Advanced** tab, shown following, to set values for network and encryption settings if you need them. The following table describes the settings.

▼ **Advanced**

Allocated storage (GB)*

Replication Subnet Group*

Availability zone*

VPC Security Group(s)

KMS master key

Description Default master key that protects my DMS replication instance volumes when no other key is defined

Account

Key ARN arn:aws:kms:us-west-2:...:key/...

For This Option	Do This
Allocated storage (GB)	<p>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:</p> <ul style="list-style-type: none"> • 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 before 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 Amazon Aurora is the target. <p>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.</p>
Replication Subnet Group	<p>Choose the replication subnet group in your selected VPC where you want the replication instance to be created. If your source database is in a VPC, choose the subnet group that contains the source database as the location for your replication instance. For more information about replication subnet groups, see Creating a Replication Subnet Group (p. 70).</p>
Availability zone	<p>Choose the Availability Zone where your source database is located.</p>
VPC Security group(s)	<p>The replication instance is created in a VPC. If your source database is in a VPC, select the VPC security group that provides access to the DB instance where the database resides.</p>
KMS master key	<p>Choose the encryption key to use to encrypt replication storage and connection information. If you choose (Default) aws/dms, the default AWS Key Management Service (AWS KMS) key associated with your account and AWS Region is used. A description and your account number are shown, along with the key's ARN. For more information on using the encryption key, see Setting an Encryption Key and Specifying KMS Permissions (p. 44).</p>

3. Specify the **Maintenance** settings. The following table describes the settings. For more information about maintenance settings, see [AWS DMS Maintenance Window \(p. 60\)](#).

▼ Maintenance

Auto minor version upgrade ⓘ

Maintenance window* for ⓘ
 : (UTC-7)
 hrs.

For This Option	Do This
Auto minor version upgrade	Select to have minor engine upgrades applied automatically to the replication instance during the maintenance window.
Maintenance window	Choose a weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC). Default: A 30-minute window selected at random from an 8-hour block of time per AWS Region, occurring on a random day of the week.

4. Choose **Create replication instance**.

Modifying a Replication Instance

You can modify the settings for a replication instance to, for example, change the instance class or to increase storage.

When you modify a replication instance, you can apply the changes immediately. To apply changes immediately, you select the **Apply changes immediately** option in the AWS Management Console, you use the `--apply-immediately` parameter when calling the AWS CLI, or you set the `ApplyImmediately` parameter to `true` when using the AWS DMS API.

If you don't choose to apply changes immediately, the changes are put into the pending modifications queue. During the next maintenance window, any pending changes in the queue are applied.

Note

If you choose to apply changes immediately, any changes in the pending modifications queue are also applied. If any of the pending modifications require downtime, choosing **Apply changes immediately** can cause unexpected downtime.

To modify a replication instance by using the AWS console

1. Sign in to the AWS Management Console and select AWS DMS.
2. In the navigation pane, choose **Replication instances**.
3. Choose the replication instance you want to modify. The following table describes the modifications you can make.

For This Option	Do This
Name	You can change the name of the replication instance. Type a name for the replication instance that contains from 8 to 16 printable ASCII characters (excluding /, ", and @). The name should be unique for your account for the AWS Region you selected. You can choose to add some intelligence to the name, such as including the AWS Region and task you are performing, for example west2-mysql2mysql-instance1 .
Instance class	You can change the instance class. Choose an instance class with the configuration you need for your migration. Changing the instance class causes the replication instance to reboot. This reboot occurs during the next maintenance window or can occur immediately if you select the Apply changes immediately option. For more information on how to determine which instance class is best for your migration, see Working with an AWS DMS Replication Instance (p. 57) .
Replication engine version	You can upgrade the engine version that is used by the replication instance. Upgrading the replication engine version causes the replication instance to shut down while it is being upgraded.
Multi-AZ	You can change this option to create a standby replica of your replication instance in another Availability Zone for failover support or remove this option. If you intend to use change data capture (CDC), ongoing replication, you should enable this option.

For This Option	Do This
Allocated storage (GB)	<p>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:</p> <ul style="list-style-type: none"> • 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 before 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 Amazon Aurora is the target. <p>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.</p>
VPC Security Group(s)	The replication instance is created in a VPC. If your source database is in a VPC, select the VPC security group that provides access to the DB instance where the database resides.
Auto minor version upgrade	Choose this option to have minor engine upgrades applied automatically to the replication instance during the maintenance window or immediately if you select the Apply changes immediately option.
Maintenance window	<p>Choose a weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).</p> <p>Default: A 30-minute window selected at random from an 8-hour block of time per AWS Region, occurring on a random day of the week.</p>
Apply changes immediately	Choose this option to apply any modifications you made immediately. Depending on the settings you choose, choosing this option could cause an immediate reboot of the replication instance.

Rebooting a Replication Instance

You can reboot an AWS DMS replication instance to restart the replication engine. A reboot results in a momentary outage for the replication instance, during which the instance status is set to **Rebooting**. If

the AWS DMS instance is configured for Multi-AZ, the reboot can be conducted with a failover. An AWS DMS event is created when the reboot is completed.

If your AWS DMS instance is a Multi-AZ deployment, you can force a failover from one AWS Availability Zone to another when you reboot. When you force a failover of your AWS DMS instance, AWS DMS automatically switches to a standby instance in another Availability Zone. Rebooting with failover is beneficial when you want to simulate a failure of an AWS DMS instance for testing purposes.

If there are migration tasks running on the replication instance when a reboot occurs, no data loss occurs and the task resumes once the reboot is completed. If the tables in the migration task are in the middle of a bulk load (full load phase), DMS restarts the migration for those tables from the beginning. If tables in the migration task are in the ongoing replication phase, the task resumes once the reboot is completed.

You can't reboot your AWS DMS replication instance if its status is not in the **Available** state. Your AWS DMS instance can be unavailable for several reasons, such as a previously requested modification or a maintenance-window action. The time required to reboot an AWS DMS replication instance is typically small (under 5 minutes).

Rebooting a Replication Instance Using the AWS Console

To reboot a replication instance, use the AWS console.

To reboot a replication instance using the AWS console

1. Sign in to the AWS Management Console and select AWS DMS.
2. In the navigation pane, choose **Replication instances**.
3. Choose the replication instance you want to reboot.
4. Choose **Reboot**.
5. In the **Reboot replication instance** dialog box, choose **Reboot With Failover?** if you have configured your replication instance for Multi-AZ deployment and you want to fail over to another AWS Availability Zone.
6. Choose **Reboot**.

Rebooting a Replication Instance Using the CLI

To reboot a replication instance, use the AWS CLI `reboot-replication-instance` command with the following parameter:

- `--replication-instance-arn`

Example Example Simple Reboot

The following AWS CLI example reboots a replication instance.

```
aws dms reboot-replication-instance \  
--replication-instance-arn arnofmyrepinstance
```

Example Example Simple Reboot with Failover

The following AWS CLI example reboots a replication instance with failover.

```
aws dms reboot-replication-instance \  
--replication-instance-arn arnofmyrepinstance --failover
```

```
--replication-instance-arn arnofmyrepliance \  
--force-failover
```

Rebooting a Replication Instance Using the API

To reboot a replication instance, use the AWS DMS API [RebootReplicationInstance](#) action with the following parameters:

- `ReplicationInstanceArn` = *arnofmyrepliance*

Example Example Simple Reboot

The following code example reboots a replication instance.

```
https://dms.us-west-2.amazonaws.com/  
?Action=RebootReplicationInstance  
&DBInstanceArn=arnofmyrepliance  
&SignatureMethod=HmacSHA256  
&SignatureVersion=4  
&Version=2014-09-01  
&X-Amz-Algorithm=AWS4-HMAC-SHA256  
&X-Amz-Credential=AKIADQKE4SARGYLE/20140425/us-east-1/dms/aws4_request  
&X-Amz-Date=20140425T192732Z  
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date  
&X-Amz-Signature=1dc9dd716f4855e9bdf188c70f1cf9f6251b070b68b81103b59ec70c3e7854b3
```

Example Example Simple Reboot with Failover

The following code example reboots a replication instance and fails over to another AWS Availability Zone.

```
https://dms.us-west-2.amazonaws.com/  
?Action=RebootReplicationInstance  
&DBInstanceArn=arnofmyrepliance  
&ForceFailover=true  
&SignatureMethod=HmacSHA256  
&SignatureVersion=4  
&Version=2014-09-01  
&X-Amz-Algorithm=AWS4-HMAC-SHA256  
&X-Amz-Credential=AKIADQKE4SARGYLE/20140425/us-east-1/dms/aws4_request  
&X-Amz-Date=20140425T192732Z  
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date  
&X-Amz-Signature=1dc9dd716f4855e9bdf188c70f1cf9f6251b070b68b81103b59ec70c3e7854b3
```

Deleting a Replication Instance

You can delete an AWS DMS replication instance when you are finished using it. If you have migration tasks that use the replication instance, you must stop and delete the tasks before deleting the replication instance.

If you close your AWS account, all AWS DMS resources and configurations associated with your account are deleted after two days. These resources include all replication instances, source and target endpoint configuration, replication tasks, and SSL certificates. If after two days you decide to use AWS DMS again, you recreate the resources you need.

Deleting a Replication Instance Using the AWS Console

To delete a replication instance, use the AWS console.

To delete a replication instance using the AWS console

1. Sign in to the AWS Management Console and select AWS DMS.
2. In the navigation pane, choose **Replication instances**.
3. Choose the replication instance you want to delete.
4. Choose **Delete**.
5. In the dialog box, choose **Delete**.

Deleting a Replication Instance Using the CLI

To delete a replication instance, use the AWS CLI `delete-replication-instance` command with the following parameter:

- `--replication-instance-arn`

Example Example Delete

The following AWS CLI example deletes a replication instance.

```
aws dms delete-replication-instance \  
--replication-instance-arn <arnofmyreinstance>
```

Deleting a Replication Instance Using the API

To delete a replication instance, use the AWS DMS API `DeleteReplicationInstance` action with the following parameters:

- `ReplicationInstanceArn` = `<arnofmyreinstance>`

Example Example Delete

The following code example deletes a replication instance.

```
https://dms.us-west-2.amazonaws.com/  
?Action=DeleteReplicationInstance  
&DBInstanceArn=arnofmyreinstance  
&SignatureMethod=HmacSHA256  
&SignatureVersion=4  
&Version=2014-09-01  
&X-Amz-Algorithm=AWS4-HMAC-SHA256  
&X-Amz-Credential=AKIADQKE4SARGYLE/20140425/us-east-1/dms/aws4_request  
&X-Amz-Date=20140425T192732Z  
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date  
&X-Amz-Signature=1dc9dd716f4855e9bdf188c70f1cf9f6251b070b68b81103b59ec70c3e7854b3
```

DDL Statements Supported by AWS DMS

You can execute data definition language (DDL) statements on the source database during the data migration process. These statements are replicated to the target database by the replication server.

Supported DDL statements include the following:

- Create table
- Drop table
- Rename table
- Add column
- Drop column
- Rename column
- Change column data type

For information about which DDL statements are supported for a specific source, see the topic describing that source.

Working with AWS DMS Endpoints

An endpoint provides connection, data store type, and location information about your data store. AWS Database Migration Service uses this information to connect to a data store and migrate data from a source endpoint to a target endpoint. You can specify additional connection attributes for an endpoint by using extra connection attributes. These attributes can control logging, file size, and other parameters; for more information about extra connection attributes, see the documentation section for your data store.

Following, you can find out more details about endpoints.

Topics

- [Sources for Data Migration \(p. 83\)](#)
- [Targets for Data Migration \(p. 147\)](#)
- [Creating Source and Target Endpoints \(p. 210\)](#)

Sources for Data Migration

AWS Database Migration Service (AWS DMS) can use many of the most popular data engines as a source for data replication. The database source can be a self-managed engine running on an Amazon Elastic Compute Cloud (Amazon EC2) instance or an on-premises database. Or it can be a data source on an Amazon-managed service such as Amazon Relational Database Service (Amazon RDS) or Amazon Simple Storage Service.

Valid sources for AWS DMS include the following:

On-premises and Amazon EC2 instance databases

- Oracle versions 10.2 and later, 11g, and up to 12.2, for the Enterprise, Standard, Standard One, and Standard Two editions.
- Microsoft SQL Server versions 2005, 2008, 2008R2, 2012, 2014, and 2016 for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.
- MySQL versions 5.5, 5.6, and 5.7.
- MariaDB (supported as a MySQL-compatible data source).
- PostgreSQL 9.4 and later.
- SAP Adaptive Server Enterprise (ASE) versions 12.5.3 or higher, 15, 15.5, 15.7, 16 and later.
- MongoDB versions 2.6.x and 3.x and later.
- Db2 LUW versions:
 - Version 9.7, all Fix Packs are supported.
 - Version 10.1, all Fix Packs are supported.
 - Version 10.5, all Fix Packs except for Fix Pack 5 are supported.

Microsoft Azure

- AWS DMS supports full data load when using Azure SQL Database as a source. Change data capture (CDC) is not supported.

Amazon RDS instance databases

- Oracle versions 11g (versions 11.2.0.3.v1 and later), and 12c, for the Enterprise, Standard, Standard One, and Standard Two editions.
- Microsoft SQL Server versions 2008R2, 2012, 2014, and 2016 for both the Enterprise and Standard editions. CDC is supported for all versions of Enterprise Edition. CDC is only supported for Standard Edition version 2016 SP1 and later. The Web, Workgroup, Developer, and Express editions are not supported by AWS DMS.
- MySQL versions 5.5, 5.6, and 5.7. Change data capture (CDC) is only supported for versions 5.6 and later.
- PostgreSQL 9.4 and later. CDC is only supported for versions 9.4.9 and higher and 9.5.4 and higher. The `rds.logical_replication` parameter, which is required for CDC, is supported only in these versions and later.
- MariaDB, supported as a MySQL-compatible data source.
- Amazon Aurora with MySQL compatibility.

Amazon Simple Storage Service

- AWS DMS supports full data load and change data capture (CDC) when using Amazon Simple Storage Service as a source.

Topics

- [Using an Oracle Database as a Source for AWS DMS \(p. 84\)](#)
- [Using a Microsoft SQL Server Database as a Source for AWS DMS \(p. 100\)](#)
- [Using Microsoft Azure SQL Database as a Source for AWS DMS \(p. 109\)](#)
- [Using a PostgreSQL Database as a Source for AWS DMS \(p. 110\)](#)
- [Using a MySQL-Compatible Database as a Source for AWS DMS \(p. 122\)](#)
- [Using an SAP ASE Database as a Source for AWS DMS \(p. 129\)](#)
- [Using MongoDB as a Source for AWS DMS \(p. 132\)](#)
- [Using Amazon Simple Storage Service as a Source for AWS DMS \(p. 138\)](#)
- [Using an IBM Db2 for Linux, Unix, and Windows Database \(Db2 LUW\) as a Source for AWS DMS \(p. 144\)](#)

Using an Oracle Database as a Source for AWS DMS

You can migrate data from one or many Oracle databases using AWS DMS. With an Oracle database as a source, you can migrate data to any of the targets supported by AWS DMS.

For self-managed Oracle databases, AWS DMS supports all Oracle database editions for versions 10.2 and later, 11g, and up to 12.2 for self-managed databases as sources. For Amazon-managed Oracle databases provided by Amazon RDS, AWS DMS supports all Oracle database editions for versions 11g (versions 11.2.0.3.v1 and later) and up to 12.2.

You can use SSL to encrypt connections between your Oracle endpoint and the replication instance. For more information on using SSL with an Oracle endpoint, see [Using SSL With AWS Database Migration Service \(p. 47\)](#).

The steps to configure an Oracle database as a source for AWS DMS source are as follows:

1. If you want to create a CDC-only or full load plus CDC task, then you must choose either Oracle LogMiner or Oracle Binary Reader to capture data changes. Choosing LogMiner or Binary Reader

determines some of the subsequent permission and configuration tasks. For a comparison of LogMiner and Binary Reader, see the next section.

2. Create an Oracle user with the appropriate permissions for AWS DMS. If you are creating a full-load-only task, then no further configuration is needed.
3. If you are creating a full load plus CDC task or a CDC-only task, configure Oracle for LogMiner or Binary Reader.
4. Create a DMS endpoint that conforms with your chosen configuration.

For additional details on working with Oracle databases and AWS DMS, see the following sections.

Topics

- [Using Oracle LogMiner or Oracle Binary Reader for Change Data Capture \(CDC\) \(p. 85\)](#)
- [Working with a Self-Managed Oracle Database as a Source for AWS DMS \(p. 87\)](#)
- [Working with an Amazon-Managed Oracle Database as a Source for AWS DMS \(p. 89\)](#)
- [Limitations on Using Oracle as a Source for AWS DMS \(p. 92\)](#)
- [Extra Connection Attributes When Using Oracle as a Source for AWS DMS \(p. 93\)](#)
- [Source Data Types for Oracle \(p. 97\)](#)

Using Oracle LogMiner or Oracle Binary Reader for Change Data Capture (CDC)

Oracle offers two methods for reading the redo logs when doing change processing: Oracle LogMiner and Oracle Binary Reader. Oracle LogMiner provides a SQL interface to Oracle's online and archived redo log files. Binary Reader is an AWS DMS feature that reads and parses the raw redo log files directly.

By default, AWS DMS uses Oracle LogMiner for change data capture (CDC). The advantages of using LogMiner with AWS DMS include the following:

- LogMiner supports most Oracle options, such as encryption options and compression options. Binary Reader doesn't support all Oracle options, in particular options for encryption and compression.
- LogMiner offers a simpler configuration, especially compared to Oracle Binary Reader's direct access setup or if the redo logs are on Automatic Storage Management (ASM).
- LogMiner fully supports most Oracle encryption options, including Oracle Transparent Data Encryption (TDE).
- LogMiner supports the following HCC compression types for both full load and on-going replication (CDC):
 - QUERY HIGH
 - ARCHIVE HIGH
 - ARCHIVE LOW
 - QUERY LOW

Binary Reader supports QUERY LOW compression only for full load replications, not ongoing (CDC) replications.

- LogMiner supports table clusters for use by AWS DMS. Binary Reader does not.

The advantages to using Binary Reader with AWS DMS, instead of LogMiner, include the following:

- For migrations with a high volume of changes, LogMiner might have some I/O or CPU impact on the computer hosting the Oracle source database. Binary Reader has less chance of having I/O or CPU impact because the archive logs are copied to the replication instance and mined there.

- For migrations with a high volume of changes, CDC performance is usually much better when using Binary Reader compared with using Oracle LogMiner.
- Binary Reader supports CDC for LOBs in Oracle version 12c. LogMiner does not.
- Binary Reader supports the following HCC compression types for both full load and continuous replication (CDC):
 - QUERY HIGH
 - ARCHIVE HIGH
 - ARCHIVE LOW

The QUERY LOW compression type is only supported for full load migrations.

In general, use Oracle LogMiner for migrating your Oracle database unless you have one of the following situations:

- You need to run several migration tasks on the source Oracle database.
- The volume of changes or the redo log volume on the source Oracle database is high.
- You are migrating LOBs from an Oracle 12.2 or later source endpoint.
- If your workload includes UPDATE statements that update only LOB columns you must use Binary Reader. These update statements are not supported by Oracle LogMiner.
- If your source is Oracle version 11 and you perform UPDATE statements on XMLTYPE and LOB columns, you must use Binary Reader. These statements are not supported by Oracle LogMiner.
- On Oracle 12c, LogMiner does not support LOB columns. You must use Binary Reader if you are migrating LOB columns from Oracle 12c.

Configuration for Change Data Capture (CDC) on an Oracle Source Database

When you use Oracle as a source endpoint either for full-load and change data capture (CDC) or just for CDC, you must set an extra connection attribute. This attribute specifies whether to use LogMiner or Binary Reader to access the transaction logs. You specify an extra connection attribute when you create the source endpoint. Multiple extra connection attribute settings should be separated by a semicolon.

LogMiner is used by default, so you don't have to explicitly specify its use. In order to enable Binary Reader to access the transaction logs, add the following extra connection attributes.

```
useLogMinerReader=N; useBfile=Y;
```

If the Oracle source database is using Oracle Automatic Storage Management (ASM), the extra connection attribute needs to include the ASM user name and ASM server address. When you create the source endpoint, the password field needs to have both passwords, the source user password and the ASM password.

For example, the following extra connection attribute format is used to access a server that uses Oracle ASM.

```
useLogMinerReader=N;asm_user=<asm_username>;asm_server=<first_RAC_server_ip_address>:<port_number>/+ASM
```

If the Oracle source database is using Oracle ASM, the source endpoint password field must have both the Oracle user password and the ASM password, separated by a comma. For example, the following works in the password field.

```
<oracle_user_password>,<asm_user_password>
```

Limitations for CDC on an Oracle Source Database

The following limitations apply when using an Oracle database as a source for AWS DMS change data capture:

- AWS DMS doesn't capture changes made by the Oracle DBMS_REDEFINITION package, such as changes to table metadata and the OBJECT_ID value.
- AWS DMS doesn't support index-organized tables with an overflow segment in CDC mode when using BFILE. An example is when you access the redo logs without using LogMiner.

Working with a Self-Managed Oracle Database as a Source for AWS DMS

A self-managed database is a database that you configure and control, either a local on-premises database instance or a database on Amazon EC2. Following, you can find out about the privileges and configurations you need to set up when using a self-managed Oracle database with AWS DMS.

User Account Privileges Required on a Self-Managed Oracle Source for AWS DMS

To use an Oracle database as a source in an AWS DMS task, the user specified in the AWS DMS Oracle database definitions must be granted the following privileges in the Oracle database. When granting privileges, use the actual name of objects (for example, V_\$OBJECT including the underscore), not the synonym for the object (for example, V\$OBJECT without the underscore).

```
GRANT SELECT ANY TRANSACTION to <dms_user>
GRANT SELECT on V_$ARCHIVED_LOG to <dms_user>
GRANT SELECT on V_$LOG to <dms_user>
GRANT SELECT on V_$LOGFILE to <dms_user>
GRANT SELECT on V_$DATABASE to <dms_user>
GRANT SELECT on V_$THREAD to <dms_user>
GRANT SELECT on V_$PARAMETER to <dms_user>
GRANT SELECT on V_$NLS_PARAMETERS to <dms_user>
GRANT SELECT on V_$TIMEZONE_NAMES to <dms_user>
GRANT SELECT on V_$TRANSACTION to <dms_user>
GRANT SELECT on ALL_INDEXES to <dms_user>
GRANT SELECT on ALL_OBJECTS to <dms_user>
GRANT SELECT on DBA_OBJECTS to <dms_user> (required if the Oracle version is earlier than
  11.2.0.3)
GRANT SELECT on ALL_TABLES to <dms_user>
GRANT SELECT on ALL_USERS to <dms_user>
GRANT SELECT on ALL_CATALOG to <dms_user>
GRANT SELECT on ALL_CONSTRAINTS to <dms_user>
GRANT SELECT on ALL_CONS_COLUMNS to <dms_user>
GRANT SELECT on ALL_TAB_COLS to <dms_user>
GRANT SELECT on ALL_IND_COLUMNS to <dms_user>
GRANT SELECT on ALL_LOG_GROUPS to <dms_user>
GRANT SELECT on SYS.DBA_REGISTRY to <dms_user>
GRANT SELECT on SYS.OBJ$ to <dms_user>
GRANT SELECT on DBA_TABLESPACES to <dms_user>
GRANT SELECT on ALL_TAB_PARTITIONS to <dms_user>
GRANT SELECT on ALL_ENCRYPTED_COLUMNS to <dms_user>
GRANT SELECT on V_$LOGMNR_LOGS to <dms_user>
GRANT SELECT on V_$LOGMNR_CONTENTS to <dms_user>
```

When using ongoing replication (CDC), you need these additional permissions.

- The following permission is required when using CDC so that AWS DMS can add to Oracle LogMiner redo logs for both 11g and 12c.

```
Grant EXECUTE ON dbms_logmnr TO <dms_user>;
```

- The following permission is required when using CDC so that AWS DMS can add to Oracle LogMiner redo logs for 12c only.

```
Grant LOGMINING TO <dms_user>;
```

If you are using any of the additional features noted following, the given additional permissions are required:

- If views are exposed, grant SELECT on ALL_VIEWS to <dms_user>.
- If you use a pattern to match table names in your replication task, grant SELECT ANY TABLE.
- If you specify a table list in your replication task, grant SELECT on each table in the list.
- If you add supplemental logging, grant ALTER ANY TABLE.
- If you add supplemental logging and you use a specific table list, grant ALTER for each table in the list.
- If you are migrating from Oracle RAC, grant SELECT permissions on materialized views with the prefixes g_\$ and v_\$.

Configuring a Self-Managed Oracle Source for AWS DMS

Before using a self-managed Oracle database as a source for AWS DMS, you need to perform several tasks:

- Provide Oracle account access – You must provide an Oracle user account for AWS DMS. The user account must have read/write privileges on the Oracle database, as specified in the previous section.
- Ensure that ARCHIVELOG mode is on – Oracle can run in two different modes, the ARCHIVELOG mode and the NOARCHIVELOG mode. To use Oracle with AWS DMS, the source database must be in ARCHIVELOG mode.
- Set up supplemental logging – If you are planning to use the source in a CDC or full-load plus CDC task, then you need to set up supplemental logging to capture the changes for replication.

There are two steps to enable supplemental logging for Oracle. First, you need to enable database-level supplemental logging. Doing this ensures that the LogMiner has the minimal information to support various table structures such as clustered and index-organized tables. Second, you need to enable table-level supplemental logging for each table to be migrated.

To enable database-level supplemental logging

1. Run the following query to determine if database-level supplemental logging is already enabled. The return result should be from GE to 9.0.0.

```
SELECT name, value, description FROM v$parameter WHERE name = 'compatible';
```

2. Run the following query. The returned result should be YES or IMPLICIT.

```
SELECT supplemental_log_data_min FROM v$database;
```

3. Run the following query to enable database-level supplemental logging.

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

There are two methods to enable table-level supplemental logging. In the first one, if your database user account has ALTER TABLE privileges on all tables to be migrated, you can use the extra connection parameter `addSupplementalLogging` as described following. Otherwise, you can use the steps following for each table in the migration.

To enable table-level supplemental logging

1. If the table has a primary key, add PRIMARY KEY supplemental logging for the table by running the following command.

```
ALTER TABLE <table_name> ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
```

2. If no primary key exists and the table has multiple unique indexes, then AWS DMS uses the first unique index in alphabetical order of index name.

Create a supplemental log group as shown preceding on that index's columns.

3. If there is no primary key and no unique index, supplemental logging must be added on all columns. Run the following query to add supplemental logging to all columns.

```
ALTER TABLE <table_name> ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

In some cases, the target table primary key or unique index is different than the source table primary key or unique index. In these cases, add supplemental logging on the source table columns that make up the target table primary key or unique index. If you change the target table primary key, you should add supplemental logging on the selected index's columns, instead of the columns of the original primary key or unique index.

Add additional logging if needed, such as if a filter is defined for a table. If a table has a unique index or a primary key, you need to add supplemental logging on each column that is involved in a filter if those columns are different than the primary key or unique index columns. However, if ALL COLUMNS supplemental logging has been added to the table, you don't need to add any additional logging.

```
ALTER TABLE <table_name> ADD SUPPLEMENTAL LOG GROUP <group_name> (<column_list>) ALWAYS;
```

Working with an Amazon-Managed Oracle Database as a Source for AWS DMS

An Amazon-managed database is a database that is on an Amazon service such as Amazon RDS, Amazon Aurora, or Amazon Simple Storage Service. Following, you can find the privileges and configurations you need to set up when using an Amazon-managed Oracle database with AWS DMS.

User Account Privileges Required on an Amazon-Managed Oracle Source for AWS DMS

To grant privileges on Oracle databases on Amazon RDS, use the stored procedure `rdsadmin.rdsadmin_util.grant_sys_object`. For more information, see [Granting SELECT or EXECUTE privileges to SYS Objects](#).

Grant the following to the AWS DMS user account used to access the source Oracle endpoint.

- GRANT SELECT ANY TABLE to <dms_user>;
- GRANT SELECT on ALL_VIEWS to <dms_user>;
- GRANT SELECT ANY TRANSACTION to <dms_user>;
- Run the following: `exec rdsadmin.rdsadmin_util.grant_sys_object('V_$ARCHIVED_LOG', '<dms_user>', 'SELECT');`
- Run the following: `exec rdsadmin.rdsadmin_util.grant_sys_object('V_$LOG', '<dms_user>', 'SELECT');`
- Run the following: `exec rdsadmin.rdsadmin_util.grant_sys_object('V_$LOGFILE', '<dms_user>', 'SELECT');`
- Run the following: `exec rdsadmin.rdsadmin_util.grant_sys_object('V_$DATABASE', '<dms_user>', 'SELECT');`
- Run the following: `exec rdsadmin.rdsadmin_util.grant_sys_object('V_$THREAD', '<dms_user>', 'SELECT');`
- Run the following: `exec rdsadmin.rdsadmin_util.grant_sys_object('V_$PARAMETER', '<dms_user>', 'SELECT');`
- Run the following: `exec rdsadmin.rdsadmin_util.grant_sys_object('V_$NLS_PARAMETERS', '<dms_user>', 'SELECT');`
- Run the following: `exec rdsadmin.rdsadmin_util.grant_sys_object('V_$TIMEZONE_NAMES', '<dms_user>', 'SELECT');`
- Run the following: `exec rdsadmin.rdsadmin_util.grant_sys_object('V_$TRANSACTION', '<dms_user>', 'SELECT');`
- Run the following:

```
GRANT SELECT on ALL_INDEXES to <dms_user>;
GRANT SELECT on ALL_OBJECTS to <dms_user>;
GRANT SELECT on ALL_TABLES to <dms_user>;
GRANT SELECT on ALL_USERS to <dms_user>;
GRANT SELECT on ALL_CATALOG to <dms_user>;
GRANT SELECT on ALL_CONSTRAINTS to <dms_user>;
GRANT SELECT on ALL_CONS_COLUMNS to <dms_user>;
GRANT SELECT on ALL_TAB_COLS to <dms_user>;
GRANT SELECT on ALL_IND_COLUMNS to <dms_user>;
GRANT SELECT on ALL_LOG_GROUPS to <dms_user>;
```

- Run the following: `exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_REGISTRY', '<dms_user>', 'SELECT');`
- Run the following: `exec rdsadmin.rdsadmin_util.grant_sys_object('OBJ$', '<dms_user>', 'SELECT');`
- GRANT SELECT on DBA_TABLESPACES to <dms_user>;
- GRANT SELECT on ALL_TAB_PARTITIONS to <dms_user>;
- GRANT LOGMINING TO <dms_user>;
- Run the following: `exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_ENCRYPTED_COLUMNS', '<dms_user>', 'SELECT');`
- Run the following: `exec rdsadmin.rdsadmin_util.grant_sys_object('V_$LOGMNR_LOGS', '<dms_user>', 'SELECT');`
- Run the following: `exec rdsadmin.rdsadmin_util.grant_sys_object('V_$LOGMNR_CONTENTS', '<dms_user>', 'SELECT');`
- Run the following: `exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_LOGMNR', '<dms_user>', 'EXECUTE');`

Configuring an Amazon-Managed Oracle Source for AWS DMS

Before using an Amazon-managed Oracle database as a source for AWS DMS, you need to perform several tasks:

- Provide Oracle account access – You must provide an Oracle user account for AWS DMS. The user account must have read/write privileges on the Oracle database, as specified in the previous section.
- Set the backup retention period for your Amazon RDS database to one day or longer – Setting the backup retention period ensures that the database is running in ARCHIVELOG mode. For more information about setting the backup retention period, see the [Working with Automated Backups](#) in the *Amazon RDS User Guide*.
- Set up archive retention – Run the following to retain archived redo logs of your Oracle database instance. Running this command lets AWS DMS retrieve the log information using LogMiner. Make sure that you have enough storage for the archived redo logs during the migration period.

In the following example, logs are kept for 24 hours.

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

- Set up supplemental logging – If you are planning to use the source in a CDC or full-load plus CDC task, then set up supplemental logging to capture the changes for replication.

There are two steps to enable supplemental logging for Oracle. First, you need to enable database-level supplemental logging. Doing this ensures that the LogMiner has the minimal information to support various table structures such as clustered and index-organized tables. Second, you need to enable table-level supplemental logging for each table to be migrated.

To enable database-level supplemental logging

- Run the following query to enable database-level supplemental logging.

```
exec rdsadmin.rdsadmin_util.alter_supplemental_logging('ADD');
```

To enable table-level supplemental logging

- Run the following command to enable PRIMARY KEY logging for tables that have primary keys.

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

For tables that don't have primary keys, use the following command to add supplemental logging.

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

If you create a table without a primary key, you should either include a supplemental logging clause in the create statement or alter the table to add supplemental logging. The following command creates a table and adds supplemental logging.

```
create table <table_name> (<column_list>, supplemental log data (ALL) columns);
```

If you create a table and later add a primary key, you need to add supplemental logging to the table. Add supplemental logging to the table using the following command.

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

Configuring Change Data Capture (CDC) for an Amazon RDS for Oracle Source for AWS DMS

You can configure AWS DMS to use an Amazon RDS for Oracle instance as a source of CDC. You use Oracle Binary Reader with an Amazon RDS for Oracle source (Oracle versions 11.2.0.4.v11 and later, and 12.1.0.2.v7 and later).

You must include the following extra connection attributes when you create the Amazon RDS for Oracle source endpoint:

```
useLogminerReader=N; useBfile=Y; accessAlternateDirectly=false;
useAlternateFolderForOnline=true;

oraclePathPrefix=/rdsdbdata/db/ORCL_A/; usePathPrefix=/rdsdbdata/log/;
replacePathPrefix=true
```

Limitations on Using Oracle as a Source for AWS DMS

The following limitations apply when using an Oracle database as a source for AWS DMS:

- AWS DMS supports Oracle transparent data encryption (TDE) tablespace encryption and AWS Key Management Service (AWS KMS) encryption when used with Oracle LogMiner. All other forms of encryption are not supported.
- Tables with LOBs must have a primary key to use CDC.
- AWS DMS supports the `rename table <table name> to <new table name>` syntax with Oracle version 11 and higher.
- Oracle source databases columns created using explicit CHAR semantics are transferred to a target Oracle database using BYTE semantics. You must create tables containing columns of this type on the target Oracle database before migrating.
- AWS DMS doesn't replicate data changes resulting from partition or subpartition operations—data definition language (DDL) operations such as ADD, DROP, EXCHANGE, or TRUNCATE. To replicate such changes, you must reload the table being replicated. AWS DMS replicates any future data changes to newly added partitions without you having to reload the table. However, UPDATE operations on old data records in partitions fail and generate a `0 rows affected` warning.
- The DDL statement `ALTER TABLE ADD <column> <data_type> DEFAULT <>` doesn't replicate the default value to the target. The new column in the target is set to NULL. If the new column is nullable, Oracle updates all the table rows before logging the DDL itself. As a result, AWS DMS captures the changes to the counters but doesn't update the target. Because the new column is set to NULL, if the target table has no primary key or unique index, subsequent updates generate a `0 rows affected` warning.
- Data changes resulting from the `CREATE TABLE AS` statement are not supported. However, the new table is created on the target.
- When limited-size LOB mode is enabled, AWS DMS replicates empty LOBs on the Oracle source as NULL values in the target.
- When AWS DMS begins CDC, it maps a timestamp to the Oracle system change number (SCN). By default, Oracle keeps only five days of the timestamp to SCN mapping. Oracle generates an error if the timestamp specified is too old (greater than the five-day retention period). For more information, see the [Oracle documentation](#).

- AWS DMS doesn't support connections to an Oracle source by using an ASM proxy.
- AWS DMS doesn't support virtual columns.

Extra Connection Attributes When Using Oracle as a Source for AWS DMS

You can use extra connection attributes to configure your Oracle source. You specify these settings when you create the source endpoint. Multiple extra connection attribute settings should be separated from each other by semicolons.

The following table shows the extra connection attributes you can use to configure an Oracle database as a source for AWS DMS.

Name	Description
<code>addSupplementalLogging</code>	<p>Set this attribute to set up table-level supplemental logging for the Oracle database. This attribute enables PRIMARY KEY supplemental logging on all tables selected for a migration task.</p> <p>Default value: N</p> <p>Valid values: Y/N</p> <p>Example: <code>addSupplementalLogging=Y</code></p> <p>Note If you use this option, you still need to enable database-level supplemental logging as discussed previously.</p>
<code>additionalArchivedLogDestId</code>	<p>Set this attribute with <code>archivedLogDestId</code> in a primary/standby setup. This attribute is useful in the case of a failover. In this case, AWS DMS needs to know which destination to get archive redo logs from to read changes, because the previous primary instance is now a standby instance after failover.</p>
<code>useLogminerReader</code>	<p>Set this attribute to Y to capture change data using the LogMiner utility (the default). Set this option to N if you want AWS DMS to access the redo logs as a binary file. When set to N, you must also add the setting <code>useBfile=Y</code>. For more information, see Using Oracle LogMiner or Oracle Binary Reader for Change Data Capture (CDC) (p. 85).</p> <p>Default value: Y</p> <p>Valid values: Y/N</p> <p>Example: <code>useLogminerReader=N; useBfile=Y</code></p> <p>If the Oracle source database is using Oracle Automatic Storage Management (ASM), the extra connection parameter needs to include the ASM user name and ASM server address. The password field also needs to have both passwords, the source user password and the ASM password, separated from each other by a comma.</p>

Name	Description
	<p>Example: <code>useLogminerReader=N;asm_user=<asm_username>; asm_server=<first_RAC_server_ip_address>:<port_number>/ +ASM</code></p>
<p><code>useBfile</code></p>	<p>Set this attribute to Y in order to capture change data using the Binary Reader utility. You must set <code>useLogminerReader</code> to N in order to set this attribute to Y. Note also that you must set additional attributes to use the Binary Reader with an Amazon RDS for Oracle as the source. For more information, see Using Oracle LogMiner or Oracle Binary Reader for Change Data Capture (CDC) (p. 85).</p> <p>Default value: N</p> <p>Valid values: Y/N</p> <p>Example: <code>useLogminerReader=N; useBfile=Y</code></p>
<p><code>accessAlternateDirectly</code></p>	<p>You must set this attribute to false in order to use the Binary Reader to capture change data for an Amazon RDS for Oracle as the source. This tells the DMS instance to not access redo logs through any specified path prefix replacement using direct file access. For more information, see Configuring Change Data Capture (CDC) for an Amazon RDS for Oracle Source for AWS DMS (p. 92).</p> <p>Default value: true</p> <p>Valid values: true/false</p> <p>Example: <code>useLogminerReader=N; useBfile=Y; accessAlternateDirectly=false</code></p>
<p><code>useAlternateFolderForOnline</code></p>	<p>You must set this attribute to true in order to use the Binary Reader to capture change data for an Amazon RDS for Oracle as the source. This tells the DMS instance to use any specified prefix replacement to access all online redo logs. For more information, see Configuring Change Data Capture (CDC) for an Amazon RDS for Oracle Source for AWS DMS (p. 92).</p> <p>Default value: false</p> <p>Valid values: true/false</p> <p>Example: <code>useLogminerReader=N; useBfile=Y; accessAlternateDirectly=false; useAlternateFolderForOnline=true;</code></p>

Name	Description
oraclePathPrefix	<p>You must set this string attribute to the required value in order to use the Binary Reader to capture change data for an Amazon RDS for Oracle as the source. This value specifies the default Oracle root used to access the redo logs. For more information, see Configuring Change Data Capture (CDC) for an Amazon RDS for Oracle Source for AWS DMS (p. 92).</p> <p>Default value: none</p> <p>Valid value: /rdsdbdata/db/ORCL_A/</p> <p>Example: useLogminerReader=N; useBfile=Y; accessAlternateDirectly=false; useAlternateFolderForOnline=true; oraclePathPrefix=/rdsdbdata/db/ORCL_A/;</p>
usePathPrefix	<p>You must set this string attribute to the required value in order to use the Binary Reader to capture change data for an Amazon RDS for Oracle as the source. This value specifies the path prefix used to replace the default Oracle root to access the redo logs. For more information, see Configuring Change Data Capture (CDC) for an Amazon RDS for Oracle Source for AWS DMS (p. 92).</p> <p>Default value: none</p> <p>Valid value: /rdsdbdata/log/</p> <p>Example: useLogminerReader=N; useBfile=Y; accessAlternateDirectly=false; useAlternateFolderForOnline=true; oraclePathPrefix=/rdsdbdata/db/ORCL_A/; usePathPrefix=/rdsdbdata/log/;</p>
replacePathPrefix	<p>You must set this attribute to true in order to use the Binary Reader to capture change data for an Amazon RDS for Oracle as the source. This setting tells DMS instance to replace the default Oracle root with the specified usePathPrefix setting to access the redo logs. For more information, see Configuring Change Data Capture (CDC) for an Amazon RDS for Oracle Source for AWS DMS (p. 92).</p> <p>Default value: false</p> <p>Valid values: true/false</p> <p>Example: useLogminerReader=N; useBfile=Y; accessAlternateDirectly=false; useAlternateFolderForOnline=true; oraclePathPrefix=/rdsdbdata/db/ORCL_A/; usePathPrefix=/rdsdbdata/log/; replacePathPrefix=true</p>

Name	Description
<code>retryInterval</code>	<p>Specifies the number of seconds that the system waits before resending a query.</p> <p>Default value: 5</p> <p>Valid values: Numbers starting from 1</p> <p>Example: <code>retryInterval=6</code></p>
<code>archivedLogDestId</code>	<p>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 to specify an archived redo logs location identifier. Doing this improves performance by ensuring that the correct logs are accessed from the outset.</p> <p>Default value:0</p> <p>Valid values: Number</p> <p>Example: <code>archivedLogDestId=1</code></p>
<code>archivedLogsOnly</code>	<p>When this field is set to Y, AWS DMS only accesses the archived redo logs. If the archived redo logs are stored on Oracle ASM only, the AWS DMS user account needs to be granted ASM privileges.</p> <p>Default value: N</p> <p>Valid values: Y/N</p> <p>Example: <code>archivedLogsOnly=Y</code></p>
<code>numberDataTypeScale</code>	<p>Specifies the number scale. You can select a scale up to 38, or you can select FLOAT. By default, the NUMBER data type is converted to precision 38, scale 10.</p> <p>Default value: 10</p> <p>Valid values: -1 to 38 (-1 for FLOAT)</p> <p>Example: <code>numberDataTypeScale =12</code></p>
<code>afterConnectScript</code>	<p>Specifies a script to run immediately after AWS DMS connects to the endpoint.</p> <p>Valid values: A SQL statement set off by a semicolon. Not all SQL statements are supported.</p> <p>Example: <code>afterConnectScript=ALTER SESSION SET CURRENT_SCHEMA = system;</code></p>

Name	Description
<code>failTasksOnLobTruncation</code>	<p>When set to <code>true</code>, this attribute causes a task to fail if the actual size of an LOB column is greater than the specified <code>LobMaxSize</code>.</p> <p>If a task is set to limited LOB mode and this option is set to <code>true</code>, the task fails instead of truncating the LOB data.</p> <p>Default value: <code>false</code></p> <p>Valid values: Boolean</p> <p>Example: <code>failTasksOnLobTruncation=true</code></p>
<code>readTableSpaceName</code>	<p>When set to <code>true</code>, this attribute supports tablespace replication.</p> <p>Default value: <code>false</code></p> <p>Valid values: Boolean</p> <p>Example: <code>readTableSpaceName =true</code></p>
<code>standbyDelayTime</code>	<p>Use this attribute to specify a time in minutes for the delay in standby sync.</p> <p>With AWS DMS version 2.3.0 and later, you can create an Oracle ongoing replication (CDC) task that uses an Oracle active data guard standby instance as a source for replicating on-going changes to a supported target. This eliminates the need to connect to an active database that may be in production.</p> <p>Default value: <code>0</code></p> <p>Valid values: Number</p> <p>Example: <code>standbyDelayTime=1</code></p>

Source Data Types for Oracle

The Oracle endpoint for AWS DMS supports most Oracle data types. The following table shows the Oracle source data types that are supported when using AWS DMS and the default mapping to AWS DMS data types.

For information on how to view the data type that is mapped in the target, see the section for the target endpoint you are using.

For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service \(p. 319\)](#).

Oracle Data Type	AWS DMS Data Type
<code>BINARY_FLOAT</code>	<code>REAL4</code>
<code>BINARY_DOUBLE</code>	<code>REAL8</code>

Oracle Data Type	AWS DMS Data Type
BINARY	BYTES
FLOAT (P)	If precision is less than or equal to 24, use REAL4. If precision is greater than 24, use REAL8.
NUMBER (P,S)	When scale is less than 0, use REAL8
NUMBER according to the "Expose number as" property in the Oracle source database settings.	When scale is 0: <ul style="list-style-type: none"> • And precision is 0, use REAL8. • And precision is greater than or equal to 2, use INT1. • And precision is greater than 2 and less than or equal to 4, use INT2. • And precision is greater than 4 and less than or equal to 9, use INT4. • And precision is greater than 9, use NUMERIC. • And precision is greater than or equal to scale, use NUMERIC. <p>In all other cases, use REAL8.</p>
DATE	DATETIME
INTERVAL_YEAR TO MONTH	STRING (with interval year_to_month indication)
INTERVAL_DAY TO SECOND	STRING (with interval day_to_second indication)
TIME	DATETIME
TIMESTAMP	DATETIME
TIMESTAMP WITH TIME ZONE	STRING (with timestamp_with_timezone indication)
TIMESTAMP WITH LOCAL TIME ZONE	STRING (with timestamp_with_local_timezone indication)
CHAR	STRING
VARCHAR2	STRING
NCHAR	WSTRING
NVARCHAR2	WSTRING
RAW	BYTES
REAL	REAL8
BLOB	BLOB To use this data type with AWS DMS, you must enable the use of BLOB data types for a specific task. AWS DMS supports BLOB data types only in tables that include a primary key.

Oracle Data Type	AWS DMS Data Type
CLOB	<p>CLOB</p> <p>To use this data type with AWS DMS, you must enable the use of CLOB data types for a specific task. During change data capture (CDC), AWS DMS supports CLOB data types only in tables that include a primary key.</p>
NCLOB	<p>NCLOB</p> <p>To use this data type with AWS DMS, you must enable the use of NCLOB data types for a specific task. During CDC, AWS DMS supports NCLOB data types only in tables that include a primary key.</p>
LONG	<p>CLOB</p> <p>The LONG data type is not supported in batch-optimized apply mode (TurboStream CDC mode). To use this data type with AWS DMS, you must enable the use of LOBs for a specific task. During CDC, AWS DMS supports LOB data types only in tables that have a primary key.</p>
LONG RAW	<p>BLOB</p> <p>The LONG RAW data type is not supported in batch-optimized apply mode (TurboStream CDC mode). To use this data type with AWS DMS, you must enable the use of LOBs for a specific task. During CDC, AWS DMS supports LOB data types only in tables that have a primary key.</p>
XMLTYPE	<p>CLOB</p> <p>Support for the XMLTYPE data type requires the full Oracle Client (as opposed to the Oracle Instant Client). When the target column is a CLOB, both full LOB mode and limited LOB mode are supported (depending on the target).</p>

Oracle tables used as a source with columns of the following data types are not supported and cannot be replicated. Replicating columns with these data types result in a null column.

- BFILE
- ROWID
- REF
- UROWID
- Nested Table
- User-defined data types
- ANYDATA

Note

Virtual columns are not supported.

Using a Microsoft SQL Server Database as a Source for AWS DMS

You can migrate data from one or many Microsoft SQL Server databases using AWS DMS (AWS DMS). With a SQL Server database as a source, you can migrate data to either another SQL Server database or one of the other supported databases.

AWS DMS supports, as a source, on-premises and Amazon EC2 instance databases for Microsoft SQL Server versions 2005, 2008, 2008R2, 2012, 2014, and 2016. The Enterprise, Standard, Workgroup, and Developer editions are supported. The Web and Express editions are not supported.

AWS DMS supports, as a source, Amazon RDS DB instance databases for SQL Server versions 2008R2, 2012, 2014, and 2016. The Enterprise and Standard editions are supported. CDC is supported for all versions of Enterprise Edition. CDC is only supported for Standard Edition version 2016 SP1 and later. The Web, Workgroup, Developer, and Express editions are not supported.

You can have the source SQL Server database installed on any computer in your network. A SQL Server account with the appropriate access privileges to the source database for the type of task you chose is also required for use with AWS DMS.

AWS DMS supports migrating data from named instances of SQL Server. You can use the following notation in the server name when you create the source endpoint.

```
IPAddress\InstanceName
```

For example, the following is a correct source endpoint server name. Here, the first part of the name is the IP address of the server, and the second part is the SQL Server instance name (in this example, SQLTest).

```
10.0.0.25\SQLTest
```

You can use SSL to encrypt connections between your SQL Server endpoint and the replication instance. For more information on using SSL with a SQL Server endpoint, see [Using SSL With AWS Database Migration Service \(p. 47\)](#).

To capture changes from a source SQL Server database, the database must be configured for full backups and must be either the Enterprise, Developer, or Standard Edition.

For additional details on working with SQL Server source databases and AWS DMS, see the following.

Topics

- [Limitations on Using SQL Server as a Source for AWS DMS \(p. 100\)](#)
- [Using Ongoing Replication \(CDC\) from a SQL Server Source \(p. 101\)](#)
- [Supported Compression Methods \(p. 105\)](#)
- [Working with SQL Server AlwaysOn Availability Groups \(p. 105\)](#)
- [Configuring a SQL Server Database as a Replication Source for AWS DMS \(p. 105\)](#)
- [Extra Connection Attributes When Using SQL Server as a Source for AWS DMS \(p. 106\)](#)
- [Source Data Types for SQL Server \(p. 107\)](#)

Limitations on Using SQL Server as a Source for AWS DMS

The following limitations apply when using a SQL Server database as a source for AWS DMS:

- The identity property for a column is not migrated to a target database column.

- In AWS DMS engine versions before version 2.4.x, changes to rows with more than 8000 bytes of information, including header and mapping information, are not processed correctly due to limitations in the SQL Server TLOG buffer size. Use the latest AWS DMS version to avoid this issue.
- The SQL Server endpoint does not support the use of sparse tables.
- Windows Authentication is not supported.
- Changes to computed fields in a SQL Server are not replicated.
- Temporal tables are not supported.
- SQL Server partition switching is not supported.
- A clustered index on the source is created as a nonclustered index on the target.
- When using the WRITETEXT and UPDATETEXT utilities, AWS DMS does not capture events applied on the source database.
- The following data manipulation language (DML) pattern is not supported:

```
SELECT <*> INTO <new_table> FROM <existing_table>
```

- When using SQL Server as a source, column-level encryption is not supported.
- Due to a known issue with SQL Server 2008 and 2008 R2, AWS DMS doesn't support server level audits on SQL Server 2008 and SQL Server 2008 R2 as a source endpoint.

For example, running the following command causes AWS DMS to fail:

```
USE [master]
GO
ALTER SERVER AUDIT [my_audit_test-20140710] WITH (STATE=on)
GO
```

Using Ongoing Replication (CDC) from a SQL Server Source

You can use ongoing replication (change data capture, or CDC) for a self-managed SQL Server database on-premises or on Amazon EC2, or an Amazon-managed database on Amazon RDS.

AWS DMS supports ongoing replication for these SQL Server configurations:

- For source SQL Server instances that are on-premises or on Amazon EC2, AWS DMS supports ongoing replication for SQL Server Enterprise, Standard, and Developer Edition.
- For source SQL Server instances running on Amazon RDS, AWS DMS supports ongoing replication for SQL Server Enterprise through SQL Server 2016 SP1. Beyond this version, AWS DMS supports CDC for both SQL Server Enterprise and Standard editions.

If you want AWS DMS to automatically set up the ongoing replication, the AWS DMS user account that you use to connect to the source database must have the sysadmin fixed server role. If you don't want to assign the sysadmin role to the user account you use, you can still use ongoing replication by following the series of manual steps discussed following.

The following requirements apply specifically when using ongoing replication with a SQL Server database as a source for AWS DMS:

- SQL Server must be configured for full backups, and you must perform a backup before beginning to replicate data.
- The recovery model must be set to **Bulk logged** or **Full**.

- SQL Server backup to multiple disks isn't supported. If the backup is defined to write the database backup to multiple files over different disks, AWS DMS can't read the data and the AWS DMS task fails.
- For self-managed SQL Server sources, be aware that SQL Server Replication Publisher definitions for the source database used in a DMS CDC task aren't removed when you remove a task. A SQL Server system administrator must delete these definitions from SQL Server for self-managed sources.
- During CDC, AWS DMS needs to look up SQL Server transaction log backups to read changes. AWS DMS doesn't support using SQL Server transaction log backups that were created using third-party backup software.
- For self-managed SQL Server sources, be aware that SQL Server doesn't capture changes on newly created tables until they've been published. When tables are added to a SQL Server source, AWS DMS manages creating the publication. However, this process might take several minutes. Operations made to newly created tables during this delay aren't captured or replicated to the target.
- AWS DMS change data capture requires FULLLOGGING to be turned on in SQL Server. To turn on FULLLOGGING in SQL Server, either enable MS-REPLICATION or CHANGE DATA CAPTURE (CDC).
- You can't reuse the SQL Server *tlog* until the changes have been processed.
- CDC operations aren't supported on memory-optimized tables. This limitation applies to SQL Server 2014 (when the feature was first introduced) and later.

Capturing Data Changes for SQL Server

For a self-managed SQL Server source, AWS DMS uses the following:

- MS-Replication, to capture changes for tables with primary keys. You can configure this automatically by giving the AWS DMS endpoint user `sysadmin` privileges on the source SQL Server instance. Alternatively, you can follow the steps provided in this section to prepare the source and use a non-`sysadmin` user for the AWS DMS endpoint.
- MS-CDC, to capture changes for tables without primary keys. MS-CDC must be enabled at the database level, and for all of the tables individually.

For a SQL Server source running on Amazon RDS, AWS DMS uses MS-CDC to capture changes for tables, with or without primary keys. MS-CDC must be enabled at the database level, and for all of the tables individually, using the Amazon RDS-specific stored procedures described in this section.

There are several ways you can use a SQL Server database for ongoing replication (CDC):

- Set up ongoing replication using the `sysadmin` role. (This applies only to self-managed SQL Server sources.)
- Set up ongoing replication to not use the `sysadmin` role. (This applies only to self-managed SQL Server sources.)
- Set up ongoing replication for an Amazon RDS for SQL Server DB instance.

Setting Up Ongoing Replication Using the `sysadmin` Role

For tables with primary keys, AWS DMS can configure the required artifacts on the source. For tables without primary keys, you need to set up MS-CDC.

First, enable MS-CDC for the database by running the following command. Use an account that has the `sysadmin` role assigned to it.

```
use [DBname]
EXEC sys.sp_cdc_enable_db
```

Next, enable MS-CDC for each of the source tables by running the following command.

```
EXECUTE sys.sp_cdc_enable_table @source_schema = N'MySchema', @source_name =  
N'MyTable', @role_name = NULL;
```

For more information on setting up MS-CDC for specific tables, see the [SQL Server documentation](#).

Setting Up Ongoing Replication Without Assigning the sysadmin Role

You can set up ongoing replication for a SQL Server database source that doesn't require the user account to have sysadmin privileges.

To set up a SQL Server database source for ongoing replication without using the sysadmin role

1. Create a new SQL Server account with password authentication using SQL Server Management Studio (SSMS). In this example, we use an account called `dmstest`.
2. In the **User Mappings** section of SSMS, choose the MSDB and MASTER databases (which gives public permission) and assign the DB_OWNER role for the database you want to use ongoing replication.
3. Open the context (right-click) menu for the new account, choose **Security** and explicitly grant the Connect SQL privilege.
4. Run the following grant commands.

```
GRANT SELECT ON FN_DBLOG TO dmstest;  
GRANT SELECT ON FN_DUMP_DBLOG TO dmstest;  
GRANT VIEW SERVER STATE TO dmstest;  
use msdb;  
GRANT EXECUTE ON MSDB.DBO.SP_STOP_JOB TO dmstest;  
GRANT EXECUTE ON MSDB.DBO.SP_START_JOB TO dmstest;  
GRANT SELECT ON MSDB.DBO.BACKUPSET TO dmstest;  
GRANT SELECT ON MSDB.DBO.BACKUPMEDIAFAMILY TO dmstest;  
GRANT SELECT ON MSDB.DBO.BACKUPFILE TO dmstest;
```

5. In SSMS, open the context (right-click) menu for the **Replication** folder, and then choose **Configure Distribution**. Follow all default steps and configure this SQL Server instance for distribution. A distribution database is created under databases.
6. Create a publication using the procedure following.
7. Create a new AWS DMS task with SQL Server as the source endpoint using the user account you created.

Note

The steps in this procedure apply only for tables with primary keys. You still need to enable MS-CDC for tables without primary keys.

Creating a SQL Server Publication for Ongoing Replication

To use CDC with SQL Server, you must create a publication for each table that is participating in ongoing replication.

To create a publication for SQL Server ongoing replication

1. Login to SSMS using the SYSADMIN user account.
2. Expand **Replication**.
3. Right click **Local Publications**.
4. In the **New Publication Wizard**, choose **Next**.
5. Select the database where you want to create the publication.
6. Choose **Transactional publication**. Choose **Next**.

7. Expand **Tables** and select the tables with PK (also these tables you want to publish). Choose **Next**.
8. You don't need to create a filter, so choose **Next**.
9. You don't need to create a Snapshot Agent, so choose **Next**.
10. Choose **Security Settings** and choose **Run under the SQL Server Agent service account**. Make sure to choose **By impersonating the process account** for publisher connection. Choose **OK**.
11. Choose **Next**.
12. Choose **Create the publication**.
13. Provide a name of the publication in the following format:

AR_PUBLICATION_000<DBID>. For example, you could name the publication AR_PUBLICATION_00018. You can also use the DB_ID function in SQL Server. For more information on the DB_ID function, see [the SQL Server documentation](#).

Setting Up Ongoing Replication on an Amazon RDS for SQL Server DB Instance

Amazon RDS for SQL Server supports MS-CDC for all versions of Amazon RDS for SQL Server Enterprise editions up to SQL Server 2016 SP1. Standard editions of SQL Server 2016 SP1 and later versions support MS-CDC for Amazon RDS for SQL Server.

Unlike self-managed SQL Server sources, Amazon RDS for SQL Server doesn't support MS-Replication. Therefore, AWS DMS needs to use MS-CDC for tables with or without primary keys.

Amazon RDS does not grant sysadmin privileges for setting replication artifacts that AWS DMS uses for on-going changes in a source SQL Server instance. You must enable MS-CDC on the Amazon RDS instance using master user privileges in the following procedure.

To enable MS-CDC on an RDS for SQL Server DB instance

1. Run the following query at the database level.

```
exec msdb.dbo.rds_cdc_enable_db '<DB instance name>'
```

2. For each table with a primary key, run the following query to enable MS-CDC.

```
exec sys.sp_cdc_enable_table
@source_schema = N'db_name',
@source_name = N'table_name',
@role_name = NULL,
@supports_net_changes = 1
GO
```

For each table with unique keys but no primary key, run the following query to enable MS-CDC.

```
exec sys.sp_cdc_enable_table
@source_schema = N'db_name',
@source_name = N'table_name',
@index_name = N'unique_index_name'
@role_name = NULL,
@supports_net_changes = 1
GO
```

For each table with no primary key nor unique keys, run the following query to enable MS-CDC.

```
exec sys.sp_cdc_enable_table
```

```
@source_schema = N'db_name',  
@source_name = N'table_name',  
@role_name = NULL  
GO
```

3. Set the retention period for changes to be available on the source using the following command.

```
EXEC sys.sp_cdc_change_job @job_type = 'capture' ,@pollinginterval = 86400
```

The parameter `@pollinginterval` is measured in seconds. The preceding command retains changes for one day. AWS recommends a one day retention period when using MS-CDC with AWS DMS.

Supported Compression Methods

The following table shows the compression methods that AWS DMS supports for each SQL Server version.

SQL Server Version	Row/Page Compression (at Partition Level)	Vardecimal Storage Format
2005	No	No
2008	Yes	No
2012	Yes	No
2014	Yes	No

Note

Sparse columns and columnar structure compression are not supported.

Working with SQL Server AlwaysOn Availability Groups

The SQL Server AlwaysOn Availability Groups feature is a high-availability and disaster-recovery solution that provides an enterprise-level alternative to database mirroring.

To use AlwaysOn Availability Groups as a source in AWS DMS, do the following:

- Enable the Distribution option on all SQL Server instances in your Availability Replicas.
- In the AWS DMS console, open the SQL Server source database settings. For **Server Name**, specify the Domain Name Service (DNS) name or IP address that was configured for the Availability Group Listener.

When you start an AWS DMS task for the first time, it might take longer than usual to start because the creation of the table articles is being duplicated by the Availability Groups Server.

Configuring a SQL Server Database as a Replication Source for AWS DMS

You can configure a SQL Server database as a replication source for AWS DMS (AWS DMS). For the most complete replication of changes, we recommend that you use the Enterprise, Standard, or Developer edition of SQL Server. One of these versions is required because these are the only versions that include MS-Replication(EE,SE) and MS-CDC(EE,DEV). The source SQL Server must also be configured for full

backups. In addition, AWS DMS must connect with a user (a SQL Server instance login) that has the sysadmin fixed server role on the SQL Server database you are connecting to.

Following, you can find information about configuring SQL Server as a replication source for AWS DMS.

Extra Connection Attributes When Using SQL Server as a Source for AWS DMS

You can use extra connection attributes to configure your SQL Server source. You specify these settings when you create the source endpoint. Multiple extra connection attribute settings should be separated by a semicolon.

The following table shows the extra connection attributes you can use with SQL Server as a source:

Name	Description
safeguardPolicy	<p>For optimal performance, AWS DMS tries to capture all unread changes from the active transaction log (TLOG). However, sometimes due to truncation, the active TLOG might not contain all of the unread changes. When this occurs, AWS DMS accesses the backup log to capture the missing changes. To minimize the need to access the backup log, AWS DMS prevents truncation using one of the following methods:</p> <ol style="list-style-type: none"> 1. Start transactions in the database: This is the default method. When this method is used, AWS DMS prevents TLOG truncation by mimicking a transaction in the database. As long as such a transaction is open, changes that appear after the transaction started aren't truncated. If you need Microsoft Replication to be enabled in your database, then you must choose this method. 2. Exclusively use sp_repldone within a single task: When this method is used, AWS DMS reads the changes and then uses sp_repldone to mark the TLOG transactions as ready for truncation. Although this method does not involve any transactional activities, it can only be used when Microsoft Replication is not running. Also, when using this method, only one AWS DMS task can access the database at any given time. Therefore, if you need to run parallel AWS DMS tasks against the same database, use the default method. <p>Default value: RELY_ON_SQL_SERVER_REPLICATION_AGENT</p> <p>Valid values: {EXCLUSIVE_AUTOMATIC_TRUNCATION, RELY_ON_SQL_SERVER_REPLICATION_AGENT}</p> <p>Example: safeguardPolicy= RELY_ON_SQL_SERVER_REPLICATION_AGENT</p>
readBackupOnly	<p>When this parameter is set to <code>Y</code>, AWS DMS only reads changes from transaction log backups and does not read from the active transaction log file during ongoing replication. Setting this parameter to <code>Y</code> can add up some source latency to ongoing replication but it lets you control</p>

Name	Description
	<p>active transaction log file growth during full load and ongoing replication tasks.</p> <p>Valid values: N or Y. The default is N.</p> <p>Example: <code>readBackupOnly=Y</code></p>

Source Data Types for SQL Server

Data migration that uses SQL Server as a source for AWS DMS supports most SQL Server data types. The following table shows the SQL Server source data types that are supported when using AWS DMS and the default mapping from AWS DMS data types.

For information on how to view the data type that is mapped in the target, see the section for the target endpoint you are using.

For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service \(p. 319\)](#).

SQL Server Data Types	AWS DMS Data Types
BIGINT	INT8
BIT	BOOLEAN
DECIMAL	NUMERIC
INT	INT4
MONEY	NUMERIC
NUMERIC (p,s)	NUMERIC
SMALLINT	INT2
SMALLMONEY	NUMERIC
TINYINT	UINT1
REAL	REAL4
FLOAT	REAL8
DATETIME	DATETIME
DATETIME2 (SQL Server 2008 and later)	DATETIME
SMALLDATETIME	DATETIME
DATE	DATE
TIME	TIME
DATETIMEOFFSET	WSTRING
CHAR	STRING
VARCHAR	STRING

SQL Server Data Types	AWS DMS Data Types
VARCHAR (max)	<p>CLOB</p> <p>TEXT</p> <p>To use this data type with AWS DMS, you must enable the use of CLOB data types for a specific task.</p> <p>For SQL Server tables, AWS DMS updates LOB columns in the target even for UPDATE statements that don't change the value of the LOB column in SQL Server.</p> <p>During CDC, AWS DMS supports CLOB data types only in tables that include a primary key.</p>
NCHAR	WSTRING
NVARCHAR (length)	WSTRING
NVARCHAR (max)	<p>NCLOB</p> <p>NTEXT</p> <p>To use this data type with AWS DMS, you must enable the use of NCLOB data types for a specific task.</p> <p>For SQL Server tables, AWS DMS updates LOB columns in the target even for UPDATE statements that don't change the value of the LOB column in SQL Server.</p> <p>During CDC, AWS DMS supports CLOB data types only in tables that include a primary key.</p>
BINARY	BYTES
VARBINARY	BYTES
VARBINARY (max)	<p>BLOB</p> <p>IMAGE</p> <p>For SQL Server tables, AWS DMS updates LOB columns in the target even for UPDATE statements that don't change the value of the LOB column in SQL Server.</p> <p>To use this data type with AWS DMS, you must enable the use of BLOB data types for a specific task.</p> <p>AWS DMS supports BLOB data types only in tables that include a primary key.</p>
TIMESTAMP	BYTES

SQL Server Data Types	AWS DMS Data Types
UNIQUEIDENTIFIER	STRING
HIERARCHYID	Use HIERARCHYID when replicating to a SQL Server target endpoint. Use WSTRING (250) when replicating to all other target endpoints.
XML	NCLOB For SQL Server tables, AWS DMS updates LOB columns in the target even for UPDATE statements that don't change the value of the LOB column in SQL Server. To use this data type with AWS DMS, you must enable the use of NCLOB data types for a specific task. During CDC, AWS DMS supports NCLOB data types only in tables that include a primary key.
GEOMETRY	Use GEOMETRY when replicating to target endpoints that support this data type. Use CLOB when replicating to target endpoints that don't support this data type.
GEOGRAPHY	Use GEOGRAPHY when replicating to target endpoints that support this data type. Use CLOB when replicating to target endpoints that don't support this data type.

AWS DMS doesn't support tables that include fields with the following data types:

- CURSOR
- SQL_VARIANT
- TABLE

Note

User-defined data types are supported according to their base type. For example, a user-defined data type based on DATETIME is handled as a DATETIME data type.

Using Microsoft Azure SQL Database as a Source for AWS DMS

With AWS DMS, you can use Microsoft Azure SQL Database as a source in much the same way as you do SQL Server. AWS DMS supports, as a source, the same list of database versions that are supported for SQL Server running on-premises or on an Amazon EC2 instance.

For more information, see [Using a Microsoft SQL Server Database as a Source for AWS DMS \(p. 100\)](#).

Note

AWS DMS doesn't support change data capture operations (CDC) with Azure SQL Database.

Using a PostgreSQL Database as a Source for AWS DMS

You can migrate data from one or many PostgreSQL databases using AWS DMS. With a PostgreSQL database as a source, you can migrate data to either another PostgreSQL database or one of the other supported databases. AWS DMS supports a PostgreSQL version 9.4 and later database as a source for on-premises databases, databases on an EC2 instance, and databases on an Amazon RDS DB instance.

Note

PostgreSQL versions 10.x and later contain numerous changes in function names and folder names from previous versions. If you are using PostgreSQL version 10.x or later as a source for AWS DMS, see the topic [Using PostgreSQL Version 10.x and Later as a Source for AWS DMS \(p. 117\)](#) for information on preparing a database as a source for AWS DMS.

You can use SSL to encrypt connections between your PostgreSQL endpoint and the replication instance. For more information on using SSL with a PostgreSQL endpoint, see [Using SSL With AWS Database Migration Service \(p. 47\)](#).

For a homogeneous migration from a PostgreSQL database to a PostgreSQL database on AWS, the following is true:

- JSONB columns on the source are migrated to JSONB columns on the target.
- JSON columns are migrated as JSON columns on the target.
- HSTORE columns are migrated as HSTORE columns on the target.

For a heterogeneous migration with PostgreSQL as the source and a different database engine as the target, the situation is different. In this case, JSONB, JSON, and HSTORE columns are converted to the AWS DMS intermediate type of NCLOB and then translated to the corresponding NCLOB column type on the target. In this case, AWS DMS treats JSONB data as if it were a LOB column. During the full load phase of a migration, the target column must be nullable.

AWS DMS supports change data capture (CDC) for PostgreSQL tables with primary keys. If a table doesn't have a primary key, the write-ahead logs (WAL) don't include a before image of the database row and AWS DMS can't update the table.

AWS DMS supports CDC on Amazon RDS PostgreSQL databases when the DB instance is configured to use logical replication. Amazon RDS supports logical replication for a PostgreSQL DB instance version 9.4.9 and higher and 9.5.4 and higher.

For additional details on working with PostgreSQL databases and AWS DMS, see the following sections.

Topics

- [Migrating from PostgreSQL to PostgreSQL Using AWS DMS \(p. 111\)](#)
- [Prerequisites for Using a PostgreSQL Database as a Source for AWS DMS \(p. 113\)](#)
- [Security Requirements When Using a PostgreSQL Database as a Source for AWS DMS \(p. 113\)](#)
- [Limitations on Using a PostgreSQL Database as a Source for AWS DMS \(p. 114\)](#)
- [Setting Up an Amazon RDS PostgreSQL DB Instance as a Source \(p. 115\)](#)
- [Removing AWS DMS Artifacts from a PostgreSQL Source Database \(p. 117\)](#)
- [Additional Configuration Settings When Using a PostgreSQL Database as a Source for AWS DMS \(p. 117\)](#)
- [Using PostgreSQL Version 10.x and Later as a Source for AWS DMS \(p. 117\)](#)
- [Extra Connection Attributes When Using PostgreSQL as a Source for AWS DMS \(p. 119\)](#)

- [Source Data Types for PostgreSQL \(p. 120\)](#)

Migrating from PostgreSQL to PostgreSQL Using AWS DMS

For a heterogeneous migration, where you are migrating from a database engine other than PostgreSQL to a PostgreSQL database, AWS DMS is almost always the best migration tool to use. But for a homogeneous migration, where you are migrating from a PostgreSQL database to a PostgreSQL database, native tools can be more effective.

We recommend that you use native PostgreSQL database migration tools such as `pg_dump` under the following conditions:

- You have a homogeneous migration, where you are migrating from a source PostgreSQL database to a target PostgreSQL database.
- You are migrating an entire database.
- The native tools allow you to migrate your data with minimal downtime.

The `pg_dump` utility uses the `COPY` command to create a schema and data dump of a PostgreSQL database. The dump script generated by `pg_dump` loads data into a database with the same name and recreates the tables, indexes, and foreign keys. You can use the `pg_restore` command and the `-d` parameter to restore the data to a database with a different name.

For more information about importing a PostgreSQL database into Amazon RDS for PostgreSQL or Amazon Aurora (PostgreSQL), see <https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide//PostgreSQL.Procedural.Importing.html>.

Using DMS to Migrate Data from PostgreSQL to PostgreSQL

AWS DMS can migrate data from, for example, a source PostgreSQL database that is on premises to a target Amazon RDS for PostgreSQL or Amazon Aurora (PostgreSQL) instance. Core or basic PostgreSQL data types most often migrate successfully.

Data types that are supported on the source database but are not supported on the target may not migrate successfully. AWS DMS streams some data types as strings if the data type is unknown. Some data types, such as XML and JSON, can successfully migrate as small files but can fail if they are large documents.

The following table shows source PostgreSQL data types and whether they can be migrated successfully:

Data type	Migrates successfully	Will partially migrate	Will not migrate	Comments
INTEGER	X			
SMALLINT	X			
BIGINT	X			
NUMERIC/DECIMAL(p,s)		X		With $0 < p < 39$ and $0 < s$
NUMERIC/DECIMAL		X		$p > 38$ or $p = s = 0$
REAL	X			
DOUBLE	X			
SMALLSERIAL	X			

Data type	Migrates successfully	Will partially migrate	Will not migrate	Comments
SERIAL	X			
BIGSERIAL	X			
MONEY	X			
CHAR		X		Without specified precision
CHAR(n)	X			
VARCHAR		X		Without specified precision
VARCHAR(n)	X			
TEXT	X			
BYTEA	X			
TIMESTAMP	X			
TIMESTAMP(Z)		X		
DATE	X			
TIME	X			
TIME (z)		X		
INTERVAL		X		
BOOLEAN	X			
ENUM			X	
CIDR			X	
INET			X	
MACADDR			X	
TSVECTOR			X	
TSQUERY			X	
XML		X		
POINT			X	
LINE			X	
LSEG			X	
BOX			X	
PATH			X	

Data type	Migrates successfully	Will partially migrate	Will not migrate	Comments
POLYGON			X	
CIRCLE			X	
JSON		X		
ARRAY			X	
COMPOSITE			X	
RANGE			X	

Prerequisites for Using a PostgreSQL Database as a Source for AWS DMS

For a PostgreSQL database to be a source for AWS DMS, you should do the following:

- Use a PostgreSQL database that is version 9.4.x or later.
- Grant superuser permissions for the user account specified for the PostgreSQL source database.
- Add the IP address of the AWS DMS replication server to the `pg_hba.conf` configuration file.
- Set the following parameters and values in the `postgresql.conf` configuration file:
 - Set `wal_level = logical`
 - Set `max_replication_slots` to a value greater than 1.

The `max_replication_slots` value should be set according to the number of tasks that you want to run. For example, to run five tasks you need to set a minimum of five slots. Slots open automatically as soon as a task starts and remain open even when the task is no longer running. You need to manually delete open slots.

- Set `max_wal_senders` to a value greater than 1.

The `max_wal_senders` parameter sets the number of concurrent tasks that can run.

- Set `wal_sender_timeout = 0`

The `wal_sender_timeout` parameter terminates replication connections that are inactive longer than the specified number of milliseconds. Although the default is 60 seconds, we recommend that you set this parameter to zero, which disables the timeout mechanism.

- The parameter `idle_in_transaction_session_timeout` in PostgreSQL versions 9.6 and later lets you cause idle transactions to time out and fail. Some AWS DMS transactions are idle for some time before the AWS DMS engine uses them again. Do not end idle transactions when you use AWS DMS.

Security Requirements When Using a PostgreSQL Database as a Source for AWS DMS

The only security requirement when using PostgreSQL as a source is that the user account specified must be a registered user in the PostgreSQL database.

Limitations on Using a PostgreSQL Database as a Source for AWS DMS

The following limitations apply when using PostgreSQL as a source for AWS DMS:

- A captured table must have a primary key. If a table doesn't have a primary key, AWS DMS ignores DELETE and UPDATE record operations for that table.
- Timestamp with a time zone type column is not supported.
- AWS DMS ignores an attempt to update a primary key segment. In these cases, the target identifies the update as one that didn't update any rows. However, because the results of updating a primary key in PostgreSQL are unpredictable, no records are written to the exceptions table.
- AWS DMS doesn't support the **Start Process Changes from Timestamp** run option.
- AWS DMS supports full load and change processing on Amazon RDS for PostgreSQL. For information on how to prepare a PostgreSQL DB instance and to set it up for using CDC, see [Setting Up an Amazon RDS PostgreSQL DB Instance as a Source \(p. 115\)](#).
- Replication of multiple tables with the same name but where each name has a different case (for example table1, TABLE1, and Table1) can cause unpredictable behavior, and therefore AWS DMS doesn't support it.
- AWS DMS supports change processing of CREATE, ALTER, and DROP DDL statements for tables unless the tables are held in an inner function or procedure body block or in other nested constructs.

For example, the following change is not captured:

```
CREATE OR REPLACE FUNCTION attu.create_distributors1() RETURNS void
LANGUAGE plpgsql
AS $$
BEGIN
create table attu.distributors1(did serial PRIMARY KEY,name
varchar(40) NOT NULL);
END;
$$$;
```

- AWS DMS doesn't support change processing of TRUNCATE operations.
- The OID LOB data type is not migrated to the target.
- If your source is an on-premises PostgreSQL database or a PostgreSQL database on an Amazon EC2 instance, ensure that the `test_decoding` output plugin (found in the `Postgres contrib` package) is installed on your source endpoint. For more information about the `test_decoding` plugin, see the [PostgreSQL documentation](#).
- AWS DMS doesn't support change processing to set column default values (using the ALTER COLUMN SET DEFAULT clause on ALTER TABLE statements).
- AWS DMS doesn't support change processing to set column nullability (using the ALTER COLUMN [SET|DROP] NOT NULL clause on ALTER TABLE statements).
- AWS DMS doesn't support replication of partitioned tables. When a partitioned table is detected, the following occurs:
 - The endpoint reports a list of parent and child tables.
 - AWS DMS creates the table on the target as a regular table with the same properties as the selected tables.
 - If the parent table in the source database has the same primary key value as its child tables, a "duplicate key" error is generated.

Note

To replicate partitioned tables from a PostgreSQL source to a PostgreSQL target, you first need to manually create the parent and child tables on the target. Then you define a separate task

to replicate to those tables. In such a case, you set the task configuration to **Truncate before loading**.

Note

The PostgreSQL `NUMERIC` datatype is not fixed in size. When transferring data that is a `NUMERIC` data type but without precision and scale DMS uses `NUMERIC(28, 6)` (a precision of 28 and scale of 6) by default. As an example the value 0.611111104488373 from the source will be converted to 0.611111 on the PostgreSQL target.

Setting Up an Amazon RDS PostgreSQL DB Instance as a Source

You can use an Amazon RDS for PostgreSQL DB instance or Read Replica as a source for AWS DMS. A DB instance can be used for both full-load and CDC (ongoing replication); a Read Replica can only be used for full-load tasks and cannot be used for CDC.

You use the AWS master user account for the PostgreSQL DB instance as the user account for the PostgreSQL source endpoint for AWS DMS. The master user account has the required roles that allow it to set up change data capture (CDC). If you use an account other than the master user account, the account must have the `rds_superuser` role and the `rds_replication` role. The `rds_replication` role grants permissions to manage logical slots and to stream data using logical slots.

If you don't use the master user account for the DB instance, you must create several objects from the master user account for the account that you use. For information about creating the needed objects, see [Migrating an Amazon RDS for PostgreSQL Database Without Using the Master User Account \(p. 115\)](#).

Using CDC with an RDS for PostgreSQL DB Instance

You can use PostgreSQL's native logical replication feature to enable CDC during a database migration of an Amazon RDS PostgreSQL DB instance. This approach reduces downtime and ensures that the target database is in sync with the source PostgreSQL database. Amazon RDS supports logical replication for a PostgreSQL DB instance version 9.4.9 and higher and 9.5.4 and higher.

Note

Amazon RDS for PostgreSQL Read Replicas cannot be used for CDC (ongoing replication).

To enable logical replication for an RDS PostgreSQL DB instance, do the following:

- In general, use the AWS master user account for the PostgreSQL DB instance as the user account for the PostgreSQL source endpoint. The master user account has the required roles that allow it to set up CDC. If you use an account other than the master user account, you must create several objects from the master account for the account that you use. For more information, see [Migrating an Amazon RDS for PostgreSQL Database Without Using the Master User Account \(p. 115\)](#).
- Set the `rds.logical_replication` parameter in your DB parameter group to 1. This is a static parameter that requires a reboot of the DB instance for the parameter to take effect. As part of applying this parameter, AWS DMS sets the `wal_level`, `max_wal_senders`, `max_replication_slots`, and `max_connections` parameters. These parameter changes can increase WAL generation, so you should only set the `rds.logical_replication` parameter when you are using logical slots.
- A best practice is to set the `wal_sender_timeout` parameter to 0. Setting this parameter to 0 prevents PostgreSQL from terminating replication connections that are inactive longer than the specified timeout. When AWS DMS is migrating data, replication connections need to be able to last longer than the specified timeout.

Migrating an Amazon RDS for PostgreSQL Database Without Using the Master User Account

If you don't use the master user account for the Amazon RDS PostgreSQL DB instance that you are using as a source, you need to create several objects to capture data definition language (DDL) events. You

create these objects in the account other than the master account and then create a trigger in the master user account.

Note

If you set the `captureDDL` parameter to `N` on the source endpoint, you don't have to create the following table and trigger on the source database.

Use the following procedure to create these objects. The user account other than the master account is referred to as the `NoPriv` account in this procedure.

To create objects

1. Choose the schema where the objects are to be created. The default schema is `public`. Ensure that the schema exists and is accessible by the `NoPriv` account.
2. Log in to the PostgreSQL DB instance using the `NoPriv` account.
3. Create the table `awsdms_ddl_audit` by running the following command, replacing `<objects_schema>` in the code following with the name of the schema to use.

```
create table <objects_schema>.awsdms_ddl_audit
(
  c_key      bigserial primary key,
  c_time     timestamp,      -- Informational
  c_user     varchar(64),    -- Informational: current_user
  c_txn      varchar(16),    -- Informational: current transaction
  c_tag      varchar(24),    -- Either 'CREATE TABLE' or 'ALTER TABLE' or 'DROP TABLE'
  c_oid      integer,       -- For future use - TG_OBJECTID
  c_name     varchar(64),    -- For future use - TG_OBJECTNAME
  c_schema   varchar(64),    -- For future use - TG_SCHEMANAME. For now - holds
  current_schema
  c_ddlqry   text           -- The DDL query associated with the current DDL event
)
```

4. Create the function `awsdms_intercept_ddl` by running the following command, replacing `<objects_schema>` in the code following with the name of the schema to use.

```
CREATE OR REPLACE FUNCTION <objects_schema>.awsdms_intercept_ddl()
  RETURNS event_trigger
LANGUAGE plpgsql
SECURITY DEFINER
AS $$
  declare _qry text;
BEGIN
  if (tg_tag='CREATE TABLE' or tg_tag='ALTER TABLE' or tg_tag='DROP TABLE') then
    SELECT current_query() into _qry;
    insert into <objects_schema>.awsdms_ddl_audit
      values
      (
        default,current_timestamp,current_user,cast(TXID_CURRENT()as
        varchar(16)),tg_tag,0,'',current_schema,_qry
      );
    delete from <objects_schema>.awsdms_ddl_audit;
  end if;
END;
$$;
```

5. Log out of the `NoPriv` account and log in with an account that has the `rds_superuser` role assigned to it.

6. Create the event trigger `awsdms_intercept_ddl` by running the following command.

```
CREATE EVENT TRIGGER awsdms_intercept_ddl ON ddl_command_end  
EXECUTE PROCEDURE <objects_schema>.awsdms_intercept_ddl();
```

When you have completed the procedure preceding, you can create the AWS DMS source endpoint using the `NoPriv` account.

Removing AWS DMS Artifacts from a PostgreSQL Source Database

To capture DDL events, AWS DMS creates various artifacts in the PostgreSQL database when a migration task starts. When the task completes, you might want to remove these artifacts. To remove the artifacts, issue the following statements (in the order they appear), where `{AmazonRDSMigration}` is the schema in which the artifacts were created:

```
drop event trigger awsdms_intercept_ddl;
```

The event trigger doesn't belong to a specific schema.

```
drop function {AmazonRDSMigration}.awsdms_intercept_ddl()  
drop table {AmazonRDSMigration}.awsdms_ddl_audit  
drop schema {AmazonRDSMigration}
```

Note

Dropping a schema should be done with extreme caution, if at all. Never drop an operational schema, especially not a public one.

Additional Configuration Settings When Using a PostgreSQL Database as a Source for AWS DMS

You can add additional configuration settings when migrating data from a PostgreSQL database in two ways:

- You can add values to the extra connection attribute to capture DDL events and to specify the schema in which the operational DDL database artifacts are created. For more information, see [Extra Connection Attributes When Using PostgreSQL as a Source for AWS DMS \(p. 119\)](#).
- You can override connection string parameters. Select this option if you need to do either of the following:
 - Specify internal AWS DMS parameters. Such parameters are rarely required and are therefore not exposed in the user interface.
 - Specify pass-through (`passthru`) values for the specific database client. AWS DMS includes pass-through parameters in the connection string passed to the database client.

Using PostgreSQL Version 10.x and Later as a Source for AWS DMS

PostgreSQL version 10.x and later databases have numerous changes in function names and folder names from previous PostgreSQL versions. These changes make certain migration actions not backward compatible.

Because most of the name changes are superficial, AWS DMS has created wrapper functions that let AWS DMS work with PostgreSQL version 10.x and later. The wrapper functions are prioritized higher than functions in `pg_catalog`. In addition, we ensure that schema visibility of existing schemas isn't changed so that we don't override any other system catalog functions such as user-defined functions.

To use these wrapper functions before you perform any migration tasks, run the following SQL code on the source PostgreSQL database. Use the same AWS DMS user account as you are using for the target database.

```
BEGIN;
CREATE SCHEMA IF NOT EXISTS fnRenames;
CREATE OR REPLACE FUNCTION fnRenames.pg_switch_xlog() RETURNS pg_lsn AS $$
    SELECT pg_switch_wal(); $$ LANGUAGE SQL;
CREATE OR REPLACE FUNCTION fnRenames.pg_xlog_replay_pause() RETURNS VOID AS $$
    SELECT pg_wal_replay_pause(); $$ LANGUAGE SQL;
CREATE OR REPLACE FUNCTION fnRenames.pg_xlog_replay_resume() RETURNS VOID AS $$
    SELECT pg_wal_replay_resume(); $$ LANGUAGE SQL;
CREATE OR REPLACE FUNCTION fnRenames.pg_current_xlog_location() RETURNS pg_lsn AS $$
    SELECT pg_current_wal_lsn(); $$ LANGUAGE SQL;
CREATE OR REPLACE FUNCTION fnRenames.pg_is_xlog_replay_paused() RETURNS boolean AS $$
    SELECT pg_is_wal_replay_paused(); $$ LANGUAGE SQL;
CREATE OR REPLACE FUNCTION fnRenames.pg_xlogfile_name(lsn pg_lsn) RETURNS TEXT AS $$
    SELECT pg_walfile_name(lsn); $$ LANGUAGE SQL;
CREATE OR REPLACE FUNCTION fnRenames.pg_last_xlog_replay_location() RETURNS pg_lsn AS $$
    SELECT pg_last_wal_replay_lsn(); $$ LANGUAGE SQL;
CREATE OR REPLACE FUNCTION fnRenames.pg_last_xlog_receive_location() RETURNS pg_lsn AS $$
    SELECT pg_last_wal_receive_lsn(); $$ LANGUAGE SQL;
CREATE OR REPLACE FUNCTION fnRenames.pg_current_xlog_flush_location() RETURNS pg_lsn AS $$
    SELECT pg_current_wal_flush_lsn(); $$ LANGUAGE SQL;
CREATE OR REPLACE FUNCTION fnRenames.pg_current_xlog_insert_location() RETURNS pg_lsn AS $
$
    SELECT pg_current_wal_insert_lsn(); $$ LANGUAGE SQL;
CREATE OR REPLACE FUNCTION fnRenames.pg_xlog_location_diff(lsn1 pg_lsn, lsn2 pg_lsn)
    RETURNS NUMERIC AS $$
    SELECT pg_wal_lsn_diff(lsn1, lsn2); $$ LANGUAGE SQL;
CREATE OR REPLACE FUNCTION fnRenames.pg_xlogfile_name_offset(lsn pg_lsn, OUT TEXT, OUT
    INTEGER) AS $$
    SELECT pg_walfile_name_offset(lsn); $$ LANGUAGE SQL;
CREATE OR REPLACE FUNCTION fnRenames.pg_create_logical_replication_slot(slot_name name,
    plugin name,
    temporary BOOLEAN DEFAULT FALSE, OUT slot_name name, OUT xlog_position pg_lsn) RETURNS
    RECORD AS $$
    SELECT slot_name::NAME, lsn::pg_lsn FROM
    pg_catalog.pg_create_logical_replication_slot(slot_name, plugin,
    temporary); $$ LANGUAGE SQL;
ALTER USER <user name> SET search_path = fnRenames, pg_catalog, "$user", public;

-- DROP SCHEMA fnRenames CASCADE;
-- ALTER USER PG_User SET search_path TO DEFAULT;
COMMIT;
```

Note

If you do not invoke this preparatory code on a source PostgreSQL 10.x database, an error is raised like this.

```
2018-10-29T02:57:50 [SOURCE_CAPTURE ]E: RetCode: SQL_ERROR SqlState: 42703
NativeError: 1 Message:
    ERROR: column &quot;xlog_position&quot; does not exist;,
    No query has been executed with that handle [1022502] (ar_odbc_stmt.c:3647)
```

Extra Connection Attributes When Using PostgreSQL as a Source for AWS DMS

You can use extra connection attributes to configure your PostgreSQL source. You specify these settings when you create the source endpoint. Multiple extra connection attribute settings should be separated by a semicolon.

The following table shows the extra connection attributes you can use when using PostgreSQL as a source for AWS DMS:

Name	Description
captureDDL	<p>To capture DDL events, AWS DMS creates various artifacts in the PostgreSQL database when the task starts. You can later remove these artifacts as described in Removing AWS DMS Artifacts from a PostgreSQL Source Database (p. 117).</p> <p>If this value is set to N, you don't have to create tables or triggers on the source database. For more information, see Migrating an Amazon RDS for PostgreSQL Database Without Using the Master User Account (p. 115).</p> <p>Streamed DDL events are captured.</p> <p>Default value: Y</p> <p>Valid values: Y/N</p> <p>Example: captureDDLs=Y</p>
ddlArtifactsSchema	<p>The schema in which the operational DDL database artifacts are created.</p> <p>Default value: public</p> <p>Valid values: String</p> <p>Example: ddlArtifactsSchema=xyzddl-schema</p>
failTasksOnLobTruncation	<p>When set to <code>true</code>, this value causes a task to fail if the actual size of a LOB column is greater than the specified <code>LobMaxSize</code>.</p> <p>If task is set to Limited LOB mode and this option is set to <code>true</code>, the task fails instead of truncating the LOB data.</p> <p>Default value: false</p> <p>Valid values: Boolean</p> <p>Example: failTasksOnLobTruncation=true</p>
executeTimeout	<p>Sets the client statement timeout for the PostgreSQL instance, in seconds. The default value is 60 seconds.</p> <p>Example: executeTimeout=100</p>

Source Data Types for PostgreSQL

The following table shows the PostgreSQL source data types that are supported when using AWS DMS and the default mapping to AWS DMS data types.

For information on how to view the data type that is mapped in the target, see the section for the target endpoint you are using.

For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service \(p. 319\)](#).

PostgreSQL Data Types	AWS DMS Data Types
INTEGER	INT4
SMALLINT	INT2
BIGINT	INT8
NUMERIC (p,s)	If precision is from 0 through 38, then use NUMERIC. If precision is 39 or greater, then use STRING.
DECIMAL(P,S)	If precision is from 0 through 38, then use NUMERIC. If precision is 39 or greater, then use STRING.
REAL	REAL4
DOUBLE	REAL8
SMALLSERIAL	INT2
SERIAL	INT4
BIGSERIAL	INT8
MONEY	NUMERIC(38,4) Note: The MONEY data type is mapped to FLOAT in SQL Server.
CHAR	WSTRING (1)
CHAR(N)	WSTRING (n)
VARCHAR(N)	WSTRING (n)
TEXT	NCLOB
BYTEA	BLOB
TIMESTAMP	TIMESTAMP
TIMESTAMP (z)	TIMESTAMP
TIMESTAMP with time zone	Not supported
DATE	DATE

PostgreSQL Data Types	AWS DMS Data Types
TIME	TIME
TIME (z)	TIME
INTERVAL	STRING (128)—1 YEAR, 2 MONTHS, 3 DAYS, 4 HOURS, 5 MINUTES, 6 SECONDS
BOOLEAN	CHAR (5) false or true
ENUM	STRING (64)
CIDR	STRING (50)
INET	STRING (50)
MACADDR	STRING (18)
BIT (n)	STRING (n)
BIT VARYING (n)	STRING (n)
UUID	STRING
TSVECTOR	CLOB
TSQUERY	CLOB
XML	CLOB
POINT	STRING (255) "(x,y)"
LINE	STRING (255) "(x,y,z)"
LSEG	STRING (255) "((x1,y1),(x2,y2))"
BOX	STRING (255) "((x1,y1),(x2,y2))"
PATH	CLOB "((x1,y1),(xn,yn))"
POLYGON	CLOB "((x1,y1),(xn,yn))"
CIRCLE	STRING (255) "(x,y),r"
JSON	NCLOB
JSONB	NCLOB
ARRAY	NCLOB
COMPOSITE	NCLOB
HSTORE	NCLOB
INT4RANGE	STRING (255)
INT8RANGE	STRING (255)
NUMRANGE	STRING (255)
STRRANGE	STRING (255)

Using a MySQL-Compatible Database as a Source for AWS DMS

You can migrate data from any MySQL-compatible database (MySQL, MariaDB, or Amazon Aurora MySQL) using AWS Database Migration Service. MySQL versions 5.5, 5.6, and 5.7, and also MariaDB and Amazon Aurora MySQL, are supported for on-premises. All AWS-managed MySQL databases (Amazon RDS for MySQL, Amazon RDS for MariaDB, Amazon Aurora MySQL) are supported as sources for AWS DMS.

You can use SSL to encrypt connections between your MySQL-compatible endpoint and the replication instance. For more information on using SSL with a MySQL-compatible endpoint, see [Using SSL With AWS Database Migration Service \(p. 47\)](#).

In the following sections, the term "self-managed" applies to any database that is installed either on-premises or on Amazon EC2. The term "Amazon-managed" applies to any database on Amazon RDS, Amazon Aurora, or Amazon Simple Storage Service.

For additional details on working with MySQL-compatible databases and AWS DMS, see the following sections.

Topics

- [Migrating from MySQL to MySQL Using AWS DMS \(p. 122\)](#)
- [Using Any MySQL-Compatible Database as a Source for AWS DMS \(p. 124\)](#)
- [Using a Self-Managed MySQL-Compatible Database as a Source for AWS DMS \(p. 124\)](#)
- [Using an Amazon-Managed MySQL-Compatible Database as a Source for AWS DMS \(p. 125\)](#)
- [Limitations on Using a MySQL Database as a Source for AWS DMS \(p. 126\)](#)
- [Extra Connection Attributes When Using MySQL as a Source for AWS DMS \(p. 127\)](#)
- [Source Data Types for MySQL \(p. 128\)](#)

Migrating from MySQL to MySQL Using AWS DMS

For a heterogeneous migration, where you are migrating from a database engine other than MySQL to a MySQL database, AWS DMS is almost always the best migration tool to use. But for a homogeneous migration, where you are migrating from a MySQL database to a MySQL database, native tools can be more effective.

We recommend that you use native MySQL database migration tools such as `mysqldump` under the following conditions:

- You have a homogeneous migration, where you are migrating from a source MySQL database to a target MySQL database.
- You are migrating an entire database.
- The native tools allow you to migrate your data with minimal downtime.

You can import data from an existing MySQL or MariaDB database to an Amazon RDS MySQL or MariaDB DB instance. You do so by copying the database with `mysqldump` and piping it directly into the Amazon RDS MySQL or MariaDB DB instance. The `mysqldump` command-line utility is commonly used to make backups and transfer data from one MySQL or MariaDB server to another. It is included with MySQL and MariaDB client software.

For more information about importing a MySQL database into Amazon RDS for MySQL or Amazon Aurora (MySQL), see [Importing Data into a MySQL DB Instance](#) and [Importing Data from a MySQL or MariaDB DB to an Amazon RDS MySQL or MariaDB DB Instance](#).

Using AWS DMS to Migrate Data from MySQL to MySQL

AWS DMS can migrate data from, for example, a source MySQL database that is on premises to a target Amazon RDS for MySQL or Amazon Aurora (MySQL) instance. Core or basic MySQL data types most often migrate successfully.

Data types that are supported on the source database but are not supported on the target may not migrate successfully. AWS DMS streams some data types as strings if the data type is unknown. Some data types, such as XML and JSON, can successfully migrate as small files but can fail if they are large documents.

The following table shows source MySQL data types and whether they can be migrated successfully:

Data type	Migrates successfully	Will partially migrate	Will not migrate	Comments
INT	X			
BIGINT	X			
MEDIUMINT	X			
TINYINT	X			
DECIMAL(p,s)	X			
BINARY	X			
BIT(M)	X			
BLOB	X			
LONGBLOB	X			
MEDIUMBLOB	X			
TINYBLOB	X			
DATE	X			
DATETIME	X			
TIME		X		
TIMESTAMP	X			
YEAR	X			
DOUBLE	X			
FLOAT		X		
VARCHAR(N)	X			
VARBINARY(N)	X			
CHAR(N)	X			
TEXT	X			
LONGTEXT	X			

Data type	Migrates successfully	Will partially migrate	Will not migrate	Comments
MEDIUMTEXT	X			
TINYTEXT	X			
GEOMETRY			X	
POINT			X	
LINestring			X	
POLYGON			X	
MULTILINESTRING			X	
MULTIPOLYGON			X	
GEOMETRYCOLLECTION			X	
ENUM		X		
SET		X		

Using Any MySQL-Compatible Database as a Source for AWS DMS

Before you begin to work with a MySQL database as a source for AWS DMS, make sure that you have the following prerequisites. These prerequisites apply to either self-managed or Amazon-managed sources.

You must have an account for AWS DMS that has the Replication Admin role. The role needs the following privileges:

- **REPLICATION CLIENT** – This privilege is required for change data capture (CDC) tasks only. In other words, full-load-only tasks don't require this privilege.
- **REPLICATION SLAVE** – This privilege is required for change data capture (CDC) tasks only. In other words, full-load-only tasks don't require this privilege.
- **SUPER** – This privilege is required only in MySQL versions before 5.6.6.

The AWS DMS user must also have SELECT privileges for the source tables designated for replication.

Using a Self-Managed MySQL-Compatible Database as a Source for AWS DMS

You can use the following self-managed MySQL-compatible databases as sources for AWS DMS:

- MySQL Community Edition
- MySQL Standard Edition
- MySQL Enterprise Edition
- MySQL Cluster Carrier Grade Edition
- MariaDB Community Edition
- MariaDB Enterprise Edition
- MariaDB Column Store

You must enable binary logging if you plan to use change data capture (CDC). To enable binary logging, the following parameters must be configured in MySQL's `my.ini` (Windows) or `my.cnf` (UNIX) file.

Parameter	Value
<code>server_id</code>	Set this parameter to a value of 1 or greater.
<code>log-bin</code>	Set the path to the binary log file, such as <code>log-bin=E:\MySQL_Logs\BinLog</code> . Don't include the file extension.
<code>binlog_format</code>	Set this parameter to <code>ROW</code> .
<code>expire_logs_days</code>	Set this parameter to a value of 1 or greater. To prevent overuse of disk space, we recommend that you don't use the default value of 0.
<code>binlog_checksum</code>	Set this parameter to <code>NONE</code> .
<code>binlog_row_image</code>	Set this parameter to <code>FULL</code> .
<code>log_slave_updates</code>	Set this parameter to <code>TRUE</code> if you are using a MySQL or MariaDB read-replica as a source.

If your source uses the NDB (clustered) database engine, the following parameters must be configured to enable CDC on tables that use that storage engine. Add these changes in MySQL's `my.ini` (Windows) or `my.cnf` (UNIX) file.

Parameter	Value
<code>ndb_log_bin</code>	Set this parameter to <code>ON</code> . This value ensures that changes in clustered tables are logged to the binary log.
<code>ndb_log_update_as_write</code>	Set this parameter to <code>OFF</code> . This value prevents writing <code>UPDATE</code> statements as <code>INSERT</code> statements in the binary log.
<code>ndb_log_updated_only</code>	Set this parameter to <code>OFF</code> . This value ensures that the binary log contains the entire row and not just the changed columns.

Using a Amazon-Managed MySQL-Compatible Database as a Source for AWS DMS

You can use the following Amazon-managed MySQL-compatible databases as sources for AWS DMS:

- MySQL Community Edition
- MariaDB Community Edition
- Amazon Aurora MySQL

When using an Amazon-managed MySQL-compatible database as a source for AWS DMS, make sure that you have the following prerequisites:

- You must enable automatic backups. For more information on setting up automatic backups, see [Working with Automated Backups](#) in the *Amazon RDS User Guide*.
- You must enable binary logging if you plan to use change data capture (CDC). For more information on setting up binary logging for an Amazon RDS MySQL database, see [Working with Automated Backups](#) in the *Amazon RDS User Guide*.

- You must ensure that the binary logs are available to AWS DMS. Because Amazon-managed MySQL-compatible databases purge the binary logs as soon as possible, you should increase the length of time that the logs remain available. For example, to increase log retention to 24 hours, run the following command.

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

- The `binlog_format` parameter should be set to "ROW."
- The `binlog_checksum` parameter should be set to "NONE". For more information about setting parameters in Amazon RDS MySQL, see [Working with Automated Backups](#) in the *Amazon RDS User Guide*.
- If you are using an Amazon RDS MySQL or Amazon RDS MariaDB read replica as a source, then backups must be enabled on the read replica.

Limitations on Using a MySQL Database as a Source for AWS DMS

When using a MySQL database as a source, AWS DMS doesn't support the following:

- Change data capture (CDC) is not supported for Amazon RDS MySQL 5.5 or lower. For Amazon RDS MySQL, you must use version 5.6 or higher to enable CDC.
- The data definition language (DDL) statements DROP TABLE and RENAME TABLE are not supported. Additionally, all DDL statements for partitioned tables are not supported.
- For partitioned tables on the source, when you set **Target table preparation mode** to **Drop tables on target**, AWS DMS creates a simple table without any partitions on the MySQL target. To migrate partitioned tables to a partitioned table on the target, pre-create the partitioned tables on the target MySQL database.
- Using an ALTER TABLE<table_name> ADD COLUMN <column_name> statement to add columns to the beginning (FIRST) or the middle of a table (AFTER) is not supported. Columns are always added to the end of the table.
- CDC is not supported when a table name contains uppercase and lowercase characters, and the source engine is hosted on an operating system with case-insensitive file names. An example is Windows or OS X using HFS+.
- The AR_H_USER header column is not supported.
- The AUTO_INCREMENT attribute on a column is not migrated to a target database column.
- Capturing changes when the binary logs are not stored on standard block storage is not supported. For example, CDC doesn't work when the binary logs are stored on Amazon Simple Storage Service.
- AWS DMS creates target tables with the InnoDB storage engine by default. If you need to use a storage engine other than InnoDB, you must manually create the table and migrate to it using **"do nothing" mode**.
- You can't use Aurora MySQL read replicas as a source for AWS DMS.
- If the MySQL-compatible source is stopped during full load, the AWS DMS task doesn't stop with an error. The task ends successfully, but the target might be out of sync with the source. If this happens, either restart the task or reload the affected tables.
- Indexes created on a portion of a column value aren't migrated. For example, the index CREATE INDEX first_ten_chars ON customer (name(10)) isn't created on the target.
- In some cases, the task is configured to not replicate LOBs ("SupportLobs" is false in task settings or "Don't include LOB columns" is checked in the task console). In these cases, AWS DMS doesn't migrate any MEDIUMBLOB, LONGBLOB, MEDIUMTEXT, and LONGTEXT columns to the target.

BLOB, TINYBLOB, TEXT, and TINYTEXT columns are not affected and are migrated to the target.

Extra Connection Attributes When Using MySQL as a Source for AWS DMS

You can use extra connection attributes to configure a MySQL source. You specify these settings when you create the source endpoint. Multiple extra connection attribute settings should be separated from each other by semicolons.

The following table shows the extra connection attributes available when using Amazon RDS MySQL as a source for AWS DMS.

Name	Description
<code>eventsPollInterval</code>	<p>Specifies how often to check the binary log for new changes/events when the database is idle.</p> <p>Default value: 5</p> <p>Valid values: 1 - 60</p> <p>Example: <code>eventsPollInterval=5</code></p> <p>In the example, AWS DMS checks for changes in the binary logs every five seconds.</p>
<code>initstmt=SET time_zone</code>	<p>Specifies the time zone for the source MySQL database. Timestamps are translated to the specified timezone.</p> <p>Default value: UTC</p> <p>Valid values: Any three-character abbreviation for the time zone you want to use, such as UTC, EST, or GMT. Valid values are the standard time zone abbreviations for the operating system hosting the source MySQL database.</p> <p>Example: <code>initstmt=SET time_zone=UTC</code></p>
<code>afterConnectScript</code>	<p>Specifies a script to run immediately after AWS DMS connects to the endpoint. The migration task continues running regardless if the SQL statement succeeds or fails.</p> <p>Valid values: One or more valid SQL statements, set off by a semicolon.</p> <p>Example: <code>afterConnectScript=ALTER SESSION SET CURRENT_SCHEMA = system;</code></p>
<code>CleanSrcMetadataOnMismatch</code>	<p>Cleans and recreates table metadata information on the replication instance when a mismatch occurs. For example, in a situation where running an alter DDL on the table could result in different information about the table cached in the replication instance. Boolean.</p> <p>Default value: <code>false</code></p> <p>Example: <code>CleanSrcMetadataOnMismatch=false</code></p>

Source Data Types for MySQL

The following table shows the MySQL database source data types that are supported when using AWS DMS and the default mapping from AWS DMS data types.

Note

The UTF-8 4-byte character set (utf8mb4) is not supported and can cause unexpected behavior in a source database. Plan to convert any data using the UTF-8 4-byte character set before migrating.

For information on how to view the data type that is mapped in the target, see the section for the target endpoint you are using.

For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service \(p. 319\)](#).

MySQL Data Types	AWS DMS Data Types
INT	INT4
MEDIUMINT	INT4
BIGINT	INT8
TINYINT	INT1
DECIMAL(10)	NUMERIC (10,0)
BINARY	BYTES(1)
BIT	BOOLEAN
BIT(64)	BYTES(8)
BLOB	BYTES(66535)
LONGBLOB	BLOB
MEDIUMBLOB	BLOB
TINYBLOB	BYTES(255)
DATE	DATE
DATETIME	DATETIME
TIME	STRING
TIMESTAMP	DATETIME
YEAR	INT2
DOUBLE	REAL8
FLOAT	REAL(DOUBLE) The supported FLOAT range is -1.79E+308 to -2.23E-308, 0 and 2.23E-308 to 1.79E+308 If FLOAT values aren't in this range, map the FLOAT data type to the STRING data type.

MySQL Data Types	AWS DMS Data Types
VARCHAR (45)	WSTRING (45)
VARCHAR (2000)	WSTRING (2000)
VARCHAR (4000)	WSTRING (4000)
VARBINARY (4000)	BYTES (4000)
VARBINARY (2000)	BYTES (2000)
CHAR	WSTRING
TEXT	WSTRING (65535)
LONGTEXT	NCLOB
MEDIUMTEXT	NCLOB
TINYTEXT	WSTRING (255)
GEOMETRY	BLOB
POINT	BLOB
LINestring	BLOB
POLYGON	BLOB
MULTIPOINT	BLOB
MULTILINestring	BLOB
MULTIPOLYGON	BLOB
GEOMETRYCOLLECTION	BLOB

Note

If the DATETIME and TIMESTAMP data types are specified with a "zero" value (that is, 0000-00-00), make sure that the target database in the replication task supports "zero" values for the DATETIME and TIMESTAMP data types. Otherwise, these values are recorded as null on the target.

The following MySQL data types are supported in full load only.

MySQL Data Types	AWS DMS Data Types
ENUM	STRING
SET	STRING

Using an SAP ASE Database as a Source for AWS DMS

You can migrate data from an SAP Adaptive Server Enterprise (ASE) database—formerly known as Sybase—using AWS DMS. With an SAP ASE database as a source, you can migrate data to any of the other supported AWS DMS target databases. AWS DMS supports SAP ASE versions 12.5.3 or higher, 15, 15.5, 15.7, 16 and later as sources.

Permissions Required for Using SAP ASE as a Source for AWS DMS

To use an SAP ASE database as a source in an AWS DMS task, grant the user account specified in the AWS DMS database definitions the following permissions in the SAP ASE database.

- sa_role
- replication_role
- sybase_ts_role
- If you enable the **Automatically enable Sybase replication** option (in the **Advanced** tab) when you create the SAP ASE source endpoint, also give permission to AWS DMS to run the stored procedure `sp_setreptable`.

Removing the Truncation Point

When a task starts, AWS DMS establishes a `$replication_truncation_point` entry in the `syslogshold` system view, indicating that a replication process is in progress. While AWS DMS is working, it advances the replication truncation point at regular intervals, according to the amount of data that has already been copied to the target.

After the `$replication_truncation_point` entry is established, keep the AWS DMS task running to prevent the database log from becoming excessively large. If you want to stop the AWS DMS task permanently, remove the replication truncation point by issuing the following command:

```
dbcc settrunc('ltm','ignore')
```

After the truncation point is removed, you can't resume the AWS DMS task. The log continues to be truncated automatically at the checkpoints (if automatic truncation is set).

Source Data Types for SAP ASE

For a list of the SAP ASE source data types that are supported when using AWS DMS and the default mapping from AWS DMS data types, see the following table. AWS DMS doesn't support SAP ASE source tables with columns of the user-defined type (UDT) data type. Replicated columns with this data type are created as NULL.

For information on how to view the data type that is mapped in the target, see the [Targets for Data Migration \(p. 147\)](#) section for your target endpoint.

For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service \(p. 319\)](#).

SAP ASE Data Types	AWS DMS Data Types
BIGINT	INT8
BINARY	BYTES
BIT	BOOLEAN
CHAR	STRING
DATE	DATE

SAP ASE Data Types	AWS DMS Data Types
DATETIME	DATETIME
DECIMAL	NUMERIC
DOUBLE	REAL8
FLOAT	REAL8
IMAGE	BLOB
INT	INT4
MONEY	NUMERIC
NCHAR	WSTRING
NUMERIC	NUMERIC
NVARCHAR	WSTRING
REAL	REAL4
SMALLDATETIME	DATETIME
SMALLINT	INT2
SMALLMONEY	NUMERIC
TEXT	CLOB
TIME	TIME
TINYINT	UINT1
UNICHAR	UNICODE CHARACTER
UNITEXT	NCLOB
UNIVARCHAR	UNICODE
VARBINARY	BYTES
VARCHAR	STRING

Using MongoDB as a Source for AWS DMS

AWS DMS supports MongoDB versions 2.6.x and 3.x as a database source.

If you are new to MongoDB, be aware of the following important MongoDB database concepts:

- A record in MongoDB is a *document*, which is a data structure composed of field and value pairs. The value of a field can include other documents, arrays, and arrays of documents. A document is roughly equivalent to a row in a relational database table.
- A *collection* in MongoDB is a group of documents, and is roughly equivalent to a relational database table.
- Internally, a MongoDB document is stored as a binary JSON (BSON) file in a compressed format that includes a type for each field in the document. Each document has a unique ID.

AWS DMS supports two migration modes when using MongoDB as a source. You specify the migration mode using the **Metadata mode** parameter using the AWS Management Console or the extra connection attribute `nestingLevel` when you create the MongoDB endpoint. The choice of migration mode affects the resulting format of the target data as explained following.

Document mode

In document mode, the MongoDB document is migrated as is, meaning that the document data is consolidated into a single column named `_doc` in a target table. Document mode is the default setting when you use MongoDB as a source endpoint.

For example, consider the following documents in a MongoDB collection called `myCollection`.

```
> db.myCollection.find()
{ "_id" : ObjectId("5a94815f40bd44d1b02bdfe0"), "a" : 1, "b" : 2, "c" : 3 }
{ "_id" : ObjectId("5a94815f40bd44d1b02bdfe1"), "a" : 4, "b" : 5, "c" : 6 }
```

After migrating the data to a relational database table using document mode, the data is structured as follows. The data fields in the MongoDB document are consolidated into the `_doc` column.

oid_id	_doc
5a94815f40bd44d1b02bdfe0	{ "a" : 1, "b" : 2, "c" : 3 }
5a94815f40bd44d1b02bdfe1	{ "a" : 4, "b" : 5, "c" : 6 }

You can optionally set the extra connection attribute `extractDocID` to `true` to create a second column named `"_id"` that acts as the primary key. If you are going to use change data capture (CDC), set this parameter to `true`.

In document mode, AWS DMS manages the creation and renaming of collections like this:

- If you add a new collection to the source database, AWS DMS creates a new target table for the collection and replicates any documents.
- If you rename an existing collection on the source database, AWS DMS doesn't rename the target table.

Table mode

In table mode, AWS DMS transforms each top-level field in a MongoDB document into a column in the target table. If a field is nested, AWS DMS flattens the nested values into a single column. AWS DMS then adds a key field and data types to the target table's column set.

For each MongoDB document, AWS DMS adds each key and type to the target table's column set. For example, using table mode, AWS DMS migrates the previous example into the following table.

oid_id	a	b	c
5a94815f40bd44d1b02bdfe0	1	2	3
5a94815f40bd44d1b02bdfe1	4	5	6

Nested values are flattened into a column containing dot-separated key names. The column is named the concatenation of the flattened field names separated by periods. For example, AWS DMS migrates a JSON document with a field of nested values such as `{ "a" : { "b" : { "c" : 1 } } }` into a column named `a.b.c`.

To create the target columns, AWS DMS scans a specified number of MongoDB documents and creates a set of all the fields and their types. AWS DMS then uses this set to create the columns of the target table. If you create or modify your MongoDB source endpoint using the console, you can specify the number of documents to scan. The default value is 1000 documents. If you use the AWS CLI, you can use the extra connection attribute `docsToInvestigate`.

In table mode, AWS DMS manages documents and collections like this:

- When you add a document to an existing collection, the document is replicated. If there are fields that don't exist in the target, those fields aren't replicated.
- When you update a document, the updated document is replicated. If there are fields that don't exist in the target, those fields aren't replicated.
- Deleting a document is fully supported.
- Adding a new collection doesn't result in a new table on the target when done during a CDC task.
- Renaming a collection is not supported.

Permissions Needed When Using MongoDB as a Source for AWS DMS

For an AWS DMS migration with a MongoDB source, you can create either a user account with root privileges, or a user with permissions only on the database to migrate.

The following code creates a user to be the root account.

```
use admin
db.createUser(
  {
    user: "root",
    pwd: "<password>",
    roles: [ { role: "root", db: "admin" } ]
  }
)
```

The following code creates a user with minimal privileges on the database to be migrated.

```
use <database_to_migrate>
db.createUser(
  {
    user: "<dms-user>",
    pwd: "<password>",
    roles: [ { role: "read", db: "local" }, "read" ]
  })
```

Configuring a MongoDB Replica Set for Change Data Capture (CDC)

To use ongoing replication or change data capture (CDC) with MongoDB, AWS DMS requires access to the MongoDB operations log (oplog). To create the oplog, you need to deploy a replica set if one doesn't exist. For more information, see [the MongoDB documentation](#).

You can use CDC with either the primary or secondary node of a MongoDB replica set as the source endpoint.

To convert a standalone instance to a replica set

1. Using the command line, connect to mongo.

```
mongo localhost
```

2. Stop the mongod service.

```
service mongod stop
```

3. Restart mongod using the following command:

```
mongod --replSet "rs0" --auth -port <port_number>
```

4. Test the connection to the replica set using the following commands:

```
mongo -u root -p <password> --host rs0/localhost:<port_number>  
--authenticationDatabase "admin"
```

If you plan to perform a document mode migration, select option `_id` as a separate column when you create the MongoDB endpoint. Selecting this option creates a second column named `_id` that acts as the primary key. This second column is required by AWS DMS to support data manipulation language (DML) operations.

Security Requirements When Using MongoDB as a Source for AWS DMS

AWS DMS supports two authentication methods for MongoDB. The two authentication methods are used to encrypt the password, so they are only used when the `authType` parameter is set to `PASSWORD`.

The MongoDB authentication methods are as follows:

- **MONOGODB-CR** – the default when using MongoDB 2.x authentication.
- **SCRAM-SHA-1** – the default when using MongoDB version 3.x authentication.

If an authentication method is not specified, AWS DMS uses the default method for the version of the MongoDB source.

Limitations When Using MongoDB as a Source for AWS DMS

The following are limitations when using MongoDB as a source for AWS DMS:

- When the `_id` option is set as a separate column, the ID string can't exceed 200 characters.
- Object ID and array type keys are converted to columns that are prefixed with `oid` and `array` in table mode.

Internally, these columns are referenced with the prefixed names. If you use transformation rules in AWS DMS that reference these columns, you must specify the prefixed column. For example, you specify `#{oid__id}` and not `#{_id}`, or `#{array__addresses}` and not `#{_addresses}`.

- Collection names can't include the dollar symbol (`$`).
- Table mode and document mode have the limitations discussed preceding.

Extra Connection Attributes When Using MongoDB as a Source for AWS DMS

When you set up your MongoDB source endpoint, you can specify extra connection attributes. Extra connection attributes are specified by key-value pairs and separated by semicolons.

The following table describes the extra connection attributes available when using MongoDB databases as an AWS DMS source.

Attribute Name	Valid Values	Default Value and Description
authType	NO PASSWORD	PASSWORD – When NO is selected, user name and password parameters aren't used and can be empty.
authMechanism	DEFAULT MONGODB_CR SCRAM_SHA_1	DEFAULT – For MongoDB version 2.x, use MONGODB_CR. For MongoDB version 3.x, use SCRAM_SHA_1. This attribute isn't used when authType=NO.
nestingLevel	NONE ONE	NONE – Specify NONE to use document mode. Specify ONE to use table mode.
extractDocID	true false	false – Use this attribute when nestingLevel is set to NONE.
docsToInvestigate	A positive integer greater than 0.	1000 – Use this attribute when nestingLevel is set to ONE.
authSource	A valid MongoDB database name.	admin – This attribute isn't used when authType=NO.

Source Data Types for MongoDB

Data migration that uses MongoDB as a source for AWS DMS supports most MongoDB data types. In the following table, you can find the MongoDB source data types that are supported when using AWS DMS and the default mapping from AWS DMS data types. For more information about MongoDB data types, see [BSON Types](#) in the MongoDB documentation.

For information on how to view the data type that is mapped in the target, see the section for the target endpoint that you are using.

For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service](#) (p. 319).

MongoDB Data Types	AWS DMS Data Types
Boolean	Bool
Binary	BLOB
Date	Date
Timestamp	Date
Int	INT4
Long	INT8
Double	REAL8
String (UTF-8)	CLOB
Array	CLOB

MongoDB Data Types	AWS DMS Data Types
OID	String
REGEX	CLOB
CODE	CLOB

When you set up your MongoDB source endpoint, you can specify extra connection attributes. Extra connection attributes are specified by key-value pairs and separated by semicolons.

The following table describes the extra connection attributes available when using MongoDB databases as an AWS DMS source.

Attribute Name	Valid Values	Default Value and Description
authType	NO PASSWORD	PASSWORD – When NO is selected, user name and password parameters aren't used and can be empty.
authMechanism	DEFAULT MONGODB_CR SCRAM_SHA_1	DEFAULT – For MongoDB version 2.x, use MONGODB_CR. For MongoDB version 3.x, use SCRAM_SHA_1. This attribute isn't used when authType=NO.
nestingLevel	NONE ONE	NONE – Specify NONE to use document mode. Specify ONE to use table mode.
extractDocID	true false	false – Use this attribute when nestingLevel is set to NONE.
docsToInvestigate	A positive integer greater than 0.	1000 – Use this attribute when nestingLevel is set to ONE.
authSource	A valid MongoDB database name.	admin – This attribute isn't used when authType=NO.

Note

If the source endpoint is MongoDB, then the following extra connection attributes must be enabled:

- nestingLevel=NONE
- extractDocID=FALSE

For more information, see [Using Amazon DocumentDB as a Target for AWS Database Migration Service \(p. 198\)](#).

Source Data Types for MongoDB

Data migration that uses MongoDB as a source for AWS DMS supports most MongoDB data types. In the following table, you can find the MongoDB source data types that are supported when using AWS DMS

and the default mapping from AWS DMS data types. For more information about MongoDB data types, see [BSON Types](#) in the MongoDB documentation.

For information on how to view the data type that is mapped in the target, see the section for the target endpoint that you are using.

For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service](#) (p. 319).

MongoDB Data Types	AWS DMS Data Types
Boolean	Bool
Binary	BLOB
Date	Date
Timestamp	Date
Int	INT4
Long	INT8
Double	REAL8
String (UTF-8)	CLOB
Array	CLOB
OID	String
REGEX	CLOB
CODE	CLOB

Using Amazon Simple Storage Service as a Source for AWS DMS

You can migrate data from an Amazon Simple Storage Service bucket using AWS DMS. To do this, provide access to an Amazon S3 bucket containing one or more data files. In that S3 bucket, include a JSON file that describes the mapping between the data and the database tables of the data in those files.

The source data files must be present in the Amazon S3 bucket before the full load starts. You specify the bucket name using the `bucketName` parameter.

The source data files must be in comma separated value (CSV) format. Name them using the naming convention shown following. In this convention, `schemaName` is the source schema and `tableName` is the name of a table within that schema.

```
/schemaName/tableName/LOAD001.csv  
/schemaName/tableName/LOAD002.csv  
/schemaName/tableName/LOAD003.csv  
...
```

For example, suppose that your data files are in `mybucket`, at the following Amazon S3 path.


```
s3://mybucket/hr/employee
```

At load time, AWS DMS assumes that the source schema name is `hr`, and that the source table name is `employee`.

In addition to `bucketName` (which is required), you can optionally provide a `bucketFolder` parameter to specify where AWS DMS should look for data files in the Amazon S3 bucket. Continuing the previous example, if you set `bucketFolder` to `sourcedata`, then AWS DMS reads the data files at the following path.

```
s3://mybucket/sourcedata/hr/employee
```

You can specify the column delimiter, row delimiter, null value indicator, and other parameters using extra connection attributes. For more information, see [Extra Connection Attributes for Amazon S3 as a Source for AWS DMS](#) (p. 142).

Defining External Tables for Amazon S3 as a Source for AWS DMS

In addition to the data files, you must also provide an external table definition. An *external table definition* is a JSON document that describes how AWS DMS should interpret the data from Amazon S3. The maximum size of this document is 2 MB. If you create a source endpoint using the AWS DMS Management Console, you can enter the JSON directly into the table mapping box. If you use the AWS Command Line Interface (AWS CLI) or AWS DMS API to perform migrations, you can create a JSON file to specify the external table definition.

Suppose that you have a data file that includes the following.

```
101,Smith,Bob,4-Jun-14,New York
102,Smith,Bob,8-Oct-15,Los Angeles
103,Smith,Bob,13-Mar-17,Dallas
104,Smith,Bob,13-Mar-17,Dallas
```

Following is an example external table definition for this data.

```
{
  "TableCount": "1",
  "Tables": [
    {
      "TableName": "employee",
      "TablePath": "hr/employee/",
      "TableOwner": "hr",
      "TableColumns": [
        {
          "ColumnName": "Id",
          "ColumnType": "INT8",
          "ColumnNullable": "false",
          "ColumnIsPk": "true"
        },
        {
          "ColumnName": "LastName",
          "ColumnType": "STRING",
          "ColumnLength": "20"
        },
        {
          "ColumnName": "FirstName",
```

```
        "ColumnType": "STRING",
        "ColumnLength": "30"
      },
      {
        "ColumnName": "HireDate",
        "ColumnType": "DATETIME"
      },
      {
        "ColumnName": "OfficeLocation",
        "ColumnType": "STRING",
        "ColumnLength": "20"
      }
    ],
    "TableColumnsTotal": "5"
  }
]
```

The elements in this JSON document are as follows:

TableCount—the number of source tables. In this example, there is only one table.

Tables—an array consisting of one JSON map per source table. In this example, there is only one map. Each map consists of the following elements:

- **TableName**—the name of the source table.
- **TablePath**—the path in your Amazon S3 bucket where AWS DMS can find the full data load file. If a `bucketFolder` value is specified, this value is prepended to the path.
- **TableOwner**—the schema name for this table.
- **TableColumns**—an array of one or more maps, each of which describes a column in the source table:
 - **ColumnName**—the name of a column in the source table.
 - **ColumnType**—the data type for the column. For valid data types, see [Source Data Types for Amazon Simple Storage Service \(p. 143\)](#).
 - **ColumnLength**—the number of bytes in this column.
 - **ColumnNullable**—(optional) a Boolean value that is `true` if this column can contain NULL values.
 - **ColumnIsPk**—(optional) a Boolean value that is `true` if this column is part of the primary key.
- **TableColumnsTotal**—the total number of columns. This number must match the number of elements in the `TableColumns` array.

In the example preceding, some of the columns are of type `STRING`. In this case, use the `ColumnLength` element to specify the maximum number of characters.

`ColumnLength` applies for the following data types:

- `BYTE`
- `STRING`

If you don't specify otherwise, AWS DMS assumes that `ColumnLength` is zero.

For a column of the `NUMERIC` type, you need to specify the precision and scale. *Precision* is the total number of digits in a number, and *scale* is the number of digits to the right of the decimal point. You use the `ColumnPrecision` and `ColumnScale` elements for this, as shown following.

```
...
{
```

```
"ColumnName": "HourlyRate",  
"ColumnType": "NUMERIC",  
"ColumnPrecision": "5"  
"ColumnScale": "2"  
}  
...
```

Using CDC with Amazon S3 as a Source for AWS DMS

After AWS DMS performs a full data load, it can optionally replicate data changes to the target endpoint. To do this, you upload change data capture files (CDC files) to your Amazon S3 bucket. AWS DMS reads these CDC files when you upload them, and then applies the changes at the target endpoint.

The CDC files are named as follows:

```
CDC00001.csv  
CDC00002.csv  
CDC00003.csv  
...
```

To indicate where AWS DMS can find the files, you must specify the `cdcPath` parameter. Continuing the previous example, if you set `cdcPath` to `changedata`, then AWS DMS reads the CDC files at the following path.

```
s3://mybucket/changedata
```

The records in a CDC file are formatted as follows:

- Operation—the change operation to be performed: `INSERT`, `UPDATE`, or `DELETE`. These keywords are case-insensitive.
- Table name—the name of the source table.
- Schema name—the name of the source schema.
- Data—one or more columns that represent the data to be changed.

Following is an example CDC file for a table named `employee`.

```
INSERT,employee,hr,101,Smith,Bob,4-Jun-14,New York  
UPDATE,employee,hr,101,Smith,Bob,8-Oct-15,Los Angeles  
UPDATE,employee,hr,101,Smith,Bob,13-Mar-17,Dallas  
DELETE,employee,hr,101,Smith,Bob,13-Mar-17,Dallas
```

Prerequisites When Using Amazon S3 as a Source for AWS DMS

When you use Amazon S3 as a source for AWS DMS, the source Amazon S3 bucket that you use must be in the same AWS Region as the AWS DMS replication instance that you use to migrate your data. In addition, the AWS account you use for the migration must have read access to the source bucket.

The AWS Identity and Access Management (IAM) role assigned to the user account used to create the migration task must have the following set of permissions.

```
{
```

```

"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject"
    ],
    "Resource": [
      "arn:aws:s3:::mybucket*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "s3:ListBucket"
    ],
    "Resource": [
      "arn:aws:s3:::mybucket*"
    ]
  }
]
}

```

Extra Connection Attributes for Amazon S3 as a Source for AWS DMS

You can specify the following options as extra connection attributes.

Option	Description
bucketFolder	(Optional) A folder name in the S3 bucket. If this attribute is provided, source data files and CDC files are read from the path <code>bucketFolder/schemaName/tableName/</code> . If this attribute is not specified, then the path used is <code>schemaName/tableName/</code> . An example follows. bucketFolder=testFolder;
bucketName	The name of the S3 bucket. An example follows. bucketName=buckettest;
cdcPath	The location of change data capture (CDC) files. This attribute is required if a task captures change data; otherwise, it's optional. If cdcPath is present, then AWS DMS reads CDC files from this path and replicate the data changes to the target endpoint. For more information, see Using CDC with Amazon S3 as a Source for AWS DMS (p. 141) . An example follows. cdcPath=dataChanges;
	The delimiter used to separate rows in the source files. The default is a carriage return (<code>\r</code>). An example follows. csvRowDelimiter=\n;
csvDelimiter	The delimiter used to separate columns in the source files. The default is a comma. An example follows. csvDelimiter=,;

Option	Description
<code>externalTableDefinition</code>	<p>A JSON object that describes how AWS DMS should interpret the data in the Amazon S3 bucket during the migration. For more information, see Defining External Tables for Amazon S3 as a Source for AWS DMS (p. 139). An example follows.</p> <pre>externalTableDefinition=<json_object></pre>
<code>ignoreHeaderRows</code>	<p>When set to 1, AWS DMS ignores the first row header in a CSV file. A value of 1 enables the feature, a value of 0 disables the feature. The default is 0.</p> <pre>ignoreHeaderRows=1</pre>

Source Data Types for Amazon Simple Storage Service

Data migration that uses Amazon Simple Storage Service as a source for AWS DMS needs to map data from Amazon S3 to AWS DMS data types. For more information, see [Defining External Tables for Amazon S3 as a Source for AWS DMS \(p. 139\)](#).

For information on how to view the data type that is mapped in the target, see the section for the target endpoint you are using.

For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service \(p. 319\)](#).

AWS DMS Data Types—Amazon Simple Storage Service as Source
<p>BYTE</p> <p>Requires <code>ColumnLength</code>. For more information, see Defining External Tables for Amazon S3 as a Source for AWS DMS (p. 139).</p>
DATE
TIME
DATETIME
TIMESTAMP
INT1
INT2
INT4
INT8
<p>NUMERIC</p> <p>Requires <code>ColumnPrecision</code> and <code>ColumnScale</code>. For more information, see Defining External Tables for Amazon S3 as a Source for AWS DMS (p. 139).</p>
REAL4
REAL8
STRING

AWS DMS Data Types—Amazon Simple Storage Service as Source
Requires <code>ColumnLength</code> . For more information, see Defining External Tables for Amazon S3 as a Source for AWS DMS (p. 139).
UINT1
UINT2
UINT4
UINT8
BLOB
CLOB
BOOLEAN

Using an IBM Db2 for Linux, Unix, and Windows Database (Db2 LUW) as a Source for AWS DMS

You can migrate data from an IBM Db2 for Linux, Unix, and Windows (Db2 LUW) database to any supported target database using AWS DMS (AWS DMS). AWS DMS supports as a migration source the following Db2 LUW versions:

- Version 9.7, all Fix Packs are supported.
- Version 10.1, all Fix Packs are supported.
- Version 10.5, all Fix Packs except for Fix Pack 5 are supported.

You can use SSL to encrypt connections between your Db2 LUW endpoint and the replication instance. You must be using AWS DMS engine version 2.4.2 or higher to use SSL. For more information on using SSL with a Db2 LUW endpoint, see [Using SSL With AWS Database Migration Service](#) (p. 47).

Prerequisites When Using Db2 LUW as a Source for AWS DMS

The following prerequisites are required before you can use an Db2 LUW database as a source.

To enable ongoing replication, also called change data capture (CDC), you must do the following

- The database must be set to be recoverable. To capture changes, AWS DMS requires that the database is configured to be recoverable. A database is recoverable if either or both of the database configuration parameters `LOGARCHMETH1` and `LOGARCHMETH2` are set to `ON`.
- The user account must be granted the following permissions:

`SYSADM` or `DBADM`

`DATAACCESS`

Limitations When Using Db2 LUW as a Source for AWS DMS

Clustered databases are not supported. Note, however, that you can define a separate Db2 LUW for each of the endpoints of a cluster. See the IBM Db2 LUW documentation for more information.

When using ongoing replication (CDC), the following limitations apply:

- When truncating a table with multiple partitions, the number of DDL events shown in the AWS DMS console will be equal to the number of partitions. This is because Db2 LUW records a separate DDL for each partition.
- The following DDL actions are not supported on partitioned tables:
 - ALTER TABLE ADD PARTITION
 - ALTER TABLE DETACH PARTITION
 - ALTER TABLE ATTACH PARTITION
- The DECFLOAT data type is not supported. Consequently, changes to DECFLOAT columns are ignored during ongoing replication.
- The RENAME COLUMN statement is not supported.
- When performing updates to MDC (Multi-Dimensional Clustering) tables, each update is shown in the AWS DMS console as INSERT + DELETE.
- When the task setting **Include LOB columns in replication** is disabled, any table that has LOB columns is suspended during ongoing replication.
- When the Audit table option is enabled, the first timestamp record in the audit table will be NULL.
- When the Change table option is enabled, the first timestamp record in the table will be zero (i.e. 1970-01-01 00:00:00.000000).
- For Db2 LUW versions 10.5 and higher: Variable-length string columns with data that is stored out-of-row is ignored. Note that this limitation is only applicable to tables created with extended row size.

Extra Connection Attributes When Using Db2 LUW as a Source for AWS DMS

You can use extra connection attributes to configure your Db2 LUW source. You specify these settings when you create the source endpoint. Multiple extra connection attribute settings should be separated by a semicolon.

The following table shows the extra connection attributes you can use with Db2 LUW as a source:

Name	Description
MaxKBytesPerRead	Maximum number of bytes per read, as a NUMBER value. The default is 64 KB.
SetDataCaptureChanges	Enables ongoing replication (change data capture), as a BOOLEAN value. The default is true.

Source Data Types for IBM Db2 LUW

Data migration that uses Db2 LUW as a source for AWS DMS supports most Db2 LUW data types. The following table shows the Db2 LUW source data types that are supported when using AWS DMS and the default mapping from AWS DMS data types. For more information about Db2 LUW data types, see [the Db2 LUW documentation](#).

For information on how to view the data type that is mapped in the target, see the section for the target endpoint you are using.

For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service \(p. 319\)](#).

Db2 LUW Data Types	AWS DMS Data Types
INTEGER	INT4
SMALLINT	INT2
BIGINT	INT8
DECIMAL (p,s)	NUMERIC (p,s)
FLOAT	REAL8
DOUBLE	REAL8
REAL	REAL4
DECFLOAT (p)	If precision = 16, then: REAL8 If precision is = 34, then: STRING
GRAPHIC	WSTRING n<=127
VARGRAPHIC	WSTRING n<=16k double byte chars
LONG VARGRAPHIC	CLOB
CHAR (n)	STRING n<=255
VARCHAR (n)	STRING n<=32k
LONG VARCHAR (n)	CLOB n<=32k
CHAR (n) FOR BIT DATA	BYTES
VARCHAR (n) FOR BIT DATA	BYTES
LONG VARCHAR FOR BIT DATA	BYTES
DATE	DATE
TIME	TIME
TIMESTAMP	DATETIME
BLOB	BLOB
CLOB	CLOB

Db2 LUW Data Types	AWS DMS Data Types
	Maximum size: 2 GB
DBCLOB	CLOB Maximum size: 1 G double byte chars
XML	CLOB

Targets for Data Migration

AWS Database Migration Service (AWS DMS) can use many of the most popular databases as a target for data replication. The target can be on an Amazon Elastic Compute Cloud (Amazon EC2) instance, an Amazon Relational Database Service (Amazon RDS) instance, or an on-premises database.

The databases include the following:

On-premises and EC2 instance databases

- Oracle versions 10g, 11g, 12c, for the Enterprise, Standard, Standard One, and Standard Two editions
- Microsoft SQL Server versions 2005, 2008, 2008R2, 2012, 2014, and 2016, for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.
- MySQL versions 5.5, 5.6, and 5.7
- MariaDB (supported as a MySQL-compatible data target)
- PostgreSQL versions 9.4 and later
- SAP Adaptive Server Enterprise (ASE) versions 15, 15.5, 15.7, 16 and later

Amazon RDS instance databases, Amazon Redshift, Amazon Simple Storage Service, Amazon DynamoDB, Amazon Kinesis Data Streams and Amazon Elasticsearch Service

- Amazon RDS Oracle versions 11g (versions 11.2.0.3.v1 and later) and 12c, for the Enterprise, Standard, Standard One, and Standard Two editions
- Amazon RDS Microsoft SQL Server versions 2008R2, 2012, and 2014, for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.
- Amazon RDS MySQL versions 5.5, 5.6, and 5.7
- Amazon RDS MariaDB (supported as a MySQL-compatible data target)
- Amazon RDS PostgreSQL versions 9.4 and later
- Amazon Aurora with MySQL compatibility
- Amazon Aurora with PostgreSQL compatibility
- Amazon Redshift
- Amazon Simple Storage Service
- Amazon DynamoDB
- Amazon Elasticsearch Service
- Amazon Kinesis Data Streams

Topics

- [Using an Oracle Database as a Target for AWS Database Migration Service \(p. 148\)](#)
- [Using a Microsoft SQL Server Database as a Target for AWS Database Migration Service \(p. 152\)](#)
- [Using a PostgreSQL Database as a Target for AWS Database Migration Service \(p. 156\)](#)

- [Using a MySQL-Compatible Database as a Target for AWS Database Migration Service \(p. 159\)](#)
- [Using an Amazon Redshift Database as a Target for AWS Database Migration Service \(p. 163\)](#)
- [Using a SAP ASE Database as a Target for AWS Database Migration Service \(p. 170\)](#)
- [Using Amazon Simple Storage Service as a Target for AWS Database Migration Service \(p. 171\)](#)
- [Using an Amazon DynamoDB Database as a Target for AWS Database Migration Service \(p. 175\)](#)
- [Using Amazon Kinesis Data Streams as a Target for AWS Database Migration Service \(p. 189\)](#)
- [Using an Amazon Elasticsearch Service Cluster as a Target for AWS Database Migration Service \(p. 195\)](#)
- [Using Amazon DocumentDB as a Target for AWS Database Migration Service \(p. 198\)](#)

Using an Oracle Database as a Target for AWS Database Migration Service

You can migrate data to Oracle database targets using AWS DMS, either from another Oracle database or from one of the other supported databases. You can use Secure Sockets Layer (SSL) to encrypt connections between your Oracle endpoint and the replication instance. For more information on using SSL with an Oracle endpoint, see [Using SSL With AWS Database Migration Service \(p. 47\)](#).

AWS DMS supports Oracle versions 10g, 11g, and 12c for on-premises and EC2 instances for the Enterprise, Standard, Standard One, and Standard Two editions as targets. AWS DMS supports Oracle versions 11g (versions 11.2.0.3.v1 and later) and 12c for Amazon RDS instance databases for the Enterprise, Standard, Standard One, and Standard Two editions.

When using Oracle as a target, we assume that the data should be migrated into the schema or user that is used for the target connection. If you want to migrate data to a different schema, you need to use a schema transformation to do so. For example, suppose that your target endpoint connects to the user RDSMASTER and you want to migrate from the user PERFDATA to PERFDATA. In this case, create a transformation as follows.

```
{
  "rule-type": "transformation",
  "rule-id": "2",
  "rule-name": "2",
  "rule-action": "rename",
  "rule-target": "schema",
  "object-locator": {
    "schema-name": "PERFDATA"
  },
  "value": "PERFDATA"
}
```

For more information about transformations, see [Specifying Table Selection and Transformations by Table Mapping Using JSON \(p. 250\)](#).

For additional details on working with Oracle databases as a target for AWS DMS, see the following sections:

Topics

- [Limitations on Oracle as a Target for AWS Database Migration Service \(p. 149\)](#)
- [User Account Privileges Required for Using Oracle as a Target \(p. 149\)](#)
- [Configuring an Oracle Database as a Target for AWS Database Migration Service \(p. 150\)](#)
- [Extra Connection Attributes When Using Oracle as a Target for AWS DMS \(p. 150\)](#)

- [Target Data Types for Oracle \(p. 151\)](#)

Limitations on Oracle as a Target for AWS Database Migration Service

Limitations when using Oracle as a target for data migration include the following:

- AWS DMS does not create schema on the target Oracle database. You have to create any schemas you want on the target Oracle database. The schema name must already exist for the Oracle target. Tables from source schema are imported to user/schema, which AWS DMS uses to connect to the target instance. You must create multiple replication tasks if you have to migrate multiple schemas.
- AWS DMS doesn't support the `use direct path full load` option for tables with INDEXTYPE CONTEXT. As a workaround, you can use array load.
- In Batch Optimized Apply mode, loading into the net changes table uses Direct Path, which doesn't support XMLType. As a workaround, you can use Transactional Apply mode.

User Account Privileges Required for Using Oracle as a Target

To use an Oracle target in an AWS Database Migration Service task, for the user account specified in the AWS DMS Oracle database definitions you need to grant the following privileges in the Oracle database:

- SELECT ANY TRANSACTION
- SELECT on V\$NLS_PARAMETERS
- SELECT on V\$TIMEZONE_NAMES
- SELECT on ALL_INDEXES
- SELECT on ALL_OBJECTS
- SELECT on DBA_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
- DROP ANY TABLE
- SELECT ANY TABLE
- INSERT ANY TABLE
- UPDATE ANY TABLE
- CREATE ANY VIEW
- DROP ANY VIEW
- CREATE ANY PROCEDURE
- ALTER ANY PROCEDURE
- DROP ANY PROCEDURE
- CREATE ANY SEQUENCE
- ALTER ANY SEQUENCE
- DROP ANY SEQUENCE

For the requirements specified following, grant the additional privileges named:

- To use a specific table list, grant SELECT on any replicated table and also ALTER on any replicated table.
- To allow a user to create a table in his default tablespace, grant the privilege GRANT UNLIMITED TABLESPACE.
- For logon, grant the privilege CREATE SESSION.
- If you are using a direct path, grant the privilege LOCK ANY TABLE.
- If the "DROP and CREATE table" or "TRUNCATE before loading" option is selected in the full load settings, and the target table schema is different from that for the AWS DMS user, grant the privilege DROP ANY TABLE.
- To store changes in change tables or an audit table when the target table schema is different from that for the AWS DMS user, grant the privileges CREATE ANY TABLE and CREATE ANY INDEX.

Read Privileges Required for AWS Database Migration Service on the Target Database

The AWS DMS user account must be granted read permissions for the following DBA tables:

- SELECT on DBA_USERS
- SELECT on DBA_TAB_PRIVS
- SELECT on DBA_OBJECTS
- SELECT on DBA_SYNONYMS
- SELECT on DBA_SEQUENCES
- SELECT on DBA_TYPES
- SELECT on DBA_INDEXES
- SELECT on DBA_TABLES
- SELECT on DBA_TRIGGERS

If any of the required privileges cannot be granted to V\$xxx, then grant them to V_\$xxx.

Configuring an Oracle Database as a Target for AWS Database Migration Service

Before using an Oracle database as a data migration target, you must provide an Oracle user account to AWS DMS. The user account must have read/write privileges on the Oracle database, as specified in the section [User Account Privileges Required for Using Oracle as a Target](#) (p. 149).

Extra Connection Attributes When Using Oracle as a Target for AWS DMS

You can use extra connection attributes to configure your Oracle target. You specify these settings when you create the target endpoint. Multiple extra connection attribute settings should be separated by a semicolon.

The following table shows the extra connection attributes available when using Oracle as a target.

Name	Description
useDirectPathFullLoad	Use direct path full load, specify this to enable/disable the OCI direct path protocol for bulk loading Oracle tables. Default value: Y

Name	Description
	Valid values: Y/N Example: useDirectPathFullLoad=N
charLengthSemantics	Column length semantics specifies whether the length of a column is in bytes or in characters. Set this value to CHAR. Example: charLengthSemantics=CHAR

Target Data Types for Oracle

A target Oracle database used with AWS DMS supports most Oracle data types. The following table shows the Oracle target data types that are supported when using AWS DMS and the default mapping from AWS DMS data types. For more information about how to view the data type that is mapped from the source, see the section for the source you are using.

AWS DMS Data Type	Oracle Data Type
BOOLEAN	NUMBER (1)
BYTES	RAW (length)
DATE	DATETIME
TIME	TIMESTAMP (0)
DATETIME	TIMESTAMP (scale)
INT1	NUMBER (3)
INT2	NUMBER (5)
INT4	NUMBER (10)
INT8	NUMBER (19)
NUMERIC	NUMBER (p,s)
REAL4	FLOAT
REAL8	FLOAT
STRING	With date indication: DATE With time indication: TIMESTAMP With timestamp indication: TIMESTAMP With timestamp_with_timezone indication: TIMESTAMP WITH TIMEZONE With timestamp_with_local_timezone indication: TIMESTAMP WITH LOCAL TIMEZONE With interval_year_to_month indication: INTERVAL YEAR TO MONTH With interval_day_to_second indication: INTERVAL DAY TO SECOND If length > 4000: CLOB

AWS DMS Data Type	Oracle Data Type
	In all other cases: VARCHAR2 (length)
UINT1	NUMBER (3)
UINT2	NUMBER (5)
UINT4	NUMBER (10)
UINT8	NUMBER (19)
WSTRING	If length > 2000: NCLOB In all other cases: NVARCHAR2 (length)
BLOB	BLOB To use this data type with AWS DMS, you must enable the use of BLOBs for a specific task. BLOB data types are supported only in tables that include a primary key
CLOB	CLOB To use this data type with AWS DMS, you must enable the use of CLOBs for a specific task. During CDC, CLOB data types are supported only in tables that include a primary key.
NCLOB	NCLOB To use this data type with AWS DMS, you must enable the use of NCLOBs for a specific task. During CDC, NCLOB data types are supported only in tables that include a primary key.
XMLTYPE	The XMLTYPE target data type is only relevant in Oracle-to-Oracle replication tasks. When the source database is Oracle, the source data types are replicated "as is" to the Oracle target. For example, an XMLTYPE data type on the source is created as an XMLTYPE data type on the target.

Using a Microsoft SQL Server Database as a Target for AWS Database Migration Service

You can migrate data to Microsoft SQL Server databases using AWS DMS. With an SQL Server database as a target, you can migrate data from either another SQL Server database or one of the other supported databases.

For on-premises and Amazon EC2 instance databases, AWS DMS supports as a target SQL Server versions 2005, 2008, 2008R2, 2012, 2014, and 2016, for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.

For Amazon RDS instance databases, AWS DMS supports as a target SQL Server versions 2008R2, 2012, 2014, and 2016, for the Enterprise, Standard, Workgroup, and Developer editions are supported. The Web and Express editions are not supported.

For additional details on working with AWS DMS and SQL Server target databases, see the following.

Topics

- [Limitations on Using SQL Server as a Target for AWS Database Migration Service](#) (p. 153)
- [Security Requirements When Using SQL Server as a Target for AWS Database Migration Service](#) (p. 153)
- [Extra Connection Attributes When Using SQLServer as a Target for AWS DMS](#) (p. 153)
- [Target Data Types for Microsoft SQL Server](#) (p. 154)

Limitations on Using SQL Server as a Target for AWS Database Migration Service

The following limitations apply when using a SQL Server database as a target for AWS DMS:

- When you manually create a SQL Server target table with a computed column, full load replication is not supported when using the BCP bulk-copy utility. To use full load replication, disable the **Use BCP for loading tables** option in the console's **Advanced** tab. For more information on working with BCP, see the [Microsoft SQL Server documentation](#).
- When replicating tables with SQL Server spatial data types (GEOMETRY and GEOGRAPHY), AWS DMS replaces any spatial reference identifier (SRID) that you might have inserted with the default SRID. The default SRID is 0 for GEOMETRY and 4326 for GEOGRAPHY.
- Temporal tables are not supported. Migrating temporal tables may work with a replication-only task in transactional apply mode if those tables are manually created on the target.

Security Requirements When Using SQL Server as a Target for AWS Database Migration Service

The following describes the security requirements for using AWS DMS with a Microsoft SQL Server target.

- AWS DMS user account must have at least the `db_owner` user role on the Microsoft SQL Server database you are connecting to.
- A Microsoft SQL Server system administrator must provide this permission to all AWS DMS user accounts.

Extra Connection Attributes When Using SQLServer as a Target for AWS DMS

You can use extra connection attributes to configure your SQL Server target. You specify these settings when you create the target endpoint. Multiple extra connection attribute settings should be separated by a semicolon.

The following table shows the extra connection attributes that you can use when SQL Server is the target.

Name	Description
<code>useBCPFULLLOAD</code>	Use this to attribute to transfer data for full-load operations using BCP. When the target table contains an identity column that does not exist in the source table, you must disable the use BCP for loading table option. Default value: Y

Name	Description
	<p>Valid values: Y/N</p> <p>Example: useBCPFULLLOAD=Y</p>
BCPPacketSize	<p>The maximum size of the packets (in bytes) used to transfer data using BCP.</p> <p>Default value: 16384</p> <p>Valid values: 1–100000</p> <p>Example : BCPPacketSize=16384</p>
controlTablesFileGroup	<p>Specify a filegroup for the AWS DMS internal tables. When the replication task starts, all the internal AWS DMS control tables (awsdms_ apply_exception, awsdms_apply, awsdms_changes) are created on the specified filegroup.</p> <p>Default value: n/a</p> <p>Valid values: String</p> <p>Example: controlTablesFileGroup=filegroup1</p> <p>The following is an example of a command for creating a filegroup.</p> <pre>ALTER DATABASE replicate ADD FILEGROUP Test1FG1; GO ALTER DATABASE replicate ADD FILE (NAME = test1dat5, FILENAME = 'C:\temp\DATA\t1dat5.ndf', SIZE = 5MB, MAXSIZE = 100MB, FILEGROWTH = 5MB) TO FILEGROUP Test1FG1; GO</pre>

Target Data Types for Microsoft SQL Server

The following table shows the Microsoft SQL Server target data types that are supported when using AWS DMS and the default mapping from AWS DMS data types. For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service \(p. 319\)](#).

AWS DMS Data Type	SQL Server Data Type
BOOLEAN	TINYINT
BYTES	VARBINARY(length)
DATE	<p>For SQL Server 2008 and later, use DATE.</p> <p>For earlier versions, if the scale is 3 or less use DATETIME. In all other cases, use VARCHAR (37).</p>

AWS DMS Data Type	SQL Server Data Type
TIME	For SQL Server 2008 and later, use DATETIME2 (%d). For earlier versions, if the scale is 3 or less use DATETIME. In all other cases, use VARCHAR (37).
DATETIME	For SQL Server 2008 and later, use DATETIME2 (scale). For earlier versions, if the scale is 3 or less use DATETIME. In all other cases, use VARCHAR (37).
INT1	SMALLINT
INT2	SMALLINT
INT4	INT
INT8	BIGINT
NUMERIC	NUMBER (p,s)
REAL4	REAL
REAL8	FLOAT
STRING	If the column is a date or time column, then do the following: <ul style="list-style-type: none"> • For SQL Server 2008 and later, use DATETIME2. • For earlier versions, if the scale is 3 or less use DATETIME. In all other cases, use VARCHAR (37). If the column is not a date or time column, use VARCHAR (length).
UINT1	TINYINT
UINT2	SMALLINT
UINT4	INT
UINT8	BIGINT
WSTRING	NVARCHAR (length)
BLOB	VARBINARY(max) IMAGE To use this data type with AWS DMS, you must enable the use of BLOBs for a specific task. AWS DMS supports BLOB data types only in tables that include a primary key.
CLOB	VARCHAR(max) To use this data type with AWS DMS, you must enable the use of CLOBs for a specific task. During CDC, AWS DMS supports CLOB data types only in tables that include a primary key.

AWS DMS Data Type	SQL Server Data Type
NCLOB	NVARCHAR(max) To use this data type with AWS DMS, you must enable the use of NCLOBs for a specific task. During CDC, AWS DMS supports NCLOB data types only in tables that include a primary key.

Using a PostgreSQL Database as a Target for AWS Database Migration Service

You can migrate data to PostgreSQL databases using AWS DMS, either from another PostgreSQL database or from one of the other supported databases. PostgreSQL versions 9.4 and later are supported for on-premises, Amazon RDS, Amazon Aurora with PostgreSQL compatibility, and EC2 instance databases.

AWS DMS takes a table-by-table approach when migrating data from source to target in the Full Load phase. Table order during the full load phase cannot be guaranteed. Tables are out of sync during the full load phase and while cached transactions for individual tables are being applied. As a result, active referential integrity constraints can result in task failure during the full load phase.

In PostgreSQL, foreign keys (referential integrity constraints) are implemented using triggers. During the full load phase, AWS DMS loads each table one at a time. We strongly recommend that you disable foreign key constraints during a full load, using one of the following methods:

- Temporarily disable all triggers from the instance, and finish the full load.
- Use the `session_replication_role` parameter in PostgreSQL.

At any given time, a trigger can be in one of the following states: `origin`, `replica`, `always`, or `disabled`. When the `session_replication_role` parameter is set to `replica`, only triggers in the `replica` state are active, and they are fired when they are called. Otherwise, the triggers remain inactive.

PostgreSQL has a failsafe mechanism to prevent a table from being truncated, even when `session_replication_role` is set. You can use this as an alternative to disabling triggers, to help the full load run to completion. To do this, set the target table preparation mode to `DO_NOTHING`. Otherwise, `DROP` and `TRUNCATE` operations fail when there are foreign key constraints.

In Amazon RDS, you can control set this parameter using a parameter group. For a PostgreSQL instance running on Amazon EC2, you can set the parameter directly.

For additional details on working with a PostgreSQL database as a target for AWS DMS, see the following sections:

Topics

- [Limitations on Using PostgreSQL as a Target for AWS Database Migration Service \(p. 157\)](#)
- [Security Requirements When Using a PostgreSQL Database as a Target for AWS Database Migration Service \(p. 157\)](#)
- [Extra Connection Attributes When Using PostgreSQL as a Target for AWS DMS \(p. 157\)](#)
- [Target Data Types for PostgreSQL \(p. 157\)](#)

Limitations on Using PostgreSQL as a Target for AWS Database Migration Service

The following limitations apply when using a PostgreSQL database as a target for AWS DMS:

- The JSON data type is converted to the Native CLOB data type.
- In an Oracle to PostgreSQL migration, if a column in Oracle contains a NULL character (Hex value U+0000), AWS DMS converts the NULL character to a space (Hex value U+0020). This is due to a PostgreSQL limitation.

Security Requirements When Using a PostgreSQL Database as a Target for AWS Database Migration Service

For security purposes, the user account used for the data migration must be a registered user in any PostgreSQL database that you use as a target.

Extra Connection Attributes When Using PostgreSQL as a Target for AWS DMS

You can use extra connection attributes to configure your PostgreSQL target. You specify these settings when you create the target endpoint. Multiple extra connection attribute settings should be separated by a semicolon.

The following table shows the extra connection attributes you can use to configure PostgreSQL as a target for AWS DMS.

Name	Description
<code>maxFileSize</code>	Specifies the maximum size (in KB) of any CSV file used to transfer data to PostgreSQL. Default value: 32,768 KB (32 MB) Valid values: 1–1048576 Example: <code>maxFileSize=512</code>
<code>executeTimeout</code>	Sets the client statement timeout for the PostgreSQL instance, in seconds. The default value is 60 seconds. Example: <code>executeTimeout=100</code>
<code>afterConnectScript=SET session_replication_role='replica'</code>	Add this attribute to have AWS DMS bypass all foreign keys and user triggers. This action greatly reduces the time it takes to bulk load data when using full load mode. Example: <code>afterConnectScript=SET session_replication_role='replica'</code>

Target Data Types for PostgreSQL

The PostgreSQL database endpoint for AWS DMS supports most PostgreSQL database data types. The following table shows the PostgreSQL database target data types that are supported when using AWS

DMS and the default mapping from AWS DMS data types. Unsupported data types are listed following the table.

For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service \(p. 319\)](#).

AWS DMS Data Type	PostgreSQL Data Type
BOOL	BOOL
BYTES	BYTEA
DATE	DATE
TIME	TIME
TIMESTAMP	If the scale is from 0 through 6, then use TIMESTAMP. If the scale is from 7 through 9, then use VARCHAR (37).
INT1	SMALLINT
INT2	SMALLINT
INT4	INTEGER
INT8	BIGINT
NUMERIC	DECIMAL (P,S)
REAL4	FLOAT4
REAL8	FLOAT8
STRING	If the length is from 1 through 21,845, then use VARCHAR (length in bytes). If the length is 21,846 through 2,147,483,647, then use VARCHAR (65535).
UINT1	SMALLINT
UINT2	INTEGER
UINT4	BIGINT
UINT8	BIGINT
WSTRING	If the length is from 1 through 21,845, then use VARCHAR (length in bytes). If the length is 21,846 through 2,147,483,647, then use VARCHAR (65535).
BCLOB	BYTEA
NCLOB	TEXT
CLOB	TEXT

Note

When replicating from a PostgreSQL source, AWS DMS creates the target table with the same data types for all columns, apart from columns with user-defined data types. In such cases, the data type is created as "character varying" in the target.

Using a MySQL-Compatible Database as a Target for AWS Database Migration Service

You can migrate data to any MySQL-compatible database using AWS DMS, from any of the source data engines that AWS DMS supports. If you are migrating to an on-premises MySQL-compatible database, then AWS DMS requires that your source engine reside within the AWS ecosystem. The engine can be on an Amazon-managed service such as Amazon RDS, Amazon Aurora, or Amazon Simple Storage Service. Alternatively, the engine can be on a self-managed database on Amazon EC2.

You can use SSL to encrypt connections between your MySQL-compatible endpoint and the replication instance. For more information on using SSL with a MySQL-compatible endpoint, see [Using SSL With AWS Database Migration Service \(p. 47\)](#).

MySQL versions 5.5, 5.6, and 5.7 are supported, as are MariaDB and Aurora MySQL.

You can use the following MySQL-compatible databases as targets for AWS DMS:

- MySQL Community Edition
- MySQL Standard Edition
- MySQL Enterprise Edition
- MySQL Cluster Carrier Grade Edition
- MariaDB Community Edition
- MariaDB Enterprise Edition
- MariaDB Column Store
- Amazon Aurora MySQL

Note

Regardless of the source storage engine (MyISAM, MEMORY, and so on), AWS DMS creates a MySQL-compatible target table as an InnoDB table by default. If you need to have a table that uses a storage engine other than InnoDB, you can manually create the table on the MySQL-compatible target and migrate the table using the "do nothing" mode. For more information about the "do nothing" mode, see [Full Load Task Settings \(p. 228\)](#).

For additional details on working with a MySQL-compatible database as a target for AWS DMS, see the following sections.

Topics

- [Using Any MySQL-Compatible Database as a Target for AWS Database Migration Service \(p. 160\)](#)
- [Limitations on Using a MySQL-Compatible Database as a Target for AWS Database Migration Service \(p. 160\)](#)
- [Extra Connection Attributes When Using a MySQL-Compatible Database as a Target for AWS DMS \(p. 161\)](#)
- [Target Data Types for MySQL \(p. 162\)](#)

Using Any MySQL-Compatible Database as a Target for AWS Database Migration Service

Before you begin to work with a MySQL-compatible database as a target for AWS DMS, make sure that you have the following prerequisites:

- You must provide a user account to AWS DMS that has read/write privileges to the MySQL-compatible database. To create the necessary privileges, run the following commands.

```
CREATE USER '<user acct>'@'%' IDENTIFIED BY '<user password>';  
GRANT ALTER, CREATE, DROP, INDEX, INSERT, UPDATE, DELETE, SELECT ON <schema>.* TO  
'<user acct>'@'%;  
GRANT ALL PRIVILEGES ON awsdms_control.* TO '<user acct>'@'%';
```

- During the full-load migration phase, you must disable foreign keys on your target tables. To disable foreign key checks on a MySQL-compatible database during a full load, you can add the following command to the **Extra Connection Attributes** in the **Advanced** section of the target endpoint.

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

Limitations on Using a MySQL-Compatible Database as a Target for AWS Database Migration Service

When using a MySQL database as a target, AWS DMS doesn't support the following:

- The data definition language (DDL) statements TRUNCATE PARTITION, DROP TABLE, and RENAME TABLE.
- Using an ALTER TABLE *<table_name>* ADD COLUMN *<column_name>* statement to add columns to the beginning or the middle of a table.
- When only the LOB column in a source table is updated, AWS DMS doesn't update the corresponding target column. The target LOB is only updated if at least one other column is updated in the same transaction.
- When loading data to a MySQL-compatible target in a full load task, AWS DMS doesn't report duplicate key errors in the task log.
- When you update a column's value to its existing value, MySQL-compatible databases return a 0 rows affected warning. Although this behavior isn't technically an error, it is different from how the situation is handled by other database engines. For example, Oracle performs an update of one row. For MySQL-compatible databases, AWS DMS generates an entry in the awsdms_apply_exceptions control table and logs the following warning.

```
Some changes from the source database had no impact when applied to  
the target database. See awsdms_apply_exceptions table for details.
```

Extra Connection Attributes When Using a MySQL-Compatible Database as a Target for AWS DMS

You can use extra connection attributes to configure your MySQL-compatible target. You specify these settings when you create the target endpoint. Multiple extra connection attribute settings should be separated from each other by a semicolon.

The following table shows extra configuration settings that you can use when creating a MySQL-compatible target for AWS DMS.

Name	Description
<code>targetDbType</code>	<p>Specifies where to migrate source tables on the target, either to a single database or multiple databases.</p> <p>Default value: <code>MULTIPLE_DATABASES</code></p> <p>Valid values: <code>{SPECIFIC_DATABASE, MULTIPLE_DATABASES}</code></p> <p>Example: <code>targetDbType=MULTIPLE_DATABASES</code></p>
<code>parallelLoadThreads</code>	<p>Improves performance when loading data into the MySQL-compatible target database. Specifies how many threads to use to load the data into the MySQL-compatible target database. Setting a large number of threads can have an adverse effect on database performance, because a separate connection is required for each thread.</p> <p>Default value: 1</p> <p>Valid values: 1–5</p> <p>Example: <code>parallelLoadThreads=1</code></p>
<code>initstmt=SET FOREIGN_KEY_CHECKS=0</code>	<p>Disables foreign key checks.</p>
<code>initstmt=SET time-zone</code>	<p>Specifies the time zone for the target MySQL-compatible database.</p> <p>Default value: UTC</p> <p>Valid values: A three- or four-character abbreviation for the time zone that you want to use. Valid values are the standard time zone abbreviations for the operating system hosting the target MySQL-compatible database.</p> <p>Example: <code>initstmt=SET time_zone=UTC</code></p>
<code>afterConnectScript=SET character_set_connection='latin1'</code>	<p>Specifies that the MySQL-compatible target should translate received statements into the latin1 character set, which is the default compiled-in character set of the database. This parameter typically improves performance when converting from UTF8 clients.</p>
<code>maxFileSize</code>	<p>Specifies the maximum size (in KB) of any CSV file used to transfer data to a MySQL-compatible database.</p>

Name	Description
	<p>Default value: 32768 KB (32 MB)</p> <p>Valid values: 1 - 1048576</p> <p>Example: <code>maxFileSize=512</code></p>
<code>CleanSrcMetadataOnMismatch</code>	<p>Cleans and recreates table metadata information on the replication instance when a mismatch occurs. For example, in a situation where running an alter DDL on the table could result in different information about the table cached in the replication instance. Boolean.</p> <p>Default value: <code>false</code></p> <p>Example: <code>CleanSrcMetadataOnMismatch=false</code></p>

Target Data Types for MySQL

The following table shows the MySQL database target data types that are supported when using AWS DMS and the default mapping from AWS DMS data types.

For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service \(p. 319\)](#).

AWS DMS Data Types	MySQL Data Types
BOOLEAN	BOOLEAN
BYTES	<p>If the length is from 1 through 65,535, then use <code>VARBINARY (length)</code>.</p> <p>If the length is from 65,536 through 2,147,483,647, then use <code>LONGLOB</code>.</p>
DATE	DATE
TIME	TIME
TIMESTAMP	<p>"If scale is \Rightarrow 0 and \leq 6, then: <code>DATETIME (Scale)</code></p> <p>If scale is \Rightarrow 7 and \leq 9, then: <code>VARCHAR (37)</code>"</p>
INT1	TINYINT
INT2	SMALLINT
INT4	INTEGER
INT8	BIGINT
NUMERIC	DECIMAL (p,s)
REAL4	FLOAT
REAL8	DOUBLE PRECISION
STRING	If the length is from 1 through 21,845, then use <code>VARCHAR (length)</code> .

AWS DMS Data Types	MySQL Data Types
	If the length is from 21,846 through 2,147,483,647, then use LONGTEXT.
UINT1	UNSIGNED TINYINT
UINT2	UNSIGNED SMALLINT
UINT4	UNSIGNED INTEGER
UINT8	UNSIGNED BIGINT
WSTRING	If the length is from 1 through 32,767, then use VARCHAR (length). If the length is from 32,768 through 2,147,483,647, then use LONGTEXT.
BLOB	If the length is from 1 through 65,535, then use BLOB. If the length is from 65,536 through 2,147,483,647, then use LONGBLOB. If the length is 0, then use LONGBLOB (full LOB support).
NCLOB	If the length is from 1 through 65,535, then use TEXT. If the length is from 65,536 through 2,147,483,647, then use LONGTEXT with ucs2 for CHARACTER SET. If the length is 0, then use LONGTEXT (full LOB support) with ucs2 for CHARACTER SET.
CLOB	If the length is from 1 through 65,535, then use TEXT. If the length is from 65,536 through 2147483647, then use LONGTEXT. If the length is 0, then use LONGTEXT (full LOB support).

Using an Amazon Redshift Database as a Target for AWS Database Migration Service

You can migrate data to Amazon Redshift databases using AWS Database Migration Service. Amazon Redshift is a fully managed, petabyte-scale data warehouse service in the cloud. With an Amazon Redshift database as a target, you can migrate data from all of the other supported source databases.

The Amazon Redshift cluster must be in the same AWS account and same AWS Region as the replication instance.

During a database migration to Amazon Redshift, AWS DMS first moves data to an Amazon S3 bucket. Once the files reside in an Amazon S3 bucket, AWS DMS then transfers them to the proper tables in the Amazon Redshift data warehouse. AWS DMS creates the S3 bucket in the same AWS Region as the Amazon Redshift database. The AWS DMS replication instance must be located in that same region.

If you use the AWS Command Line Interface (AWS CLI) or the AWS DMS API to migrate data to Amazon Redshift, you must set up an AWS Identity and Access Management (IAM) role to allow S3 access. For more information about creating this IAM role, see [Creating the IAM Roles to Use With the AWS CLI and AWS DMS API \(p. 34\)](#).

The Amazon Redshift endpoint provides full automation for the following:

- Schema generation and data type mapping
- Full load of source database tables
- Incremental load of changes made to source tables
- Application of schema changes in data definition language (DDL) made to the source tables
- Synchronization between full load and change data capture (CDC) processes.

AWS Database Migration Service supports both full load and change processing operations. AWS DMS reads the data from the source database and creates a series of comma-separated value (CSV) files. For full-load operations, AWS DMS creates files for each table. AWS DMS then copies the table files for each table to a separate folder in Amazon Simple Storage Service. When the files are uploaded to Amazon Simple Storage Service, AWS DMS sends a copy command and the data in the files are copied into Amazon Redshift. For change-processing operations, AWS DMS copies the net changes to the CSV files. AWS DMS then uploads the net change files to Amazon Simple Storage Service and copies the data to Amazon Redshift.

For additional details on working with Amazon Redshift as a target for AWS DMS, see the following sections:

Topics

- [Prerequisites for Using an Amazon Redshift Database as a Target for AWS Database Migration Service \(p. 164\)](#)
- [Limitations on Using Amazon Redshift as a Target for AWS Database Migration Service \(p. 165\)](#)
- [Configuring an Amazon Redshift Database as a Target for AWS Database Migration Service \(p. 165\)](#)
- [Using Enhanced VPC Routing with an Amazon Redshift as a Target for AWS Database Migration Service \(p. 166\)](#)
- [Extra Connection Attributes When Using Amazon Redshift as a Target for AWS DMS \(p. 166\)](#)
- [Target Data Types for Amazon Redshift \(p. 168\)](#)

Prerequisites for Using an Amazon Redshift Database as a Target for AWS Database Migration Service

The following list describes the prerequisites necessary for working with Amazon Redshift as a target for data migration:

- Use the AWS Management Console to launch an Amazon Redshift cluster. You should note the basic information about your AWS account and your Amazon Redshift cluster, such as your password, user name, and database name. You need these values when creating the Amazon Redshift target endpoint.
- The Amazon Redshift cluster must be in the same AWS account and the same AWS Region as the replication instance.

- The AWS DMS replication instance needs network connectivity to the Amazon Redshift endpoint (hostname and port) that your cluster uses.
- AWS DMS uses an Amazon Simple Storage Service bucket to transfer data to the Amazon Redshift database. For AWS DMS to create the bucket, the DMS console uses an Amazon IAM role, `dms-access-for-endpoint`. If you use the AWS CLI or DMS API to create a database migration with Amazon Redshift as the target database, you must create this IAM role. For more information about creating this role, see [Creating the IAM Roles to Use With the AWS CLI and AWS DMS API \(p. 34\)](#).
- AWS DMS converts BLOBs, CLOBs, and NCLOBs to a VARCHAR on the target Amazon Redshift instance. Amazon Redshift doesn't support VARCHAR data types larger than 64 KB, so you can't store traditional LOBs on Amazon Redshift.

Limitations on Using Amazon Redshift as a Target for AWS Database Migration Service

When using an Amazon Redshift database as a target, AWS DMS doesn't support the following:

- The following DDL is not supported:

```
ALTER TABLE <table name> MODIFY COLUMN <column name> <data type>;
```

- AWS DMS cannot migrate or replicate changes to a schema with a name that begins with underscore (`_`). If you have schemas that have a name that begins with an underscore, use mapping transformations to rename the schema on the target.
- Amazon Redshift doesn't support VARCHARs larger than 64 KB. LOBs from traditional databases can't be stored in Amazon Redshift.

Configuring an Amazon Redshift Database as a Target for AWS Database Migration Service

AWS Database Migration Service must be configured to work with the Amazon Redshift instance. The following table describes the configuration properties available for the Amazon Redshift endpoint.

Property	Description
server	The name of the Amazon Redshift cluster you are using.
port	The port number for Amazon Redshift. The default value is 5439.
username	An Amazon Redshift user name for a registered user.
password	The password for the user named in the username property.
database	The name of the Amazon Redshift data warehouse (service) you are working with.

If you want to add extra connection string attributes to your Amazon Redshift endpoint, you can specify the `maxFileSize` and `fileTransferUploadStreams` attributes. For more information on these attributes, see [Extra Connection Attributes When Using Amazon Redshift as a Target for AWS DMS \(p. 166\)](#).

Using Enhanced VPC Routing with an Amazon Redshift as a Target for AWS Database Migration Service

If you're using the *Enhanced VPC Routing* feature with your Amazon Redshift target, the feature forces all COPY traffic between your Amazon Redshift cluster and your data repositories through your Amazon VPC. Because *Enhanced VPC Routing* affects the way that Amazon Redshift accesses other resources, COPY commands might fail if you haven't configured your VPC correctly.

AWS DMS can be affected by this behavior because it uses the COPY command to move data in S3 to an Amazon Redshift cluster.

Following are the steps AWS DMS takes to load data into an Amazon Redshift target:

1. AWS DMS copies data from the source to CSV files on the replication server.
2. AWS DMS uses the AWS SDK to copy the CSV files into an S3 bucket on your account.
3. AWS DMS then uses the COPY command in Amazon Redshift to copy data from the CSV files in S3 to an appropriate table in Amazon Redshift.

If *Enhanced VPC Routing* is not enabled, Amazon Redshift routes traffic through the Internet, including traffic to other services within the AWS network. If the feature is not enabled, you do not have to configure the network path. If the feature is enabled, you must specifically create a network path between your cluster's VPC and your data resources. For more information on the configuration required, see [Enhanced VPC Routing](#) in the Amazon Redshift documentation.

Extra Connection Attributes When Using Amazon Redshift as a Target for AWS DMS

You can use extra connection attributes to configure your Amazon Redshift target. You specify these settings when you create the source endpoint. Multiple extra connection attribute settings should be separated by a semicolon.

The following table shows the extra connection attributes available when Amazon Redshift is the target.

Name	Description
<code>maxFileSize</code>	Specifies the maximum size (in KB) of any CSV file used to transfer data to Amazon Redshift. Default value: 32768 KB (32 MB) Valid values: 1 - 1048576 Example: <code>maxFileSize=512</code>
<code>fileTransferUploadStreams</code>	Specifies the number of threads used to upload a single file. Default value: 10 Valid values: 1 - 64 Example: <code>fileTransferUploadStreams=20</code>
<code>acceptanydate</code>	Specifies if any date format is accepted, including invalid dates formats such as 0000-00-00. Boolean value. Default value: false

Name	Description
	<p>Valid values: true false</p> <p>Example: <code>acceptanydate=true</code></p>
<p><code>dateformat</code></p>	<p>Specifies the date format. This is a string input and is empty by default. The default format is YYYY-MM-DD but you can change it to, for example, DD-MM-YYYY. If your date or time values use different formats, use the <code>auto</code> argument with the <code>dateformat</code> parameter. The <code>auto</code> argument recognizes several formats that are not supported when using a <code>dateformat</code> string. The <code>auto</code> keyword is case-sensitive.</p> <p>Default value: empty</p> <p>Valid values: 'dateformat_string' or auto</p> <p>Example: <code>dateformat=auto</code></p>
<p><code>timeformat</code></p>	<p>Specifies the time format. This is a string input and is empty by default. The <code>auto</code> argument recognizes several formats that are not supported when using a <code>timeformat</code> string. If your date and time values use formats different from each other, use the <code>auto</code> argument with the <code>timeformat</code> parameter.</p> <p>Default value: 10</p> <p>Valid values: 'timeformat_string' 'auto' 'epochsecs' 'epochmillisecs'</p> <p>Example: <code>timeformat=auto</code></p>
<p><code>emptyasnull</code></p>	<p>Specifies whether AWS DMS should migrate empty CHAR and VARCHAR fields as null. A value of true sets empty CHAR and VARCHAR fields as null.</p> <p>Default value: false</p> <p>Valid values: true false</p> <p>Example: <code>emptyasnull=true</code></p>
<p><code>truncateColumns</code></p>	<p>Truncates data in columns to the appropriate number of characters so that it fits the column specification. Applies only to columns with a VARCHAR or CHAR data type, and rows 4 MB or less in size.</p> <p>Default value: false</p> <p>Valid values: true false</p> <p>Example:</p> <pre>truncateColumns=true;</pre>

Name	Description
<code>removeQuotes</code>	<p>Removes surrounding quotation marks from strings in the incoming data. All characters within the quotation marks, including delimiters, are retained. For more information about removing quotes for an Amazon Redshift target, see the Redshift documentation.</p> <p>Default value: false</p> <p>Valid values: true false</p> <p>Example:</p> <pre>removeQuotes=true;</pre>
<code>trimBlanks</code>	<p>Removes the trailing white space characters from a VARCHAR string. This parameter applies only to columns with a VARCHAR data type.</p> <p>Default value: false</p> <p>Valid values: true false</p> <p>Example:</p> <pre>trimBlanks=false;</pre>

Target Data Types for Amazon Redshift

The Amazon Redshift endpoint for AWS DMS supports most Amazon Redshift data types. The following table shows the Amazon Redshift target data types that are supported when using AWS DMS and the default mapping from AWS DMS data types.

For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service \(p. 319\)](#).

AWS DMS Data Types	Amazon Redshift Data Types
BOOLEAN	BOOL
BYTES	VARCHAR (Length)
DATE	DATE
TIME	VARCHAR(20)
DATETIME	<p>If the scale is => 0 and =< 6, then:</p> <p>TIMESTAMP (s)</p> <p>If the scale is => 7 and =< 9, then:</p> <p>VARCHAR (37)</p>
INT1	INT2
INT2	INT2

AWS DMS Data Types	Amazon Redshift Data Types
INT4	INT4
INT8	INT8
NUMERIC	<p>If the scale is => 0 and =< 37, then: NUMERIC (p,s)</p> <p>If the scale is => 38 and =< 127, then: VARCHAR (Length)</p>
REAL4	FLOAT4
REAL8	FLOAT8
STRING	<p>If the length is 1–65,535, then use VARCHAR (length in bytes)</p> <p>If the length is 65,536–2,147,483,647, then use VARCHAR (65535)</p>
UINT1	INT2
UINT2	INT2
UINT4	INT4
UINT8	NUMERIC (20,0)
WSTRING	<p>If the length is 1–65,535, then use NVARCHAR (length in bytes)</p> <p>If the length is 65,536–2,147,483,647, then use NVARCHAR (65535)</p>
BLOB	<p>VARCHAR (maximum LOB size *2)</p> <p>The maximum LOB size cannot exceed 31 KB. Amazon Redshift doesn't support VARCHARs larger than 64 KB.</p>
NCLOB	<p>NVARCHAR (maximum LOB size)</p> <p>The maximum LOB size cannot exceed 63 KB. Amazon Redshift doesn't support VARCHARs larger than 64 KB.</p>
CLOB	<p>VARCHAR (maximum LOB size)</p> <p>The maximum LOB size cannot exceed 63 KB. Amazon Redshift doesn't support VARCHARs larger than 64 KB.</p>

Using a SAP ASE Database as a Target for AWS Database Migration Service

You can migrate data to SAP Adaptive Server Enterprise (ASE)—formerly known as Sybase—databases using AWS DMS, either from any of the supported database sources.

SAP ASE versions 15, 15.5, 15.7, 16 and later are supported.

Prerequisites for Using a SAP ASE Database as a Target for AWS Database Migration Service

Before you begin to work with a SAP ASE database as a target for AWS DMS, make sure that you have the following prerequisites:

- You must provide SAP ASE account access to the AWS DMS user. This user must have read/write privileges in the SAP ASE database.
- When replicating to SAP ASE version 15.7 installed on a Windows EC2 instance configured with a non-Latin language (for example, Chinese), AWS DMS requires SAP ASE 15.7 SP121 to be installed on the target SAP ASE machine.

Extra Connection Attributes When Using SAP ASE as a Target for AWS DMS

You can use extra connection attributes to configure your SAP ASE target. You specify these settings when you create the target endpoint. Multiple extra connection attribute settings should be separated by a semicolon.

The following table shows the extra connection attributes available when using SAP ASE as a target:

Name	Description
<code>enableReplication</code>	Set to <code>Y</code> to automatically enable SAP ASE replication. This is only required if SAP ASE replication has not been enabled already.
<code>additionalConnectionProperties</code>	Any additional ODBC connection parameters that you want to specify.

Note

If the user name or password specified in the connection string contains non-Latin characters (for example, Chinese), the following property is required: `charset=gb18030`

Target Data Types for SAP ASE

The following table shows the SAP ASE database target data types that are supported when using AWS DMS and the default mapping from AWS DMS data types.

For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service \(p. 319\)](#).

AWS DMS Data Types	SAP ASE Data Types
BOOLEAN	BIT
BYTES	VARBINARY (Length)
DATE	DATE
TIME	TIME
TIMESTAMP	If scale is => 0 and =< 6, then: BIGDATETIME If scale is => 7 and =< 9, then: VARCHAR (37)
INT1	TINYINT
INT2	SMALLINT
INT4	INTEGER
INT8	BIGINT
NUMERIC	NUMERIC (p,s)
REAL4	REAL
REAL8	DOUBLE PRECISION
STRING	VARCHAR (Length)
UINT1	TINYINT
UINT2	UNSIGNED SMALLINT
UINT4	UNSIGNED INTEGER
UINT8	UNSIGNED BIGINT
WSTRING	VARCHAR (Length)
BLOB	IMAGE
CLOB	UNITEXT
NCLOB	TEXT

AWS DMS does not support tables that include fields with the following data types. Replicated columns with these data types show as null.

- User-defined type (UDT)

Using Amazon Simple Storage Service as a Target for AWS Database Migration Service

You can migrate data to Amazon Simple Storage Service using AWS DMS from any of the supported database sources. When using Amazon S3 as a target in an AWS DMS task, both full load and change data capture (CDC) data is written to comma-separated-values (CSV) format. AWS DMS names files created during a full load using an incremental hexadecimal counter—for example LOAD00001.csv,

LOAD00002..., LOAD00009, LOAD0000A, and so on. AWS DMS names CDC files using timestamps, for example 20141029-1134010000.csv. For each source table, AWS DMS creates a folder under the specified target folder. AWS DMS writes all full load and CDC files to the specified Amazon S3 bucket.

The parameter `bucketFolder` contains the location where the .csv files are stored before being uploaded to the S3 bucket. Table data is stored in the following format in the S3 bucket:

```
<schema_name>/<table_name>/LOAD00000001.csv
<schema_name>/<table_name>/LOAD00000002.csv
...
<schema_name>/<table_name>/LOAD00000009.csv
<schema_name>/<table_name>/LOAD0000000A.csv
<schema_name>/<table_name>/LOAD0000000B.csv
...
<schema_name>/<table_name>/LOAD0000000F.csv
<schema_name>/<table_name>/LOAD00000010.csv
...
```

You can specify the column delimiter, row delimiter, and other parameters using the extra connection attributes. For more information on the extra connection attributes, see [Extra Connection Attributes When Using Amazon S3 as a Target for AWS DMS \(p. 174\)](#) at the end of this section.

When you use AWS DMS to replicate data changes, the first column of the CSV output file indicates how the data was changed as shown following:

```
I,101,Smith,Bob,4-Jun-14,New York
U,101,Smith,Bob,8-Oct-15,Los Angeles
U,101,Smith,Bob,13-Mar-17,Dallas
D,101,Smith,Bob,13-Mar-17,Dallas
```

For this example, suppose that there is an `EMPLOYEE` table in the source database. AWS DMS writes data to the CSV file, in response to the following events:

- A new employee (Bob Smith, employee ID 101) is hired on 4-Jun-14 at the New York office. In the CSV file, the `I` in the first column indicates that a new row was `INSERTED` into the `EMPLOYEE` table at the source database.
- On 8-Oct-15, Bob transfers to the Los Angeles office. In the CSV file, the `U` indicates that the corresponding row in the `EMPLOYEE` table was `UPDATED` to reflect Bob's new office location. The rest of the line reflects the row in the `EMPLOYEE` table as it appears after the `UPDATE`.
- On 13-Mar-17, Bob transfers again to the Dallas office. In the CSV file, the `U` indicates that this row was `UPDATED` again. The rest of the line reflects the row in the `EMPLOYEE` table as it appears after the `UPDATE`.
- After some time working in Dallas, Bob leaves the company. In the CSV file, the `D` indicates that the row was `DELETED` in the source table. The rest of the line reflects how the row in the `EMPLOYEE` table appeared before it was deleted.

Prerequisites for Using Amazon Simple Storage Service as a Target

The Amazon Simple Storage Service bucket you are using as a target must be in the same region as the DMS replication instance you are using to migrate your data.

The AWS account you use for the migration must have write and delete access to the Amazon Simple Storage Service bucket you are using as a target. The role assigned to the user account used to create the migration task must have the following set of permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::buckettest2*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::buckettest2*"
      ]
    }
  ]
}
```

Limitations to Using Amazon Simple Storage Service as a Target

The following limitations apply to a file in Amazon Simple Storage Service that you are using as a target:

- Only the following data definition language (DDL) commands are supported: TRUNCATE TABLE, DROP TABLE, and CREATE TABLE.
- Full LOB mode is not supported.
- Changes to the source table structure during full load are not supported. Changes to the data are supported during full load.
- Multiple tasks that replicate data from the same source table to the same target S3 endpoint bucket result in those tasks writing to the same file. We recommend that you specify different target endpoints (buckets) if your data source is from the same table.

Security

To use Amazon Simple Storage Service as a target, the account used for the migration must have write and delete access to the Amazon Simple Storage Service bucket that is used as the target. You must specify the Amazon Resource Name (ARN) of an IAM role that has the permissions required to access Amazon Simple Storage Service.

AWS DMS supports a set of predefined grants for Amazon Simple Storage Service, known as canned ACLs. Each canned ACL has a set of grantees and permissions you can use to set permissions for the Amazon Simple Storage Service bucket. You can specify a canned ACL using the `cannedAclForObjects` on the connection string attribute for your S3 target endpoint. For more information about using the extra connection attribute `cannedAclForObjects`, see [Extra Connection Attributes When Using Amazon S3 as a Target for AWS DMS \(p. 174\)](#) for more information. For more information about Amazon Simple Storage Service canned ACLs, see [Canned ACL](#).

The IAM role that you use for the migration must be able to perform the `s3:PutObjectAcl` API action.

Extra Connection Attributes When Using Amazon S3 as a Target for AWS DMS

You can specify the following options as extra connection attributes. Multiple extra connection attribute settings should be separated by a semicolon.

Option	Description
<code>addColumnNames</code>	<p>An optional parameter that allows you to add column name information to the .csv output file. The default is false.</p> <p>Example:</p> <pre>addColumnNames=true;</pre>
<code>bucketFolder</code>	<p>An optional parameter to set a folder name in the S3 bucket. If provided, tables are created in the path <code><bucketFolder>/<schema_name>/<table_name>/</code>. If this parameter is not specified, then the path used is <code><schema_name>/<table_name>/</code>.</p> <p>Example:</p> <pre>bucketFolder=testFolder;</pre>
<code>bucketName</code>	<p>The name of the S3 bucket.</p> <p>Example:</p> <pre>bucketName=buckettest;</pre>
<code>cannedAclForObjects</code>	<p>Allows AWS DMS to specify a predefined (canned) access control list for objects written to the S3 bucket. For more information about Amazon Simple Storage Service canned ACLs, see Canned ACL in the Amazon Simple Storage Service Developer Guide.</p> <p>Example:</p> <pre>cannedAclForObjects=PUBLIC_READ;</pre> <p>Valid values for this attribute are: NONE; PRIVATE; PUBLIC_READ; PUBLIC_READ_WRITE; AUTHENTICATED_READ; AWS_EXEC_READ; BUCKET_OWNER_READ; BUCKET_OWNER_FULL_CONTROL.</p> <p>If this attribute isn't specified, it defaults to NONE.</p>
<code>cdcInsertsOnly</code>	<p>An optional parameter to write only INSERT operations to the .CSV output files. By default, the first field in a .CSV record contains the letter I (insert), U (update) or D (delete) to indicate whether the row was inserted, updated, or deleted at the source database. If <code>cdcInsertsOnly</code> is set to <code>true</code>, then only INSERTs are recorded in the CSV file, without any I annotation.</p> <p>Example:</p> <pre>cdcInsertsOnly=true;</pre>
<code>compressionType</code>	<p>An optional parameter to use GZIP to compress the target files. Set to NONE (the default) or do not use to leave the files uncompressed.</p> <p>Example:</p>

Option	Description
	<code>compressionType=GZIP;</code>
<code>csvRowDelimiter</code>	<p>The delimiter used to separate rows in the source files. The default is a carriage return (<code>\n</code>).</p> <p>Example:</p> <pre>csvRowDelimiter=\n;</pre>
<code>csvDelimiter</code>	<p>The delimiter used to separate columns in the source files. The default is a comma.</p> <p>Example:</p> <pre>csvDelimiter=,;</pre>
<code>maxFileSize</code>	<p>Specifies the maximum size (in KB) of any CSV file to be created while migrating to S3 target during full load.</p> <p>Default value: 1048576 KB (1 GB)</p> <p>Valid values: 1 - 1048576</p> <p>Example:</p> <pre>maxFileSize=512</pre>
<code>rfc4180</code>	<p>An optional parameter used to control RFC compliance behavior with data migrated to Amazon Simple Storage Service. When using Amazon Simple Storage Service as a target, if the data has quotes or a new line character in it then AWS DMS encloses the entire column with an additional ". Every quote mark within the data is repeated twice. This is in compliance with RFC 4180.</p> <p>The default is <code>true</code>.</p> <p>Example:</p> <pre>rfc4180=false;</pre>

Using an Amazon DynamoDB Database as a Target for AWS Database Migration Service

You can use AWS DMS to migrate data to an Amazon DynamoDB table. Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. AWS DMS supports using a relational database or MongoDB as a source.

In DynamoDB, tables, items, and attributes are the core components that you work with. A table is a collection of items, and each item is a collection of attributes. DynamoDB uses primary keys, called partition keys, to uniquely identify each item in a table. You can also use keys and secondary indexes to provide more querying flexibility.

You use object mapping to migrate your data from a source database to a target DynamoDB table. Object mapping lets you determine where the source data is located in the target.

When AWS DMS creates tables on an Amazon DynamoDB target endpoint, it creates as many tables as in the source database endpoint. AWS DMS also sets several Amazon DynamoDB parameter values. The cost for the table creation depends on the amount of data and the number of tables to be migrated.

When AWS DMS sets Amazon DynamoDB parameter values for a migration task, the default Read Capacity Units (RCU) parameter value is set to 200.

The Write Capacity Units (WCU) parameter value is also set, but its value depends on several other settings:

- The default value for the WCU parameter is 200.
- If the `parallelLoadThreads` parameter is set greater than 1 (default is 0), then the WCU parameter is set to 200 times the `parallelLoadThreads` value.
- In the US East (N. Virginia) Region (`us-east-1`), the largest possible WCU parameter value is 40000. If the AWS Region is `us-east-1` and the WCU parameter value is greater than 40000, the WCU parameter value is set to 40000.
- In AWS Regions other than `us-east-1`, the largest possible WCU parameter value is 10000. For any AWS Region other than `us-east-1`, if the WCU parameter value is set greater than 10000 the WCU parameter value is set to 10000.

Migrating from a Relational Database to a DynamoDB Table

AWS DMS supports migrating data to DynamoDB's scalar data types. When migrating from a relational database like Oracle or MySQL to DynamoDB, you might want to restructure how you store this data.

Currently AWS DMS supports single table to single table restructuring to DynamoDB scalar type attributes. If you are migrating data into DynamoDB from a relational database table, you take data from a table and reformat it into DynamoDB scalar data type attributes. These attributes can accept data from multiple columns, and you can map a column to an attribute directly.

AWS DMS supports the following DynamoDB scalar data types:

- String
- Number
- Boolean

Note

NULL data from the source are ignored on the target.

Prerequisites for Using a DynamoDB as a Target for AWS Database Migration Service

Before you begin to work with a DynamoDB database as a target for AWS DMS, make sure that you create an IAM role that allows AWS DMS to assume and grants access to the DynamoDB tables that are being migrated into. The minimum set of access permissions is shown in the following sample role policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "dms.amazonaws.com"
      },
    },
  ],
}
```

```
    "Action": "sts:AssumeRole"  
  }  
]  
}
```

The role that you use for the migration to DynamoDB must have the following permissions:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "dynamodb:PutItem",  
        "dynamodb:CreateTable",  
        "dynamodb:DescribeTable",  
        "dynamodb>DeleteTable",  
        "dynamodb>DeleteItem"  
      ],  
      "Resource": [  
        "arn:aws:dynamodb:us-west-2:account-id:table/Name1",  
        "arn:aws:dynamodb:us-west-2:account-id:table/OtherName*"  
      ]  
    },  
    {  
      "Effect": "Allow",  
      "Action": [  
        "dynamodb:ListTables"  
      ],  
      "Resource": "*"   
    }  
  ]  
}
```

Limitations When Using DynamoDB as a Target for AWS Database Migration Service

The following limitations apply when using Amazon DynamoDB as a target:

- DynamoDB limits the precision of the Number data type to 38 places. Store all data types with a higher precision as a String. You need to explicitly specify this using the object mapping feature.
- Because Amazon DynamoDB doesn't have a Date data type, data using the Date data type are converted to strings.
- Amazon DynamoDB doesn't allow updates to the primary key attributes. This restriction is important when using ongoing replication with change data capture (CDC) because it can result in unwanted data in the target. Depending on how you have the object mapping, a CDC operation that updates the primary key can either fail or insert a new item with the updated primary key and incomplete data.
- AWS DMS only supports replication of tables with non-composite primary keys, unless you specify an object mapping for the target table with a custom partition key or sort key, or both.
- AWS DMS doesn't support LOB data unless it is a CLOB. AWS DMS converts CLOB data into a DynamoDB string when migrating data.

Using Object Mapping to Migrate Data to DynamoDB

AWS DMS uses table-mapping rules to map data from the source to the target DynamoDB table. To map data to a DynamoDB target, you use a type of table-mapping rule called *object-mapping*. Object

mapping lets you define the attribute names and the data to be migrated to them. You must have selection rules when you use object mapping,

Amazon DynamoDB doesn't have a preset structure other than having a partition key and an optional sort key. If you have a noncomposite primary key, AWS DMS uses it. If you have a composite primary key or you want to use a sort key, define these keys and the other attributes in your target DynamoDB table.

To create an object mapping rule, you specify the `rule-type` as *object-mapping*. This rule specifies what type of object mapping you want to use.

The structure for the rule is as follows:

```
{ "rules": [
  {
    "rule-type": "object-mapping",
    "rule-id": "<id>",
    "rule-name": "<name>",
    "rule-action": "<valid object-mapping rule action>",
    "object-locator": {
      "schema-name": "<case-sensitive schema name>",
      "table-name": ""
    },
    "target-table-name": "<table_name>",
  }
]
```

AWS DMS currently supports *map-record-to-record* and *map-record-to-document* as the only valid values for the `rule-action` parameter. *map-record-to-record* and *map-record-to-document* specify what AWS DMS does by default to records that aren't excluded as part of the `exclude-columns` attribute list; these values don't affect the attribute-mappings in any way.

- *map-record-to-record* can be used when migrating from a relational database to DynamoDB. It uses the primary key from the relational database as the partition key in Amazon DynamoDB and creates an attribute for each column in the source database. When using *map-record-to-record*, for any column in the source table not listed in the `exclude-columns` attribute list, AWS DMS creates a corresponding attribute on the target DynamoDB instance regardless of whether that source column is used in an attribute mapping.
- *map-record-to-document* puts source columns into a single, flat, DynamoDB map on the target using the attribute name `"_doc."` When using *map-record-to-document*, for any column in the source table not listed in the `exclude-columns` attribute list, AWS DMS places the data into a single, flat, DynamoDB map attribute on the source called `"_doc"`.

One way to understand the difference between the `rule-action` parameters *map-record-to-record* and *map-record-to-document* is to see the two parameters in action. For this example, assume that you are starting with a relational database table row with the following structure and data:

FirstName	LastName	NickName	WorkAddress	WorkPhone	HomeAddress	HomePhone
▶ Daniel	Sheridan	Dan	101 Main St Cambridge, MA	800-867-5309	100 Secret St, Unknownville, MA	123-456-7890

To migrate this information to DynamoDB, you create rules to map the data into a DynamoDB table item. Note the columns listed for the `exclude-columns` parameter. These columns are not directly mapped over to the target. Instead, attribute mapping is used to combine the data into new items, such as where *FirstName* and *LastName* are grouped together to become *CustomerName* on the DynamoDB target. *NickName* and *income* are not excluded.


```

{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "test",
        "table-name": "%"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "object-mapping",
      "rule-id": "1",
      "rule-name": "TransformToDDB",
      "rule-action": "map-record-to-record",
      "object-locator": {
        "schema-name": "test",
        "table-name": "customer",
      },
      "target-table-name": "customer_t",
      "mapping-parameters": {
        "partition-key-name": "CustomerName",
        "exclude-columns": [
          "FirstName",
          "LastName",
          "HomeAddress",
          "HomePhone",
          "WorkAddress",
          "WorkPhone"
        ],
        "attribute-mappings": [
          {
            "target-attribute-name": "CustomerName",
            "attribute-type": "scalar",
            "attribute-sub-type": "string",
            "value": "${FirstName},${LastName}"
          },
          {
            "target-attribute-name": "ContactDetails",
            "attribute-type": "document",
            "attribute-sub-type": "dynamodb-map",
            "value": {
              "M": {
                "Home": {
                  "M": {
                    "Address": {
                      "S": "${HomeAddress}"
                    },
                    "Phone": {
                      "S": "${HomePhone}"
                    }
                  }
                },
                "Work": {
                  "M": {
                    "Address": {
                      "S": "${WorkAddress}"
                    },
                    "Phone": {
                      "S": "${WorkPhone}"
                    }
                  }
                }
              }
            }
          }
        ]
      }
    }
  ]
}

```

```
}  
  ]  
  }  
  ]  
  }  
  ]  
  }  
}
```

By using the rule-action parameter *map-record-to-record*, the data for *NickName* and *income* are mapped to items of the same name in the DynamoDB target.

```
  ▾ Item {4}  
    + CustomerName String : Daniel,Sheridan  
    + ▾ ContactDetails Map {2}  
      + ▾ Home Map {2}  
        + Address String : 100 Secret St, Unknownville, MA  
        + Phone String : 123-456-7890  
      + ▾ Work Map {2}  
        + Address String : 101 Main St Cambridge, MA  
        + Phone String : 800-867-5309  
    + NickName String : Dan  
    + income Number : 12345678
```

However, suppose that you use the same rules but change the rule-action parameter to *map-record-to-document*. In this case, the columns not listed in the *exclude-columns* parameter, *NickName* and *income*, are mapped to a *_doc* item.

```
  ▾ Item {3}  
    + CustomerName String : Daniel,Sheridan  
    + ▾ ContactDetails Map {2}  
      + ▾ Home Map {2}  
        + Address String : 100 Secret St, Unknownville, MA  
        + Phone String : 123-456-7890  
      + ▾ Work Map {2}  
        + Address String : 101 Main St Cambridge, MA  
        + Phone String : 800-867-5309  
    + ▾ _doc Map {2}  
      + NickName String : Dan  
      + income Number : 12345678
```

Using Custom Condition Expressions with Object Mapping

You can use a feature of Amazon DynamoDB called conditional expressions to manipulate data that is being written to a DynamoDB table. For more information about condition expressions in DynamoDB, see [Condition Expressions](#).

A condition expression member consists of:

- an expression (required)
- expression attribute values (optional) . Specifies a DynamoDB json structure of the attribute value
- expression attribute names (optional)
- options for when to use the condition expression (optional). The default is apply-during-cdc = false and apply-during-full-load = true

The structure for the rule is as follows:

```
"target-table-name": "customer_t",
  "mapping-parameters": {
    "partition-key-name": "CustomerName",
    "condition-expression": {
      "expression": "<conditional expression>",
      "expression-attribute-values": [
        {
          "name": "<attribute name>",
          "value": <attribute value>
        }
      ],
      "apply-during-cdc": <optional Boolean value>,
      "apply-during-full-load": <optional Boolean value>
    }
  }
```

The following sample highlights the sections used for condition expression.

```

{
  "rules": [
    {
      "rule-type": "object-mapping",
      "rule-id": "1",
      "rule-name": "TransformToDDB",
      "rule-action": "map-record-to-record",
      "object-locator": {
        "schema-name": "test",
        "table-name": "customer",
      },
      "target-table-name": "customer_t",
      "mapping-parameters": {
        "partition-key-name": "CustomerName",
        "condition-expression": {
          "expression": "attribute_not_exists(version) or version <= :record_version",
          "expression-attribute-values": [
            {
              "name": ":record_version",
              "value": {"N": "${version}"}
            }
          ],
          "apply-during-cdc": true,
          "apply-during-full-load": true
        },
        "attribute-mappings": [
          {
            "target-attribute-name": "CustomerName",
            "attribute-type": "scalar",
            "attribute-sub-type": "string",
            "value": "${FirstName},${LastName}"
          }
        ]
      }
    }
  ]
}

```

Using Attribute Mapping with Object Mapping

Attribute mapping lets you specify a template string using source column names to restructure data on the target. There is no formatting done other than what the user specifies in the template.

The following example shows the structure of the source database and the desired structure of the DynamoDB target. First is shown the structure of the source, in this case an Oracle database, and then the desired structure of the data in DynamoDB. The example concludes with the JSON used to create the desired target structure.

The structure of the Oracle data is as follows:

First	Last	Stc	HomeA	HomeP	WorkAddre	Work	DateOfBirth
Primary Key	N/A						
Randy	Mars	5	221B Baker Street	1234567890	7890 Spooner Street, Quahog	9876543210	03/21/1988

The structure of the DynamoDB data is as follows:

Custom	StoreId	ContactDetails	DateOfBirth
Partition Key	Sort Key	N/A	
Randy, Marsh		<pre>{ "Name": "Randy", "Home": { "Address": "221B Baker Street", "Phone": 1234567890 }, "Work": { "Address": "31 Spooner Street, Quahog", "Phone": 9876541230 } }</pre>	02/29/1988

The following JSON shows the object mapping and column mapping used to achieve the DynamoDB structure:

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "test",
        "table-name": "%"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "object-mapping",
      "rule-id": "2",
      "rule-name": "TransformToDDB",
      "rule-action": "map-record-to-record",
      "object-locator": {
        "schema-name": "test",
        "table-name": "customer"
      },
      "target-table-name": "customer_t",
      "mapping-parameters": {
        "partition-key-name": "CustomerName",
        "sort-key-name": "StoreId",
        "exclude-columns": [
          "FirstName",
          "LastName",
          "HomeAddress",
          "HomePhone",
          "WorkAddress",
          "WorkPhone"
        ],
        "attribute-mappings": [
          {
            "target-attribute-name": "CustomerName",
            "attribute-type": "scalar",
            "attribute-sub-type": "string",
            "value": "${FirstName},${LastName}"
          }
        ]
      }
    }
  ]
}
```

```

    },
    {
      "target-attribute-name": "StoreId",
      "attribute-type": "scalar",
      "attribute-sub-type": "string",
      "value": "${StoreId}"
    },
    {
      "target-attribute-name": "ContactDetails",
      "attribute-type": "scalar",
      "attribute-sub-type": "string",
      "value": "{\"Name\": \"${FirstName}\", \"Home\": {\"Address\": \"${HomeAddress}\",
\\Phone\": \"${HomePhone}\"}, \"Work\": {\"Address\": \"${WorkAddress}\", \"Phone\":
\\${WorkPhone}\"}}"
    }
  ]
}
]
}

```

Another way to use column mapping is to use DynamoDB format as your document type. The following code example uses *dynamodb-map* as the *attribute-sub-type* for attribute mapping.

```

{
  "rules": [
    {
      "rule-type": "object-mapping",
      "rule-id": "1",
      "rule-name": "TransformToDDB",
      "rule-action": "map-record-to-record",
      "object-locator": {
        "schema-name": "test",
        "table-name": "customer",
      },
      "target-table-name": "customer_t",
      "mapping-parameters": {
        "partition-key-name": "CustomerName",
        "sort-key-name": "StoreId",
        "exclude-columns": [
          "FirstName",
          "LastName",
          "HomeAddress",
          "HomePhone",
          "WorkAddress",
          "WorkPhone"
        ],
      },
      "attribute-mappings": [
        {
          "target-attribute-name": "CustomerName",
          "attribute-type": "scalar",
          "attribute-sub-type": "string",
          "value": "${FirstName},${LastName}"
        },
        {
          "target-attribute-name": "StoreId",
          "attribute-type": "scalar",
          "attribute-sub-type": "string",
          "value": "${StoreId}"
        },
      ]
    }
  ]
}

```



```
| team          | varchar(10) | YES | NULL |
+-----+-----+-----+-----+
```

The structure of the MySQL database table *sport_team* is shown below:

```
mysql> desc sport_team;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id             | mediumint(9) | NO   | PRI | NULL    | auto_increment |
| name           | varchar(30)  | NO   |     | NULL    |                |
| abbreviated_name | varchar(10)  | YES  |     | NULL    |                |
| home_field_id  | smallint(6)  | YES  | MUL | NULL    |                |
| sport_type_name | varchar(15)  | NO   | MUL | NULL    |                |
| sport_league_short_name | varchar(10) | NO   |     | NULL    |                |
| sport_division_short_name | varchar(10) | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
```

The table-mapping rules used to map the two tables to the two DynamoDB tables is shown below:

```
{
  "rules":[
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "dms_sample",
        "table-name": "nfl_data"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "selection",
      "rule-id": "2",
      "rule-name": "2",
      "object-locator": {
        "schema-name": "dms_sample",
        "table-name": "sport_team"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "object-mapping",
      "rule-id": "3",
      "rule-name": "MapNFLData",
      "rule-action": "map-record-to-record",
      "object-locator": {
        "schema-name": "dms_sample",
        "table-name": "nfl_data"
      },
      "target-table-name": "NFLTeams",
      "mapping-parameters": {
        "partition-key-name": "Team",
        "sort-key-name": "PlayerName",
        "exclude-columns": [
          "player_number", "team", "Name"
        ]
      },
      "attribute-mappings": [
        {

```



```

        "target-attribute-name": "Team",
        "attribute-type": "scalar",
        "attribute-sub-type": "string",
        "value": "${team}"
    },
    {
        "target-attribute-name": "PlayerName",
        "attribute-type": "scalar",
        "attribute-sub-type": "string",
        "value": "${Name}"
    },
    {
        "target-attribute-name": "PlayerInfo",
        "attribute-type": "scalar",
        "attribute-sub-type": "string",
        "value": "{ \"Number\": \"${player_number}\", \"Position\": \"${Position}\",
        \"/>

```

The sample output for the *NFLTeams* DynamoDB table is shown below:

```

    "PlayerInfo": "{ \"Number\": \"6\", \"Position\": \"P\", \"Status\": \"ACT\", \"Stats\":
    { \"Stat1\": \"PUNTS:73\", \"Stat2\": \"AVG:46\", \"Stat3\": \"LNG:67\", \"Stat4\": \"IN
    20:31\" } }",

```

```
"PlayerName": "Allen, Ryan",  
"Position": "P",  
"stat1": "PUNTS",  
"stat1_val": "73",  
"stat2": "AVG",  
"stat2_val": "46",  
"stat3": "LNG",  
"stat3_val": "67",  
"stat4": "IN 20",  
"stat4_val": "31",  
"status": "ACT",  
"Team": "NE"  
}
```

The sample output for the `SportsTeams` *DynamoDB* table is shown below:

```
{  
  "abbreviated_name": "IND",  
  "home_field_id": 53,  
  "sport_division_short_name": "AFC South",  
  "sport_league_short_name": "NFL",  
  "sport_type_name": "football",  
  "TeamInfo": "{\"League\": \"NFL\", \"Division\": \"AFC South\"}",  
  "TeamName": "Indianapolis Colts"  
}
```

Target Data Types for Amazon DynamoDB

The Amazon DynamoDB endpoint for Amazon AWS DMS supports most Amazon DynamoDB data types. The following table shows the Amazon AWS DMS target data types that are supported when using AWS DMS and the default mapping from AWS DMS data types.

For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service](#) (p. 319).

When AWS DMS migrates data from heterogeneous databases, we map data types from the source database to intermediate data types called AWS DMS data types. We then map the intermediate data types to the target data types. The following table shows each AWS DMS data type and the data type it maps to in DynamoDB:

AWS DMS Data Type	DynamoDB Data Type
String	String
WString	String
Boolean	Boolean
Date	String
DateTime	String
INT1	Number
INT2	Number
INT4	Number

AWS DMS Data Type	DynamoDB Data Type
INT8	Number
Numeric	Number
Real4	Number
Real8	Number
UINT1	Number
UINT2	Number
UINT4	Number
UINT8	Number
CLOB	String

Using Amazon Kinesis Data Streams as a Target for AWS Database Migration Service

You can use AWS DMS to migrate data to an Amazon Kinesis data stream. Amazon Kinesis data streams are part of the Amazon Kinesis Data Streams service. You can use Kinesis data streams to collect and process large streams of data records in real time.

A Kinesis data stream is made up of shards. *Shards* are uniquely identified sequences of data records in a stream. For more information on shards in Amazon Kinesis Data Streams, see [Shard](#) in the *Amazon Kinesis Data Streams Developer Guide*.

AWS Database Migration Service publishes records to a Kinesis data stream using JSON. During conversion, AWS DMS serializes each record from the source database into an attribute-value pair in JSON format.

You must use AWS Database Migration Service engine version 3.1.2 or higher to migrate data to Amazon Kinesis Data Streams.

You use object mapping to migrate your data from any supported data source to a target stream. With object mapping, you determine how to structure the data records in the stream. You also define a partition key for each table, which Kinesis Data Streams uses to group the data into its shards.

Prerequisites for Using a Kinesis Data Stream as a Target for AWS Database Migration Service

Before you set up a Kinesis data stream as a target for AWS DMS, make sure that you create an IAM role. This role must allow AWS DMS to assume and grant access to the Kinesis data streams that are being migrated into. The minimum set of access permissions is shown in the following example role policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
```

```
    "Service": "dms.amazonaws.com"
  },
  "Action": "sts:AssumeRole"
}
]
}
```

The role that you use for the migration to a Kinesis data stream must have the following permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:ListStreams",
        "kinesis:PutRecords"
      ],
      "Resource": "arn:aws:kinesis:region:account-id:stream/stream-name"
    },
  ]
}
```

Limitations When Using Kinesis Data Streams as a Target for AWS Database Migration Service

The following limitations apply when using Kinesis Data Streams as a target:

- AWS DMS publishes each update to a single record in the source database as one data record in a given Kinesis data stream. As a result, applications consuming the data from the stream lose track of transaction boundaries.
- Kinesis data streams don't support deduplication. Applications that consume data from a stream need to handle duplicate records. For more information, see [Handling Duplicate Records](#) in the *Amazon Kinesis Data Streams Developer Guide*.
- AWS DMS supports the following two forms for partition keys:
 - `SchemaName.TableName`: A combination of the schema and table name.
 - `${AttributeName}`: The value of one of the fields in the JSON, or the primary key of the table in the source database.

Using Object Mapping to Migrate Data to a Kinesis Data Stream

AWS DMS uses table-mapping rules to map data from the source to the target Kinesis data stream. To map data to a target stream, you use a type of table-mapping rule called object mapping. You use object mapping to define how data records in the source map to the data records published to the Kinesis data stream.

Kinesis data streams don't have a preset structure other than having a partition key.

To create an object mapping rule, you specify `rule-type` as `object-mapping`. This rule specifies what type of object mapping you want to use.

The structure for the rule is as follows.

```
{ "rules": [
  {
    "rule-type": "object-mapping",
    "rule-id": "<id>",
    "rule-name": "<name>",
    "rule-action": "<valid object-mapping rule action>",
    "object-locator": {
      "schema-name": "<case-sensitive schema name>",
      "table-name": ""
    },
  },
]
}
```

AWS DMS currently supports `map-record-to-record` and `map-record-to-document` as the only valid values for the `rule-action` parameter. `map-record-to-record` and `map-record-to-document` specify what AWS DMS does by default to records that aren't excluded as part of the `exclude-columns` attribute list. These values don't affect the attribute mappings in any way.

Use `map-record-to-record` when migrating from a relational database to a Kinesis data stream. This rule type uses the `taskResourceId.schemaName.tableName` value from the relational database as the partition key in the Kinesis data stream and creates an attribute for each column in the source database. When using `map-record-to-record`, for any column in the source table not listed in the `exclude-columns` attribute list, AWS DMS creates a corresponding attribute in the target stream. This corresponding attribute is created regardless of whether that source column is used in an attribute mapping.

One way to understand `map-record-to-record` is to see it in action. For this example, assume that you are starting with a relational database table row with the following structure and data.

FirstName	LastName	StoreId	HomeAddress	HomePhone	WorkAddress	WorkPhone	DateOfBirth
Randy	Marsh	5	221B Baker Street	123456789031	Spooner Street, Quahog	9876543210	02/29/1988

To migrate this information to a Kinesis data stream, you create rules to map the data to the target stream. The following rule illustrates the mapping.

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "test",
        "table-name": "%"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "object-mapping",
      "rule-id": "1",
      "rule-name": "DefaultMapToKinesis",
      "rule-action": "map-record-to-document",
      "object-locator": {
```

```
    "schema-name": "Test",
    "table-name": "Customer",
  },
}
]
```

The following illustrates the resulting record format in the Kinesis data stream.

- StreamName: XXX
- PartitionKey: Test.Customers //schmaName.tableName
- Data: //The following JSON message

```
{
  "FirstName": "Randy",
  "LastName": "Marsh",
  "StoreId": "5",
  "HomeAddress": "221B Baker Street",
  "HomePhone": "1234567890",
  "WorkAddress": "31 Spooner Street, Quahog",
  "WorkPhone": "9876543210",
  "DateOfBirth": "02/29/1988"
}
```

Restructuring Data with Attribute Mapping

You can restructure the data while you are migrating it to a Kinesis data stream using an attribute map. For example, you might want to combine several fields in the source into a single field in the target. The following attribute map illustrates how to restructure the data.

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "Test",
        "table-name": "%"
      },
      "rule-action": "include"
    }
  ]
}
{
  {
    "rule-type": "object-mapping",
    "rule-id": "1",
    "rule-name": "TransformToKinesis",
    "rule-action": "map-record-to-document",
    "object-locator": {
      "schema-name": "Test",
      "table-name": "Customer",
    },
    "mapping-parameters": {
      "partition-key":{
```

```

        "attribute-name": "CustomerName",
        "value": "${FirstName},${LastName}"
    },
    "exclude-columns": [
        "FirstName", "LastName", "HomeAddress", "HomePhone", "WorkAddress",
        "WorkPhone"
    ],
    "attribute-mappings": [
        {
            "attribute-name": "CustomerName",
            "value": "${FirstName},${LastName}"
        },
        {
            "attribute-name": "ContactDetails",
            "value": {
                "Home": {
                    "Address": "${HomeAddress}",
                    "Phone": ${HomePhone}
                },
                "Work": {
                    "Address": "${WorkAddress}",
                    "Phone": ${WorkPhone}
                }
            }
        },
        {
            "attribute-name": "DateOfBirth",
            "value": "${DateOfBirth}"
        }
    ]
}
}
]
}

```

To set a constant value for **partition-key**, specify a **partition-key** value. For example, you might do this to force all the data to be stored in a single shard. The following mapping illustrates this approach.

```

{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "Test",
        "table-name": "%"
      },
      "rule-action": "include"
    }
  ],
  {
    {
      "rule-type": "object-mapping",
      "rule-id": "1",
      "rule-name": "TransformToKinesis",
      "rule-action": "map-record-to-document",
      "object-locator": {
        "schema-name": "Test",
        "table-name": "Customer",
      },
      "mapping-parameters": {
        "partition-key": {
          "value": "ConstantPartitionKey"
        }
      }
    }
  }
}

```

```
    },
    "exclude-columns": [
      "FirstName", "LastName", "HomeAddress", "HomePhone", "WorkAddress",
      "WorkPhone"
    ],
    "attribute-mappings": [
      {
        "attribute-name": "CustomerName",
        "value": "${FirstName},${LastName}"
      },
      {
        "attribute-name": "ContactDetails",
        "value": {
          "Home": {
            "Address": "${HomeAddress}",
            "Phone": ${HomePhone}
          },
          "Work": {
            "Address": "${WorkAddress}",
            "Phone": ${WorkPhone}
          }
        }
      },
      {
        "attribute-name": "DateOfBirth",
        "value": "${DateOfBirth}"
      }
    ]
  }
}
```

Message Format for Kinesis Data Streams

The JSON output is simply a list of key-value pairs. AWS DMS provides the following reserved fields to make it easier to consume the data from the Kinesis Data Streams:

RecordType

The record type can be either data or control. *Data records* represent the actual rows in the source. *Control records* are for important events in the stream, for example a restart of the task.

Operation

For data records, the operation can be `create`, `read`, `update`, or `delete`.

For control records, the operation can be `TruncateTable` or `DropTable`.

SchemaName

The source schema for the record. This field can be empty for a control record.

TableName

The source table for the record. This field can be empty for a control record.

Timestamp

The timestamp for when the JSON message was constructed. The field is formatted with the ISO 8601 format.

Note

The `partition-key` value for a control record that is for a specific table is `TaskId.SchemaName.TableName`. The `partition-key` value for a control record that is for a specific task is that record's `TaskId`. Specifying a `partition-key` value in the object mapping has no impact on the `partition-key` for a control record.

Using an Amazon Elasticsearch Service Cluster as a Target for AWS Database Migration Service

You can use AWS DMS to migrate data to Amazon Elasticsearch Service (Amazon ES). Amazon ES is a managed service that makes it easy to deploy, operate, and scale an Elasticsearch cluster.

In Elasticsearch, you work with indexes and documents. An *index* is a collection of documents, and a *document* is a JSON object containing scalar values, arrays, and other objects. Elasticsearch provides a JSON-based query language, so that you can query data in an index and retrieve the corresponding documents.

When AWS DMS creates indexes for a target endpoint for Amazon Elasticsearch Service, it creates one index for each table from the source endpoint. The cost for creating an Elasticsearch index depends on several factors. These are the number of indexes created, the total amount of data in these indexes, and the small amount of metadata that Elasticsearch stores for each document.

You must use AWS Database Migration Service engine version 3.1.2 or higher to migrate data to Amazon Elasticsearch Service.

Configure your Elasticsearch cluster with compute and storage resources that are appropriate for the scope of your migration. We recommend that you consider the following factors, depending on the replication task you want to use:

- For a full data load, consider the total amount of data that you want to migrate, and also the speed of the transfer.
- For replicating ongoing changes, consider the frequency of updates, and your end-to-end latency requirements.

Also, configure the index settings on your Elasticsearch cluster, paying close attention to the shard and replica count.

Migrating from a Relational Database Table to an Amazon ES Index

AWS DMS supports migrating data to Elasticsearch's scalar data types. When migrating from a relational database like Oracle or MySQL to Elasticsearch, you might want to restructure how you store this data.

AWS DMS supports the following Elasticsearch scalar data types:

- Boolean
- Date
- Float
- Int
- String

AWS DMS converts data of type `Date` into type `String`. You can specify custom mapping to interpret these dates.

AWS DMS doesn't support migration of LOB data types.

Prerequisites for Using Amazon Elasticsearch Service as a Target for AWS Database Migration Service

Before you begin work with an Elasticsearch database as a target for AWS DMS, make sure that you create an AWS Identity and Access Management (IAM) role. This role should let AWS DMS access the Elasticsearch indexes at the target endpoint. The minimum set of access permissions is shown in the following sample role policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
        "Service": "dms.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

The role that you use for the migration to Elasticsearch must have the following permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "es:ESHttpDelete",
        "es:ESHttpGet",
        "es:ESHttpHead",
        "es:ESHttpPost",
        "es:es:ESHttpPut"
      ],
      "Resource": "arn:aws:es:region:account-id:domain/domain-name/*"
    }
  ]
}
```

In the preceding example, replace *region* with the AWS Region identifier, *account-id* with your AWS account ID, and *domain-name* with the name of your Amazon Elasticsearch Service domain. An example is `arn:aws:es:us-west-2:123456789012:domain/my-es-domain`

Extra Connection Attributes When Using Elasticsearch as a Target for AWS DMS

When you set up your Elasticsearch target endpoint, you can specify extra connection attributes. Extra connection attributes are specified by key-value pairs and separated by semicolons.

The following table describes the extra connection attributes available when using an Elasticsearch instance as an AWS DMS source.

Attribute Name	Valid Values	Default Value and Description
fullLoadErrorThreshold	Positive integer greater than 0 but no larger than 100.	10 – For a full load task, this attribute determines the threshold of errors allowed before the task fails. For example, suppose that there are 1,500 rows at the source endpoint and this parameter is set to 10. Then the task fails if AWS DMS encounters more than 150 errors (10 percent of the row count) when writing to the target endpoint.
errorRetryDuration	Positive integer greater than 0.	300 – If an error occurs at the target endpoint, AWS DMS retries for this many seconds. Otherwise, the task fails.

Limitations When Using Amazon Elasticsearch Service as a Target for AWS Database Migration Service

The following limitations apply when using Amazon Elasticsearch Service as a target:

- AWS DMS only supports replication of tables with noncomposite primary keys. The primary key of the source table must consist of a single column.
- Elasticsearch uses dynamic mapping (auto guess) to determine the data types to use for migrated data.
- Elasticsearch stores each document with a unique ID. The following is an example ID.

```
"_id": "D359F8B537F1888BC71FE20B3D79EAE6674BE7ACA9B645B0279C7015F6FF19FD"
```

Each document ID is 64 bytes long, so anticipate this as a storage requirement. For example, if you migrate 100,000 rows from an AWS DMS source, the resulting Elasticsearch index requires storage for an additional 6,400,000 bytes.

- With Amazon ES, you can't make updates to the primary key attributes. This restriction is important when using ongoing replication with change data capture (CDC) because it can result in unwanted data in the target. In CDC mode, primary keys are mapped to SHA256 values, which are 32 bytes long. These are converted to human-readable 64-byte strings, and are used as Elasticsearch document IDs.
- If AWS DMS encounters any items that can't be migrated, it writes error messages to Amazon CloudWatch Logs. This behavior differs from that of other AWS DMS target endpoints, which write errors to an exceptions table.

Target Data Types for Amazon Elasticsearch Service

When AWS DMS migrates data from heterogeneous databases, the service maps data types from the source database to intermediate data types called AWS DMS data types. The service then maps the intermediate data types to the target data types. The following table shows each AWS DMS data type and the data type it maps to in Elasticsearch.

AWS DMS Data Type	Elasticsearch Data Type
Boolean	boolean
Date	string
Time	date

AWS DMS Data Type	Elasticsearch Data Type
Timestamp	date
INT4	integer
Real4	float
UINT4	integer

For additional information about AWS DMS data types, see [Data Types for AWS Database Migration Service](#) (p. 319).

Using Amazon DocumentDB as a Target for AWS Database Migration Service

You can use AWS DMS to migrate data to Amazon DocumentDB (with MongoDB compatibility) from any of the source data engines that AWS DMS supports. The source engine can be on an Amazon-managed service such as Amazon RDS, Aurora, or Amazon S3. Alternatively, the engine can be on a self-managed database, such as MongoDB running on Amazon EC2 or on-premises.

You can use AWS DMS to replicate source data to Amazon DocumentDB databases, collections, or documents.

If the source endpoint is MongoDB, make sure to enable the following extra connection attributes:

- `nestingLevel=NONE`
- `extractDocID=FALSE`

For more information, see [Extra Connection Attributes When Using MongoDB as a Source for AWS DMS](#) (p. 135).

MongoDB stores data in a binary JSON format (BSON). AWS DMS supports all of the BSON data types that are supported by Amazon DocumentDB. For a list of these data types, see [Supported MongoDB APIs, Operations, and Data Types in the Amazon DocumentDB Developer Guide](#).

If the source endpoint is a relational database, AWS DMS maps database objects to Amazon DocumentDB as follows:

- A relational database, or database schema, maps to an Amazon DocumentDB *database*.
- Tables within a relational database map to *collections* in Amazon DocumentDB.
- Records in a relational table map to *documents* in Amazon DocumentDB. Each document is constructed from data in the source record.

If the source endpoint is Amazon S3, then the resulting Amazon DocumentDB objects correspond to AWS DMS mapping rules for Amazon S3. For example, consider the following URI.

```
s3://mybucket/hr/employee
```

In this case, AWS DMS maps the objects in `mybucket` to Amazon DocumentDB as follows:

- The top-level URI part (`hr`) maps to an Amazon DocumentDB database.
- The next URI part (`employee`) maps to an Amazon DocumentDB collection.

- Each object in `employee` maps to a document in Amazon DocumentDB.

For more information on mapping rules for Amazon S3, see [Using Amazon Simple Storage Service as a Source for AWS DMS](#) (p. 138).

For additional details on working with Amazon DocumentDB as a target for AWS DMS, including a walkthrough of the migration process, see the following sections.

Topics

- [Mapping Data from a Source to an Amazon DocumentDB Target](#) (p. 199)
- [Ongoing Replication with Amazon DocumentDB as a Target](#) (p. 202)
- [Limitations to Using Amazon DocumentDB as a Target](#) (p. 203)
- [Target Data Types for Amazon DocumentDB](#) (p. 203)
- [Walkthrough: Migrating from MongoDB to Amazon DocumentDB](#) (p. 204)

Mapping Data from a Source to an Amazon DocumentDB Target

AWS DMS reads records from the source endpoint, and constructs JSON documents based on the data it reads. For each JSON document, AWS DMS must determine an `_id` field to act as a unique identifier. It then writes the JSON document to an Amazon DocumentDB collection, using the `_id` field as a primary key.

Source Data That Is a Single Column

If the source data consists of a single column, the data must be of a string type. (Depending on the source engine, the actual data type might be VARCHAR, NVARCHAR, TEXT, LOB, CLOB, or similar.) AWS DMS assumes that the data is a valid JSON document, and replicates the data to Amazon DocumentDB as is.

If the resulting JSON document contains a field named `_id`, then that field is used as the unique `_id` in Amazon DocumentDB.

If the JSON doesn't contain an `_id` field, then Amazon DocumentDB generates an `_id` value automatically.

Source Data That Is Multiple Columns

If the source data consists of multiple columns, then AWS DMS constructs a JSON document from all of these columns. To determine the `_id` field for the document, AWS DMS proceeds as follows:

- If one of the columns is named `_id`, then the data in that column is used as the target `_id`.
- If there is no `_id` column, but the source data has a primary key or a unique index, then AWS DMS uses that key or index value as the `_id` value. The data from the primary key or unique index also appears as explicit fields in the JSON document.
- If there is no `_id` column, and no primary key or a unique index, then Amazon DocumentDB generates an `_id` value automatically.

Coercing a Data Type at the Target Endpoint

AWS DMS can modify data structures when it writes to an Amazon DocumentDB target endpoint. You can request these changes by renaming columns and tables at the source endpoint, or by providing transformation rules that are applied when a task is running.

Using a Nested JSON Document (json_ Prefix)

To coerce a data type, you can prefix the source column name with `json_` (that is, `json_columnName`) either manually or using a transformation. In this case, the column is created as a nested JSON document within the target document, rather than as a string field.

For example, suppose that you want to migrate the following document from a MongoDB source endpoint.

```
{
  "_id": "1",
  "FirstName": "John",
  "LastName": "Doe",
  "ContactDetails": "{\"Home\": {\"Address\": \"Boston\", \"Phone\": \"1111111\"}, \"Work\": { \"Address\": \"Boston\", \"Phone\": \"222222222\"}}"
```

If you don't coerce any of the source data types, the embedded `ContactDetails` document is migrated as a string.

```
{
  "_id": "1",
  "FirstName": "John",
  "LastName": "Doe",
  "ContactDetails": "{\"Home\": {\"Address\": \"Boston\", \"Phone\": \"1111111\"}, \"Work\": { \"Address\": \"Boston\", \"Phone\": \"222222222\"}}"
```

However, you can add a transformation rule to coerce `ContactDetails` to a JSON object. For example, suppose that the original source column name is `ContactDetails`. Suppose also that the renamed source column is to be `json_ContactDetails`. AWS DMS replicates the `ContactDetails` field as nested JSON, as follows.

```
{
  "_id": "1",
  "FirstName": "John",
  "LastName": "Doe",
  "ContactDetails": {
    "Home": {
      "Address": "Boston",
      "Phone": "1111111111"
    },
    "Work": {
      "Address": "Boston",
      "Phone": "2222222222"
    }
  }
}
```

Using a JSON Array (array_ Prefix)

To coerce a data type, you can prefix a column name with `array_` (that is, `array_columnName`), either manually or using a transformation. In this case, AWS DMS considers the column as a JSON array, and creates it as such in the target document.

Suppose that you want to migrate the following document from a MongoDB source endpoint.

```
{
  "_id" : "1",
```

```
"FirstName": "John",  
"LastName": "Doe",  
  
"ContactAddresses": ["Boston", "New York"],  
  
"ContactPhoneNumbers": ["1111111111", "2222222222"]  
}
```

If you don't coerce any of the source data types, the embedded `ContactDetails` document is migrated as a string.

```
{  
  "_id": "1",  
  "FirstName": "John",  
  "LastName": "Doe",  
  
  "ContactAddresses": "[\"Boston\", \"New York\"]",  
  
  "ContactPhoneNumbers": "[\"1111111111\", \"2222222222\"]"  
}
```

However, you can add transformation rules to coerce `ContactAddress` and `ContactPhoneNumbers` to JSON arrays, as shown in the following table.

Original Source Column Name	Renamed Source Column
ContactAddress	array_ContactAddress
ContactPhoneNumbers	array_ContactPhoneNumbers

AWS DMS replicates `ContactAddress` and `ContactPhoneNumbers` as follows.

```
{  
  "_id": "1",  
  "FirstName": "John",  
  "LastName": "Doe",  
  "ContactAddresses": [  
    "Boston",  
    "New York"  
  ],  
  "ContactPhoneNumbers": [  
    "1111111111",  
    "2222222222"  
  ]  
}
```

Connecting to Amazon DocumentDB Using TLS

By default, a newly created Amazon DocumentDB cluster accepts secure connections only using Transport Layer Security (TLS). When TLS is enabled, every connection to Amazon DocumentDB requires a public key.

You can download the public key for Amazon DocumentDB as the [rds-combined-ca-bundle.pem](#) file from an AWS-hosted Amazon S3 bucket.

After you download this .pem file, you can import the file into AWS DMS as described following.

AWS Management Console

To import the public key (.pem) file

1. Open the AWS DMS console at <https://console.aws.amazon.com/dms>.
2. In the navigation pane, choose **Certificates**.
3. Choose **Import certificate** and do the following:
 - For **Certificate identifier**, enter a unique name for the certificate, for example `docdb-cert`.
 - For **Import file**, navigate to the location where you saved the .pem file.

When the settings are as you want them, choose **Add new CA certificate**.

AWS CLI

Use the `aws dms import-certificate` command, as shown in the following example.

```
aws dms import-certificate \  
  --certificate-identifier docdb-cert \  
  --certificate-pem file://./rds-combined-ca-bundle.pem
```

When you create an AWS DMS target endpoint, provide the certificate identifier (for example, `docdb-cert`). Also, set the SSL mode parameter to `verify-full`.

Ongoing Replication with Amazon DocumentDB as a Target

If ongoing replication is enabled, AWS DMS ensures that documents in Amazon DocumentDB stay in sync with the source. When a source record is created or updated, AWS DMS must first determine which Amazon DocumentDB record is affected by doing the following:

- If the source record has a column named `_id`, the value of that column determines the corresponding `_id` in the Amazon DocumentDB collection.
- If there is no `_id` column, but the source data has a primary key or unique index, then AWS DMS uses that key or index value as the `_id` for the Amazon DocumentDB collection.
- If the source record doesn't have an `_id` column, a primary key, or a unique index, then AWS DMS matches all of the source columns to the corresponding fields in the Amazon DocumentDB collection.

When a new source record is created, AWS DMS writes a corresponding document to Amazon DocumentDB. If an existing source record is updated, AWS DMS updates the corresponding fields in the target document in Amazon DocumentDB. Any fields that exist in the target document but not in the source record remain untouched.

When a source record is deleted, AWS DMS deletes the corresponding document from Amazon DocumentDB.

Structural Changes (DDL) at the Source

With ongoing replication, any changes to source data structures (such as tables, columns, and so on) are propagated to their counterparts in Amazon DocumentDB. In relational databases, these changes are initiated using data definition language (DDL) statements. You can see how AWS DMS propagates these changes to Amazon DocumentDB in the following table.

DDL at Source	Effect at Amazon DocumentDB Target
<code>CREATE TABLE</code>	Creates an empty collection.

DDL at Source	Effect at Amazon DocumentDB Target
Statement that renames a table (<code>RENAME TABLE</code> , <code>ALTER TABLE . . . RENAME</code> , and similar)	Renames the collection.
<code>TRUNCATE TABLE</code>	Removes all the documents from the collection, but only if <code>HandleSourceTableTruncated</code> is <code>true</code> . For more information, see Task Settings for Change Processing DDL Handling (p. 234) .
<code>DROP TABLE</code>	Deletes the collection, but only if <code>HandleSourceTableDropped</code> is <code>true</code> . For more information, see Task Settings for Change Processing DDL Handling (p. 234) .
Statement that adds a column to a table (<code>ALTER TABLE . . . ADD</code> and similar)	The DDL statement is ignored, and a warning is issued. When the first <code>INSERT</code> is performed at the source, the new field is added to the target document.
<code>ALTER TABLE . . . RENAME COLUMN</code>	The DDL statement is ignored, and a warning is issued. When the first <code>INSERT</code> is performed at the source, the newly named field is added to the target document.
<code>ALTER TABLE . . . DROP COLUMN</code>	The DDL statement is ignored, and a warning is issued.
Statement that changes the column data type (<code>ALTER COLUMN . . . MODIFY</code> and similar)	The DDL statement is ignored, and a warning is issued. When the first <code>INSERT</code> is performed at the source with the new data type, the target document is created with a field of that new data type.

Limitations to Using Amazon DocumentDB as a Target

The following limitations apply when using Amazon DocumentDB as a target for AWS DMS:

- In Amazon DocumentDB, collection names can't contain the dollar symbol (\$). In addition, database names can't contain any Unicode characters.
- AWS DMS doesn't support merging of multiple source tables into a single Amazon DocumentDB collection.
- When AWS DMS processes changes from a source table that doesn't have a primary key, any LOB columns in that table are ignored.
- If the **Change table** option is enabled and AWS DMS encounters a source column named "`_id`", then that column appears as "`__id`" (two underscores) in the change table.
- If you choose Oracle as a source endpoint, then the Oracle source must have full supplemental logging enabled. Otherwise, if there are columns at the source that weren't changed, then the data is loaded into Amazon DocumentDB as null values.

Target Data Types for Amazon DocumentDB

In the following table, you can find the Amazon DocumentDB target data types that are supported when using AWS DMS, and the default mapping from AWS DMS data types. For more information about AWS DMS data types, see [Data Types for AWS Database Migration Service \(p. 319\)](#).

AWS DMS Data Type	Amazon DocumentDB Data Type
BOOLEAN	Boolean
BYTES	Binary data
DATE	Date
TIME	String (UTF8)
DATETIME	Date
INT1	32-bit integer
INT2	32-bit integer
INT4	32-bit integer
INT8	64-bit integer
NUMERIC	String (UTF8)
REAL4	Double
REAL8	Double
STRING	If the data is recognized as JSON, then AWS DMS migrates it to Amazon DocumentDB as a document. Otherwise, the data is mapped to String (UTF8).
UINT1	32-bit integer
UINT2	32-bit integer
UINT4	64-bit integer
UINT8	String (UTF8)
WSTRING	If the data is recognized as JSON, then AWS DMS migrates it to Amazon DocumentDB as a document. Otherwise, the data is mapped to String (UTF8).
BLOB	Binary
CLOB	If the data is recognized as JSON, then AWS DMS migrates it to Amazon DocumentDB as a document. Otherwise, the data is mapped to String (UTF8).
NCLOB	If the data is recognized as JSON, then AWS DMS migrates it to Amazon DocumentDB as a document. Otherwise, the data is mapped to String (UTF8).

Walkthrough: Migrating from MongoDB to Amazon DocumentDB

Use the following walkthrough to guide you through the process of migrating from MongoDB to Amazon DocumentDB (with MongoDB compatibility). In this walkthrough, 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

- [Step 1: Launch an Amazon EC2 Instance \(p. 205\)](#)
- [Step 2: Install and Configure MongoDB Community Edition \(p. 206\)](#)
- [Step 3: Create an AWS DMS Replication Instance \(p. 207\)](#)
- [Step 4: Create Source and Target Endpoints \(p. 208\)](#)
- [Step 5: Create and Run a Migration Task \(p. 209\)](#)

Step 1: Launch an Amazon EC2 Instance

For this walkthrough, you launch an Amazon EC2 instance into your default VPC.

To launch an Amazon EC2 instance

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:
 1. Choose **Select an existing security group**.
 2. 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 TCP 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*.
 3. 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*.

Step 2: Install and Configure MongoDB Community Edition

Perform these steps on the Amazon EC2 instance that you launched in [Step 1: Launch an Amazon EC2 Instance](#) (p. 205).

To install and configure MongoDB Community Edition on your EC2 instance

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:
  port: 27017
  bindIp: 127.0.0.1 # Enter 0.0.0.0,:: to bind to all IPv4 and IPv6 addresses or,
alternatively, use the net.bindIpAll setting.
```

- b. Modify the `bindIp` line so that it looks like the following.

```
bindIp: public-dns-name
```

- c. Replace `public-dns-name` with the actual public DNS name for your instance, for example `ec2-11-22-33-44.us-west-2.compute.amazonaws.com`.
- d. Save the `/etc/mongod.conf` file, and then restart `mongod`.

```
sudo service mongod restart
```

3. Populate your MongoDB instance with data by doing the following:
 - a. Use the `wget` command to download a JSON file containing sample data.

```
wget http://media.mongodb.org/zips.json
```

- b. Use the `mongoimport` command to import the data into a new database (`zips-db`).

```
mongoimport --host public-dns-name:27017 --db zips-db --file zips.json
```

- c. After the import completes, use the mongo shell to connect to MongoDB and verify that the data was loaded successfully.

```
mongo --host public-dns-name:27017
```

- d. Replace *public-dns-name* with the actual public DNS name for your instance.
- e. At the mongo shell prompt, enter the following commands.

```
use zips-db

db.zips.count()

db.zips.aggregate( [
  { $group: { _id: { state: "$state", city: "$city" }, pop: { $sum: "$pop" } } },
  { $group: { _id: "$_id.state", avgCityPop: { $avg: "$pop" } } }
] )
```

The output should display the following:

- The name of the database (zips-db)
 - The number of documents in the zips collection (29353)
 - The average population for cities in each state
- f. Exit from the mongo shell and return to the command prompt by using the following command.

```
exit
```

Step 3: Create an AWS DMS Replication Instance

To perform replication in AWS DMS, you need a replication instance.

To create an AWS DMS replication instance

1. Open the AWS DMS console at <https://console.aws.amazon.com/dms/>.
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.

Step 4: Create Source and Target Endpoints

The source endpoint is the endpoint for your MongoDB installation running on your Amazon EC2 instance.

To create a source endpoint

1. Open the AWS DMS console at <https://console.aws.amazon.com/dms/>.
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 **Authentication mechanism**, choose **default**.
 - 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:

- Have available the master user name and password for your Amazon DocumentDB cluster.
- Have available 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.

```
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.

```
wget https://s3.amazonaws.com/rds-downloads/rds-combined-ca-bundle.pem
```

To create a target endpoint

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**, choose **Add new CA certificate**, and then for **Certificate Identifier**, enter `rds-combined-ca-bundle`. For **Import file**, navigate to the `rds-combined-ca-bundle.pem` file that you downloaded. When you are finished, choose **Add new CA certificate**.
 - 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

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

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`).

Step 5: 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.

1. Open the AWS DMS console at <https://console.aws.amazon.com/dms/>.

2. In the navigation pane, choose **Tasks**.
3. Choose **Create task** and enter the following information:
 - For **Task name**, enter a name that's easy to remember, for example `my-dms-task`.
 - For **Replication instance**, choose the replication instance that you created in [Step 3: Create an AWS DMS Replication Instance \(p. 207\)](#).
 - For **Source endpoint**, choose the source endpoint that you created in [Step 4: Create Source and Target Endpoints \(p. 208\)](#).
 - For **Target endpoint**, choose the target endpoint that you created in [Step 4: Create Source and Target Endpoints \(p. 208\)](#).
 - For **Migration type**, choose **Migrate existing data**.
 - For **Start task on create**, enable this option.

In the **Task Settings** section, keep all of the options at their default values.

In the **Table mappings** section, choose the **Guided** tab, and then enter the following information:

- For **Schema name is**, choose **Enter a schema**.
- For **Schema name is like**, keep this at its default setting (%).
- For **Table name is like**, keep this at its default setting (%).

Choose **Add selection rule** to confirm that the information is correct.

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*.

Creating Source and Target Endpoints

You can create source and target endpoints when you create your replication instance or you can create endpoints after your replication instance is created. The source and target data stores can be on an Amazon Elastic Compute Cloud (Amazon EC2) instance, an Amazon Relational Database Service (Amazon RDS) DB instance, or an on-premises database.

The procedure following assumes that you have chosen the AWS DMS console wizard. Note that you can also do this step by selecting **Endpoints** from the AWS DMS console's navigation pane and then selecting **Create endpoint**. When using the console wizard, you create both the source and target endpoints on the same page. When not using the console wizard, you create each endpoint separately.

To specify source or target database endpoints using the AWS console

1. On the **Connect source and target database endpoints** page, specify your connection information for the source or target database. The following table describes the settings.

Endpoint type* Source Target ⓘ

Select RDS DB Instance ⓘ

Endpoint identifier* ⓘ

Source engine* postgres ⓘ

Server name*

Port* 8199 ⓘ

SSL mode* - Select One - ⓘ

User name* ⓘ

Password* ⓘ

Database name*

▶ Advanced

For This Option	Do This
Endpoint type	Choose whether this endpoint is the source or target endpoint.
Select RDS DB Instance	Choose this option if the endpoint is an Amazon RDS DB instance.
Endpoint identifier	Type the name you want to use to identify the endpoint. You might want to include in the name the type of endpoint, such as oracle-source or PostgreSQL-target . The name must be unique for all replication instances.
Source engine and Target engine	Choose the type of database engine that is the endpoint.
Server name	Type the server name. For an on-premises database, this can be the IP address or the public hostname. For an Amazon RDS DB instance, this can be the endpoint (also called the DNS name) for the DB instance, such as mysqlsrvinst.abcd12345678.us-west-2.rds.amazonaws.com .
Port	Type the port used by the database.

For This Option	Do This
SSL mode	Choose an SSL mode if you want to enable connection encryption for this endpoint. Depending on the mode you select, you might be asked to provide certificate and server certificate information.
User name	Type the user name with the permissions required to allow data migration. For information on the permissions required, see the security section for the source or target database engine in this user guide.
Password	Type the password for the account with the required permissions. If you want to use special characters in your password, such as "+" or "&", enclose the entire password in curly braces "{}".
Database name	The name of the database you want to use as the endpoint.

- Choose the **Advanced** tab, shown following, to set values for connection string and encryption key if you need them. You can test the endpoint connection by choosing **Run test**.

▼ **Advanced**

Extra connection attributes ⓘ

KMS master key (Default) aws/dms ⓘ

Description Default master key that protects my DMS replication instance volumes when no other key is defined

Account [REDACTED]

Key ARN [REDACTED]

▼ **Test endpoint connection (optional)**

Test your endpoint connection by selecting a replication instance within your desired VPC. After clicking with the details provided and attempt to connect to the instance. If the connection fails, you can edit ar will be deleted.

VPC* [REDACTED] ▼

Replication instance* - Select One - ⓘ

Refresh schemas after successful ⓘ

For This Option	Do This
Extra connection attributes	Type any additional connection parameters here. For more information about extra connection attributes, see the documentation section for your data store.
KMS master key	Choose the encryption key to use to encrypt replication storage and connection information. If you choose (Default) aws/dms , the default AWS Key Management Service (AWS KMS) key associated with your account and region is used. For more information on using the encryption key, see Setting an Encryption Key and Specifying KMS Permissions (p. 44) .
Test endpoint connection (optional)	Add the VPC and replication instance name. To test the connection, choose Run test .

Working with AWS DMS Tasks

An AWS Database Migration Service (AWS DMS) task is where all the work happens. You specify what tables and schemas to use for your migration and any special processing, such as logging requirements, control table data, and error handling.

When creating a migration task, you need to know several things:

- Before you can create a task, you must create a source endpoint, a target endpoint, and a replication instance.
- You can specify many task settings to tailor your migration task. You can set these by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS DMS API. These settings include specifying how migration errors are handled, error logging, and control table information.
- After you create a task, you can run it immediately. The target tables with the necessary metadata definitions are automatically created and loaded, and you can specify ongoing replication.
- By default, AWS DMS starts your task as soon as you create it. However, in some situations, you might want to postpone the start of the task. For example, when using the AWS CLI, you might have a process that creates a task and a different process that starts the task based on some triggering event. As needed, you can postpone your task's start.
- You can monitor, stop, or restart tasks using the AWS DMS console, AWS CLI, or AWS DMS API.

The following are actions that you can do when working with an AWS DMS task.

Task	Relevant Documentation
<p>Creating a Task Assessment Report</p> <p>You can create a task assessment report that shows any unsupported data types that could cause problems during migration. You can run this report on your task before running the task to find out potential issues.</p>	<p>Creating a Task Assessment Report (p. 215)</p>
<p>Creating a Task</p> <p>When you create a task, you specify the source, target, and replication instance, along with any migration settings.</p>	<p>Creating a Task (p. 218)</p>
<p>Creating an Ongoing Replication Task</p> <p>You can set up a task to provide continuous replication between the source and target.</p>	<p>Creating Tasks for Ongoing Replication Using AWS DMS (p. 239)</p>
<p>Applying Task Settings</p> <p>Each task has settings that you can configure according to the needs of your database migration. You create</p>	<p>Specifying Task Settings for AWS Database Migration Service Tasks (p. 224)</p>

Task	Relevant Documentation
these settings in a JSON file or, with some settings, you can specify the settings using the AWS DMS console.	
Data Validation Data validation is a task setting you can use to have AWS DMS compare the data on your target data store with the data from your source data store.	Validating AWS DMS Tasks (p. 272) .
Modifying a Task When a task is stopped, you can modify the settings for the task.	Modifying a Task (p. 242)
Reloading Tables During a Task You can reload a table during a task if an error occurs during the task.	Reloading Tables During a Task (p. 242)
Using Table Mapping Table mapping uses several types of rules to specify the data source, source schema, data, and any transformations that should occur during the task.	Selection Rules Selection Rules and Actions (p. 250) Transformation Rules Transformation Rules and Actions (p. 252)
Applying Filters You can use source filters to limit the number and type of records transferred from your source to your target. For example, you can specify that only employees with a location of headquarters are moved to the target database. You apply filters on a column of data.	Using Source Filters (p. 257)
Monitoring a Task There are several ways to get information on the performance of a task and the tables used by the task.	Monitoring AWS DMS Tasks (p. 261)
Managing Task Logs You can view and delete task logs using the AWS DMS API or AWS CLI.	Managing AWS DMS Task Logs (p. 267)

Creating a Task Assessment Report

The task assessment feature identifies data types that might not get migrated correctly. During a task assessment, AWS DMS reads the source database schema and creates a list of data types. It then compares this list to a pre-defined list of data types supported by AWS DMS. AWS DMS creates a report you can look at to see if your migration task has unsupported data types.

The task assessment report includes a summary that lists the unsupported data types and the column count for each one. It includes a list of data structures in JSON for each unsupported data type. You can use the report to modify the source data types and improve the migration success.

There are two levels of unsupported data types. Data types that are shown on the report as “not supported” can’t be migrated. Data types that are shown on the report as “partially supported” might be converted to another data type and not migrate as you expect.

For example, the following is a sample task assessment report.

```
{
  "summary":{
    "task-name":"test15",
    "not-supported":{
      "data-type": [
        "sql-variant"
      ],
      "column-count":3
    },
    "partially-supported":{
      "data-type":[
        "float8",
        "jsonb"
      ],
      "column-count":2
    }
  },
  "types":[
    {
      "data-type":"float8",
      "support-level":"partially-supported",
      "schemas":[
        {
          "schema-name":"schema1",
          "tables":[
            {
              "table-name":"table1",
              "columns":[
                "column1",
                "column2"
              ]
            },
            {
              "table-name":"table2",
              "columns":[
                "column3",
                "column4"
              ]
            }
          ]
        },
        {
          "schema-name":"schema2",
          "tables":[
            {
              "table-name":"table3",
              "columns":[
                "column5",
                "column6"
              ]
            },
            {
              "table-name":"table4",
              "columns":[
                "column7",
```

```
        "column8"
      ]
    }
  ]
},
{
  "datatype": "int8",
  "support-level": "partially-supported",
  "schemas": [
    {
      "schema-name": "schema1",
      "tables": [
        {
          "table-name": "table1",
          "columns": [
            "column9",
            "column10"
          ]
        },
        {
          "table-name": "table2",
          "columns": [
            "column11",
            "column12"
          ]
        }
      ]
    }
  ]
}
]
```

You can view the latest task assessment report from the **Assessment** tab on the **Tasks** page on the AWS console. AWS DMS stores previous task assessment reports in an Amazon S3 bucket. The Amazon S3 bucket name is in the following format.

```
dms-<customerId>-<customerDNS>
```

The report is stored in the bucket in a folder named with the task name. The report's file name is the date of the assessment in the format yyyy-mm-dd-hh-mm. You can view and compare previous task assessment reports from the Amazon S3 console.

AWS DMS also creates an AWS Identity and Access Management (IAM) role to allow access to the S3 bucket; the role name is `dms-access-for-tasks`. The role uses the `AmazonDMSRedshiftS3Role` policy.

You can enable the task assessment feature using the AWS console, the AWS CLI, or the DMS API:

- On the console, choose **Task Assessment** when creating or modifying a task. To view the task assessment report using the console, choose the task on the **Tasks** page and choose the **Assessment results** tab in the details section.
- The CLI commands are `start-replication-task-assessment` to begin a task assessment and `describe-replication-task-assessment-results` to receive the task assessment report in JSON format.
- The AWS DMS API uses the `StartReplicationTaskAssessment` action to begin a task assessment and the `DescribeReplicationTaskAssessment` action to receive the task assessment report in JSON format.

Creating a Task

There are several things you must do to create an AWS DMS migration task:

- Create a source endpoint, a target endpoint, and a replication instance before you create a migration task.
- Select a migration method:
 - **Migrating Data to the Target Database** – This process creates files or tables in the target database and automatically defines the metadata that is required at the target. It also populates the tables with data from the source. The data from the tables is loaded in parallel for improved efficiency. This process is the **Migrate existing data** option in the AWS console and is called `Full Load` in the API.
 - **Capturing Changes During Migration** – This process captures changes to the source database that occur while the data is being migrated from the source to the target. When the migration of the originally requested data has completed, the change data capture (CDC) process then applies the captured changes to the target database. Changes are captured and applied as units of single committed transactions, and you can update several different target tables as a single source commit. This approach guarantees transactional integrity in the target database. This process is the **Migrate existing data and replicate ongoing changes** option in the AWS console and is called `full-load-and-cdc` in the API.
 - **Replicating Only Data Changes on the Source Database** – This process reads the recovery log file of the source database management system (DBMS) and groups together the entries for each transaction. In some cases, AWS DMS can't apply changes to the target within a reasonable time (for example, if the target is not accessible). In these cases, AWS DMS buffers the changes on the replication server for as long as necessary. It doesn't reread the source DBMS logs, which can take a large amount of time. This process is the **Replicate data changes only** option in the AWS DMS console.
- Determine how the task should handle large binary objects (LOBs) on the source. For more information, see [Setting LOB Support for Source Databases in a AWS DMS Task \(p. 238\)](#).
- Specify migration task settings. These include setting up logging, specifying what data is written to the migration control table, how errors are handled, and other settings. For more information about task settings, see [Specifying Task Settings for AWS Database Migration Service Tasks \(p. 224\)](#).
- Set up table mapping to define rules to select and filter data that you are migrating. For more information about table mapping, see [Using Table Mapping to Specify Task Settings \(p. 245\)](#). Before you specify your mapping, make sure that you review the documentation section on data type mapping for your source and your target database.

You can choose to start a task as soon as you finish specifying information for that task on the **Create task** page. Alternatively, you can start the task from the Dashboard page after you finish specifying task information.

The procedure following assumes that you have chosen the AWS DMS console wizard and specified replication instance information and endpoints using the console wizard. You can also do this step by selecting **Tasks** from the AWS DMS console's navigation pane and then selecting **Create task**.

To create a migration task

1. On the **Create Task** page, specify the task options. The following table describes the settings.

Create Task

A task can contain one or more table mappings which define what data is moved from the source to the target. If a table does not exist on the target, it can be created automatically.

Task name ⓘ

Task description ⓘ

Source endpoint rdsoracle-endpoint

Target endpoint rdspostgres-endpoint

Replication instance replinstance1-26

Migration type ⓘ

Start task on create

▶ Task Settings

▶ Table mappings

For This Option	Do This
Task name	Type a name for the task.
Task description	Type a description for the task.
Source endpoint	Shows the source endpoint to be used.
Target endpoint	Shows the target endpoint to be used.
Replication instance	Shows the replication instance to be used.
Migration type	Choose the migration method you want to use. You can choose to have just the existing data migrated to the target database or have ongoing changes sent to the target database in addition to the migrated data.
Start task on create	When this option is selected, the task begins as soon as it is created.

2. Choose the **Task Settings** tab, shown following, and specify values for your target table, LOB support, and to enable logging. The task settings shown depend on the **Migration type** value you select. For example, when you select **Migrate existing data**, the following options are shown:

▼ **Task Settings**

Target table preparation mode Do nothing ⓘ
 Drop tables on target
 Truncate

Include LOB columns in replication Don't include LOB columns ⓘ
 Full LOB mode
 Limited LOB mode

Max LOB size (kb)* ⓘ

Enable logging

For This Option	Do This
<p>Target table preparation mode</p>	<p>Do nothing – In Do nothing mode, AWS DMS assumes that the target tables have been pre-created on the target. If the migration is a full load or full load plus CDC, you must ensure that the target tables are empty before starting the migration. Do nothing mode is an appropriate choice for CDC-only tasks when the target tables have been pre-backfilled from the source, and ongoing replication is applied to keep the source and target in-sync. You can use the AWS Schema Conversion Tool (AWS SCT), to pre-create target tables for you.</p> <p>Drop tables on target – In Drop tables on target mode, AWS DMS drops the target tables and recreates them before starting the migration. This ensures that the target tables are empty when the migration starts. AWS DMS creates only the objects required to efficiently migrate the data: tables, primary keys, and in some cases, unique indexes. AWS DMS doesn't create secondary indexes, non-primary key constraints, or column data defaults. If you are performing a full load plus CDC or CDC-only task, we recommend that you pause the migration and create secondary indexes that support filtering for update and delete statements.</p> <p>You might need to perform some configuration on the target database when you use Drop tables on target mode. For example, for an Oracle target, AWS DMS cannot create a schema (database user) for security reasons. In that case, you have to pre-create the schema user so AWS DMS can create the tables when the migration starts. For most other target types, AWS DMS creates the schema and all associated tables with the proper configuration parameters.</p> <p>Truncate – In Truncate mode, AWS DMS truncates all target tables before the migration starts. Truncate mode is appropriate for full load or full load plus CDC migrations where the target schema has been pre-created before the migration starts. You can use the AWS Schema Conversion Tool (AWS SCT) to pre-create target tables for you.</p>
<p>Include LOB columns in replication</p>	<p>Don't include LOB columns – LOB columns are excluded from the migration.</p> <p>Full LOB mode – Migrate complete LOBs regardless of size. AWS DMS migrates LOBs piecewise in chunks controlled by the Max LOB size parameter. This mode is slower than using Limited LOB mode.</p> <p>Limited LOB mode – Truncate LOBs to the value of the Max LOB size parameter. This mode is faster than using Full LOB mode.</p>

For This Option	Do This
Max LOB size (kb)	In Limited LOB Mode , LOB columns that exceed the setting of Max LOB size are truncated to the specified Max LOB Size .
Enable validation	Enables data validation, to verify that the data is migrated accurately from the source to the target. For more information, see Validating AWS DMS Tasks (p. 272) .
Enable logging	Enables logging by Amazon CloudWatch.

When you select **Migrate existing data and replicate** for **Migration type**, the following options are shown:

▼ Task Settings

Target table preparation mode Do nothing ⓘ
 Drop tables on target
 Truncate

Stop task after full load completes Don't stop ⓘ
 Stop Before Applying Cached Changes
 Stop After Applying Cached Changes

Include LOB columns in replication Don't include LOB columns ⓘ
 Full LOB mode
 Limited LOB mode

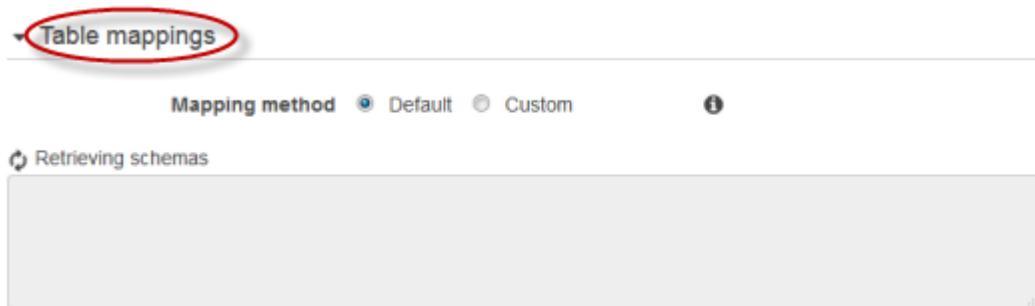
Max LOB size (kb)* ⓘ

Enable logging

For This Option	Do This
Target table preparation mode	<p>Do nothing – Data and metadata of the target tables are not changed.</p> <p>Drop tables on target – The tables are dropped and new tables are created in their place.</p> <p>Truncate – Tables are truncated without affecting table metadata.</p>

For This Option	Do This
Stop task after full load completes	<p>Don't stop – Don't stop the task but immediately apply cached changes and continue on.</p> <p>Stop before applying cached changes - Stop the task before the application of cached changes. Using this approach, you can add secondary indexes that might speed the application of changes.</p> <p>Stop after applying cached changes - Stop the task after cached changes have been applied. Using this approach, you can add foreign keys, triggers, and so on, if you are using transactional apply.</p>
Include LOB columns in replication	<p>Don't include LOB columns – LOB columns is excluded from the migration.</p> <p>Full LOB mode – Migrate complete LOBs regardless of size. LOBs are migrated piecewise in chunks controlled by the LOB chunk size. This method is slower than using Limited LOB Mode.</p> <p>Limited LOB mode – Truncate LOBs to 'Max LOB Size' This method is faster than using Full LOB Mode.</p>
Max LOB size (KB)	In Limited LOB Mode , LOB columns that exceed the setting of Max LOB Size are truncated to the specified Max LOB Size.
Enable validation	Enables data validation, to verify that the data is migrated accurately from the source to the target. For more information, see Validating AWS DMS Tasks (p. 272) .
Enable logging	Enables logging by Amazon CloudWatch.

- Choose the **Table mappings** tab, shown following, to set values for schema mapping and the mapping method. If you choose **Custom**, you can specify the target schema and table values. For more information about table mapping, see [Using Table Mapping to Specify Task Settings \(p. 245\)](#).



- After you have finished with the task settings, choose **Create task**.

Specifying Task Settings for AWS Database Migration Service Tasks

Each task has settings that you can configure according to the needs of your database migration. You create these settings in a JSON file or, with some settings, you can specify the settings using the AWS DMS console.

There are several main types of task settings, as listed following.

Topics

- [Target Metadata Task Settings \(p. 227\)](#)
- [Full Load Task Settings \(p. 228\)](#)
- [Logging Task Settings \(p. 228\)](#)
- [Parallel Loading of Tables \(p. 229\)](#)
- [Control Table Task Settings \(p. 230\)](#)
- [Stream Buffer Task Settings \(p. 232\)](#)
- [Change Processing Tuning Settings \(p. 233\)](#)
- [Data Validation Task Settings \(p. 234\)](#)
- [Task Settings for Change Processing DDL Handling \(p. 234\)](#)
- [Error Handling Task Settings \(p. 234\)](#)
- [Saving Task Settings \(p. 237\)](#)

Task Settings	Relevant Documentation
<p>Creating a Task Assessment Report</p> <p>You can create a task assessment report that shows any unsupported data types that could cause problems during migration. You can run this report on your task before running the task to find out potential issues.</p>	<p>Creating a Task Assessment Report (p. 215)</p>
<p>Creating a Task</p> <p>When you create a task, you specify the source, target, and replication instance, along with any migration settings.</p>	<p>Creating a Task (p. 218)</p>
<p>Creating an Ongoing Replication Task</p> <p>You can set up a task to provide continuous replication between the source and target.</p>	<p>Creating Tasks for Ongoing Replication Using AWS DMS (p. 239)</p>
<p>Applying Task Settings</p> <p>Each task has settings that you can configure according to the needs of your database migration. You create these settings in a JSON file or, with some settings, you can specify the settings using the AWS DMS console.</p>	<p>Specifying Task Settings for AWS Database Migration Service Tasks (p. 224)</p>

Task Settings	Relevant Documentation
<p>Data Validation</p> <p>Data validation is a task setting you can use to have AWS DMS compare the data on your target data store with the data from your source data store.</p>	<p>Validating AWS DMS Tasks (p. 272).</p>
<p>Modifying a Task</p> <p>When a task is stopped, you can modify the settings for the task.</p>	<p>Modifying a Task (p. 242)</p>
<p>Reloading Tables During a Task</p> <p>You can reload a table during a task if an error occurs during the task.</p>	<p>Reloading Tables During a Task (p. 242)</p>
<p>Using Table Mapping</p> <p>Table mapping uses several types of rules to specify task settings for the data source, source schema, data, and any transformations that should occur during the task.</p>	<p>Selection Rules Selection Rules and Actions (p. 250)</p> <p>Transformation Rules Transformation Rules and Actions (p. 252)</p>
<p>Applying Filters</p> <p>You can use source filters to limit the number and type of records transferred from your source to your target. For example, you can specify that only employees with a location of headquarters are moved to the target database. You apply filters on a column of data.</p>	<p>Using Source Filters (p. 257)</p>
<p>Monitoring a Task</p> <p>There are several ways to get information on the performance of a task and the tables used by the task.</p>	<p>Monitoring AWS DMS Tasks (p. 261)</p>
<p>Managing Task Logs</p> <p>You can view and delete task logs using the AWS DMS API or AWS CLI.</p>	<p>Managing AWS DMS Task Logs (p. 267)</p>

A task settings JSON file can look like this:

```

{
  "TargetMetadata": {
    "TargetSchema": "",
    "SupportLobs": true,
    "FullLobMode": false,
    "LobChunkSize": 64,
    "LimitedSizeLobMode": true,
  }
}
```

```

    "LobMaxSize": 32,
    "BatchApplyEnabled": true
  },
  "FullLoadSettings": {
    "TargetTablePrepMode": "DO_NOTHING",
    "CreatePkAfterFullLoad": false,
    "StopTaskCachedChangesApplied": false,
    "StopTaskCachedChangesNotApplied": false,
    "MaxFullLoadSubTasks": 8,
    "TransactionConsistencyTimeout": 600,
    "CommitRate": 10000
  },
  "Logging": {
    "EnableLogging": false
  },
  "ControlTablesSettings": {
    "ControlSchema": "",
    "HistoryTimeslotInMinutes": 5,
    "HistoryTableEnabled": false,
    "SuspendedTablesTableEnabled": false,
    "StatusTableEnabled": false
  },
  "StreamBufferSettings": {
    "StreamBufferCount": 3,
    "StreamBufferSizeInMB": 8
  },
  "ChangeProcessingTuning": {
    "BatchApplyPreserveTransaction": true,
    "BatchApplyTimeoutMin": 1,
    "BatchApplyTimeoutMax": 30,
    "BatchApplyMemoryLimit": 500,
    "BatchSplitSize": 0,
    "MinTransactionSize": 1000,
    "CommitTimeout": 1,
    "MemoryLimitTotal": 1024,
    "MemoryKeepTime": 60,
    "StatementCacheSize": 50
  },
  "ChangeProcessingDdlHandlingPolicy": {
    "HandleSourceTableDropped": true,
    "HandleSourceTableTruncated": true,
    "HandleSourceTableAltered": true
  },
  "ValidationSettings": {
    "EnableValidation": false,
    "ThreadCount": 5
  },
  "ErrorBehavior": {
    "DataErrorPolicy": "LOG_ERROR",
    "DataTruncationErrorPolicy": "LOG_ERROR",
    "DataErrorEscalationPolicy": "SUSPEND_TABLE",
    "DataErrorEscalationCount": 50,
    "TableErrorPolicy": "SUSPEND_TABLE",
    "TableErrorEscalationPolicy": "STOP_TASK",
    "TableErrorEscalationCount": 50,
    "RecoverableErrorCount": 0,
    "RecoverableErrorInterval": 5,
    "RecoverableErrorThrottling": true,
    "RecoverableErrorThrottlingMax": 1800,
    "ApplyErrorDeletePolicy": "IGNORE_RECORD",
    "ApplyErrorInsertPolicy": "LOG_ERROR",
    "ApplyErrorUpdatePolicy": "LOG_ERROR",
    "ApplyErrorEscalationPolicy": "LOG_ERROR",
    "ApplyErrorEscalationCount": 0,
    "FullLoadIgnoreConflicts": true
  }
}

```



```
}
```

Target Metadata Task Settings

Target metadata settings include the following:

- **TargetSchema** – The target table schema name. If this metadata option is empty, the schema from the source table is used. AWS DMS automatically adds the owner prefix for the target database to all tables if no source schema is defined. This option should be left empty for MySQL-type target endpoints.
- **LOB settings** – Settings that determine how large objects (LOBs) are managed. If you set `SupportLobs=true`, you must set one of the following to `true`:
 - **FullLobMode** – If you set this option to `true`, then you must enter a value for the `LobChunkSize` option. Enter the size, in kilobytes, of the LOB chunks to use when replicating the data to the target. The `FullLobMode` option works best for very large LOB sizes but tends to cause slower loading.
 - **InlineLobMaxSize** – This value determines which LOBs AWS Database Migration Service transfers inline during a full load. Transferring small LOBs is more efficient than looking them up from a source table. During a full load, AWS Database Migration Service checks all LOBs and performs an inline transfer for the LOBs that are smaller than `InlineLobMaxSize`. AWS Database Migration Service transfers all LOBs larger than the `InlineLobMaxSize` in `FullLobMode`. The default value for `InlineLobMaxSize` is 0 and the range is 1 kilobyte–2 gigabyte. Set a value for `InlineLobMaxSize` only if you know that most of the LOBs are smaller than the value specified in `InlineLobMaxSize`.
 - **LimitedSizeLobMode** – If you set this option to `true`, then you must enter a value for the `LobMaxSize` option. Enter the maximum size, in kilobytes, for an individual LOB.
- **LoadMaxFileSize** – An option for PostgreSQL and MySQL target endpoints that defines the maximum size on disk of stored, unloaded data, such as CSV files. This option overrides the connection attribute. You can provide values from 0, which indicates that this option doesn't override the connection attribute, to 100,000 KB.
- **BatchApplyEnabled** – Determines if each transaction is applied individually or if changes are committed in batches. The default value is `false`.

The `BatchApplyEnabled` parameter is used with the `BatchApplyPreserveTransaction` parameter. If `BatchApplyEnabled` is set to `true`, then the `BatchApplyPreserveTransaction` parameter determines the transactional integrity.

If `BatchApplyPreserveTransaction` is set to `true`, then transactional integrity is preserved and a batch is guaranteed to contain all the changes within a transaction from the source.

If `BatchApplyPreserveTransaction` is set to `false`, then there can be temporary lapses in transactional integrity to improve performance.

The `BatchApplyPreserveTransaction` parameter applies only to Oracle target endpoints, and is only relevant when the `BatchApplyEnabled` parameter is set to `true`.

When LOB columns are included in the replication, `BatchApplyEnabled` can only be used in **Limited-size LOB mode**.

- **ParallelLoadThreads** – Specifies the number of threads AWS DMS uses to load each table into the target database. The maximum value for a MySQL target is 16; the maximum value for a DynamoDB target is 32. The maximum limit can be increased upon request.
- **ParallelLoadBufferSize** – Specifies the maximum number of records to store in the buffer used by the parallel load threads to load data to the target. The default value is 50. Maximum value is 1000. This field is currently only valid when DynamoDB is the target. This parameter should be used with `ParallelLoadThreads` and is valid only when `ParallelLoadThreads > 1`.

Full Load Task Settings

Full load settings include the following:

- To indicate how to handle loading the target at full-load startup, specify one of the following values for the `TargetTablePrepMode` option:
 - `DO_NOTHING` – Data and metadata of the existing target table are not affected.
 - `DROP_AND_CREATE` – The existing table is dropped and a new table is created in its place.
 - `TRUNCATE_BEFORE_LOAD` – Data is truncated without affecting the table metadata.
- To delay primary key or unique index creation until after full load completes, set the `CreatePkAfterFullLoad` option. When this option is selected, you cannot resume incomplete full load tasks.
- For full load and CDC-enabled tasks, you can set the following `Stop task after full load completes` options:
 - `StopTaskCachedChangesApplied` – Set this option to `true` to stop a task after a full load completes and cached changes are applied.
 - `StopTaskCachedChangesNotApplied` – Set this option to `true` to stop a task before cached changes are applied.
- `MaxFullLoadSubTasks` – Set this option to indicate the maximum number of tables to load in parallel. The default is 8; the maximum value is 50.
- To set the number of seconds that AWS DMS waits for transactions to close before beginning a full-load operation, if transactions are open when the task starts, set the `TransactionConsistencyTimeout` option. The default value is 600 (10 minutes). AWS DMS begins the full load after the timeout value is reached, even if there are open transactions. A full-load-only task doesn't wait for 10 minutes but instead starts immediately.
- To indicate the maximum number of events that can be transferred together, set the `CommitRate` option.

Logging Task Settings

Logging task settings are written to a JSON file and they let you specify which component activities are logged and what amount of information is written to the log. The logging feature uses Amazon CloudWatch to log information during the migration process.

There are several ways to enable Amazon CloudWatch logging. You can select the `EnableLogging` option on the AWS Management Console when you create a migration task or set the `EnableLogging` option to `true` when creating a task using the AWS DMS API. You can also specify `"EnableLogging": true` in the JSON of the logging section of task settings.

To delete the task logs, you can specify `"DeleteTaskLogs": true` in the JSON of the logging section of task settings.

You can specify logging for the following component activities:

- `SOURCE_UNLOAD` – Data is unloaded from the source database.
- `SOURCE_CAPTURE` – Data is captured from the source database.
- `TARGET_LOAD` – Data is loaded into the target database.
- `TARGET_APPLY` – Data and data definition language (DDL) statements are applied to the target database.
- `TASK_MANAGER` – The task manager triggers an event.

After you specify a component activity, you can then specify the amount of information that is logged. The following list is in order from the lowest level of information to the highest level of information. The higher levels always include information from the lower levels. These severity values include:

- `LOGGER_SEVERITY_ERROR` – Error messages are written to the log.
- `LOGGER_SEVERITY_WARNING` – Warnings and error messages are written to the log.
- `LOGGER_SEVERITY_INFO` – Informational messages, warnings, and error messages are written to the log.
- `LOGGER_SEVERITY_DEFAULT` – Informational messages, warnings, and error messages are written to the log.
- `LOGGER_SEVERITY_DEBUG` – Debug messages, informational messages, warnings, and error messages are written to the log.
- `LOGGER_SEVERITY_DETAILED_DEBUG` – All information is written to the log.

For example, the following JSON section gives task settings for logging for all component activities.

```
...
  "Logging": {
    "EnableLogging": true,
    "LogComponents": [{
      "Id": "SOURCE_UNLOAD",
      "Severity": "LOGGER_SEVERITY_DEFAULT"
    }, {
      "Id": "SOURCE_CAPTURE",
      "Severity": "LOGGER_SEVERITY_DEFAULT"
    }, {
      "Id": "TARGET_LOAD",
      "Severity": "LOGGER_SEVERITY_DEFAULT"
    }, {
      "Id": "TARGET_APPLY",
      "Severity": "LOGGER_SEVERITY_INFO"
    }, {
      "Id": "TASK_MANAGER",
      "Severity": "LOGGER_SEVERITY_DEBUG"
    }
  ]
},
...
```

Parallel Loading of Tables

AWS DMS can logically split a full-load task into subtasks to load a table using several threads in parallel. You can use this parallel process to have multiple threads load tables and partitioned tables, and then migrate the tables to the target endpoint. You can split the tables by primary key values or, with some database engines, by partition or subpartition.

To use parallel loading, you create a rule of type `table-settings` with the `parallel-load` option. Within the `table-settings` rule, you specify the selection criteria for the table or tables that you want to load in parallel. To specify the selection criteria, set the `type` element for `parallel-load` to one of the following:

- `partitions-auto`
- `subpartitions-auto`
- `none`

The following example illustrates how to create a `table-settings` rule to load table partitions in parallel.

```
{
  "rules": [{
    "rule-type": "table-settings",
    "rule-id": "1",
    "rule-name": "1",
    "object-locator": {
      "schema-name": "test",
      "table-name": "table1"
    },
    "parallel-load": {
      "type": "partitions-auto"
    }
  }]
}
```

Control Table Task Settings

Control tables provide information about the AWS DMS task, as well as useful statistics that you can use to plan and manage both the current migration task and future tasks. You can apply these task settings in a JSON file or using the **Advanced Settings** link on the **Create task** page in the AWS DMS console. In addition to the **Apply Exceptions (dmslogs.awsdms_apply_exceptions)** table, which is always created, you can choose to create additional tables including the following:

- **Replication Status (dmslogs.awsdms_status)** – This table provides details about the current task. These include task status, amount of memory consumed by the task, and the number of changes not yet applied to the target. This table also gives the position in the source database where AWS DMS is currently reading and indicates if the task is a full load or change data capture (CDC).
- **Suspended Tables (dmslogs.awsdms_suspended_tables)** – This table provides a list of suspended tables as well as the reason they were suspended.
- **Replication History (dmslogs.awsdms_history)** – This table provides information about replication history. This information includes the number and volume of records processed during the task, latency at the end of a CDC task, and other statistics.

The **Apply Exceptions (dmslogs.awsdms_apply_exceptions)** table contains the following parameters:

Column	Type	Description
TASK_NAME	nvchar	The name of the AWS DMS task.
TABLE_OWNER	nvchar	The table owner.
TABLE_NAME	nvchar	The table name.
ERROR_TIME	timestamp	The time the exception (error) occurred.
STATEMENT	nvchar	The statement that was being run when the error occurred.
ERROR	nvchar	The error name and description.

The **Replication History (dmslogs.awsdms_history)** table contains the following parameters:

Column	Type	Description
SERVER_NAME	nvchar	The name of the machine where the replication task is running.
TASK_NAME	nvchar	The name of the AWS DMS task.
TIMESLOT_TYPE	vvarchar	One of the following values: <ul style="list-style-type: none"> FULL LOAD CHANGE PROCESSING (CDC) <p>If the task is running both full load and CDC, two history records are written to the time slot.</p>
TIMESLOT	timestamp	The ending timestamp of the time slot.
TIMESLOT_DURATION	int	The duration of the time slot.
TIMESLOT_LATENCY	int	The target latency at the end of the time slot. This value is only applicable to CDC time slots.
RECORDS	int	The number of records processed during the time slot.
TIMESLOT_VOLUME	int	The volume of data processed in MB.

The **Replication Status (dmslogs.aws_dms_status)** table contains the current status of the task and the target database. It has the following settings:

Column	Type	Description
SERVER_NAME	nvchar	The name of the machine where the replication task is running.
TASK_NAME	nvchar	The name of the AWS DMS task.
TASK_STATUS	vvarchar	One of the following values: <ul style="list-style-type: none"> FULL LOAD CHANGE PROCESSING (CDC) <p>Task status is set to FULL LOAD as long as there is at least one table in full load. After all tables have been loaded, the task status changes to CHANGE PROCESSING if CDC is enabled.</p>
STATUS_TIME	timestamp	The timestamp of the task status.
PENDING_CHANGES	int	The number of change records that were not applied to the target.

Column	Type	Description
DISK_SWAP_SIZE	int	The amount of disk space used by old or offloaded transactions.
TASK_MEMORY	int	Current memory used, in MB.
SOURCE_CURRENT_POSITION	varchar	The position in the source database that AWS DMS is currently reading from.
SOURCE_CURRENT_TIMESTAMP	timestamp	The timestamp in the source database that AWS DMS is currently reading from.
SOURCE_TAIL_POSITION	varchar	The position of the oldest start transaction that is not committed. This value is the newest position that you can revert to without losing any changes.
SOURCE_TAIL_TIMESTAMP	timestamp	The timestamp of the oldest start transaction that is not committed. This value is the newest timestamp that you can revert to without losing any changes.
SOURCE_TIMESTAMP_APPLIED	timestamp	The timestamp of the last transaction commit. In a bulk apply process, this value is the timestamp for the commit of the last transaction in the batch.

Additional control table settings include the following:

- `ControlSchema` – Use this option to indicate the database schema name for the AWS DMS target Control Tables. If you do not enter any information in this field, then the tables are copied to the default location in the database.
- `HistoryTimeslotInMinutes` – Use this option to indicate the length of each time slot in the Replication History table. The default is 5 minutes.

Stream Buffer Task Settings

You can set stream buffer settings using the AWS CLI, include the following:

- `StreamBufferCount` – Use this option to specify the number of data stream buffers for the migration task. The default stream buffer number is 3. Increasing the value of this setting might increase the speed of data extraction. However, this performance increase is highly dependent on the migration environment, including the source system and instance class of the replication server. The default is sufficient for most situations.
- `StreamBufferSizeInMB` – Use this option to indicate the maximum size of each data stream buffer. The default size is 8 MB. You might need to increase the value for this option when you work with very large LOBs. You also might need to increase the value if you receive a message in the log files that the stream buffer size is insufficient. When calculating the size of this option, you can use the following equation: $[\text{Max LOB size (or LOB chunk size)}] * [\text{number of LOB}]$

`columns]*[number of stream buffers]*[number of tables loading in parallel per task(MaxFullLoadSubTasks)]*3`

- `CtrlStreamBufferSizeInMB` – Use this option to set the size of the control stream buffer. Value is in megabytes, and can be 1–8. The default value is 5. You might need to increase this when working with a very large number of tables, such as tens of thousands of tables.

Change Processing Tuning Settings

The following settings determine how AWS DMS handles changes for target tables during change data capture (CDC). Several of these settings depend on the value of the target metadata parameter `BatchApplyEnabled`. For more information on the `BatchApplyEnabled` parameter, see [Target Metadata Task Settings \(p. 227\)](#).

Change processing tuning settings include the following:

The following settings apply only when the target metadata parameter `BatchApplyEnabled` is set to `true`.

- `BatchApplyPreserveTransaction` – If set to `true`, transactional integrity is preserved and a batch is guaranteed to contain all the changes within a transaction from the source. The default value is `true`. This setting applies only to Oracle target endpoints.

If set to `false`, there can be temporary lapses in transactional integrity to improve performance. There is no guarantee that all the changes within a transaction from the source are applied to the target in a single batch.

- `BatchApplyTimeoutMin` – Sets the minimum amount of time in seconds that AWS DMS waits between each application of batch changes. The default value is 1.
- `BatchApplyTimeoutMax` – Sets the maximum amount of time in seconds that AWS DMS waits between each application of batch changes before timing out. The default value is 30.
- `BatchApplyMemoryLimit` – Sets the maximum amount of memory in (MB) to use for pre-processing in **Batch optimized apply mode**. The default value is 500.
- `BatchSplitSize` – Sets the maximum number of changes applied in a single batch. The default value 0, meaning there is no limit applied.

The following settings apply only when the target metadata parameter `BatchApplyEnabled` is set to `false`.

- `MinTransactionSize` – Sets the minimum number of changes to include in each transaction. The default value is 1000.
- `CommitTimeout` – Sets the maximum time in seconds for AWS DMS to collect transactions in batches before declaring a timeout. The default value is 1.
- `HandleSourceTableAltered` – Set this option to `true` to alter the target table when the source table is altered.

AWS DMS attempts to keep transaction data in memory until the transaction is fully committed to the source and/or the target. However, transactions that are larger than the allocated memory or that are not committed within the specified time limit are written to disk.

The following settings apply to change processing tuning regardless of the change processing mode.

- `MemoryLimitTotal` – Sets the maximum size (in MB) that all transactions can occupy in memory before being written to disk. The default value is 1024.

- `MemoryKeepTime` – Sets the maximum time in seconds that each transaction can stay in memory before being written to disk. The duration is calculated from the time that AWS DMS started capturing the transaction. The default value is 60.
- `StatementCacheSize` – Sets the maximum number of prepared statements to store on the server for later execution when applying changes to the target. The default value is 50. The maximum value is 200.

Data Validation Task Settings

You can ensure that your data was migrated accurately from the source to the target. If you enable it for a task, then AWS DMS begins comparing the source and target data immediately after a full load is performed for a table. For more information on data validation, see [Validating AWS DMS Tasks \(p. 272\)](#).

Data validation settings include the following:

- To enable data validation, set the `EnableValidation` setting to `true`.
- To adjust the number of execution threads that AWS DMS uses during validation, set the `ThreadCount` value. The default value for `ThreadCount` is 5. If you set `ThreadCount` to a higher number, AWS DMS can complete the validation faster. However, AWS DMS then also executes more simultaneous queries, consuming more resources on the source and the target.

For example, the following JSON enables data validation.

```
"ValidationSettings": {  
  "EnableValidation": true,  
  "ThreadCount": 5  
}
```

For an Oracle endpoint, AWS DMS uses `DBMS_CRYPTO` to validate BLOBs. If your Oracle endpoint uses BLOBs, then you must grant the `execute` permission on `dbms_crypto` to the user account that is used to access the Oracle endpoint. You can do this by running the following statement.

```
grant execute on sys.dbms_crypto to <dms_endpoint_user>;
```

Task Settings for Change Processing DDL Handling

The following settings determine how AWS DMS handles data definition language (DDL) changes for target tables during change data capture (CDC). Task settings for change processing DDL handling include the following:

- `HandleSourceTableDropped` – Set this option to `true` to drop the target table when the source table is dropped
- `HandleSourceTableTruncated` – Set this option to `true` to truncate the target table when the source table is truncated
- `HandleSourceTableAltered` – Set this option to `true` to alter the target table when the source table is altered.

Error Handling Task Settings

You can set the error handling behavior of your replication task during change data capture (CDC) using the following settings:

- **DataErrorPolicy** – Determines the action AWS DMS takes when there is an error related to data processing at the record level. Some examples of data processing errors include conversion errors, errors in transformation, and bad data. The default is `LOG_ERROR`.
 - `IGNORE_RECORD` – The task continues and the data for that record is ignored. The error counter for the `DataErrorEscalationCount` property is incremented. Thus, if you set a limit on errors for a table, this error counts toward that limit.
 - `LOG_ERROR` – The task continues and the error is written to the task log.
 - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
 - `STOP_TASK` – The task stops and manual intervention is required.
- **DataTruncationErrorPolicy** – Determines the action AWS DMS takes when data is truncated. The default is `LOG_ERROR`.
 - `IGNORE_RECORD` – The task continues and the data for that record is ignored. The error counter for the `DataErrorEscalationCount` property is incremented. Thus, if you set a limit on errors for a table, this error counts toward that limit.
 - `LOG_ERROR` – The task continues and the error is written to the task log.
 - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
 - `STOP_TASK` – The task stops and manual intervention is required.
- **DataErrorEscalationPolicy** – Determines the action AWS DMS takes when the maximum number of errors (set in the `DataErrorsEscalationCount` parameter) is reached. The default is `SUSPEND_TABLE`.
 - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
 - `STOP_TASK` – The task stops and manual intervention is required.
- **DataErrorEscalationCount** – Sets the maximum number of errors that can occur to the data for a specific record. When this number is reached, the data for the table that contains the error record is handled according to the policy set in the `DataErrorEscalationCount`. The default is 0.
- **TableErrorPolicy** – Determines the action AWS DMS takes when an error occurs when processing data or metadata for a specific table. This error only applies to general table data and is not an error that relates to a specific record. The default is `SUSPEND_TABLE`.
 - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
 - `STOP_TASK` – The task stops and manual intervention is required.
- **TableErrorEscalationPolicy** – Determines the action AWS DMS takes when the maximum number of errors (set using the `TableErrorEscalationCount` parameter). The default and only user setting is `STOP_TASK`, where the task is stopped and manual intervention is required.
- **TableErrorEscalationCount** – The maximum number of errors that can occur to the general data or metadata for a specific table. When this number is reached, the data for the table is handled according to the policy set in the `TableErrorEscalationPolicy`. The default is 0.
- **RecoverableErrorCount** – The maximum number of attempts made to restart a task when an environmental error occurs. After the system attempts to restart the task the designated number of times, the task is stopped and manual intervention is required. The default value is -1, which instructs AWS DMS to attempt to restart the task indefinitely. Set this value to 0 to never attempt to restart a task. If a fatal error occurs, AWS DMS stops attempting to restart the task after six attempts.
- **RecoverableErrorInterval** – The number of seconds that AWS DMS waits between attempts to restart a task. The default is 5.
- **RecoverableErrorThrottling** – When enabled, the interval between attempts to restart a task is increased each time a restart is attempted. The default is `true`.

- `RecoverableErrorThrottlingMax` – The maximum number of seconds that AWS DMS waits between attempts to restart a task if `RecoverableErrorThrottling` is enabled. The default is 1800.
- `ApplyErrorDeletePolicy` – Determines what action AWS DMS takes when there is a conflict with a DELETE operation. The default is `IGNORE_RECORD`.
 - `IGNORE_RECORD` – The task continues and the data for that record is ignored. The error counter for the `ApplyErrorEscalationCount` property is incremented. Thus, if you set a limit on errors for a table, this error counts toward that limit.
 - `LOG_ERROR` – The task continues and the error is written to the task log.
 - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
 - `STOP_TASK` – The task stops and manual intervention is required.
- `ApplyErrorInsertPolicy` – Determines what action AWS DMS takes when there is a conflict with an INSERT operation. The default is `LOG_ERROR`.
 - `IGNORE_RECORD` – The task continues and the data for that record is ignored. The error counter for the `ApplyErrorEscalationCount` property is incremented. Thus, if you set a limit on errors for a table, this error counts toward that limit.
 - `LOG_ERROR` – The task continues and the error is written to the task log.
 - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
 - `STOP_TASK` – The task stops and manual intervention is required.
 - `INSERT_RECORD` – If there is an existing target record with the same primary key as the inserted source record, the target record is updated.
- `ApplyErrorUpdatePolicy` – Determines what action AWS DMS takes when there is a conflict with an UPDATE operation. The default is `LOG_ERROR`.
 - `IGNORE_RECORD` – The task continues and the data for that record is ignored. The error counter for the `ApplyErrorEscalationCount` property is incremented. Thus, if you set a limit on errors for a table, this error counts toward that limit.
 - `LOG_ERROR` – The task continues and the error is written to the task log.
 - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
 - `STOP_TASK` – The task stops and manual intervention is required.
 - `UPDATE_RECORD` – If the target record is missing, the missing target record is inserted into the target table. Selecting this option requires full supplemental logging to be enabled for all the source table columns when Oracle is the source database.
- `ApplyErrorEscalationPolicy` – Determines what action AWS DMS takes when the maximum number of errors (set using the `ApplyErrorsEscalationCount` parameter) is reached.
 - `LOG_ERROR` – The task continues and the error is written to the task log.
 - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
 - `STOP_TASK` – The task stops and manual intervention is required.
- `ApplyErrorEscalationCount` – Sets the maximum number of APPLY conflicts that can occur for a specific table during a change process operation. When this number is reached, the data for the table is handled according to the policy set in the `ApplyErrorEscalationPolicy` parameter. The default is 0.
- `ApplyErrorFailOnTruncationDdl` – Set this to `true` to cause the task to fail when a truncation is performed on any of the tracked tables during CDC. The failure message is: "Truncation DDL detected." The default is `false`.

This approach doesn't work with PostgreSQL or any other source endpoint that doesn't replicate DDL table truncation.

- `FailOnNoTablesCaptured` – Set this to `true` to cause a task to fail when the transformation rules defined for a task find no tables when the task starts. The default is `false`.
- `FailOnTransactionConsistencyBreached` – This option applies to tasks using Oracle as a source with CDC. Set this to `true` to cause a task to fail when a transaction is open for more time than the specified timeout and could be dropped.

When a CDC task starts with Oracle, AWS DMS waits for a limited time for the oldest open transaction to close before starting CDC. If the oldest open transaction doesn't close until the timeout is reached, then we normally start CDC anyway, ignoring that transaction. If this setting is set to `true`, the task fails.

- `FullLoadIgnoreConflicts` – Set this to `false` to have AWS DMS ignore "zero rows affected" and "duplicates" errors when applying cached events. If set to `true`, AWS DMS reports all errors instead of ignoring them. The default is `false`.

Saving Task Settings

You can save the settings for a task as a JSON file, in case you want to reuse the settings for another task.

For example, the following JSON file contains settings saved for a task.

```
{
  "TargetMetadata": {
    "TargetSchema": "",
    "SupportLobs": true,
    "FullLobMode": false,
    "LobChunkSize": 64,
    "LimitedSizeLobMode": true,
    "LobMaxSize": 32,
    "BatchApplyEnabled": true
  },
  "FullLoadSettings": {
    "TargetTablePrepMode": "DO_NOTHING",
    "CreatePkAfterFullLoad": false,
    "StopTaskCachedChangesApplied": false,
    "StopTaskCachedChangesNotApplied": false,
    "MaxFullLoadSubTasks": 8,
    "TransactionConsistencyTimeout": 600,
    "CommitRate": 10000
  },
  "Logging": {
    "EnableLogging": false
  },
  "ControlTablesSettings": {
    "ControlSchema": "",
    "HistoryTimeslotInMinutes": 5,
    "HistoryTableEnabled": false,
    "SuspendedTablesTableEnabled": false,
    "StatusTableEnabled": false
  },
  "StreamBufferSettings": {
    "StreamBufferCount": 3,
    "StreamBufferSizeInMB": 8
  },
  "ChangeProcessingTuning": {
    "BatchApplyPreserveTransaction": true,
    "BatchApplyTimeoutMin": 1,
    "BatchApplyTimeoutMax": 30,
    "BatchApplyMemoryLimit": 500,
    "BatchSplitSize": 0,
  }
}
```

```

    "MinTransactionSize": 1000,
    "CommitTimeout": 1,
    "MemoryLimitTotal": 1024,
    "MemoryKeepTime": 60,
    "StatementCacheSize": 50
  },
  "ChangeProcessingDdlHandlingPolicy": {
    "HandleSourceTableDropped": true,
    "HandleSourceTableTruncated": true,
    "HandleSourceTableAltered": true
  },
  "ErrorBehavior": {
    "DataErrorPolicy": "LOG_ERROR",
    "DataTruncationErrorPolicy": "LOG_ERROR",
    "DataErrorEscalationPolicy": "SUSPEND_TABLE",
    "DataErrorEscalationCount": 50,
    "TableErrorPolicy": "SUSPEND_TABLE",
    "TableErrorEscalationPolicy": "STOP_TASK",
    "TableErrorEscalationCount": 50,
    "RecoverableErrorCount": 0,
    "RecoverableErrorInterval": 5,
    "RecoverableErrorThrottling": true,
    "RecoverableErrorThrottlingMax": 1800,
    "ApplyErrorDeletePolicy": "IGNORE_RECORD",
    "ApplyErrorInsertPolicy": "LOG_ERROR",
    "ApplyErrorUpdatePolicy": "LOG_ERROR",
    "ApplyErrorEscalationPolicy": "LOG_ERROR",
    "ApplyErrorEscalationCount": 0,
    "FullLoadIgnoreConflicts": true
  }
}

```

Setting LOB Support for Source Databases in a AWS DMS Task

Large binary 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.

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.

If you decide to include LOBs, you can then decide the other LOB settings:

- The LOB mode determines how LOBs are handled:
 - **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** – 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.

Note

With Oracle, LOBs are treated as VARCHAR data types whenever possible. This approach means that 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 64 K. Therefore, a limited LOB size of less than 64 K is optimal when Oracle is your source database.

- When a task is configured to run in **limited LOB mode**, the **Max LOB size (K)** option sets the maximum size LOB that AWS DMS accepts. Any LOBs that are larger than this value is truncated to this value.
- When a task is configured to use **full LOB mode**, AWS DMS retrieves LOBs in pieces. The **LOB chunk size (K)** 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.

Creating Multiple Tasks

In some migration scenarios, you might have to create several migration tasks. Tasks work independently and can run concurrently. Each task has its own initial load, CDC, and log reading process. Tables that are related through data manipulation language (DML) must be part of the same task.

Some reasons to create multiple tasks for a migration include the following:

- The target tables for the tasks reside on different databases, such as when you are fanning out or breaking a system into multiple systems.
- You want to break the migration of a large table into multiple tasks by using filtering.

Note

Because each task has its own change capture and log reading process, changes are *not* coordinated across tasks. Therefore, when using multiple tasks to perform a migration, make sure that source transactions are wholly contained within a single task.

Creating Tasks for Ongoing Replication Using AWS DMS

You can create an AWS DMS task that captures ongoing changes to the source data store. You can do this capture while you are migrating your data. You can also create a task that captures ongoing changes after you complete your initial migration to a supported target data store. This process is called ongoing replication or change data capture (CDC). AWS DMS uses this process when replicating ongoing changes from a source data store. This process works by collecting changes to the database logs using the database engine's native API.

Each source engine has specific configuration requirements for exposing this change stream to a given user account. Most engines require some additional configuration to make it possible for the capture process to consume the change data in a meaningful way, without data loss. For example, Oracle requires the addition of supplemental logging, and MySQL requires row-level binary logging (bin logging).

To read ongoing changes from the source database, AWS DMS uses engine-specific API actions to read changes from the source engine's transaction logs. Following are some examples of how AWS DMS does that:

- For Oracle, AWS DMS uses either the Oracle LogMiner API or binary reader API (bfile API) to read ongoing changes. AWS DMS reads ongoing changes from the online or archive redo logs based on the system change number (SCN).
- For Microsoft SQL Server, AWS DMS uses MS-Replication or MS-CDC to write information to the SQL Server transaction log. It then uses the `fn_dblog()` or `fn_dump_dblog()` function in SQL Server to read the changes in the transaction log based on the log sequence number (LSN).

- For MySQL, AWS DMS reads changes from the row-based binary logs (binlogs) and migrates those changes to the target.
- For PostgreSQL, AWS DMS sets up logical replication slots and uses the `test_decoding` plugin to read changes from the source and migrate them to the target.
- For Amazon RDS as a source, we recommend ensuring that backups are enabled to setup CDC. We also recommend ensuring that the source database is configured to retain change logs for a sufficient time—24 hours is usually enough.

There are two types of ongoing replication tasks:

- Full load plus CDC – The task migrates existing data and then updates the target database based on changes to the source database.
- CDC only – The task migrates ongoing changes after you have data on your target database.

Performing Replication Starting from a CDC Start Point

You can start an AWS DMS ongoing replication task (change data capture only) from several points. These include the following:

- **From a custom CDC start time** – You can use the AWS Management Console or AWS CLI to provide AWS DMS with a timestamp where you want the replication to start. AWS DMS then starts an ongoing replication task from this custom CDC start time. AWS DMS converts the given timestamp (in UTC) to a native start point, such as an LSN for SQL Server or an SCN for Oracle. AWS DMS uses engine-specific methods to determine where exactly to start the migration task based on the source engine's change stream.

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.

- **From a CDC native start point** – You can also start from a native point in the source engine's transaction log. In some cases, you might prefer this approach because a timestamp can indicate multiple native points in the transaction log. AWS DMS supports this feature for the following source endpoints:
 - SQL Server
 - Oracle
 - MySQL

Determining a CDC Native Start Point

A *CDC native start point* is a point in time. As an example, suppose that a bulk data dump has been applied to the target starting from a point in time. In this case, you can look up the native start point for the ongoing replication-only task from a point before the dump was taken.

Following are examples of how you can find the CDC native start point from a supported source engine:

SQL Server

In SQL Server, a log sequence number (LSN) has three parts:

- Virtual log file (VLF) sequence number
- Starting offset of a log block

- Slot number

An example LSN is as follows: 00000014:00000061:0001

To get the start point for a SQL Server migration task based on your transaction log backup settings, use the `fn_dblog()` or `fn_dump_dblog()` function in SQL Server.

To use CDC native start point with SQL Server, create a publication on any table participating in ongoing replication. For more information about creating a publication, see [Creating a SQL Server Publication for Ongoing Replication \(p. 103\)](#). AWS DMS creates the publication automatically when you use CDC without using a CDC native start point.

Oracle

A system change number (SCN) is a logical, internal time stamp used by Oracle databases. SCNs order events that occur within the database, which is necessary to satisfy the ACID properties of a transaction. Oracle databases use SCNs to mark the location where all changes have been written to disk so that a recovery action doesn't apply already written changes. Oracle also uses SCNs to mark the point where no redo exists for a set of data so that recovery can stop. For more information about Oracle SCNs, see the [Oracle documentation](#).

To get the current SCN in an Oracle database, run the following command:

```
SELECT current_scn FROM V$DATABASE
```

MySQL

Before the release of MySQL version 5.6.3, the log sequence number (LSN) for MySQL was a 4-byte unsigned integer. In MySQL version 5.6.3, when the redo log file size limit increased from 4 GB to 512 GB, the LSN became an 8-byte unsigned integer. The increase reflects that additional bytes were required to store extra size information. Applications built on MySQL 5.6.3 or later that use LSN values should use 64-bit rather than 32-bit variables to store and compare LSN values. For more information about MySQL LSNs, see the [MySQL documentation](#).

To get the current LSN in a MySQL database, run the following command:

```
mysql> show master status;
```

The query returns a binlog file name, the position, and several other values. The CDC native start point is a combination of the binlogs file name and the position, for example `mysql-bin-changelog.000024:373`. In this example, `mysql-bin-changelog.000024` is the binlogs file name and `373` is the position where AWS DMS needs to start capturing changes.

Using a Checkpoint as a CDC Start Point

An ongoing replication task migrates changes, and AWS DMS caches checkpoint information specific to AWS DMS from time to time. The checkpoint that AWS DMS creates contains information so the replication engine knows the recovery point for the change stream. You can use the checkpoint to go back in the timeline of changes and recover a failed migration task. You can also use a checkpoint to start another ongoing replication task for another target at any given point in time.

You can get the checkpoint information in one of the following two ways:

- Run the API command `DescribeReplicationTasks` and view the results. You can filter the information by task and search for the checkpoint. You can retrieve the latest checkpoint when the task is in stopped or failed state.
- View the metadata table named `awsdms_txn_state` on the target instance. You can query the table to get checkpoint information. To create the metadata table, set the `TaskRecoveryTableEnabled`

parameter to `Yes` when you create a task. This setting causes AWS DMS to continuously write checkpoint information to the target metadata table. This information is lost if a task is deleted.

For example, the following is a sample checkpoint in the metadata table:
`checkpoint:V1#34#00000132/0F000E48#0#0##0#121`

Stopping a Task at a Commit or Server Time Point

With the introduction of CDC native start points, AWS DMS can also stop a task at the following points:

- A commit time on the source
- A server time on the replication instance

You can modify a task and set a time in UTC to stop as required. The task automatically stops based on the commit or server time that you set. Additionally, if you know an appropriate time to stop the migration task at task creation, you can set a stop time when you create the task.

Modifying a Task

You can modify a task if you need to change the task settings, table mapping, or other settings. You modify a task in the DMS console by selecting the task and choosing **Modify**. You can also use the AWS CLI or AWS DMS API command `ModifyReplicationTask`.

There are a few limitations to modifying a task. These include:

- You cannot modify the source or target endpoint of a task.
- You cannot change the migration type of a task.
- A task that have been run must have a status of **Stopped** or **Failed** to be modified.

Reloading Tables During a Task

While a task is running, you can reload a target database table using data from the source. You might want to reload a table if, during the task, an error occurs or data changes due to partition operations (for example, when using Oracle). You can reload up to 10 tables from a task.

To reload a table, the following conditions must apply:

- The task must be running.
- The migration method for the task must be either Full Load or Full Load with CDC.
- Duplicate tables aren't allowed.
- AWS DMS retains the previously read table definition and doesn't recreate it during the reload operation. Any DDL statements like `ALTER TABLE ADD COLUMN`, `DROP COLUMN` made to the table before the table is reloaded can cause the reload operation to fail.

AWS Management Console

To reload a table using the AWS DMS console

1. Sign in to the AWS Management Console and choose AWS DMS. 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 Needed to Use AWS DMS \(p. 31\)](#).

2. Choose **Tasks** from the navigation pane.
3. Choose the running task that has the table you want to reload.
4. Choose the **Table Statistics** tab.

The screenshot shows the AWS Database Migration Service console interface. At the top, there are buttons for 'Create task', 'Assess', 'Modify', 'Start/Resume', and 'Stop'. Below these is a search filter box labeled 'Filter: Q Filter'. A table lists tasks, with one task named 'move-data' in a 'Running' state, sourced from 'from-mysql-sou'. Below this, the 'move-data' task details are shown, with tabs for 'Overview', 'Task monitoring', 'Table statistics', 'Logs', and 'Alerts'. The 'Table statistics' tab is active, and the 'Reload table data' button is circled in red. Below the buttons is another search filter box. A table lists table statistics for the 'employees' schema, showing columns for Schema, Table, Load State, Inserts, and Deleted Rows.

Schema	Table	Load State	Inserts	Deleted Rows
employees	departments	Table completed	0	0
employees	dept_emp	Table completed	0	0
employees	dept_manager	Table completed	0	0

5. Choose the table you want to reload. If the task is no longer running, you can't reload the table.
6. Choose **Reload table data**.

When AWS DMS is preparing to reload a table, the console changes the table status to **Table is being reloaded**.

Using Table Mapping to Specify Task Settings

Table mapping uses several types of rules to specify the data source, source schema, data, and any transformations that should occur during the task. You can use table mapping to specify individual tables in a database to migrate and the schema to use for the migration. In addition, you can use filters to specify what data from a given table column you want replicated. You can use transformations to modify the data written to the target database.

Specifying Table Selection and Transformations by Table Mapping from the Console

You can use the AWS Management Console to perform table mapping, including specifying table selection and transformations. On the console, use the **Where** section to specify the schema, table, and action (include or exclude). Use the **Filter** section to specify the column name in a table and the conditions that you want to apply to a replication task. Together, these two actions create a selection rule.

You can include transformations in a table mapping after you have specified at least one selection rule. You can use transformations to rename a schema or table, add a prefix or suffix to a schema or table, or remove a table column.

The following example shows how to set up selection rules for a table called **Customers** in a schema called **EntertainmentAgencySample**. You create selection rules and transformations on the **Guided** tab. This tab appears only when you have a source endpoint that has schema and table information.

To specify a table selection, filter criteria, and transformations using the AWS console

1. Sign in to the AWS Management Console and choose AWS DMS. 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 Needed to Use AWS DMS \(p. 31\)](#).
2. On the **Dashboard** page, choose **Tasks**.
3. Choose **Create Task**.
4. Enter the task information, including **Task name**, **Replication instance**, **Source endpoint**, **Target endpoint**, and **Migration type**. Choose **Guided** from the **Table mappings** section.

Create task

A task can contain one or more table mappings which define what data is moved from the source to the target. If a table does not exist on the target, it can be created automatically.

Task name* ⓘ

Replication instance* ▼

Source endpoint* ▼

Target endpoint* ▼

Migration type* ▼ ⓘ

Start task on create

▶ Task Settings

▼ Table mappings

Guided **JSON**

5. In the **Table mapping** section, choose the schema name and table name. You can use "%" as a wildcard value when specifying the table name. Specify the action to be taken, to include or exclude data defined by the filter.

Start task on create

▶ Task Settings

▼ Table mappings

Guided JSON

Selection rules ⓘ

At least one selection rule with an include action is required. Once you have one or more selection rules, you can add transformation rules.

Where ⓘ

Schema name is

Table name is like
Use % as a wildcard.

Action ⓘ

Filter ⓘ

[Add column filter](#)

[Add selection rule](#)

6. Specify filter information using the **Add column filter** and the **Add condition** links.
 - a. Choose **Add column filter** to specify a column and conditions.
 - b. Choose **Add condition** to add additional conditions.

The following example shows a filter for the **Customers** table that includes **AgencyIDs** between **01** and **85**.

Selection rules ⓘ

At least one selection rule with an include action is required. Once you have one or more selection rules, you can add transformation rules.

Where ⓘ

Schema name is

Table name is like

Use % as a wildcard.

Action ⓘ

Filter ⓘ

Column name is ✕

[Add condition](#)

[Add column filter](#)

[Add selection rule](#)

- When you have created the selections you want, choose **Add selection rule**.
- After you have created at least one selection rule, you can add a transformation to the task. Choose **add transformation rule**.

▼ Table mappings

Guided **JSON**

Selection rules ⓘ

where **schema name** is like 'EntertainmentAgencyExample' and **table name** is like 'Customers',
include [edit](#) [copy](#) [delete](#)

[add selection rule](#)

Transformation rules ⓘ

[add transformation rule](#)

- Choose the target that you want to transform, and enter the additional information requested. The following example shows a transformation that deletes the **AgencyStatus** column from the **Customer** table.

Selection rules ⓘ

where **schema name** is like 'EntertainmentAgencyExample' and **table name** is like 'Customers',
include ✎ 🔄 ✕

[+ add selection rule](#)

Transformation rules ⓘ

Target ⓘ

Where ⓘ

Schema name is

Table name is like

Column name is like

Use % as a wildcard.

Action ⓘ

Action

[cancel](#) [Add transformation rule](#)

10. Choose **Add transformation rule**.

11. (Optional) Add additional selection rules or transformations by choosing **add selection rule** or **add transformation rule**. When you are finished, choose **Create task**.

Guided **JSON**

Selection rules ⓘ

where **schema name** is like 'EntertainmentAgencyExample' and **table name** is like 'Customer',
include ✎ 🔄 ✕

[+ add selection rule](#)

Transformation rules ⓘ

For **column** where **schema name** is like 'EntertainmentAgencyExample' and **table name** is like
'Customer' and **column name** is like 'AgencyStatus', **remove column** ✎ 🔄 ✕

[+ add transformation rule](#)

[Cancel](#) [Create task](#)

Specifying Table Selection and Transformations by Table Mapping Using JSON

You can create table mappings in the JSON format. If you create a migration task using the AWS DMS Management Console, you can enter the JSON directly into the table mapping box. If you use the CLI or API to perform migrations, you can create a JSON file to specify the table mappings that you want to apply during migration.

You can specify what tables or schemas you want to work with, and you can perform schema and table transformations. You create table mapping rules using the `selection` and `transformation` rule types.

Selection Rules and Actions

Using table mapping, you can specify what tables or schemas you want to work with by using selection rules and actions. For table mapping rules that use the selection rule type, the following values can be applied.

Parameter	Possible Values	Description
<code>rule-type</code>	<code>selection</code>	You must have at least one selection rule when specifying a table mapping.
<code>rule-id</code>	A numeric value.	A unique numeric value to identify the rule.
<code>rule-name</code>	An alpha-numeric value.	A unique name to identify the rule.
<code>rule-action</code>	<code>include</code> , <code>exclude</code>	Include or exclude the object selected by the rule.
<code>load-order</code>	A positive integer. The maximum value is 2147483647.	Indicates the priority for loading tables. Tables with higher values are loaded first.

Example Migrate All Tables in a Schema

The following example migrates all tables from a schema named `Test` in your source to your target endpoint.

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


Example Migrate Some Tables in a Schema

The following example migrates all tables except those starting with **DMS** from a schema named **Test** in your source to your target endpoint.

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "Test",
        "table-name": "%"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "selection",
      "rule-id": "2",
      "rule-name": "2",
      "object-locator": {
        "schema-name": "Test",
        "table-name": "DMS%"
      },
      "rule-action": "exclude"
    }
  ]
}
```

Example Migrate All Tables in a Schema

The following example migrates all tables from a schema named **Test** in your source to your target endpoint.

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

Example Migrate Tables in a Set Order

The following example migrates two tables. Table `loadfirst` (with priority 2) is migrated before table `loadsecond`.

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
```

```

        "object-locator": {
            "schema-name": "Test",
            "table-name": "loadfirst"
        },
        "rule-action": "include",
        "load-order": "2"
    },
    {
        "rule-type": "selection",
        "rule-id": "2",
        "rule-name": "2",
        "object-locator": {
            "schema-name": "Test",
            "table-name": "loadsecond"
        },
        "rule-action": "include",
        "load-order": "1"
    }
]
}
    
```

Transformation Rules and Actions

You use the transformation actions to specify any transformations you want to apply to the selected schema or table. Transformation rules are optional.

For table mapping rules that use the transformation rule type, the following values can be applied.

Parameter	Possible Values	Description
rule-type	transformation, table-settings	A value that applies the rule to the object specified by the selection rule.
rule-id	A numeric value.	A unique numeric value to identify the rule.
rule-name	An alpha-numeric value.	A unique name to identify the rule.
object-locator	schema-name The name of the schema. table-name The name of the table. You can use the "%" percent sign to be a wildcard.	The schema and table the rule applies to.
rule-action	<ul style="list-style-type: none"> • rename • remove-column • convert-lowercase, convert-uppercase • add-prefix, remove-prefix, replace-prefix • add-suffix, remove-suffix, replace-suffix 	The transformation you want to apply to the object. All transformation rule actions are case-sensitive.
rule-target	schema, table, column	The type of object you are transforming.
value	An alpha-numeric value that follows the naming rules for the target type.	The new value for actions that require input, such as rename.

Parameter	Possible Values	Description
old-value	An alpha-numeric value that follows the naming rules for the target type.	The old value for actions that require replacement, such as replace-prefix.
parallel-load	<ul style="list-style-type: none">partitions-autosubpartitions-autonone	A value that determines how to segment a table while loading. You can segment the table by partitions or subpartitions.

Example Rename a Schema

The following example renames a schema from **Test** in your source to **Test1** in your target endpoint.

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "Test",
        "table-name": "%"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "transformation",
      "rule-id": "2",
      "rule-name": "2",
      "rule-action": "rename",
      "rule-target": "schema",
      "object-locator": {
        "schema-name": "Test"
      },
      "value": "Test1"
    }
  ]
}
```

Example Rename a Table

The following example renames a table from **Actor** in your source to **Actor1** in your target endpoint.

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "Test",
        "table-name": "%"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "transformation",
      "rule-id": "2",
      "rule-name": "2",

```

```
        "rule-action": "rename",
        "rule-target": "table",
        "object-locator": {
            "schema-name": "Test",
            "table-name": "Actor"
        },
        "value": "Actor1"
    }
]
}
```

Example Rename a Column

The following example renames a column in table **Actor** from **first_name** in your source to **fname** in your target endpoint.

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "test",
        "table-name": "%"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "transformation",
      "rule-id": "4",
      "rule-name": "4",
      "rule-action": "rename",
      "rule-target": "column",
      "object-locator": {
        "schema-name": "test",
        "table-name": "Actor",
        "column-name": "first_name"
      },
      "value": "fname"
    }
  ]
}
```

Example Remove a Column

The following example transforms the table named **Actor** in your source to remove all columns starting with the characters **co1** from it in your target endpoint.

```
{
  "rules": [{
    "rule-type": "selection",
    "rule-id": "1",
    "rule-name": "1",
    "object-locator": {
      "schema-name": "test",
      "table-name": "%"
    },
    "rule-action": "include"
  }, {
    "rule-type": "transformation",
    "rule-id": "2",
    "rule-name": "2",
```

```
"rule-action": "remove-column",
"rule-target": "column",
"object-locator": {
  "schema-name": "test",
  "table-name": "Actor",
  "column-name": "col%"
}
}]
}
```

Example Convert to Lowercase

The following example converts a table name from **ACTOR** in your source to **actor** in your target endpoint.

```
{
  "rules": [{
    "rule-type": "selection",
    "rule-id": "1",
    "rule-name": "1",
    "object-locator": {
      "schema-name": "test",
      "table-name": "%"
    },
    "rule-action": "include"
  }, {
    "rule-type": "transformation",
    "rule-id": "2",
    "rule-name": "2",
    "rule-action": "convert-lowercase",
    "rule-target": "table",
    "object-locator": {
      "schema-name": "test",
      "table-name": "ACTOR"
    }
  }
]}
}
```

Example Convert to Uppercase

The following example converts all columns in all tables and all schemas from lowercase in your source to uppercase in your target endpoint.

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "test",
        "table-name": "%"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "transformation",
      "rule-id": "2",
      "rule-name": "2",
      "rule-action": "convert-uppercase",
      "rule-target": "column",
      "object-locator": {
        "schema-name": "%",

```

```
        "table-name": "%",  
        "column-name": "%"  
    }  
  }  
]  
}
```

Example Add a Prefix

The following example transforms all tables in your source to add the prefix **DMS_** to them in your target endpoint.

```
{  
  "rules": [{  
    "rule-type": "selection",  
    "rule-id": "1",  
    "rule-name": "1",  
    "object-locator": {  
      "schema-name": "test",  
      "table-name": "%"  
    },  
    "rule-action": "include"  
  }, {  
    "rule-type": "transformation",  
    "rule-id": "2",  
    "rule-name": "2",  
    "rule-action": "add-prefix",  
    "rule-target": "table",  
    "object-locator": {  
      "schema-name": "test",  
      "table-name": "%"  
    },  
    "value": "DMS_"  
  }  
]  
}
```

Example Replace a Prefix

The following example transforms all columns containing the prefix **Pre_** in your source to replace the prefix with **NewPre_** in your target endpoint.

```
{  
  "rules": [  
    {  
      "rule-type": "selection",  
      "rule-id": "1",  
      "rule-name": "1",  
      "object-locator": {  
        "schema-name": "test",  
        "table-name": "%"  
      },  
      "rule-action": "include"  
    },  
    {  
      "rule-type": "transformation",  
      "rule-id": "2",  
      "rule-name": "2",  
      "rule-action": "replace-prefix",  
      "rule-target": "column",  
      "object-locator": {  
        "schema-name": "%",  
        "table-name": "%",  
      }  
    }  
  ]  
}
```

```

        "column-name": "%"
      },
      "value": "NewPre_",
      "old-value": "Pre_"
    }
  ]
}

```

Example Remove a Suffix

The following example transforms all tables in your source to remove the suffix **_DMS** from them in your target endpoint.

```

{
  "rules": [{
    "rule-type": "selection",
    "rule-id": "1",
    "rule-name": "1",
    "object-locator": {
      "schema-name": "test",
      "table-name": "%"
    },
    "rule-action": "include"
  }, {
    "rule-type": "transformation",
    "rule-id": "2",
    "rule-name": "2",
    "rule-action": "remove-suffix",
    "rule-target": "table",
    "object-locator": {
      "schema-name": "test",
      "table-name": "%"
    },
    "value": "_DMS"
  }
]}

```

Example Segment a Table for Loading

The following example segments a table in your source to load or unload the table more efficiently.

```

{
  "rules": [{
    "rule-type": "table-settings",
    "rule-id": "1",
    "rule-name": "1",
    "object-locator": {
      "schema-name": "test",
      "table-name": "table1"
    },
    "parallel-load": {
      "type": "partitions-auto"
    }
  }
]}

```

Using Source Filters

You can use source filters to limit the number and type of records transferred from your source to your target. For example, you can specify that only employees with a location of headquarters are moved to the target database. Filters are part of a selection rule. You apply filters on a column of data.

Source filters must follow these constraints:

- A selection rule can have no filters or one or more filters.
- Every filter can have one or more filter conditions.
- If more than one filter is used, the list of filters is combined as if using an AND operator between the filters.
- If more than one filter condition is used within a single filter, the list of filter conditions is combined as if using an OR operator between the filter conditions.
- Filters are only applied when `rule-action = 'include'`.
- Filters require a column name and a list of filter conditions. Filter conditions must have a filter operator and a value.
- Column names, table names, and schema names are case-sensitive.

Filtering by Time and Date

When selecting data to import, you can specify a date or time as part of your filter criteria. AWS DMS uses the date format YYYY-MM-DD and the time format YYYY-MM-DD HH:MM:SS for filtering. The AWS DMS comparison functions follow the SQLite conventions. For more information about SQLite data types and date comparisons, see [Datatypes In SQLite Version 3](#).

The following example shows how to filter on a date.

Example Single Date Filter

The following filter replicates all employees where `empstartdate >= January 1, 2002` to the target database.

```
{
  "rules": [{
    "rule-type": "selection",
    "rule-id": "1",
    "rule-name": "1",
    "object-locator": {
      "schema-name": "test",
      "table-name": "employee"
    },
    "rule-action": "include",
    "filters": [{
      "filter-type": "source",
      "column-name": "empstartdate",
      "filter-conditions": [{
        "filter-operator": "gte",
        "value": "2002-01-01"
      }]
    }]
  }]
}
```

Creating Source Filter Rules in JSON

You can create source filters by specifying a column name, filter condition, filter operator, and a filter value.

The following table shows the parameters used for source filtering.

Parameter	Value
filter-type	source
column-name	The name of the source column you want the filter applied to. The name is case-sensitive.
filter-conditions	
filter-operator	This parameter can be one of the following: <ul style="list-style-type: none"> • <code>lte</code> – less than or equal to • <code>gte</code> – greater than or equal to • <code>eq</code> – equal to • <code>between</code> – equal to or between two values
value	The value of the filter-operator parameter. If the filter-operator is <code>between</code> , provide two values, one for start-value and one for end-value.

The following examples show some common ways to use source filters.

Example Single Filter

The following filter replicates all employees where `empid >= 100` to the target database.

```
{
  "rules": [{
    "rule-type": "selection",
    "rule-id": "1",
    "rule-name": "1",
    "object-locator": {
      "schema-name": "test",
      "table-name": "employee"
    },
    "rule-action": "include",
    "filters": [{
      "filter-type": "source",
      "column-name": "empid",
      "filter-conditions": [{
        "filter-operator": "gte",
        "value": "100"
      }]
    }]
  }]
}
```

Example Multiple Filter Operators

The following filter applies multiple filter operators to a single column of data. The filter replicates all employees where (`empid <=10`) OR (`empid` is between 50 and 75) OR (`empid >= 100`) to the target database.

```
{
  "rules": [{
```

```

"rule-type": "selection",
"rule-id": "1",
"rule-name": "1",
"object-locator": {
  "schema-name": "test",
  "table-name": "employee"
},
"rule-action": "include",
"filters": [{
  "filter-type": "source",
  "column-name": "empid",
  "filter-conditions": [{
    "filter-operator": "lte",
    "value": "100"
  }],
  "filter-operator": "and",
  "start-value": "50",
  "end-value": "75"
}, {
  "filter-type": "source",
  "column-name": "dept",
  "filter-conditions": [{
    "filter-operator": "eq",
    "value": "tech"
  }],
  "filter-operator": "and",
  "start-value": "50",
  "end-value": "75"
}],
}]
}

```

Example Multiple Filters

The following filter applies multiple filters to two columns in a table. The filter replicates all employees where (empid <= 100) AND (dept= tech) to the target database.

```

{
  "rules": [{
    "rule-type": "selection",
    "rule-id": "1",
    "rule-name": "1",
    "object-locator": {
      "schema-name": "test",
      "table-name": "employee"
    },
    "rule-action": "include",
    "filters": [{
      "filter-type": "source",
      "column-name": "empid",
      "filter-conditions": [{
        "filter-operator": "lte",
        "value": "100"
      }],
      "filter-operator": "and",
      "start-value": "50",
      "end-value": "75"
    }, {
      "filter-type": "source",
      "column-name": "dept",
      "filter-conditions": [{
        "filter-operator": "eq",
        "value": "tech"
      }],
      "filter-operator": "and",
      "start-value": "50",
      "end-value": "75"
    }],
  }],
}

```

Monitoring AWS DMS Tasks

You can monitor the progress of your task by checking the task status and by monitoring the task's control table. For more information about control tables, see [Control Table Task Settings \(p. 230\)](#).

You can also monitor the progress of your tasks using Amazon CloudWatch. By using the AWS Management Console, the AWS Command Line Interface (CLI), or AWS DMS API, you can monitor the progress of your task and also the resources and network connectivity used.

Finally, you can monitor the status of your source tables in a task by viewing the table state.

Note that the "last updated" column the DMS console only indicates the time that AWS DMS last updated the table statistics record for a table. It does not indicate the time of the last update to the table.

For more information, see the following topics.

Topics

- [Task Status \(p. 261\)](#)
- [Table State During Tasks \(p. 262\)](#)
- [Monitoring Replication Tasks Using Amazon CloudWatch \(p. 263\)](#)
- [Data Migration Service Metrics \(p. 265\)](#)
- [Managing AWS DMS Task Logs \(p. 267\)](#)
- [Logging AWS DMS API Calls with AWS CloudTrail \(p. 268\)](#)

Task Status

The task status indicated the condition of the task. The following table shows the possible statuses a task can have:

Task Status	Description
Creating	AWS DMS is creating the task.
Running	The task is performing the migration duties specified.
Stopped	The task is stopped.
Stopping	The task is being stopped. This is usually an indication of user intervention in the task.
Deleting	The task is being deleted, usually from a request for user intervention.
Failed	The task has failed. See the task log files for more information.
Starting	The task is connecting to the replication instance and to the source and target endpoints. Any filters and transformations are being applied.
Ready	The task is ready to run. This status usually follows the "creating" status.

Task Status	Description
Modifying	The task is being modified, usually due to a user action that modified the task settings.

The task status bar gives an estimation of the task's progress. The quality of this estimate depends on the quality of the source database's table statistics; the better the table statistics, the more accurate the estimation. For tasks with only one table that has no estimated rows statistic, we are unable to provide any kind of percentage complete estimate. In this case, the task state and the indication of rows loaded can be used to confirm that the task is indeed running and making progress.

Table State During Tasks

The AWS DMS console updates information regarding the state of your tables during migration. The following table shows the possible state values:

The screenshot shows the AWS DMS console interface. At the top, there are buttons for 'Create task', 'Start/Resume', 'Stop', and 'Delete'. Below these is a filter input field. A table lists tasks with columns for ID, Status, Source, Target, and Type. Two tasks are shown, both with a status of 'Load complete'. Below this, the details for 'sg-mysql2pg-task1' are shown, with tabs for 'Overview', 'Task monitoring', 'Table statistics' (which is circled in red), and 'Logs'. Under the 'Table statistics' tab, there is another filter input and a table with columns for Table, State, Inserts, Deletes, Updates, and Rows. The 'State' column is circled in red, and three rows are listed, all with a state of 'Table completed'.

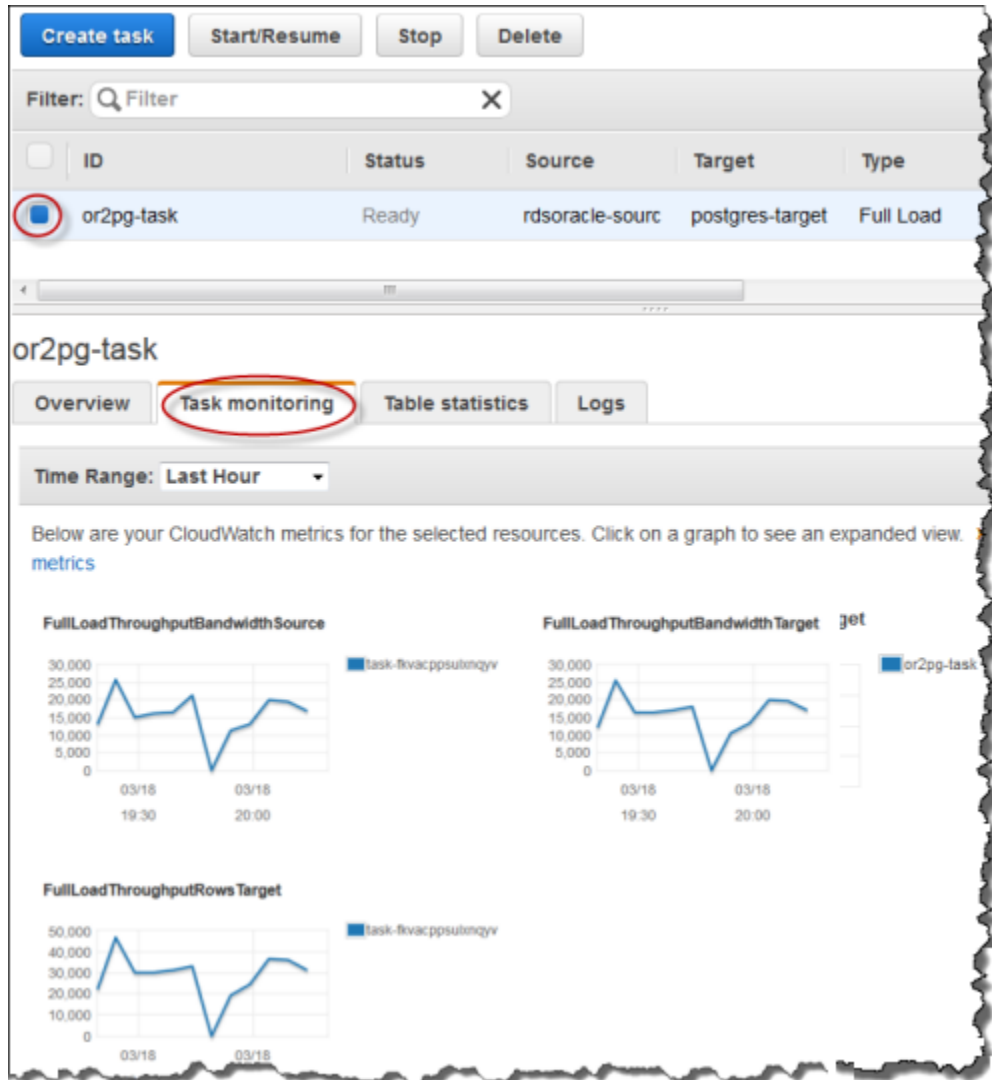
State	Description
Table does not exist	AWS DMS cannot find the table on the source endpoint.
Before load	The full load process has been enabled, but it hasn't started yet.
Full load	The full load process is in progress.
Table completed	Full load has completed.

State	Description
Table cancelled	Loading of the table has been cancelled.
Table error	An error occurred when loading the table.

Monitoring Replication Tasks Using Amazon CloudWatch

You can use Amazon CloudWatch alarms or events to more closely track your migration. For more information about Amazon CloudWatch, see [What Are Amazon CloudWatch, Amazon CloudWatch Events, and Amazon CloudWatch Logs?](#) in the Amazon CloudWatch User Guide. Note that there is a charge for using Amazon CloudWatch.

The AWS DMS console shows basic CloudWatch statistics for each task, including the task status, percent complete, elapsed time, and table statistics, as shown following. Select the replication task and then select the **Task monitoring** tab.



The AWS DMS console shows performance statistics for each table, including the number of inserts, deletions, and updates, when you select the **Table statistics** tab.

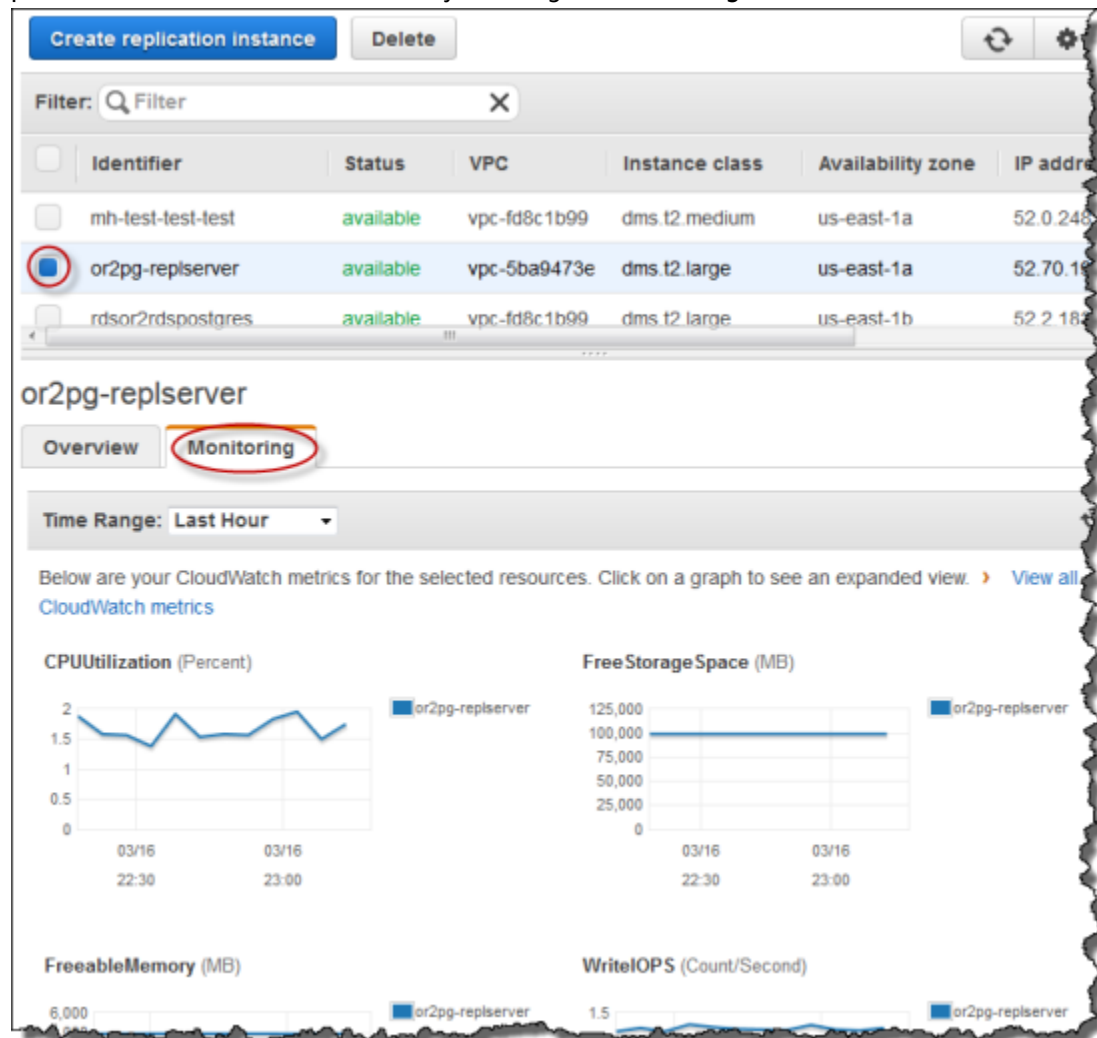
task-fkvacppsulxnqyv

Overview Task monitoring **Table statistics** Logs

Filter: X

Schema	Table	State	Inserts	Deletes	Updates	DDLs	Full Load Rows	Total	La
aat	T20	Full load	0	0	0	0	16,080,394	16,080,394	3/1
aat	T21	Full load	0	0	0	0	16,079,437	16,079,437	3/1
aat	T22	Full load	0	0	0	0	15,804,000	15,804,000	3/1

In addition, if you select a replication instance from the **Replication Instance** page, you can view performance metrics for the instance by selecting the **Monitoring** tab.



Data Migration Service Metrics

AWS DMS provides statistics for the following:

- **Host Metrics** – Performance and utilization statistics for the replication host, provided by Amazon CloudWatch. For a complete list of the available metrics, see [Replication Instance Metrics \(p. 265\)](#).
- **Replication Task Metrics** – Statistics for replication tasks including incoming and committed changes, and latency between the replication host and both the source and target databases. For a complete list of the available metrics, see [Replication Task Metrics \(p. 266\)](#).
- **Table Metrics** – Statistics for tables that are in the process of being migrated, including the number of insert, update, delete, and DDL statements completed.

Task metrics are divided into statistics between the replication host and the source endpoint, and statistics between the replication host and the target endpoint. You can determine the total statistic for a task by adding two related statistics together. For example, you can determine the total latency, or replica lag, for a task by combining the **CDCLatencySource** and **CDCLatencyTarget** values.

Task metric values can be influenced by current activity on your source database. For example, if a transaction has begun, but has not been committed, then the **CDCLatencySource** metric continues to grow until that transaction has been committed.

For the replication instance, the **FreeableMemory** metric requires clarification. Freeable memory is not a indication of the actual free memory available. It is the memory that is currently in use that can be freed and used for other uses; it's a combination of buffers and cache in use on the replication instance.

While the **FreeableMemory** metric does not reflect actual free memory available, the combination of the **FreeableMemory** and **SwapUsage** metrics can indicate if the replication instance is overloaded.

Monitor these two metrics for the following conditions.

- The **FreeableMemory** metric approaching zero.
- The **SwapUsage** metric increases or fluctuates.

If you see either of these two conditions, they indicate that you should consider moving to a larger replication instance. You should also consider reducing the number and type of tasks running on the replication instance. Full Load tasks require more memory than tasks that just replicate changes.

Replication Instance Metrics

Replication instance monitoring include Amazon CloudWatch metrics for the following statistics:

CPUUtilization

The amount of CPU used.

Units: Bytes

FreeStorageSpace

The amount of available storage space.

Units: Bytes

FreeableMemory

The amount of available random access memory.

Units: Bytes

WriteIOPS

The average number of disk I/O operations per second.

Units: Count/Second

ReadIOPS

The average number of disk I/O operations per second.

Units: Count/Second

WriteThroughput

The average number of bytes written to disk per second.

Units: Bytes/Second

ReadThroughput

The average number of bytes read from disk per second.

Units: Bytes/Second

WriteLatency

The average amount of time taken per disk I/O (output) operation.

Units: Milliseconds

ReadLatency

The average amount of time taken per disk I/O (input) operation.

Units: Milliseconds

SwapUsage

The amount of swap space used on the replication instance.

Units: Bytes

NetworkTransmitThroughput

The outgoing (Transmit) network traffic on the replication instance, including both customer database traffic and AWS DMS traffic used for monitoring and replication.

Units: Bytes/second

NetworkReceiveThroughput

The incoming (Receive) network traffic on the replication instance, including both customer database traffic and AWS DMS traffic used for monitoring and replication.

Units: Bytes/second

Replication Task Metrics

Replication task monitoring includes metrics for the following statistics:

FullLoadThroughputBandwidthSource

Incoming network bandwidth from a full load from the source in kilobytes (KB) per second.

FullLoadThroughputBandwidthTarget

Outgoing network bandwidth from a full load for the target in KB per second.

FullLoadThroughputRowsSource

Incoming changes from a full load from the source in rows per second.

FullLoadThroughputRowsTarget

Outgoing changes from a full load for the target in rows per second.

CDCIncomingChanges

The total number of change events at a point-in-time that are waiting to be applied to the target. Note that this is not the same as a measure of the transaction change rate of the source endpoint. A large number for this metric usually indicates AWS DMS is unable to apply captured changes in a timely manner, thus causing high target latency.

CDCChangesMemorySource

Amount of rows accumulating in a memory and waiting to be committed from the source.

CDCChangesMemoryTarget

Amount of rows accumulating in a memory and waiting to be committed to the target.

CDCChangesDiskSource

Amount of rows accumulating on disk and waiting to be committed from the source.

CDCChangesDiskTarget

Amount of rows accumulating on disk and waiting to be committed to the target.

CDCThroughputBandwidthSource

Network bandwidth for the target in KB per second. CDCThroughputBandwidth records bandwidth on sampling points. If no network traffic is found, the value is zero. Because CDC does not issue long-running transactions, network traffic may not be recorded.

CDCThroughputBandwidthTarget

Network bandwidth for the target in KB per second. CDCThroughputBandwidth records bandwidth on sampling points. If no network traffic is found, the value is zero. Because CDC does not issue long-running transactions, network traffic may not be recorded.

CDCThroughputRowsSource

Incoming task changes from the source in rows per second.

CDCThroughputRowsTarget

Outgoing task changes for the target in rows per second.

CDCLatencySource

The gap, in seconds, between the last event captured from the source endpoint and current system time stamp of the AWS DMS instance. If no changes have been captured from the source due to task scoping, AWS DMS sets this value to zero.

CDCLatencyTarget

The gap, in seconds, between the first event timestamp waiting to commit on the target and the current timestamp of the AWS DMS instance. This value occurs if there are transactions that are not handled by target. Otherwise, target latency is the same as source latency if all transactions are applied. Target latency should never be smaller than the source latency.

Managing AWS DMS Task Logs

AWS DMS uses Amazon CloudWatch to log task information during the migration process. You can use the AWS CLI or the AWS DMS API to view information about the task logs. To do this, use the

`describe-replication-instance-task-logs` AWS CLI command or the AWS DMS API action `DescribeReplicationInstanceTaskLogs`.

For example, the following AWS CLI command shows the task log metadata in JSON format.

```
$ aws dms describe-replication-instance-task-logs \
  --replication-instance-arn arn:aws:dms:us-east-1:237565436:rep:CDSF5FSFFFSSUFCA5
```

A sample response from the command is as follows.

```
{
  "ReplicationInstanceTaskLogs": [
    {
      "ReplicationTaskArn": "arn:aws:dms:us-
east-1:237565436:task:MY34U6Z4MSY52GRTIX304AY",
      "ReplicationTaskName": "mysql-to-ddb",
      "ReplicationInstanceTaskLogSize": 3726134
    }
  ],
  "ReplicationInstanceArn": "arn:aws:dms:us-east-1:237565436:rep:CDSF5FSFFFSSUFCA5"
}
```

In this response, there is a single task log (`mysql-to-ddb`) associated with the replication instance. The size of this log is 3,726,124 bytes.

You can use the information returned by `describe-replication-instance-task-logs` to diagnose and troubleshoot problems with task logs. For example, if you enable detailed debug logging for a task, the task log will grow quickly—potentially consuming all of the available storage on the replication instance, and causing the instance status to change to `storage-full`. By describing the task logs, you can determine which ones you no longer need; then you can delete them, freeing up storage space.

To delete the task logs for a task, set the task setting `DeleteTaskLogs` to `true`. For example, the following JSON deletes the task logs when modifying a task using the AWS CLI `modify-replication-task` command or the AWS DMS API `ModifyReplicationTask` action.

```
{
  "Logging": {
    "DeleteTaskLogs": true
  }
}
```

Logging AWS DMS API Calls with AWS CloudTrail

AWS DMS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS DMS. CloudTrail captures all API calls for AWS DMS as events, including calls from the AWS DMS console and from code calls to the AWS DMS APIs. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS DMS. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to AWS DMS, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

AWS DMS Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS DMS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your AWS account, including events for AWS DMS, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all regions. The trail logs events from all regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- [Overview for Creating a Trail](#)
- [CloudTrail Supported Services and Integrations](#)
- [Configuring Amazon SNS Notifications for CloudTrail](#)
- [Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts](#)

All AWS DMS actions are logged by CloudTrail and are documented in the [AWS Database Migration Service API Reference](#). For example, calls to the `CreateReplicationInstance`, `TestConnection` and `StartReplicationTask` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](#).

Understanding AWS DMS Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `RebootReplicationInstance` action.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAIOSFODNN7EXAMPLE: johndoe",
    "arn": "arn:aws:sts::123456789012:assumed-role/admin/johndoe",
    "accountId": "123456789012",
    "accessKeyId": "ASIAYFI33SINADOJJEZW",
```

```
"sessionContext": {
  "attributes": {
    "mfaAuthenticated": "false",
    "creationDate": "2018-08-01T16:42:09Z"
  },
  "sessionIssuer": {
    "type": "Role",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::123456789012:role/admin",
    "accountId": "123456789012",
    "userName": "admin"
  }
},
"eventTime": "2018-08-02T00:11:44Z",
"eventSource": "dms.amazonaws.com",
"eventName": "RebootReplicationInstance",
"awsRegion": "us-east-1",
"sourceIPAddress": "72.21.198.64",
"userAgent": "console.amazonaws.com",
"requestParameters": {
  "forceFailover": false,
  "replicationInstanceArn": "arn:aws:dms:us-east-1:123456789012:rep:EX4MBJ2NMRDL3BMAYJOXUGYPUE"
},
"responseElements": {
  "replicationInstance": {
    "replicationInstanceIdentifier": "replication-instance-1",
    "replicationInstanceStatus": "rebooting",
    "allocatedStorage": 50,
    "replicationInstancePrivateIpAddresses": [
      "172.31.20.204"
    ],
    "instanceCreateTime": "Aug 1, 2018 11:56:21 PM",
    "autoMinorVersionUpgrade": true,
    "engineVersion": "2.4.3",
    "publiclyAccessible": true,
    "replicationInstanceClass": "dms.t2.medium",
    "availabilityZone": "us-east-1b",
    "kmsKeyId": "arn:aws:kms:us-east-1:123456789012:key/f7bc0f8e-1a3a-4ace-9faa-e8494fa3921a",
    "replicationSubnetGroup": {
      "vpcId": "vpc-1f6a9c6a",
      "subnetGroupStatus": "Complete",
      "replicationSubnetGroupArn": "arn:aws:dms:us-east-1:123456789012:subgrp:EDHRVRBAAAPONQAIYWP4NUW22M",
      "subnets": [
        {
          "subnetIdentifier": "subnet-cbfff283",
          "subnetAvailabilityZone": {
            "name": "us-east-1b"
          },
          "subnetStatus": "Active"
        },
        {
          "subnetIdentifier": "subnet-d7c825e8",
          "subnetAvailabilityZone": {
            "name": "us-east-1e"
          },
          "subnetStatus": "Active"
        },
        {
          "subnetIdentifier": "subnet-6746046b",
          "subnetAvailabilityZone": {
            "name": "us-east-1f"
          }
        }
      ]
    }
  }
}
```

```

    "subnetStatus": "Active"
  },
  {
    "subnetIdentifier": "subnet-bac383e0",
    "subnetAvailabilityZone": {
      "name": "us-east-1c"
    },
    "subnetStatus": "Active"
  },
  {
    "subnetIdentifier": "subnet-42599426",
    "subnetAvailabilityZone": {
      "name": "us-east-1d"
    },
    "subnetStatus": "Active"
  },
  {
    "subnetIdentifier": "subnet-da327bf6",
    "subnetAvailabilityZone": {
      "name": "us-east-1a"
    },
    "subnetStatus": "Active"
  }
],
"replicationSubnetGroupIdentifier": "default-vpc-1f6a9c6a",
"replicationSubnetGroupDescription": "default group created by console for
vpc id vpc-1f6a9c6a"
},
"replicationInstanceEniId": "eni-0d6db8c7137cb9844",
"vpcSecurityGroups": [
  {
    "vpcSecurityGroupId": "sg-f839b688",
    "status": "active"
  }
],
"pendingModifiedValues": {},
"replicationInstancePublicIpAddresses": [
  "18.211.48.119"
],
"replicationInstancePublicIpAddress": "18.211.48.119",
"preferredMaintenanceWindow": "fri:22:44-fri:23:14",
"replicationInstanceArn": "arn:aws:dms:us-
east-1:123456789012:rep:EX4MBJ2NMRDL3BMAYJOXUGYPUE",
"replicationInstanceEniIds": [
  "eni-0d6db8c7137cb9844"
],
"multiAZ": false,
"replicationInstancePrivateIpAddress": "172.31.20.204",
"patchingPrecedence": 0
}
},
"requestID": "a3c83c11-95e8-11e8-9d08-4b8f2b45bfd5",
"eventID": "b3c4adb1-e34b-4744-bdeb-35528062a541",
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
}

```

Validating AWS DMS Tasks

Topics

- [Replication Task Statistics \(p. 273\)](#)
- [Revalidating Tables During a Task \(p. 274\)](#)
- [Troubleshooting \(p. 275\)](#)
- [Limitations \(p. 276\)](#)

AWS DMS provides support for data validation, to ensure that your data was migrated accurately from the source to the target. If you enable it for a task, then AWS DMS begins comparing the source and target data immediately after a full load is performed for a table.

Data validation is optional. AWS DMS compares the source and target records, and reports any mismatches. In addition, for a CDC-enabled task, AWS DMS compares the incremental changes and reports any mismatches.

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 appropriate queries to retrieve the data. Note that these queries will consume additional resources at the source and the target, as well as additional network resources.

Data validation works with the following databases:

- Oracle
- PostgreSQL
- MySQL
- MariaDB
- Microsoft SQL Server
- Amazon Aurora (MySQL)
- Amazon Aurora (PostgreSQL)

Data validation requires additional time, beyond the amount required for the migration itself. The extra time required depends on how much data was migrated.

Data validation settings include the following:

- To enable data validation, set the `EnableValidation` setting to `true`.
- To adjust the number of execution threads that AWS DMS uses during validation, set the `ThreadCount` value. The default value for `ThreadCount` is 5. If you set `ThreadCount` to a higher number, AWS DMS will be able to complete the validation faster—however, it will also execute more simultaneous queries, consuming more resources on the source and the target.

For example, the following JSON turns on validation and increases the number of threads from the default setting of 5 to 8.

```
ValidationSettings": {
  "EnableValidation":true,
  "ThreadCount":8
}
```

Replication Task Statistics

When data validation is enabled, AWS DMS provides the following statistics at the table level:

- **ValidationState**—The validation state of the table. The parameter can have the following values:
 - **Not enabled**—Validation is not enabled for the table in the migration task.
 - **Pending records**—Some records in the table are waiting for validation.
 - **Mismatched records**—Some records in the table don't match between the source and target. A mismatch might occur for a number of reasons; For more information, check the `awsdms_validation_failures` table on the target endpoint.
 - **Suspended records**—Some records in the table can't be validated.
 - **No primary key**—The table can't be validated because it had no primary key.
 - **Table error**—The table wasn't validated because it was in an error state and some data wasn't migrated.
 - **Validated**—All rows in the table are validated. If the table is updated, the status can change from Validated.
 - **Error**—The table can't be validated because of an unexpected error.
- **ValidationPending**—The number of records that have been migrated to the target, but that haven't yet been validated.

ValidationSuspended—The number of records that AWS DMS can't compare. For example, if a record at the source is constantly being updated, AWS DMS can't compare the source and the target. For more information, see [Error Handling Task Settings \(p. 234\)](#)

- **ValidationFailed**—The number of records that didn't pass the data validation phase. For more information, see [Error Handling Task Settings \(p. 234\)](#).
- **ValidationSucceededRecordCount**— Number of rows that AWS DMS validated, per minute.
- **ValidationAttemptedRecordCount**— Number of rows that validation was attempted, per minute.
- **ValidationFailedOverallCount**— Number of rows where validation failed.
- **ValidationSuspendedOverallCount**— Number of rows where validation was suspended.
- **ValidationPendingOverallCount**— Number of rows where the validation is still pending.
- **ValidationBulkQuerySourceLatency**— AWS DMS can do data validation in bulk, especially in certain scenarios during a full-load or on-going replication when there are many changes. This metric indicates the latency required to read a bulk set of data from the source endpoint.
- **ValidationBulkQueryTargetLatency**— AWS DMS can do data validation in bulk, especially in certain scenarios during a full-load or on-going replication when there are many changes. This metric indicates the latency required to read a bulk set of data on the target endpoint.
- **ValidationItemQuerySourceLatency**— During on-going replication, data validation can identify on-going changes and validate those changes. This metric indicates the latency in reading those changes from the source. Validation can run more queries than required, based on number of changes, if there are errors during validation.
- **ValidationItemQueryTargetLatency**— During on-going replication, data validation can identify on-going changes and validate the changes row by row. This metric gives us the latency in reading those changes from the target. Validation may run more queries than required, based on number of changes, if there are errors during validation.

You can view the data validation information using the console, the AWS CLI, or the AWS DMS API.

- On the console, you can choose to validate a task when you create or modify the task. To view the data validation report using the console, choose the task on the **Tasks** page and choose the **Table statistics** tab in the details section.

- Using the CLI, set the `EnableValidation` parameter to `true` when creating or modifying a task to begin data validation. The following example creates a task and enables data validation.

```
create-replication-task
--replication-task-settings '{"ValidationSettings":{"EnableValidation":true}}'
--replication-instance-arn arn:aws:dms:us-east-1:5731014:
    rep:36KWVMB7Q
--source-endpoint-arn arn:aws:dms:us-east-1:5731014:
    endpoint:CSZAEFQURFYMM
--target-endpoint-arn arn:aws:dms:us-east-1:5731014:
    endpoint:CGPP7MF6WT4JQ
--migration-type full-load-and-cdc
--table-mappings '{"rules": [{"rule-type": "selection", "rule-id": "1",
    "rule-name": "1", "object-locator": {"schema-name": "data_types", "table-name":
"%"},
    "rule-action": "include"}]}'
```

Use the `describe-table-statistics` command to receive the data validation report in JSON format. The following command shows the data validation report.

```
aws dms describe-table-statistics --replication-task-arn arn:aws:dms:us-east-1:5731014:
rep:36KWVMB7Q
```

The report would be similar to the following.

```
{
  "ReplicationTaskArn": "arn:aws:dms:us-west-2:5731014:task:VFPPTYKK2RYSI",
  "TableStatistics": [
    {
      "ValidationPendingRecords": 2,
      "Inserts": 25,
      "ValidationState": "Pending records",
      "ValidationSuspendedRecords": 0,
      "LastUpdateTime": 1510181065.349,
      "FullLoadErrorRows": 0,
      "FullLoadCondtnlChkFailedRows": 0,
      "Ddls": 0,
      "TableName": "t_binary",
      "ValidationFailedRecords": 0,
      "Updates": 0,
      "FullLoadRows": 10,
      "TableState": "Table completed",
      "SchemaName": "d_types_s_sqlserver",
      "Deletes": 0
    }
  ]
}
```

- Using the AWS DMS API, create a task using the `CreateReplicationTask` action and set the `EnableValidation` parameter to `true` to validate the data migrated by the task. Use the `DescribeTableStatistics` action to receive the data validation report in JSON format.

Revalidating Tables During a Task

While a task is running, you can request AWS DMS to perform data validation.

AWS Management Console

1. Sign in to the AWS Management Console and choose AWS DMS. 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 Needed to Use AWS DMS \(p. 31\)](#).
2. Choose **Tasks** from the navigation pane.
3. Choose the running task that has the table you want to revalidate.
4. Choose the **Table Statistics** tab.
5. Choose the table you want to revalidate. If the task is no longer running, you can't revalidate the table.
6. Choose **Revalidate**.

Troubleshooting

During validation, AWS DMS creates a new table at the target endpoint: `awsdms_validation_failures_v1`. If any record enters the `ValidationSuspended` or the `ValidationFailed` state, AWS DMS writes diagnostic information to `awsdms_validation_failures_v1`. You can query this table to help troubleshoot validation errors.

Following is a description of the `awsdms_validation_failures_v1` table:

Column Name	Data Type	Description
<code>TASK_NAME</code>	<code>VARCHAR(128)</code> NOT NULL	AWS DMS task identifier.
<code>TABLE_OWNER</code>	<code>VARCHAR(128)</code> NOT NULL	Schema (owner) of the table.
<code>TABLE_NAME</code>	<code>VARCHAR(128)</code> NOT NULL	Table name.
<code>FAILURE_TIME</code>	<code>DATE</code> NOT NULL	Time when the failure occurred.
<code>KEY</code>	<code>TEXT</code> NOT NULL	This is the primary key for row record type.
<code>FAILURE_TYPE</code>	<code>VARCHAR(128)</code> NOT NULL	Severity of validation error. Can be either <code>Failed</code> or <code>Suspended</code> .

The following query will show you all the failures for a task by querying the `awsdms_validation_failures_v1` table. The task name should be the external resource ID of the task. The external resource ID of the task is the last value in the task ARN. For example, for a task with an ARN value of `arn:aws:dms:us-west-2:5599:task:VFPFKH4FJR3FTYKK2RYSI`, the external resource ID of the task would be `VFPFKH4FJR3FTYKK2RYSI`.

```
select * from awsdms_validation_failures_v1 where TASK_NAME = 'VFPFKH4FJR3FTYKK2RYSI'
```

Once you have the primary key of the failed record, you can query the source and target endpoints to see what part of the record does not match.

Limitations

- Data validation requires that the table has a primary key or unique index.
 - Primary key columns can't be of type `CLOB`, `BLOB`, or `BYTE`.
 - For primary key columns of type `VARCHAR` or `CHAR`, the length must be less than 1024.
- If the collation of the primary key column in the target PostgreSQL instance isn't set to "C", the sort order of the primary key is different compared to the sort order in Oracle. If the sort order is different between PostgreSQL and Oracle, data validation fails to validate the records.
- Data validation generates additional queries against the source and target databases. You must ensure that both databases have enough resources to handle this additional load.
- Data validation isn't supported if a migration uses customized filtering or when consolidating several databases into one.
- For a source or target Oracle endpoint, AWS DMS uses `DBMS_CRYPTO` to validate BLOBs. If your Oracle endpoint uses BLOBs, then you must grant the `execute` permission on `dbms_crypto` to the user account that is used to access the Oracle endpoint. You can do this by running the following statement:

```
grant execute on sys.dbms_crypto to <dms_endpoint_user>;
```

- If the target database is modified outside of AWS DMS during validation, then discrepancies might not be reported accurately. This result can occur if one of your applications writes data to the target table, while AWS DMS is performing validation on that same table.
- If one or more rows are being continuously modified during the validation, then AWS DMS can't validate those rows. However, you can validate those rows manually, after the task completes.
- If AWS DMS detects more than 10,000 failed or suspended records, it stops the validation. Before you proceed further, resolve any underlying problems with the data.

Tagging Resources in AWS Database Migration Service

You can use tags in AWS Database Migration Service (AWS DMS) to add metadata to your resources. In addition, you can use these tags with AWS Identity and Access Management (IAM) policies to manage access to AWS DMS resources and to control what actions can be applied to the AWS DMS resources. Finally, you can use these tags to track costs by grouping expenses for similarly tagged resources.

All AWS DMS resources can be tagged:

- Replication instances
- Endpoints
- Replication tasks
- Certificates

An AWS DMS tag is a name-value pair that you define and associate with an AWS DMS resource. The name is referred to as the key. Supplying a value for the key is optional. You can use tags to assign arbitrary information to an AWS DMS resource. A tag key could be used, for example, to define a category, and the tag value could be an item in that category. For example, you could define a tag key of "project" and a tag value of "Salix", indicating that the AWS DMS resource is assigned to the Salix project. You could also use tags to designate AWS DMS resources as being used for test or production by using a key such as environment=test or environment=production. We recommend that you use a consistent set of tag keys to make it easier to track metadata associated with AWS DMS resources.

Use tags to organize your AWS bill to reflect your own cost structure. To do this, sign up to get your AWS account bill with tag key values included. Then, to see the cost of combined resources, organize your billing information according to resources with the same tag key values. For example, you can tag several resources with a specific application name, and then organize your billing information to see the total cost of that application across several services. For more information, see [Cost Allocation and Tagging in About AWS Billing and Cost Management](#).

Each AWS DMS resource has a tag set, which contains all the tags that are assigned to that AWS DMS resource. A tag set can contain as many as ten tags, or it can be empty. If you add a tag to an AWS DMS resource that has the same key as an existing tag on resource, the new value overwrites the old value.

AWS does not apply any semantic meaning to your tags; tags are interpreted strictly as character strings. AWS DMS might set tags on an AWS DMS resource, depending on the settings that you use when you create the resource.

The following list describes the characteristics of an AWS DMS tag.

- The tag key is the required name of the tag. The string value can be from 1 to 128 Unicode characters in length and cannot be prefixed with "aws:" or "dms:". The string might contain only the set of Unicode letters, digits, white-space, '_', ':', '/', '=', '+', '-' (Java regex: `^[\\p{L}\\p{Z}\\p{N}._:/+=\\-]*$`).
- The tag value is an optional string value of the tag. The string value can be from 1 to 256 Unicode characters in length and cannot be prefixed with "aws:" or "dms:". The string might contain only the set of Unicode letters, digits, white-space, '_', ':', '/', '=', '+', '-' (Java regex: `^[\\p{L}\\p{Z}\\p{N}._:/+=\\-]*$`).

Values do not have to be unique in a tag set and can be null. For example, you can have a key-value pair in a tag set of project/Trinity and cost-center/Trinity.

You can use the AWS CLI or the AWS DMS API to add, list, and delete tags on AWS DMS resources. When using the AWS CLI or the AWS DMS API, you must provide the Amazon Resource Name (ARN) for the AWS DMS resource you want to work with. For more information about constructing an ARN, see [Constructing an Amazon Resource Name \(ARN\) for AWS DMS \(p. 11\)](#).

Note that tags are cached for authorization purposes. Because of this, additions and updates to tags on AWS DMS resources might take several minutes before they are available.

API

You can add, list, or remove tags for a AWS DMS resource using the AWS DMS API.

- To add a tag to an AWS DMS resource, use the [AddTagsToResource](#) operation.
- To list tags that are assigned to an AWS DMS resource, use the [ListTagsForResource](#) operation.
- To remove tags from an AWS DMS resource, use the [RemoveTagsFromResource](#) operation.

To learn more about how to construct the required ARN, see [Constructing an Amazon Resource Name \(ARN\) for AWS DMS \(p. 11\)](#).

When working with XML using the AWS DMS API, tags use the following schema:

```
<Tagging>
  <TagSet>
    <Tag>
      <Key>Project</Key>
      <Value>Trinity</Value>
    </Tag>
    <Tag>
      <Key>User</Key>
      <Value>Jones</Value>
    </Tag>
  </TagSet>
</Tagging>
```

The following table provides a list of the allowed XML tags and their characteristics. Note that values for Key and Value are case dependent. For example, project=Trinity and PROJECT=Trinity are two distinct tags.

Tagging element	Description
TagSet	A tag set is a container for all tags assigned to an Amazon RDS resource. There can be only one tag set per resource. You work with a TagSet only through the AWS DMS API.
Tag	A tag is a user-defined key-value pair. There can be from 1 to 10 tags in a tag set.
Key	A key is the required name of the tag. The string value can be from 1 to 128 Unicode characters in length and cannot be prefixed with "dms:" or "aws:". The string might only contain only the set of Unicode letters, digits, white-space, '_', ':', '/', '=', '+', '-' (Java regex: "^[\\p{L}\\p{Z}\\p{N}_:/=+\\-]*\$"). Keys must be unique to a tag set. For example, you cannot have a key-pair in a tag set with the key the same but with different values, such as project/Trinity and project/Xanadu.

Tagging element	Description
Value	<p>A value is the optional value of the tag. The string value can be from 1 to 256 Unicode characters in length and cannot be prefixed with "dms:" or "aws:". The string might only contain only the set of Unicode letters, digits, white-space, '_', ':', '/', '=', '+', '-' (Java regex: "<code>^[\\p{L}\\p{Z}\\p{N}_:/=+\\-]*\$</code>").</p> <p>Values do not have to be unique in a tag set and can be null. For example, you can have a key-value pair in a tag set of project/Trinity and cost-center/Trinity.</p>

Working with Events and Notifications in AWS Database Migration Service

AWS Database Migration Service (AWS DMS) uses Amazon Simple Notification Service (Amazon SNS) to provide notifications when an AWS DMS event occurs, for example the creation or deletion of a replication instance. You can work with these notifications in any form supported by Amazon SNS for an AWS Region, such as an email message, a text message, or a call to an HTTP endpoint.

AWS DMS groups events into categories that you can subscribe to, so you can be notified when an event in that category occurs. For example, if you subscribe to the Creation category for a given replication instance, you are notified whenever a creation-related event occurs that affects your replication instance. If you subscribe to a Configuration Change category for a replication instance, you are notified when the replication instance's configuration is changed. You also receive notification when an event notification subscription changes. For a list of the event categories provided by AWS DMS, see [AWS DMS Event Categories and Event Messages \(p. 281\)](#), following.

AWS DMS sends event notifications to the addresses you provide when you create an event subscription. You might want to create several different subscriptions, such as one subscription receiving all event notifications and another subscription that includes only critical events for your production DMS resources. You can easily turn off notification without deleting a subscription by setting the **Enabled** option to **No** in the AWS DMS console or by setting the `Enabled` parameter to `false` using the AWS DMS API.

Note

AWS DMS event notifications using SMS text messages are currently available for AWS DMS resources in all regions where AWS DMS is supported. For more information on using text messages with SNS, see [Sending and Receiving SMS Notifications Using Amazon SNS](#).

AWS DMS uses a subscription identifier to identify each subscription. You can have multiple AWS DMS event subscriptions published to the same Amazon SNS topic. When you use event notification, Amazon SNS fees apply; for more information on Amazon SNS billing, see [Amazon SNS Pricing](#).

To subscribe to AWS DMS events, you use the following process:

1. Create an Amazon SNS topic. In the topic, you specify what type of notification you want to receive and to what address or number the notification will go to.
2. Create an AWS DMS event notification subscription by using the AWS Management Console, AWS CLI, or AWS DMS API.
3. AWS DMS sends an approval email or SMS message to the addresses you submitted with your subscription. To confirm your subscription, click the link in the approval email or SMS message.
4. When you have confirmed the subscription, the status of your subscription is updated in the AWS DMS console's **Event Subscriptions** section.
5. You then begin to receive event notifications.

For the list of categories and events that you can be notified of, see the following section. For more details about subscribing to and working with AWS DMS event subscriptions, see [Subscribing to AWS DMS Event Notification \(p. 282\)](#).

AWS DMS Event Categories and Event Messages

AWS DMS generates a significant number of events in categories that you can subscribe to using the AWS DMS console or the AWS DMS API. Each category applies to a source type; currently AWS DMS supports the replication instance and replication task source types.

The following table shows the possible categories and events for the replication instance source type.

Category	DMS Event ID	Description
Configuration Change	DMS-EVENT-0012	REP_INSTANCE_CLASS_CHANGING – The replication instance class for this replication instance is being changed.
Configuration Change	DMS-EVENT-0014	REP_INSTANCE_CLASS_CHANGE_COMPLETE – The replication instance class for this replication instance has changed.
Configuration Change	DMS-EVENT-0018	BEGIN_SCALE_STORAGE – The storage for the replication instance is being increased.
Configuration Change	DMS-EVENT-0017	FINISH_SCALE_STORAGE – The storage for the replication instance has been increased.
Configuration Change	DMS-EVENT-0024	BEGIN_CONVERSION_TO_HIGH_AVAILABILITY – The replication instance is transitioning to a Multi-AZ configuration.
Configuration Change	DMS-EVENT-0025	FINISH_CONVERSION_TO_HIGH_AVAILABILITY – The replication instance has finished transitioning to a Multi-AZ configuration.
Configuration Change	DMS-EVENT-0030	BEGIN_CONVERSION_TO_NON_HIGH_AVAILABILITY – The replication instance is transitioning to a Single-AZ configuration.
Configuration Change	DMS-EVENT-0029	FINISH_CONVERSION_TO_NON_HIGH_AVAILABILITY – The replication instance has finished transitioning to a Single-AZ configuration.
Creation	DMS-EVENT-0067	CREATING_REPLICATION_INSTANCE – A replication instance is being created.
Creation	DMS-EVENT-0005	CREATED_REPLICATION_INSTANCE – A replication instance has been created.
Deletion	DMS-EVENT-0066	DELETING_REPLICATION_INSTANCE – The replication instance is being deleted.
Deletion	DMS-EVENT-0003	DELETED_REPLICATION_INSTANCE – The replication instance has been deleted.
Maintenance	DMS-EVENT-0047	FINISH_PATCH_INSTANCE – Management software on the replication instance has been updated.
Maintenance	DMS-EVENT-0026	BEGIN_PATCH_OFFLINE – Offline maintenance of the replication instance is taking place. The replication instance is currently unavailable.

Category	DMS Event ID	Description
Maintenance	DMS-EVENT-0027	FINISH_PATCH_OFFLINE – Offline maintenance of the replication instance is complete. The replication instance is now available.
LowStorage	DMS-EVENT-0007	LOW_STORAGE – Free storage for the replication instance is low.
Failover	DMS-EVENT-0013	FAILOVER_STARTED – Failover started for a Multi-AZ replication instance.
Failover	DMS-EVENT-0049	FAILOVER_COMPLETED – Failover has been completed for a Multi-AZ replication instance.
Failover	DMS-EVENT-0050	MAZ_INSTANCE_ACTIVATION_STARTED – Multi-AZ activation has started.
Failover	DMS-EVENT-0051	MAZ_INSTANCE_ACTIVATION_COMPLETED – Multi-AZ activation completed.
Failure	DMS-EVENT-0031	REPLICATION_INSTANCE_FAILURE – The replication instance has gone into storage failure.
Failure	DMS-EVENT-0036	INCOMPATIBLE_NETWORK – The replication instance has failed due to an incompatible network.

The following table shows the possible categories and events for the replication task source type.

Category	DMS Event ID	Description
StateChange	DMS-EVENT-0069	REPLICATION_TASK_STARTED – The replication task has started.
StateChange	DMS-EVENT-0077	REPLICATION_TASK_STOPPED – The replication task has stopped.
Failure	DMS-EVENT-0078	REPLICATION_TASK_FAILED – A replication task has failed.
Deletion	DMS-EVENT-0073	REPLICATION_TASK_DELETED – The replication task has been deleted.
Creation	DMS-EVENT-0074	REPLICATION_TASK_CREATED – The replication task has been created.

Subscribing to AWS DMS Event Notification

You can create an AWS DMS event notification subscription so you can be notified when an AWS DMS event occurs. The simplest way to create a subscription is with the AWS DMS console. If you choose to create event notification subscriptions using AWS DMS API, you must create an Amazon SNS topic and subscribe to that topic with the Amazon SNS console or API. In this case, you also need to note the topic's Amazon Resource Name (ARN), because this ARN is used when submitting CLI commands or API actions. For information on creating an Amazon SNS topic and subscribing to it, see [Getting Started with Amazon SNS](#).

In a notification subscription, you can specify the type of source you want to be notified of and the AWS DMS source that triggers the event. You define the AWS DMS source type by using a **SourceType** value. You define the source generating the event by using a **SourceIdentifier** value. If you specify both **SourceType** and **SourceIdentifier**, such as `SourceType = db-instance` and `SourceIdentifier = myDBInstance1`, you receive all the `DB_Instance` events for the specified source. If you specify **SourceType** but not **SourceIdentifier**, you receive notice of the events for that source type for all your AWS DMS sources. If you don't specify either **SourceType** or **SourceIdentifier**, you are notified of events generated from all AWS DMS sources belonging to your customer account.

AWS Management Console

To subscribe to AWS DMS event notification by using the console

1. Sign in to the AWS Management Console and choose AWS DMS. 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 DMS.
2. In the navigation pane, choose **Event Subscriptions**.
3. On the **Event Subscriptions** page, choose **Create Event Subscription**.
4. On the **Create Event Subscription** page, do the following:
 - a. For **Name**, type a name for the event notification subscription.
 - b. Either choose an existing Amazon SNS topic for **Send notifications to**, or choose **create topic**. You must have either an existing Amazon SNS topic to send notices to or you must create the topic. If you choose **create topic**, you can enter an email address where notifications will be sent.
 - c. For **Source Type**, choose a source type. The only option is **replication instance**.
 - d. Choose **Yes** to enable the subscription. If you want to create the subscription but not have notifications sent yet, choose **No**.
 - e. Depending on the source type you selected, choose the event categories and sources you want to receive event notifications for.

The screenshot shows the 'Create Event Subscription' form in the AWS Management Console. The form is titled 'Create Event Subscription' and contains the following fields and options:

- Name***: Text input field containing 'DMS-subscription'.
- Topic name***: Text input field containing 'DMS-event-topic'. Below it is a 'Cancel' link.
- Recipient type***: Dropdown menu with 'Email' selected.
- With these recipients***: Text input field containing 'sgrayson@amazon.com'.
- Source Type***: Dropdown menu with 'Replication Instance' selected.
- Enabled***: Radio buttons for 'Yes' (selected) and 'No'.
- Event Categories**: Two sections, each with 'Select All' (selected) and 'Select Specific' radio buttons, and an empty selection box below.

- f. Choose **Create**.

The AWS DMS console indicates that the subscription is being created.

AWS DMS API

To subscribe to AWS DMS event notification by using the AWS DMS API

- Call `CreateEventSubscription`.

Migrating Large Data Stores Using AWS Database Migration Service and AWS Snowball

Large-scale data migrations can include many terabytes of information, and can be slowed by network performance and by the sheer amount of data that has to be moved. AWS DMS can load data onto an AWS Snowball device, transfer that data to AWS, and then load the data to the target AWS data store.

Using AWS DMS and the AWS Schema Conversion Tool (AWS SCT), you migrate your data in two stages. First, you use the AWS SCT to process the data locally and then move that data to the AWS Snowball Edge appliance. AWS Snowball then automatically loads the data into an Amazon S3 bucket. Next, when the data is available on Amazon S3, AWS DMS takes the files and migrates the data to the target data store. If you are using change data capture (CDC), those updates are written to the Amazon S3 bucket and the target data store is constantly updated.

AWS Snowball is an AWS service you can use to transfer data to the cloud at faster-than-network speeds using an AWS-owned appliance. An AWS Snowball Edge device can hold up to 100 TB of data. It uses 256-bit encryption and an industry-standard Trusted Platform Module (TPM) to ensure both security and full chain-of-custody for your data.

Amazon S3 is a storage and retrieval service. To store an object in Amazon S3, you upload the file you want to store to a bucket. When you upload a file, you can set permissions on the object and also on any metadata.

AWS DMS supports the following scenarios:

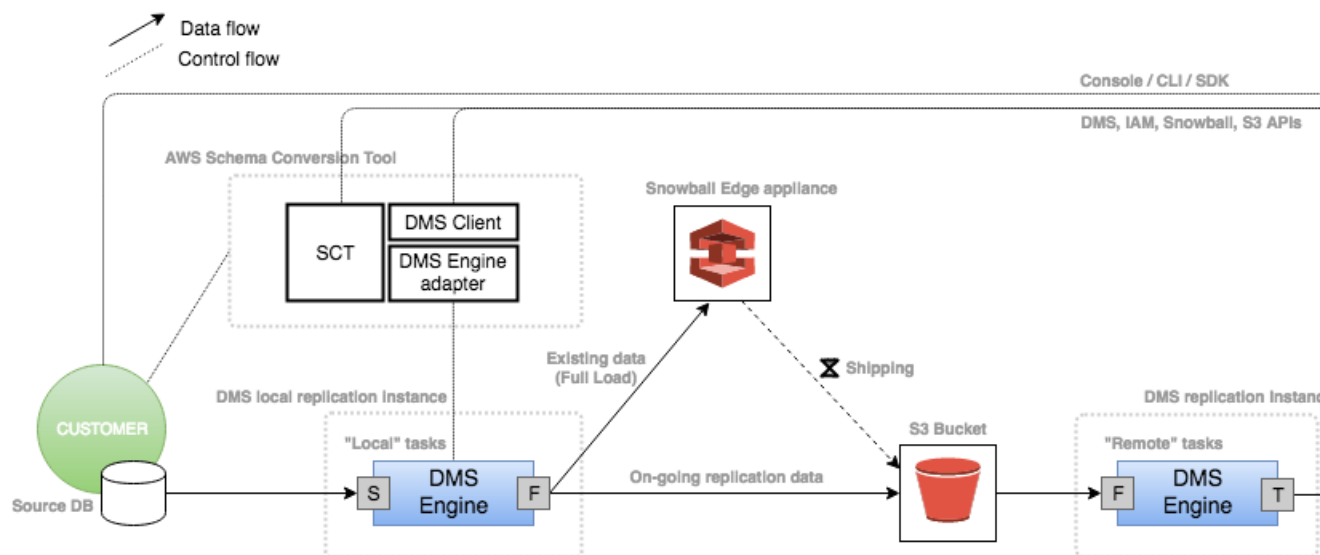
- Migration from an on-premises data warehouse to Amazon Redshift. This approach involves a client-side software installation of the AWS Schema Conversion Tool. The tool reads information from the warehouse (the extractor), and then moves data to S3 or Snowball. Then in the AWS Cloud, information is either read from S3 or Snowball and injected into Amazon Redshift.
- Migration from an on-premises relational database to an Amazon RDS database. This approach again involves a client-side software installation of the AWS Schema Conversion Tool. The tool reads information from a local database that AWS supports. The tool then moves data to S3 or Snowball. When the data is in the AWS Cloud, AWS DMS writes it to a supported database in either Amazon EC2 or Amazon RDS.

Process Overview

The process of using AWS DMS and AWS Snowball involves several steps, and it uses not only AWS DMS and AWS Snowball but also the AWS Schema Conversion Tool (AWS SCT). The sections following this overview provide a step-by-step guide to each of these tasks.

Note

We recommend that you test your migration before you use the AWS Snowball device. To do so, you can set up a task to send data, such as a single table, to an Amazon S3 bucket instead of the AWS Snowball device.



The migration involves a local task, where AWS SCT moves the data to the AWS Snowball Edge device, an intermediate action where the data is copied from the AWS Snowball Edge device to an S3 bucket. The process then involves a remote task where AWS DMS loads the data from the Amazon S3 bucket to the target data store on AWS.

The following steps need to occur to migrate data from a local data store to an AWS data store using AWS Snowball.

1. Create an AWS Snowball job using the AWS Snowball console. For more information, see [Create an Import Job](#) in the AWS Snowball documentation.
2. Download and install the AWS SCT application on a local machine. The machine must have network access and be able to access the AWS account to be used for the migration. For more information about the operating systems AWS SCT can be installed on, see [Installing and Updating the AWS Schema Conversion Tool](#).
3. Install the AWS SCT DMS Agent (DMS Agent) on a local, dedicated Linux machine. We recommend that you do not install the DMS Agent on the same machine that you install the AWS SCT application.
4. Unlock the AWS Snowball Edge device using the local, dedicated Linux machine where you installed the DMS Agent.
5. Create a new project in AWS SCT.
6. Configure the AWS SCT to use the DMS Agent.
7. Register the DMS Agent with the AWS SCT.
8. Install the database driver for your source database on the dedicated machine where you installed the DMS Agent.
9. Create and set permissions for the Amazon S3 bucket to use.
10. Edit the **AWS Service Profile** in AWS SCT.
11. Create **Local & DMS Task** in SCT.
12. Run and monitor the **Local & DMS Task** in SCT.
13. Run the AWS SCT task and monitor progress in SCT.

Step-by-Step Procedures for Migrating Data Using AWS DMS and AWS Snowball

The following sections provide detailed information on the migration steps.

Step 1: Create an AWS Snowball Job

Create an AWS Snowball job by following the steps outlined in the section [Getting Started with AWS Snowball Edge: Your First Job](#) in the AWS Snowball documentation.

Step 2: Download and Install the AWS Schema Conversion Tool (AWS SCT)

Download and install the AWS Schema Conversion Tool using the instructions at [Installing and Updating the AWS Schema Conversion Tool](#) in the AWS SCT documentation. Install the AWS SCT on a local machine that has access to AWS. This machine should be a different one than that the one where you plan to install the DMS Agent.

Step 3: Install and Configure the AWS SCT DMS Agent

In this step, you install the DMS Agent on a dedicated machine that has access to AWS and to the machine where AWS SCT was installed.

You can install the DMS Agent on the following Linux platforms:

- Red Hat Enterprise Linux versions 6.2 through 6.8, 7.0 and 7.1 (64-bit)
- SUSE Linux version 11.1 (64-bit)

To install the DMS Agent

1. Copy the RPM file called `aws-schema-conversion-tool-dms-agent-2.4.0-R2.x86_64.rpm` from the AWS SCT installation directory to a dedicated Linux machine.
2. Run the following command as root to install the DMS Agent. If you install or upgrade the DMS Agent on SUSE Linux, you must add the `--noexecs` parameter to the command.

```
sudo rpm -i aws-schema-conversion-tool-dms-agent-2.4.0-R2.x86_64.rpm
```

The default installation location is `/opt/amazon/aws-schema-conversion-tool-dms-agent`. To install the DMS Agent in a non-default directory, use `--prefix <path to new product dir>`.

3. To verify that the Amazon RDS Migration Server is running, issue the following command.

```
ps -ef | grep repctl
```

The output of this command should show two processes running.

To configure the DMS Agent, you must provide a password and port number. You use the password in AWS SCT, so keep it handy. The port is the one that the DMS Agent should listen on for AWS SCT connections. You might have to configure your firewall to allow connectivity.

Run the following script to configure the DMS Agent.

```
sudo /opt/amazon/aws-schema-conversion-tool-dms-agent/bin/configure.sh
```

Step 4: Unlock the AWS Snowball Edge Device

You should run the commands that unlock and provide credentials to the Snowball Edge device from the machine where you installed the DMS Agent. This way you can be sure that the DMS Agent call connect to the AWS Snowball Edge device. For more information about unlocking the AWS Snowball Edge device, see [Unlock the Snowball Edge](#) .

For example, the following command lists the Amazon S3 bucket used by the device.

```
aws s3 ls s3://<bucket-name> --profile <Snowball Edge profile> --endpoint http://<Snowball IP>:8080 --recursive
```

Step 5: Create a New AWS SCT Project

Next, you create a new AWS SCT project.

To create a new project in AWS SCT

1. Start AWS SCT, and choose **New Project** for **File**. The **New Project** dialog box appears.
2. Add the following project information.

For This Parameter	Do This
Project Name	Type a name for your project, which is stored locally on your computer.
Location	Type the location for your local project file.
OLTP	Choose Transactional Database (OLTP) .
Source DB Engine	Choose your source data store.
Target DB Engine	Choose your target data store.

3. Choose **OK** to create your AWS SCT project.
4. (Optional) Test your connection.

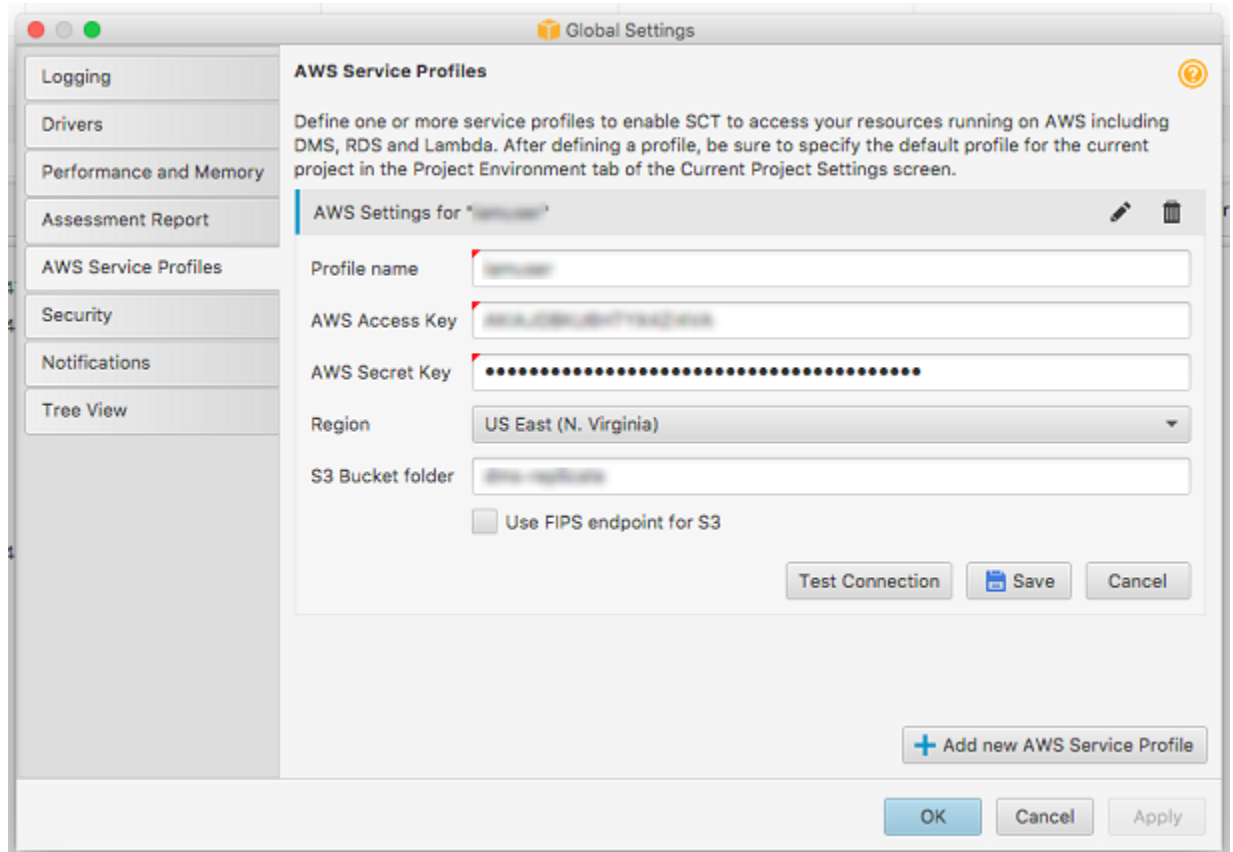
Step 6: Configure the AWS SCT Profile to Work with the DMS Agent

The AWS SCT Profile must be updated to use the DMS Agent.

To update the AWS SCT profile to work with the DMS Agent

1. Start AWS SCT.

2. Choose **Settings**, and then choose **Global Settings**. Choose **AWS Service Profiles**.
3. Choose **Add New AWS Service Profile**.



4. Add the following profile information.

For This Parameter	Do This
Profile Name	Type a name for your profile.
AWS Access Key	Type the AWS access key for the AWS account and AWS Region that you plan to use for the migration.
AWS Secret Key	Type the AWS secret key for the AWS account and AWS Region that you plan to use for the migration.
Region	Choose the AWS Region for the account you are using. Your DMS replication instance, S3 bucket, and target data store must be in this AWS Region.
S3 Bucket folder	Type a name for S3 bucket that you were assigned when you created the AWS Snowball job.

5. After you have entered the information, choose **Test Connection** to verify that AWS SCT can connect to the Amazon S3 bucket.

The **OLTP Local & DMS Data Migration** section in the pop-up window should show all entries with a status of **Pass**. If the test fails, the failure is probably because the account you are using does not have the correct privileges to access the Amazon S3 bucket.

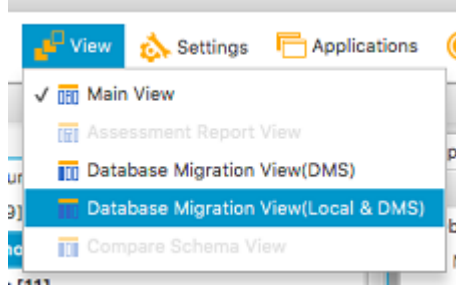
6. If the test passes, choose **OK** and then **OK** again to close the window and dialog box.

Step 7: Register the DMS Agent in AWS SCT

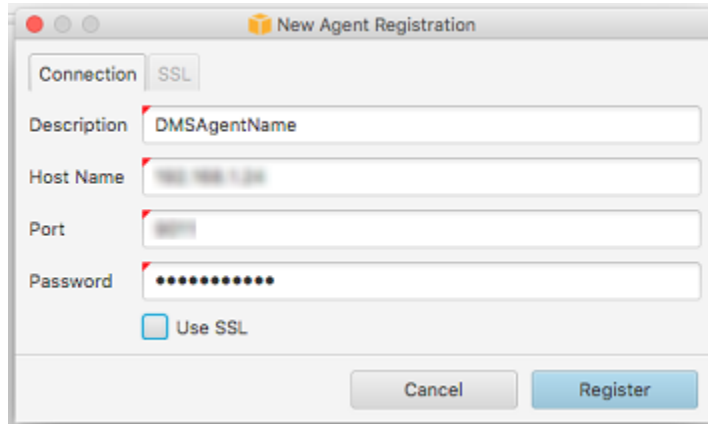
Next, you register the DMS Agent in AWS SCT. SCT then tries to connect to the agent, showing status. When the agent is available, the status turns to active.

To register the DMS Agent

1. Start AWS SCT, choose **View**, and then choose **Database Migration View (DMS)**.



2. Choose the **Agent** tab, and then choose **Register**. The **New Agent Registration** dialog box appears.



3. Type your information in the **New Agent Registration** dialog box.

For This Parameter	Do This
Description	Type the name of the agent, which is DMS Agent .
Host Name	Type the IP address of the machine where you installed the DMS Agent.
Port	Type the port number that you used when you configured the DMS Agent.
Password	Type the password that you used when you configured the DMS Agent.

4. Choose **Register** to register the agent with your AWS SCT project.

Step 8: Install the Source Database Driver for the DMS Agent on the Linux Computer

For the migration to succeed, the DMS Agent must be able to connect to the source database. To make this possible, you install the database driver for your source database. The required driver varies by database.

To restart the DMS Agent after database driver installation, change the working directory to `<product_dir>/bin` and use the steps listed following for each source database.

```
cd <product_dir>/bin
./arepctl stop
./arepctl start
```

To install on Oracle

Install Oracle Instant Client for Linux (x86-64) version 11.2.0.3.0 or later.

In addition, if not already included in your system, you need to create a symbolic link in the `$ORACLE_HOME/lib` directory. This link should be called `libclntsh.so`, and should point to a specific version of this file. For example, on an Oracle 12c client:

```
lrwxrwxrwx 1 oracle oracle 63 Oct 2 14:16 libclntsh.so ->
/u01/app/oracle/home/lib/libclntsh.so.12.1
```

In addition, the `LD_LIBRARY_PATH` environment variable should be appended with the Oracle lib directory and added to the `site_arep_login.sh` script under the lib folder of the installation. Add this script if it doesn't exist.

```
vi cat <product_dir>/bin/site_arep_login.sh
```

```
export ORACLE_HOME=/usr/lib/oracle/12.2/client64; export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib
```

To install on Microsoft SQL Server

Install the Microsoft ODBC Driver

Update the `site_arep_login.sh` with the following code.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/microsoft/msodbcsql/lib64/
```

Simba ODBC Driver

Install the Microsoft ODBC Driver

Edit the `simba.sqlserverodbc.ini` file as follows

```
DriverManagerEncoding=UTF-16
ODBCInstLib=libodbcinst.so
```

To install on SAP Sybase

The SAP Sybase ASE ODBC 64-bit client should be installed

If the installation dir is /opt/sap, update the site_arep_login.sh with

```
export SYBASE_HOME=/opt/sap
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SYBASE_HOME/
DataAccess64/ODBC/lib:$SYBASE_HOME/DataAccess/ODBC/
lib:$SYBASE_HOME/OCS-16_0/lib:$SYBASE_HOME/OCS-16_0/
lib3p64:$SYBASE_HOME/OCS-16_0/lib3p
```

The /etc/odbcinst.ini should include these entries

```
[Sybase]
Driver=/opt/sap/DataAccess64/ODBC/lib/libsybdrvodb.so
Description=Sybase ODBC driver
```

To install on MySQL

Install MySQL Connector/ODBC for Linux, version 5.2.6 or later

Make sure that the /etc/odbcinst.ini file contains an entry for MySQL, as in the following example

```
[MySQL ODBC 5.2.6 Unicode Driver]
Driver = /usr/lib64/libmyodbc5w.so
UsageCount = 1
```

To install on PostgreSQL

Install postgresql94-9.4.4-1PGDG.<OS Version>.x86_64.rpm. This is the package that contains the psql executable.

For example, postgresql94-9.4.4-1PGDG.rhel7.x86_64.rpm is the package required for Red Hat 7.

Install the ODBC driver postgresql94-odbc-09.03.0400-1PGDG.<OS version>.x86_64 or above for Linux, where <OS version> is the OS of the agent machine.

For example, postgresql94-odbc-09.03.0400-1PGDG.rhel7.x86_64 is the client required for Red Hat 7.

Make sure that the /etc/odbcinst.ini file contains an entry for PostgreSQL, as in the following example

```
[PostgreSQL]
Description = PostgreSQL ODBC driver
Driver = /usr/pgsql-9.4/lib/psqlodbc.so
Setup = /usr/pgsql-9.4/lib/psqlodbcw.so
Debug = 0
CommLog = 1
UsageCount = 2
```

Step 9: Configure AWS SCT to Access the Amazon S3 Bucket

For information on configuring an Amazon S3 bucket, see [Working with Amazon S3 Buckets](#) in the Amazon S3 documentation.

Note

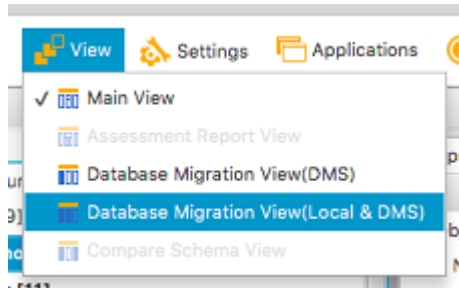
To use the resulting Amazon S3 bucket in migration, you must have an AWS DMS replication instance created in the same AWS Region as the S3 bucket. If you haven't already created one, do so by using the AWS DMS Management Console, as described in [Step 2: Create a Replication Instance](#) (p. 18).

Step 10: Creating a Local & DMS Task

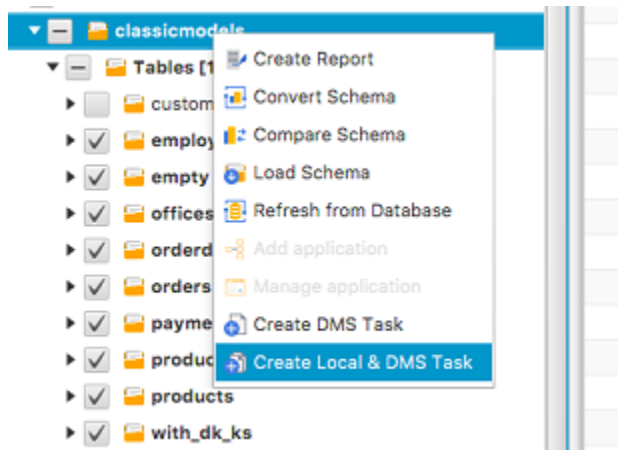
Next, you create the task that is the end-to-end migration task. The task includes two subtasks. One subtask migrates data from the source database to the AWS Snowball appliance. The other subtask takes the data that the appliance loads into an Amazon S3 bucket and migrates it to the target database.

To create the end-to-end migration task

1. Start AWS SCT, choose **View**, and then choose **Database Migration View (Local & DMS)**.



2. In the left panel that displays the schema from your source database, choose a schema object to migrate. Open the context (right-click) menu for the object, and then choose **Create Local & DMS Task**.



3. Add your task information.

For This Parameter	Do This
Task Name	Type a name for the task.

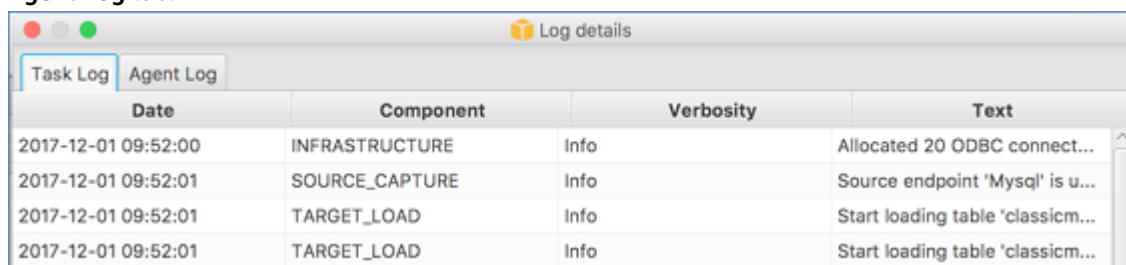
For This Parameter	Do This
Agent	Choose DMS Agent .
Replication Instance	Choose the AWS DMS replication instance that you want to use. You can only specify a Replication Instance that is version 2.4.3.
Migration Type	<p>Choose the type of migration you want.</p> <p>Choose Migrate existing data to migrate the contents of the chosen schema. This process is called a full load in AWS DMS.</p> <p>Choose Migrate existing data and replicate ongoing changes to migrate the contents of the chosen schema and capture all ongoing changes to the database. This process is called full load and CDC in AWS DMS.</p>
Target table preparation mode	<p>Choose the preparation mode you want to use.</p> <p>Truncate - Tables are truncated without affecting table metadata.</p> <p>Drop tables on target - The tables are dropped and new tables are created in their place.</p> <p>Do nothing - Data and metadata of the target tables are not changed.</p>
IAM role	Choose the predefined IAM role that has permissions to access the Amazon S3 bucket and the target database. For more information about the permissions required to access an Amazon S3 bucket, see Prerequisites When Using Amazon S3 as a Source for AWS DMS (p. 141) .
Logging	Choose Enable to have AWS CloudWatch create logs for the migration. You incur charges for this service. For more information about AWS CloudWatch, see How Amazon CloudWatch Works
Description	Type a description of the task.
Use Snowball	Choose this check box to use Snowball.
Job Name	Choose the AWS Snowball job name you created.
Snowball IP	Type the IP address of the AWS Snowball appliance.
Port	Type the port value for the AWS Snowball appliance.
Local AWS S3 Access key	Type the AWS access key for the account you are using for the migration.
Local AWS S3 Secret key	Type the AWS secret key for the account you are using for the migration.

4. Choose **Create** to create the task.

Step 11: Running and Monitoring the Local & DMS Task in SCT

You can start the Local & DMS Task when all connections to endpoints are successful. This means all connections for the Local task, which includes connections from the DMS Agent to the source database, the staging Amazon S3 bucket, and the AWS Snowball device, as well as the connections for the DMS task, which includes connections from the staging Amazon S3 bucket to the target database on AWS.

You can monitor the DMS Agent logs by choosing **Show Log**. The log details include agent server (**Agent Log**) and local running task (**Task Log**) logs. Because the endpoint connectivity is done by the server (since the local task is not running and there are no task logs), connection issues are listed under the **Agent Log** tab.






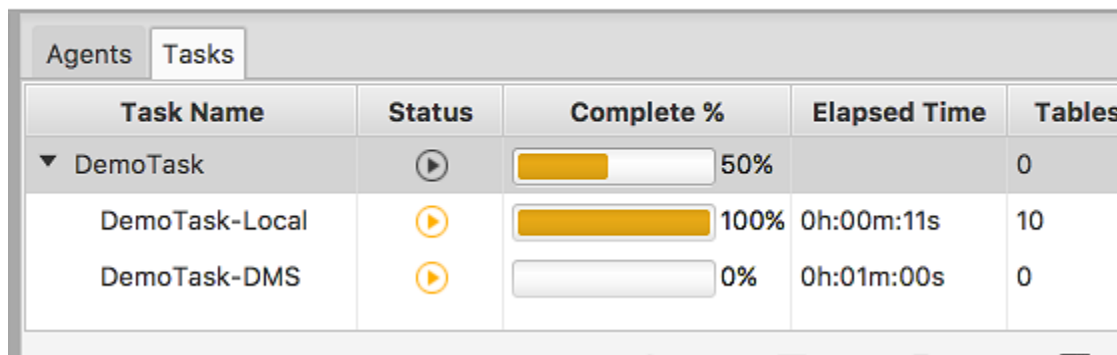
Date	Component	Verbosity	Text
2017-12-01 09:52:00	INFRASTRUCTURE	Info	Allocated 20 ODBC connect...
2017-12-01 09:52:01	SOURCE_CAPTURE	Info	Source endpoint 'Mysql' is u...
2017-12-01 09:52:01	TARGET_LOAD	Info	Start loading table 'classicm...
2017-12-01 09:52:01	TARGET_LOAD	Info	Start loading table 'classicm...




Step 12: Manage the AWS Snowball Appliance

Once the Snowball appliance is fully loaded, AWS SCT updates the status of the task to show it is 50% complete. Remember that the other part of the task involves AWS DMS taking the data from Amazon S3 to the target data store.

To do so, disconnect the AWS Snowball appliance and ship back to AWS. For more information about returning the AWS Snowball appliance to AWS, see the steps outlined in [Getting Started with AWS Snowball Edge: Your First Job](#) in the AWS Snowball documentation. You can use the AWS Snowball console or AWS SCT (show details of the DMS task) to check the status of the appliance and find out when AWS DMS begins to load data to the Amazon S3 bucket.

 Settings  Applications  Help



Task Name	Status	Complete %	Elapsed Time	Tables
▼ DemoTask		<div style="width: 50%;"><div style="background-color: #ffc107; height: 10px;"></div></div> 50%		0
DemoTask-Local		<div style="width: 100%;"><div style="background-color: #ffc107; height: 10px;"></div></div> 100%	0h:00m:11s	10
DemoTask-DMS		<div style="width: 0%;"><div style="background-color: #ffc107; height: 10px;"></div></div> 0%	0h:01m:00s	0

After the AWS Snowball appliance arrives at AWS and unloads data to S3 bucket, you can see that the remote (DMS) task starts to run. If the migration type you selected for the task was **Migrate existing data**, the status for the DMS task will show 100% complete when the data has been transferred from Amazon S3 to the target data store. If you set the a task mode to include ongoing replication, then after full load is complete the task status shows that the task continues to run, while AWS DMS applies ongoing changes.

Limitations When Working with AWS Snowball and AWS Database Migration Service (AWS DMS)

There are some limitations you should be aware of when working with AWS Snowball.

- The LOB mode limits LOB file size to 32K.
- If an error occurs during the data migration during the load from the local database to the AWS Snowball Edge device or when AWS DMS loads data from Amazon S3 to the target database, the task will restart if the error is recoverable. If AWS DMS cannot recover from the error, the migration will stop.
- Every AWS SCT task creates two endpoint connections on AWS DMS. If you create multiple tasks, you could reach a resource limit for the number of endpoints that can be created.

Troubleshooting Migration Tasks in AWS Database Migration Service

The following sections provide information on troubleshooting issues with AWS Database Migration Service (AWS DMS).

Topics

- [Slow Running Migration Tasks \(p. 297\)](#)
- [Task Status Bar Not Moving \(p. 298\)](#)
- [Missing Foreign Keys and Secondary Indexes \(p. 298\)](#)
- [Amazon RDS Connection Issues \(p. 298\)](#)
- [Networking Issues \(p. 298\)](#)
- [CDC Stuck After Full Load \(p. 299\)](#)
- [Primary Key Violation Errors When Restarting a Task \(p. 299\)](#)
- [Initial Load of Schema Fails \(p. 299\)](#)
- [Tasks Failing With Unknown Error \(p. 299\)](#)
- [Task Restart Loads Tables From the Beginning \(p. 300\)](#)
- [Number of Tables Per Task \(p. 300\)](#)
- [Troubleshooting Oracle Specific Issues \(p. 300\)](#)
- [Troubleshooting MySQL Specific Issues \(p. 302\)](#)
- [Troubleshooting PostgreSQL Specific Issues \(p. 306\)](#)
- [Troubleshooting Microsoft SQL Server Specific Issues \(p. 308\)](#)
- [Troubleshooting Amazon Redshift Specific Issues \(p. 309\)](#)
- [Troubleshooting Amazon Aurora MySQL Specific Issues \(p. 310\)](#)

Slow Running Migration Tasks

Several issues can cause a migration task to run slowly, or cause subsequent tasks to run slower than the initial task. The most common reason for a migration task running slowly is that there are inadequate resources allocated to the AWS DMS replication instance. Check your replication instance's use of CPU, memory, swap files, and IOPS to ensure that your instance has enough resources for the tasks you are running on it. For example, multiple tasks with Amazon Redshift as an endpoint are IO intensive. You can increase IOPS for your replication instance or split your tasks across multiple replication instances for a more efficient migration.

For more information about determining the size of your replication instance, see [Choosing the Optimum Size for a Replication Instance \(p. 314\)](#)

You can increase the speed of an initial migration load by doing the following:

- If your target is an Amazon RDS DB instance, ensure that Multi-AZ is not enabled for the target DB instance.
- Turn off any automatic backups or logging on the target database during the load, and turn back on those features once the migration is complete.
- If the feature is available on the target, use Provisioned IOPS.
- If your migration data contains LOBs, ensure that the task is optimized for LOB migration. See [Target Metadata Task Settings \(p. 227\)](#) for more information on optimizing for LOBs.

Task Status Bar Not Moving

The task status bar gives an estimation of the task's progress. The quality of this estimate depends on the quality of the source database's table statistics; the better the table statistics, the more accurate the estimation. For a task with only one table that has no estimated rows statistic, we are unable to provide any kind of percentage complete estimate. In this case, the task state and the indication of rows loaded can be used to confirm that the task is indeed running and making progress.

Missing Foreign Keys and Secondary Indexes

AWS DMS creates tables, primary keys, and in some cases unique indexes, but it doesn't create any other objects that are not required to efficiently migrate the data from the source. For example, it doesn't create secondary indexes, non-primary key constraints, or data defaults.

To migrate secondary objects from your database, use the database's native tools if you are migrating to the same database engine as your source database. Use the Schema Conversion Tool if you are migrating to a different database engine than that used by your source database to migrate secondary objects.

Amazon RDS Connection Issues

There can be several reasons why you are unable to connect to an Amazon RDS DB instance that you set as an endpoint. These include:

- Username and password combination is incorrect.
- Check that the endpoint value shown in the Amazon RDS console for the instance is the same as the endpoint identifier you used to create the AWS DMS endpoint.
- Check that the port value shown in the Amazon RDS console for the instance is the same as the port assigned to the AWS DMS endpoint.
- Check that the security group assigned to the Amazon RDS DB instance allows connections from the AWS DMS replication instance.
- If the AWS DMS replication instance and the Amazon RDS DB instance are not in the same VPC, check that the DB instance is publicly accessible.

Error Message: Incorrect thread connection string: incorrect thread value 0

This error can often occur when you are testing the connection to an endpoint. The error indicates that there is an error in the connection string, such as a space after the host IP address or a bad character was copied into the connection string.

Networking Issues

The most common networking issue involves the VPC security group used by the AWS DMS replication instance. By default, this security group has rules that allow egress to 0.0.0.0/0 on all ports. If you modify this security group or use your own security group, egress must, at a minimum, be permitted to the source and target endpoints on the respective database ports.

Other configuration related issues include:

- **Replication instance and both source and target endpoints in the same VPC** — The security group used by the endpoints must allow ingress on the database port from the replication instance. Ensure that the security group used by the replication instance has ingress to the endpoints, or you can create a rule in the security group used by the endpoints that allows the private IP address of the replication instance access.
- **Source endpoint is outside the VPC used by the replication instance (using Internet Gateway)** — The VPC security group must include routing rules that send traffic not destined for the VPC to the Internet Gateway. In this configuration, the connection to the endpoint appears to come from the public IP address on the replication instance.
- **Source endpoint is outside the VPC used by the replication instance (using NAT Gateway)** — You can configure a network address translation (NAT) gateway using a single Elastic IP Address bound to a single Elastic Network Interface which then receives a NAT identifier (nat-#####). If the VPC includes a default route to that NAT Gateway instead of the Internet Gateway, the replication instance will instead appear to contact the Database Endpoint using the public IP address of the Internet Gateway. In this case, the ingress to the Database Endpoint outside the VPC needs to allow ingress from the NAT address instead of the Replication Instance's public IP Address.

CDC Stuck After Full Load

Slow or stuck replication changes can occur after a full load migration when several AWS DMS settings conflict with each other. For example, if the **Target table preparation mode** parameter is set to **Do nothing** or **Truncate**, then you have instructed AWS DMS to do no setup on the target tables, including creating primary and unique indexes. If you haven't created primary or unique keys on the target tables, then AWS DMS must do a full table scan for each update, which can significantly impact performance.

Primary Key Violation Errors When Restarting a Task

This error can occur when data remains in the target database from a previous migration task. If the **Target table preparation mode** parameter is set to **Do nothing**, AWS DMS does not do any preparation on the target table, including cleaning up data inserted from a previous task. In order to restart your task and avoid these errors, you must remove rows inserted into the target tables from the previous running of the task.

Initial Load of Schema Fails

If your initial load of your schemas fails with an error of `Operation:getSchemaListDetails:errType=, status=0, errMessage=, errDetails=`, then the user account used by AWS DMS to connect to the source endpoint does not have the necessary permissions.

Tasks Failing With Unknown Error

The cause of these types of error can be varied, but often we find that the issue involves insufficient resources allocated to the AWS DMS replication instance. Check the replication instance's use of CPU,

memory, swap files, and IOPS to ensure your instance has enough resources to perform the migration. For more information on monitoring, see [Data Migration Service Metrics \(p. 265\)](#).

Task Restart Loads Tables From the Beginning

AWS DMS restarts table loading from the beginning when it has not finished the initial load of a table. When a task is restarted, AWS DMS does not reload tables that completed the initial load but will reload tables from the beginning when the initial load did not complete.

Number of Tables Per Task

While there is no set limit on the number of tables per replication task, we have generally found that limiting the number of tables in a task to less than 60,000 is a good rule of thumb. Resource use can often be a bottleneck when a single task uses more than 60,000 tables.

Troubleshooting Oracle Specific Issues

The following issues are specific to using AWS DMS with Oracle databases.

Topics

- [Pulling Data from Views \(p. 300\)](#)
- [Migrating LOBs from Oracle 12c \(p. 300\)](#)
- [Switching Between Oracle LogMiner and Binary Reader \(p. 301\)](#)
- [Error: Oracle CDC stopped 122301 Oracle CDC maximum retry counter exceeded. \(p. 301\)](#)
- [Automatically Add Supplemental Logging to an Oracle Source Endpoint \(p. 301\)](#)
- [LOB Changes not being Captured \(p. 302\)](#)
- [Error: ORA-12899: value too large for column <column-name> \(p. 302\)](#)
- [NUMBER data type being misinterpreted \(p. 302\)](#)

Pulling Data from Views

You can pull data once from a view; you cannot use it for ongoing replication. To be able to extract data from views, you must add the following code to the **Extra connection attributes** in the **Advanced** section of the Oracle source endpoint. Note that when you extract data from a view, the view is shown as a table on the target schema.

```
exposeViews=true
```

Migrating LOBs from Oracle 12c

AWS DMS can use two methods to capture changes to an Oracle database, Binary Reader and Oracle LogMiner. By default, AWS DMS uses Oracle LogMiner to capture changes. However, on Oracle 12c, Oracle LogMiner does not support LOB columns. To capture changes to LOB columns on Oracle 12c, use Binary Reader.

Switching Between Oracle LogMiner and Binary Reader

AWS DMS can use two methods to capture changes to a source Oracle database, Binary Reader and Oracle LogMiner. Oracle LogMiner is the default. To switch to using Binary Reader for capturing changes, do the following:

To use Binary Reader for capturing changes

1. Sign in to the AWS Management Console and select DMS.
2. Select **Endpoints**.
3. Select the Oracle source endpoint that you want to use Binary Reader.
4. Select **Modify**.
5. Select **Advanced**, and then add the following code to the Extra connection attributes text box:

```
useLogminerReader=N
```

6. Use an Oracle developer tool such as SQL-Plus to grant the following additional privilege to the AWS DMS user account used to connect to the Oracle endpoint:

```
SELECT ON V_$TRANSPORTABLE_PLATFORM
```

Error: Oracle CDC stopped 122301 Oracle CDC maximum retry counter exceeded.

This error occurs when the needed Oracle archive logs have been removed from your server before AWS DMS was able to use them to capture changes. Increase your log retention policies on your database server. For an Amazon RDS database, run the following procedure to increase log retention. For example, the following code increases log retention on an Amazon RDS DB instance to 24 hours.

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

Automatically Add Supplemental Logging to an Oracle Source Endpoint

By default, AWS DMS has supplemental logging turned off. To automatically turn on supplemental logging for a source Oracle endpoint, do the following:

To add supplemental logging to a source Oracle endpoint

1. Sign in to the AWS Management Console and select **DMS**.
2. Select **Endpoints**.
3. Select the Oracle source endpoint that you want to add supplemental logging to.

4. Select **Modify**.
5. Select **Advanced**, and then add the following code to the **Extra connection attributes** text box:

```
addSupplementalLogging=Y
```

6. Choose **Modify**.

LOB Changes not being Captured

Currently, a table must have a primary key for AWS DMS to capture LOB changes. If a table that contains LOBs doesn't have a primary key, there are several actions you can take to capture LOB changes:

- Add a primary key to the table. This can be as simple as adding an ID column and populating it with a sequence using a trigger.
- Create a materialized view of the table that includes a system generated ID as the primary key and migrate the materialized view rather than the table.
- Create a logical standby, add a primary key to the table, and migrate from the logical standby.

Error: ORA-12899: value too large for column <column-name>

The error "ORA-12899: value too large for column <column-name>" is often caused by a mismatch in the character sets used by the source and target databases or when NLS settings differ between the two databases. A common cause of this error is when the source database NLS_LENGTH_SEMANTICS parameter is set to CHAR and the target database NLS_LENGTH_SEMANTICS parameter is set to BYTE.

NUMBER data type being misinterpreted

The Oracle NUMBER data type is converted into various AWS DMS datatypes, depending on the precision and scale of NUMBER. These conversions are documented here [Using an Oracle Database as a Source for AWS DMS \(p. 84\)](#). The way the NUMBER type is converted can also be affected by using extra connection attributes for the source Oracle endpoint. These extra connection attributes are documented in [Extra Connection Attributes When Using Oracle as a Source for AWS DMS \(p. 93\)](#).

Troubleshooting MySQL Specific Issues

The following issues are specific to using AWS DMS with MySQL databases.

Topics

- [CDC Task Failing for Amazon RDS DB Instance Endpoint Because Binary Logging Disabled \(p. 303\)](#)
- [Connections to a target MySQL instance are disconnected during a task \(p. 303\)](#)
- [Adding Autocommit to a MySQL-compatible Endpoint \(p. 303\)](#)
- [Disable Foreign Keys on a Target MySQL-compatible Endpoint \(p. 304\)](#)
- [Characters Replaced with Question Mark \(p. 304\)](#)
- ["Bad event" Log Entries \(p. 304\)](#)

- [Change Data Capture with MySQL 5.5 \(p. 304\)](#)
- [Increasing Binary Log Retention for Amazon RDS DB Instances \(p. 305\)](#)
- [Log Message: Some changes from the source database had no impact when applied to the target database. \(p. 305\)](#)
- [Error: Identifier too long \(p. 305\)](#)
- [Error: Unsupported Character Set Causes Field Data Conversion to Fail \(p. 305\)](#)
- [Error: Codepage 1252 to UTF8 \[120112\] A field data conversion failed \(p. 306\)](#)

CDC Task Failing for Amazon RDS DB Instance Endpoint Because Binary Logging Disabled

This issue occurs with Amazon RDS DB instances because automated backups are disabled. Enable automatic backups by setting the backup retention period to a non-zero value.

Connections to a target MySQL instance are disconnected during a task

If you have a task with LOBs that is getting disconnected from a MySQL target with the following type of errors in the task log, you might need to adjust some of your task settings.

```
[TARGET_LOAD ]E: RetCode: SQL_ERROR SqlState: 08S01 NativeError:  
2013 Message: [MySQL][ODBC 5.3(w) Driver][mysqld-5.7.16-log]Lost connection  
to MySQL server during query [122502] ODBC general error.
```

```
[TARGET_LOAD ]E: RetCode: SQL_ERROR SqlState: HY000 NativeError:  
2006 Message: [MySQL][ODBC 5.3(w) Driver]MySQL server has gone away  
[122502] ODBC general error.
```

To solve the issue where a task is being disconnected from a MySQL target, do the following:

- Check that you have your database variable `max_allowed_packet` set large enough to hold your largest LOB.
- Check that you have the following variables set to have a large timeout value. We suggest you use a value of at least 5 minutes for each of these variables.
 - `net_read_timeout`
 - `net_write_timeout`
 - `wait_timeout`
 - `interactive_timeout`

Adding Autocommit to a MySQL-compatible Endpoint

To add autocommit to a target MySQL-compatible endpoint

1. Sign in to the AWS Management Console and select **DMS**.
2. Select **Endpoints**.

3. Select the MySQL-compatible target endpoint that you want to add autocommit to.
4. Select **Modify**.
5. Select **Advanced**, and then add the following code to the **Extra connection attributes** text box:

```
Initstmt= SET AUTOCOMMIT=1
```

6. Choose **Modify**.

Disable Foreign Keys on a Target MySQL-compatible Endpoint

You can disable foreign key checks on MySQL by adding the following to the **Extra Connection Attributes** in the **Advanced** section of the target MySQL, Amazon Aurora with MySQL compatibility, or MariaDB endpoint.

To disable foreign keys on a target MySQL-compatible endpoint

1. Sign in to the AWS Management Console and select **DMS**.
2. Select **Endpoints**.
3. Select the MySQL, Aurora MySQL, or MariaDB target endpoint that you want to disable foreign keys.
4. Select **Modify**.
5. Select **Advanced**, and then add the following code to the **Extra connection attributes** text box:

```
Initstmt=SET FOREIGN_KEY_CHECKS=0
```

6. Choose **Modify**.

Characters Replaced with Question Mark

The most common situation that causes this issue is when the source endpoint characters have been encoded by a character set that AWS DMS doesn't support. For example, AWS DMS does not support the UTF8MB4 character set.

"Bad event" Log Entries

"Bad event" entries in the migration logs usually indicate that an unsupported DDL operation was attempted on the source database endpoint. Unsupported DDL operations cause an event that the replication instance cannot skip so a bad event is logged. To fix this issue, restart the task from the beginning, which will reload the tables and will start capturing changes at a point after the unsupported DDL operation was issued.

Change Data Capture with MySQL 5.5

AWS DMS change data capture (CDC) for Amazon RDS MySQL-compatible databases requires full image row-based binary logging, which is not supported in MySQL version 5.5 or lower. To use AWS DMS CDC, you must upgrade your Amazon RDS DB instance to MySQL version 5.6.

Increasing Binary Log Retention for Amazon RDS DB Instances

AWS DMS requires the retention of binary log files for change data capture. To increase log retention on an Amazon RDS DB instance, use the following procedure. The following example increases the binary log retention to 24 hours.

```
Call mysql.rds_set_configuration('binlog retention hours', 24);
```

Log Message: Some changes from the source database had no impact when applied to the target database.

When AWS DMS updates a MySQL database column's value to its existing value, a message of zero rows affected is returned from MySQL. This behavior is unlike other database engines such as Oracle and SQL Server that perform an update of one row, even when the replacing value is the same as the current one.

Error: Identifier too long

The following error occurs when an identifier is too long:

```
TARGET_LOAD E: RetCode: SQL_ERROR SqlState: HY000 NativeError:  
1059 Message: MySQLhttp://ODBC 5.3(w) Driverhttp://mysqld-5.6.10Identifier  
name '<name>' is too long 122502 ODBC general error. (ar_odbc_stmt.c:4054)
```

When AWS DMS is set to create the tables and primary keys in the target database, it currently does not use the same names for the Primary Keys that were used in the source database. Instead, AWS DMS creates the Primary Key name based on the tables name. When the table name is long, the auto-generated identifier created can be longer than the allowed limits for MySQL. To solve this issue, currently, pre-create the tables and Primary Keys in the target database and use a task with the task setting **Target table preparation mode** set to **Do nothing** or **Truncate** to populate the target tables.

Error: Unsupported Character Set Causes Field Data Conversion to Fail

The following error occurs when an unsupported character set causes a field data conversion to fail:

```
"[SOURCE_CAPTURE ]E: Column '<column name>' uses an unsupported character set [120112]  
A field data conversion failed. (mysql_endpoint_capture.c:2154)
```

This error often occurs because of tables or databases using UTF8MB4 encoding. AWS DMS does not support the UTF8MB4 character set. In addition, check your database's parameters related to connections. The following command can be used to see these parameters:

```
SHOW VARIABLES LIKE '%char%';
```

Error: Codepage 1252 to UTF8 [120112] A field data conversion failed

The following error can occur during a migration if you have non codepage-1252 characters in the source MySQL database.

```
[SOURCE_CAPTURE ]E: Error converting column 'column_xyz' in table  
'table_xyz with codepage 1252 to UTF8 [120112] A field data conversion failed.  
(mysql_endpoint_capture.c:2248)
```

As a workaround, you can use the `CharsetMapping` extra connection attribute with your source MySQL endpoint to specify character set mapping. You might need to restart the AWS DMS migration task from the beginning if you add this extra connection attribute.

For example, the following extra connection attribute could be used for a MySQL source endpoint where the source character set is `utf8` or `latin1`. 65001 is the UTF8 code page identifier.

```
CharsetMapping=utf8,65001  
CharsetMapping=latin1,65001
```

Troubleshooting PostgreSQL Specific Issues

The following issues are specific to using AWS DMS with PostgreSQL databases.

Topics

- [JSON data types being truncated \(p. 306\)](#)
- [Columns of a user defined data type not being migrated correctly \(p. 307\)](#)
- [Error: No schema has been selected to create in \(p. 307\)](#)
- [Deletes and updates to a table are not being replicated using CDC \(p. 307\)](#)
- [Truncate statements are not being propagated \(p. 307\)](#)
- [Preventing PostgreSQL from capturing DDL \(p. 307\)](#)
- [Selecting the schema where database objects for capturing DDL are created \(p. 308\)](#)
- [Oracle tables missing after migrating to PostgreSQL \(p. 308\)](#)
- [Task Using View as a Source Has No Rows Copied \(p. 308\)](#)

JSON data types being truncated

AWS DMS treats the JSON data type in PostgreSQL as an LOB data type column. This means that the LOB size limitation when you use Limited LOB mode applies to JSON data. For example, if Limited LOB mode is set to 4096 KB, any JSON data larger than 4096 KB is truncated at the 4096 KB limit and will fail the validation test in PostgreSQL.

For example, the following log information shows JSON that was truncated due to the Limited LOB mode setting and failed validation.

```
03:00:49
2017-09-19T03:00:49 [TARGET_APPLY ]E: Failed to execute statement:
  'UPDATE "public"."delivery_options_quotes" SET "id"=? , "enabled"=? ,
  "new_cart_id"=? , "order_id"=? , "user_id"=? , "zone_id"=? , "quotes"=? ,
  "start_at"=? , "end_at"=? , "last_quoted_at"=? , "created_at"=? ,
  "updated_at"=? WHERE "id"=? ' [1022502] (ar_odbc_stmt
2017-09-19T03:00:49 [TARGET_APPLY ]E: Failed to execute statement:
  'UPDATE "public"."delivery_options_quotes" SET "id"=? , "enabled"=? ,
  "new_cart_id"=? , "order_id"=? , "user_id"=? , "zone_id"=? , "quotes"=? ,
  "start_at"=? , "end_at"=? , "last_quoted_at"=? , "created_at"=? ,
  "updated_at"=? WHERE "id"=? ' [1022502] (ar_odbc_stmt.c:2415)
#
03:00:49
2017-09-19T03:00:49 [TARGET_APPLY ]E: RetCode: SQL_ERROR SqlState:
  22P02 NativeError: 1 Message: ERROR: invalid input syntax for type json;,
  Error while executing the query [1022502] (ar_odbc_stmt.c:2421)
2017-09-19T03:00:49 [TARGET_APPLY ]E: RetCode: SQL_ERROR SqlState:
  22P02 NativeError: 1 Message: ERROR: invalid input syntax for type json;,
  Error while executing the query [1022502] (ar_odbc_stmt.c:2421)
```

Columns of a user defined data type not being migrated correctly

When replicating from a PostgreSQL source, AWS DMS creates the target table with the same data types for all columns, apart from columns with user-defined data types. In such cases, the data type is created as "character varying" in the target.

Error: No schema has been selected to create in

The error "SQL_ERROR SqlState: 3F000 NativeError: 7 Message: ERROR: no schema has been selected to create in" can occur when your JSON table mapping contains a wild card value for the schema but the source database doesn't support that value.

Deletes and updates to a table are not being replicated using CDC

Delete and Update operations during change data capture (CDC) are ignored if the source table does not have a primary key. AWS DMS supports change data capture (CDC) for PostgreSQL tables with primary keys; if a table does not have a primary key, the WAL logs do not include a before image of the database row and AWS DMS cannot update the table. Create a primary key on the source table if you want delete operations to be replicated.

Truncate statements are not being propagated

When using change data capture (CDC), TRUNCATE operations are not supported by AWS DMS.

Preventing PostgreSQL from capturing DDL

You can prevent a PostgreSQL target endpoint from capturing DDL statements by adding the following **Extra Connection Attribute** statement. The **Extra Connection Attribute** parameter is available in the **Advanced** tab of the source endpoint.

```
captureDDIs=N
```

Selecting the schema where database objects for capturing DDL are created

You can control what schema the database objects related to capturing DDL are created in. Add the following **Extra Connection Attribute** statement. The **Extra Connection Attribute** parameter is available in the **Advanced** tab of the target endpoint.

```
ddlArtifactsSchema=xyzddlSchema
```

Oracle tables missing after migrating to PostgreSQL

Oracle defaults to uppercase table names while PostgreSQL defaults to lowercase table names. When performing a migration from Oracle to PostgreSQL you will most likely need to supply transformation rules under the table mapping section of your task to convert the case of your table names.

Your tables and data are still accessible; if you migrated your tables without using transformation rules to convert the case of your table names, you will need to enclose your table names in quotes when referencing them.

Task Using View as a Source Has No Rows Copied

A View as a PostgreSQL source endpoint is not supported by AWS DMS.

Troubleshooting Microsoft SQL Server Specific Issues

The following issues are specific to using AWS DMS with Microsoft SQL Server databases.

Topics

- [Special Permissions for AWS DMS user account to use CDC \(p. 308\)](#)
- [Errors Capturing Changes for SQL Server Database \(p. 309\)](#)
- [Missing Identity Columns \(p. 309\)](#)
- [Error: SQL Server Does Not Support Publications \(p. 309\)](#)
- [Changes Not Appearing in Target \(p. 309\)](#)

Special Permissions for AWS DMS user account to use CDC

The user account used with AWS DMS requires the SQL Server SysAdmin role in order to operate correctly when using change data capture (CDC). CDC for SQL Server is only available for on-premises databases or databases on an EC2 instance.

Errors Capturing Changes for SQL Server Database

Errors during change data capture (CDC) can often indicate that one of the pre-requisites was not met. For example, the most common overlooked pre-requisite is a full database backup. The task log indicates this omission with the following error:

```
SOURCE_CAPTURE E: No FULL database backup found (under the 'FULL' recovery model).  
To enable all changes to be captured, you must perform a full database backup.  
120438 Changes may be missed. (sqlserver_log_queries.c:2623)
```

Review the pre-requisites listed for using SQL Server as a source in [Using a Microsoft SQL Server Database as a Source for AWS DMS \(p. 100\)](#).

Missing Identity Columns

AWS DMS does not support identity columns when you create a target schema. You must add them after the initial load has completed.

Error: SQL Server Does Not Support Publications

The following error is generated when you use SQL Server Express as a source endpoint:

```
RetCode: SQL_ERROR SqlState: HY000 NativeError: 21106  
Message: This edition of SQL Server does not support publications.
```

AWS DMS currently does not support SQL Server Express as a source or target.

Changes Not Appearing in Target

AWS DMS requires that a source SQL Server database be in either 'FULL' or 'BULK LOGGED' data recovery model in order to consistently capture changes. The 'SIMPLE' model is not supported.

The SIMPLE recovery model logs the minimal information needed to allow users to recover their database. All inactive log entries are automatically truncated when a checkpoint occurs. All operations are still logged, but as soon as a checkpoint occurs the log is automatically truncated, which means that it becomes available for re-use and older log entries can be over-written. When log entries are overwritten, changes cannot be captured, and that is why AWS DMS doesn't support the SIMPLE data recovery model. For information on other required pre-requisites for using SQL Server as a source, see [Using a Microsoft SQL Server Database as a Source for AWS DMS \(p. 100\)](#).

Troubleshooting Amazon Redshift Specific Issues

The following issues are specific to using AWS DMS with Amazon Redshift databases.

Topics

- [Loading into a Amazon Redshift Cluster in a Different Region Than the AWS DMS Replication Instance \(p. 310\)](#)
- [Error: Relation "awsdms_apply_exceptions" already exists \(p. 310\)](#)
- [Errors with Tables Whose Name Begins with "awsdms_changes" \(p. 310\)](#)
- [Seeing Tables in Cluster with Names Like dms.awsdms_changes0000000000XXXX \(p. 310\)](#)

- [Permissions Required to Work with Amazon Redshift \(p. 310\)](#)

Loading into a Amazon Redshift Cluster in a Different Region Than the AWS DMS Replication Instance

This can't be done. AWS DMS requires that the AWS DMS replication instance and a Redshift cluster be in the same region.

Error: Relation "awsdms_apply_exceptions" already exists

The error "Relation "awsdms_apply_exceptions" already exists" often occurs when a Redshift endpoint is specified as a PostgreSQL endpoint. To fix this issue, modify the endpoint and change the **Target engine** to "redshift."

Errors with Tables Whose Name Begins with "awsdms_changes"

Error messages that relate to tables with names that begin with "awsdms_changes" often occur when two tasks that are attempting to load data into the same Amazon Redshift cluster are running concurrently. Due to the way temporary tables are named, concurrent tasks can conflict when updating the same table.

Seeing Tables in Cluster with Names Like dms.awsdms_changes000000000XXXX

AWS DMS creates temporary tables when data is being loaded from files stored in S3. The name of these temporary tables have the prefix "dms.awsdms_changes." These tables are required so AWS DMS can store data when it is first loaded and before it is placed in its final target table.

Permissions Required to Work with Amazon Redshift

To use AWS DMS with Amazon Redshift, the user account you use to access Amazon Redshift must have the following permissions:

- CRUD (Select, Insert, Update, Delete)
- Bulk Load
- Create, Alter, Drop (if required by the task's definition)

To see all the pre-requisites required for using Amazon Redshift as a target, see [Using an Amazon Redshift Database as a Target for AWS Database Migration Service \(p. 163\)](#).

Troubleshooting Amazon Aurora MySQL Specific Issues

The following issues are specific to using AWS DMS with Amazon Aurora MySQL databases.

Topics

- [Error: CHARACTER SET UTF8 fields terminated by ',' enclosed by '"' lines terminated by '\n' \(p. 311\)](#)

Error: CHARACTER SET UTF8 fields terminated by ',' enclosed by '"' lines terminated by '\n'

If you are using Amazon Aurora MySQL as a target and see an error like the following in the logs, this usually indicates that you have ANSI_QUOTES as part of the SQL_MODE parameter. Having ANSI_QUOTES as part of the SQL_MODE parameter causes double quotes to be handled like quotes and can create issues when you run a task. To fix this error, remove ANSI_QUOTES from the SQL_MODE parameter.

```
2016-11-02T14:23:48 [TARGET_LOAD ]E: Load data sql statement. load data local infile
"/rdsdbdata/data/tasks/7XO4FJHCVON7TYTLQ6RX3CQH DU/data_files/4/LOAD000001DF.csv" into
table
`VOSPUSER`.`SANDBOX_SRC_FILE` CHARACTER SET UTF8 fields terminated by ','
enclosed by '"' lines terminated by '\n' ( `SANDBOX_SRC_FILE_ID`, `SANDBOX_ID`,
`FILENAME`, `LOCAL_PATH`, `LINES_OF_CODE`, `INSERT_TS`, `MODIFIED_TS`, `MODIFIED_BY`,
`RECORD_VER`, `REF_GUID`, `PLATFORM_GENERATED`, `ANALYSIS_TYPE`, `SANITIZED`, `DYN_TYPE`,
`CRAWL_STATUS`, `ORIG_EXEC_UNIT_VER_ID` ); (provider_syntax_manager.c:2561)
```

Best Practices for AWS Database Migration Service

To use AWS Database Migration Service (AWS DMS) most effectively, see this section's recommendations on the most efficient way to migrate your data.

Topics

- [Improving the Performance of an AWS DMS Migration \(p. 312\)](#)
- [Choosing the Optimum Size for a Replication Instance \(p. 314\)](#)
- [Reducing the Load on Your Source Database \(p. 315\)](#)
- [Using the Task Log to Troubleshoot Migration Issues \(p. 315\)](#)
- [Converting Schema \(p. 315\)](#)
- [Migrating Large Binary Objects \(LOBs\) \(p. 315\)](#)
- [Ongoing Replication \(p. 316\)](#)
- [Changing the User and Schema for an Oracle Target \(p. 317\)](#)
- [Improving Performance When Migrating Large Tables \(p. 317\)](#)

Improving the Performance of an AWS DMS Migration

A number of factors affect the performance of your AWS DMS migration:

- Resource availability on the source
- The available network throughput
- The resource capacity of the replication server
- The ability of the target to ingest changes
- The type and distribution of source data
- The number of objects to be migrated

In our tests, we've migrated a terabyte of data in approximately 12 to 13 hours using a single AWS DMS task and under ideal conditions. These ideal conditions included using source databases running on Amazon EC2 and in Amazon RDS with target databases in Amazon RDS, all in the same Availability Zone. Our source databases contained a representative amount of relatively evenly distributed data with a few large tables containing up to 250 GB of data. The source data didn't contain complex data types, such as BLOB.

You can improve performance by using some or all of the best practices mentioned following. Whether you can use one of these practices or not depends in large part on your specific use case. We mention limitations as appropriate.

Loading Multiple Tables in Parallel

By default, AWS DMS loads eight tables at a time. You might see some performance improvement by increasing this slightly when using a very large replication server, such as a `dms.c4.xlarge` or larger

instance. However, at some point, increasing this parallelism reduces performance. If your replication server is relatively small, such as a `dms.t2.medium`, we recommend that you reduce the number of tables loaded in parallel.

To change this number in the AWS Management Console, open the console, choose **Tasks**, choose to create or modify a task, and then choose **Advanced Settings**. Under **Tuning Settings**, change the **Maximum number of tables to load in parallel** option.

To change this number using the AWS CLI, change the `MaxFullLoadSubTasks` parameter under `TaskSettings`.

Working with Indexes, Triggers and Referential Integrity Constraints

Indexes, triggers, and referential integrity constraints can affect your migration performance and cause your migration to fail. How these affect migration depends on whether your replication task is a full load task or an ongoing replication (CDC) task.

For a full load task, we recommend that you drop primary key indexes, secondary indexes, referential integrity constraints, and data manipulation language (DML) triggers. Alternatively, you can delay their creation until after the full load tasks are complete. You don't need indexes during a full load task and indexes will incur maintenance overhead if they are present. Because the full load task loads groups of tables at a time, referential integrity constraints are violated. Similarly, insert, update, and delete triggers can cause errors, for example, if a row insert is triggered for a previously bulk loaded table. Other types of triggers also affect performance due to added processing.

You can build primary key and secondary indexes before a full load task if your data volumes are relatively small and the additional migration time doesn't concern you. Referential integrity constraints and triggers should always be disabled.

For a full load + CDC task, we recommend that you add secondary indexes before the CDC phase. Because AWS DMS uses logical replication, secondary indexes that support DML operations should be in-place to prevent full table scans. You can pause the replication task before the CDC phase to build indexes, create triggers, and create referential integrity constraints before you restart the task.

Disable Backups and Transaction Logging

When migrating to an Amazon RDS database, it's a good idea to disable backups and Multi-AZ on the target until you're ready to cut over. Similarly, when migrating to non-Azure RDS systems, disabling any logging on the target until after cut over is usually a good idea.

Use Multiple Tasks

Sometimes using multiple tasks for a single migration can improve performance. If you have sets of tables that don't participate in common transactions, you might be able to divide your migration into multiple tasks. Transactional consistency is maintained within a task, so it's important that tables in separate tasks don't participate in common transactions. Additionally, each task independently reads the transaction stream, so be careful not to put too much stress on the source database.

You can use multiple tasks to create separate streams of replication to parallelize the reads on the source, the processes on the replication instance, and the writes to the target database.

Optimizing Change Processing

By default, AWS DMS processes changes in a transactional mode, which preserves transactional integrity. If you can afford temporary lapses in transactional integrity, you can use the batch optimized apply option instead. This option efficiently groups transactions and applies them in batches for efficiency purposes. Using the batch optimized apply option almost always violates referential integrity constraints, so you should disable these during the migration process and enable them again as part of the cut over process.

Choosing the Optimum Size for a Replication Instance

Choosing the appropriate replication instance depends on several factors of your use case. To help understand how replication instance resources are used, see the following discussion. It covers the common scenario of a full load + CDC task.

During a full load task, AWS DMS loads tables individually. By default, eight tables are loaded at a time. AWS DMS captures ongoing changes to the source during a full load task so the changes can be applied later on the target endpoint. The changes are cached in memory; if available memory is exhausted, changes are cached to disk. When a full load task completes for a table, AWS DMS immediately applies the cached changes to the target table.

After all outstanding cached changes for a table have been applied, the target endpoint is in a transactionally consistent state. At this point, the target is in-sync with the source endpoint with respect to the last cached changes. AWS DMS then begins ongoing replication between the source and target. To do so, AWS DMS takes change operations from the source transaction logs and applies them to the target in a transactionally consistent manner (assuming batch optimized apply is not selected). AWS DMS streams ongoing changes through memory on the replication instance, if possible. Otherwise, AWS DMS writes changes to disk on the replication instance until they can be applied on the target.

You have some control over how the replication instance handles change processing, and how memory is used in that process. For more information on how to tune change processing, see [Change Processing Tuning Settings \(p. 233\)](#).

From the preceding explanation, you can see that total available memory is a key consideration. If the replication instance has sufficient memory that AWS DMS can stream cached and ongoing changes without writing them to disk, migration performance increases greatly. Similarly, configuring the replication instance with enough disk space to accommodate change caching and log storage also increases performance. The maximum IOPs possible depend on the selected disk size.

Consider the following factors when choosing a replication instance class and available disk storage:

- Table size – Large tables take longer to load and so transactions on those tables must be cached until the table is loaded. After a table is loaded, these cached transactions are applied and are no longer held on disk.
- Data manipulation language (DML) activity – A busy database generates more transactions. These transactions must be cached until the table is loaded. Transactions to an individual table are applied as soon as possible after the table is loaded, until all tables are loaded.
- Transaction size – Long-running transactions can generate many changes. For best performance, if AWS DMS applies changes in transactional mode, sufficient memory must be available to stream all changes in the transaction.
- Total size of the migration – Large migrations take longer and they generate a proportionally large number of log files.
- Number of tasks – The more tasks, the more caching is likely to be required, and the more log files are generated.
- Large objects – Tables with LOBs take longer to load.

Anecdotal evidence shows that log files consume the majority of space required by AWS DMS. The default storage configurations are usually sufficient.

However, replication instances that run several tasks might require more disk space. Additionally, if your database includes large and active tables, you might need to increase disk space for transactions that are cached to disk during a full load task. For example, if your load takes 24 hours and you produce 2GB of

transactions each hour, you might want to ensure that you have 48GB of space for cached transactions. Also, the more storage space you allocate to the replication instance, the higher the IOPS you get.

The guidelines preceding don't cover all possible scenarios. It's critically important to consider the specifics of your particular use case when you determine the size of your replication instance. After your migration is running, monitor the CPU, freeable memory, storage free, and IOPS of your replication instance. Based on the data you gather, you can size your replication instance up or down as needed.

Reducing the Load on Your Source Database

AWS DMS uses some resources on your source database. During a full load task, AWS DMS performs a full table scan of the source table for each table processed in parallel. Additionally, each task you create as part of a migration queries the source for changes as part of the CDC process. For AWS DMS to perform CDC for some sources, such as Oracle, you might need to increase the amount of data written to your database's change log.

If you find you are overburdening your source database, you can reduce the number of tasks or tables for each task for your migration. Each task gets source changes independently, so consolidating tasks can decrease the change capture workload.

Using the Task Log to Troubleshoot Migration Issues

In some cases, AWS DMS can encounter issues for which warnings or error messages appear only in the task log. In particular, data truncation issues or row rejections due to foreign key violations are only written in the task log. Therefore, be sure to review the task log when migrating a database. To enable viewing of the task log, configure Amazon CloudWatch as part of task creation.

Converting Schema

AWS DMS doesn't perform schema or code conversion. If you want to convert an existing schema to a different database engine, you can use the AWS Schema Conversion Tool (AWS SCT). AWS SCT converts your source objects, table, indexes, views, triggers, and other system objects into the target data definition language (DDL) format. You can also use AWS SCT to convert most of your application code, like PL/SQL or TSQL, to the equivalent target language.

You can get AWS SCT as a free download from AWS. For more information on AWS SCT, see the [AWS Schema Conversion Tool User Guide](#).

If your source and target endpoints are on the same database engine, you can use tools such as Oracle SQL Developer, MySQL Workbench, or PgAdmin4 to move your schema.

Migrating Large Binary Objects (LOBs)

In general, AWS DMS migrates LOB data in two phases.

1. AWS DMS creates a new row in the target table and populates the row with all data except the associated LOB value.
2. AWS DMS updates the row in the target table with the LOB data.

This migration process for LOBs requires that, during the migration, all LOB columns on the target table must be nullable. This is so even if the LOB columns aren't nullable on the source table. If AWS DMS creates the target tables, it sets LOB columns to nullable by default. If you create the target tables using some other mechanism, such as import or export, you must ensure that the LOB columns are nullable before you start the migration task.

This requirement has one exception. Suppose that you perform a homogeneous migration from an Oracle source to an Oracle target, and you choose **Limited Lob mode**. In this case, the entire row is populated at once, including any LOB values. For such a case, AWS DMS can create the target table LOB columns with not-nullable constraints, if needed.

Using Limited LOB Mode

AWS DMS uses two methods that balance performance and convenience when your migration contains LOB values.

- **Limited LOB mode** migrates all LOB values up to a user-specified size limit (default is 32 KB). LOB values larger than the size limit must be manually migrated. **Limited LOB mode**, the default for all migration tasks, typically provides the best performance. However you need to ensure that the **Max LOB size** parameter setting is correct. This parameter should be set to the largest LOB size for all your tables.
- **Full LOB mode** migrates all LOB data in your tables, regardless of size. **Full LOB mode** provides the convenience of moving all LOB data in your tables, but the process can have a significant impact on performance.

For some database engines, such as PostgreSQL, AWS DMS treats JSON data types like LOBs. Make sure that if you have chosen **Limited LOB mode** the **Max LOB size** option is set to a value that doesn't cause the JSON data to be truncated.

AWS DMS provides full support for using large object data types (BLOBs, CLOBs, and NCLOBs). The following source endpoints have full LOB support:

- Oracle
- Microsoft SQL Server
- ODBC

The following target endpoints have full LOB support:

- Oracle
- Microsoft SQL Server

The following target endpoint has limited LOB support. You can't use an unlimited LOB size for this target endpoint.

- Amazon Redshift

For endpoints that have full LOB support, you can also set a size limit for LOB data types.

Ongoing Replication

AWS DMS provides ongoing replication of data, keeping the source and target databases in sync. It replicates only a limited amount of data definition language (DDL). AWS DMS doesn't propagate items

such as indexes, users, privileges, stored procedures, and other database changes not directly related to table data.

If you plan to use ongoing replication, you should enable the **Multi-AZ** option when you create your replication instance. By choosing the **Multi-AZ** option you get high availability and failover support for the replication instance. However, this option can have an impact on performance.

Changing the User and Schema for an Oracle Target

When using Oracle as a target, AWS DMS assumes that the data should be migrated into the schema and user that is used to connect to the target. If you want to migrate data to a different schema, use a schema transformation to do so. Schema in Oracle is connected to the username that is used in the endpoint connection. In order to read from an X schema and write into X schema on the target, we need to rename the X schema (being read from source) and instruct AWS DMS to write data into X schema on target.

For example, if you want to migrate from the user source schema PERFDATA to the target data PERFDATA, you'll need to create a transformation as follows:

```
{
  "rule-type": "transformation",
  "rule-id": "2",
  "rule-name": "2",
  "rule-action": "rename",
  "rule-target": "schema",
  "object-locator": {
    "schema-name": "PERFDATA"
  },
  "value": "PERFDATA"
}
```

For more information about transformations, see [Specifying Table Selection and Transformations by Table Mapping Using JSON \(p. 250\)](#).

Improving Performance When Migrating Large Tables

If you want to improve the performance when migrating a large table, you can break the migration into more than one task. To break the migration into multiple tasks using row filtering, use a key or a partition key. For example, if you have an integer primary key ID from 1 to 8,000,000, you can create eight tasks using row filtering to migrate 1 million records each.

To apply row filtering in the AWS Management Console, open the console, choose **Tasks**, and create a new task. In the **Table mappings** section, add a value for **Selection Rule**. You can then add a column filter with either a less than or equal to, greater than or equal to, equal to, or range condition (between two values). For more information about column filtering, see [Specifying Table Selection and Transformations by Table Mapping from the Console \(p. 245\)](#).

Alternatively, if you have a large partitioned table that is partitioned by date, you can migrate data based on date. For example, suppose that you have a table partitioned by month, and only the current month's

data is updated. In this case, you can create a full load task for each static monthly partition and create a full load + CDC task for the currently updated partition.

AWS DMS Reference

In this reference section, you can find additional information you might need when using AWS Database Migration Service (AWS DMS), including data type conversion information.

AWS DMS maintains data types when you do a homogenous database migration where both source and target use the same engine type. When you do a heterogeneous migration, where you migrate from one database engine type to a different database engine, data types are converted to an intermediate data type. To see how the data types appear on the target database, consult the data type tables for the source and target database engines.

Be aware of a few important things about data types when migrating a database:

- The UTF-8 4-byte character set (utf8mb4) isn't supported and can cause unexpected behavior in a source database. Plan to convert any data using the UTF-8 4-byte character set before migrating.
- The FLOAT data type is inherently an approximation. When you insert a specific value in FLOAT, it might be represented differently in the database. This difference is because FLOAT isn't an exact data type, such as a decimal data type like NUMBER or NUMBER(p,s). As a result, the internal value of FLOAT stored in the database might be different than the value that you insert. Thus, the migrated value of a FLOAT might not match exactly the value in the source database.

For more information on this issue, see the following articles:

- [IEEE floating point](#) in Wikipedia
- [IEEE Floating-Point Representation](#) on MSDN
- [Why Floating-Point Numbers May Lose Precision](#) on MSDN

Topics

- [Data Types for AWS Database Migration Service \(p. 319\)](#)

Data Types for AWS Database Migration Service

AWS Database Migration Service uses built-in data types to migrate data from one database engine type to a different database engine type. The following table shows the built-in data types and their descriptions.

AWS DMS Data Types	Description
STRING	A character string.
WSTRING	A double-byte character string.
BOOLEAN	A Boolean value.
BYTES	A binary data value.
DATE	A date value: year, month, day.
TIME	A time value: hour, minutes, seconds.
DATETIME	A timestamp value: year, month, day, hour, minute, second, fractional seconds. The fractional seconds have a maximum scale of 9 digits.

AWS DMS Data Types	Description
INT1	A one-byte, signed integer.
INT2	A two-byte, signed integer.
INT4	A four-byte, signed integer.
INT8	An eight-byte, signed integer.
NUMERIC	An exact numeric value with a fixed precision and scale.
REAL4	A single-precision floating-point value.
REAL8	A double-precision floating-point value.
UINT1	A one-byte, unsigned integer.
UINT2	A two-byte, unsigned integer.
UINT4	A four-byte, unsigned integer.
UINT8	An eight-byte, unsigned integer.
BLOB	Binary large object. This data type can be used only with Oracle endpoints.
CLOB	Character large object.
NCLOB	Native character large object.

AWS DMS Release Notes

Following, you can find release notes for versions of AWS Database Migration Service (AWS DMS).

AWS Database Migration Service (AWS DMS) 3.1.2 Release Notes

The following tables show the features and bug fixes for version 3.1.2 of AWS Database Migration Service (AWS DMS).

New Feature or Enhancement	Description
Support for latency recalculation when there is a daylight saving time change	You can now recalculate time offsets during a daylight saving time change when using Oracle or PostgreSQL as a source.

The issues resolved are as follows.

Date Reported	Description
July 19, 2018	Fixed an issue where PostgreSQL as a source was sending <code>Null</code> values as <code>Empty</code> to Oracle during change data capture in <code>Full LOB</code> mode.
September 19, 2018	Fixed an issue where <code>Null</code> values in SQL Server <code>varchar</code> columns were migrated differently to all targets.
October 7, 2018	Fixed an issue where the <code>LOB</code> setting didn't work when transformation rules were present.
October 12, 2018	Fixed an issue where ongoing replication tasks with Oracle as a source failed to resume after a stop in certain cases.
October 12, 2018	Fixed an issue where ongoing replication tasks with SQL Server as a source failed to resume after a stop in certain cases.
Multiple dates	Fixed multiple issues with PostgreSQL as a source that were present in version 3.1.1.

AWS Database Migration Service (AWS DMS) 3.1.1 Release Notes

The following tables show the features and bug fixes for version 3.1.1 of AWS Database Migration Service (AWS DMS).

New Feature or Enhancement	Description
Migration of 4-byte UTF8 characters	AWS DMS now supports all 4-byte character sets, such as UTF8MB4, and so on. This feature works without any configuration changes.
Support for Microsoft SQL Server 2017 as a source	Added support for SQL Server 2017 as a source. For more details, see Using a Microsoft SQL Server Database as a Source for AWS DMS (p. 100) .
Support for parallel full load of tables	Added support for parallel full load of large tables based on partitions and subpartitions. This feature uses a separate unload thread for each table partition or subpartition to speed up the bulk load process. You can also specify specific ranges or partitions to migrate a subset of the table data. Supported sources are Oracle, SQL Server, Sybase, MySQL, and IBM Db2 for Linux, UNIX, PostgreSQL, and Windows (Db2 LUW). For more information, see Parallel Loading of Tables (p. 229) .
Control large object (LOB) settings per table	You can now control LOB settings per table with additional table mapping settings. For more information, see Target Metadata Task Settings (p. 227) . Supported sources are Oracle, SQL Server, MySQL, and PostgreSQL.
Control the order for loading tables in a single migration task	You can now control the order for loading tables with table mappings in a migration task. You can specify the order by tagging the table with a load-order unsigned integer in the table mappings. Tables with higher load-order values are migrated first. For more information, see Using Table Mapping to Specify Task Settings (p. 245) .
Support for updates to primary key values when using PostgreSQL as a source	Updates to primary key values are now replicated when you use PostgreSQL as a source for ongoing replication.

The issues resolved are as follows.

Date Reported	Description
April 24, 2018	Fixed an issue where users couldn't create Azure SQL as a source endpoint for SQL Server 2016.
May 5, 2018	Fixed an issue where CHR(0) in an Oracle source was migrated as CHR(32) in an Aurora with MySQL compatibility target.
May 10, 2018	Fixed an issue where ongoing replication from Oracle as a source didn't work as expected when using Oracle LogMiner to migrate changes from an Oracle physical standby.
May 27, 2018	Fixed an issue where characters of various data types in PostgreSQL were tripled during migration to PostgreSQL.
June 12, 2018	Fixed an issue where data was changed during a migration from TEXT to NCLOB (PostgreSQL to Oracle) due to differences in how these engines handle nulls within a string.
June 17, 2018	Fixed an issue where the replication task failed to created primary keys in target MySQL version 5.5 instance when migrating from a source MySQL version 5.5.

Date Reported	Description
June 23, 2018	Fixed an issue where JSON columns were truncated in full LOB mode when migrating from a PostgreSQL instance to Aurora with PostgreSQL compatibility.
June 27, 2018	Fixed an issue where batch application of changes to PostgreSQL as a target failed because of an issue creating the intermediate net changes table on the target.
June 30, 2018	Fixed an issue where the MySQL timestamp '0000-00-00 00:00:00' wasn't migrated as expected while performing a full load.
July 2, 2018	Fixed an issue where a DMS replication task didn't continue as expected after the source Aurora MySQL failover occurred.
July 9, 2018	Fixed an issue with a migration from MySQL to Amazon Redshift where the task failed with an unknown column and data type error.
July 21, 2018	Fixed an issue where null characters in a string migrated differently from SQL Server to PostgreSQL in limited LOB and full LOB modes.
July 23, 2018	Fixed an issue where the safeguard transactions in SQL Server as a source filled up the transaction log in SQL Server.
July 26, 2018	Fixed an issue where null values were migrated as empty values in a roll-forward migration from PostgreSQL to Oracle.
Multiple dates	Fixed various logging issues to keep users more informed about migration by Amazon CloudWatch logs.
Multiple dates	Fixed various data validation issues.

AWS Database Migration Service (AWS DMS) 2.4.4 Release Notes

The following tables show the features and bug fixes for version 2.4.4 of AWS Database Migration Service (AWS DMS).

New Feature or Enhancement	Description
Validation for migrations with filter clauses	You can now validate data when migrating a subset of a table using table filters.
Open Database Connectivity (ODBC) driver upgrades	The underlying ODBC driver for MySQL was upgraded to 5.3.11-1, and the underlying ODBC driver for Amazon Redshift was upgraded to 1.4.2-1010.
Latency recalculation in case of daylight saving time changes	You can now recalculate the time offset during daylight saving time changes for Oracle and PostgreSQL as a source. Source and target latency calculations are accurate after the daylight saving time change.
UUID data type conversion (SQL Server to MySQL)	You can now convert a <code>UNIQUEIDENTIFIER</code> data type (that is, a universally unique identifier or UUID) to bytes when migrating between SQL Server as a source and MySQL as a target.

New Feature or Enhancement	Description
Ability to change encryption modes for Amazon S3 as a source and Amazon Redshift as a target	You can now change encryption mode when migrating between S3 as a source and Amazon Redshift as a target. You specify the encryption mode with a connection attribute. Server side encryption and AWS KMS are both supported.

The issues resolved are as follows.

Date Reported	Description
July 17, 2018	Fixed an issue where PostgreSQL as a source sent null values as empty values to target Oracle databases during change data capture (CDC) in full large binary object (LOB) mode.
July 29, 2018	Fixed an issue where migration tasks to and from Amazon S3 failed to resume after upgrading from DMS version 1.9.0.
August 5, 2018	Fixed an issue where the <code>ResumeFetchForXRows</code> extra connection attribute was not working properly with a <code>VARCHAR</code> primary key for a MySQL source.
September 12, 2018	Fixed an issue where DMS working with SQL Server safeguard transactions blocked the transaction log from being reused.
September 21, 2018	Fixed an issue with failed bulk loads from PostgreSQL as a source to Amazon Redshift as a target. The failed tasks did not report a failure when the full load was interrupted.
October 3, 2018	Fixed an issue where a DMS migration task didn't fail when prerequisites for ongoing replication weren't properly configured for SQL Server as a source.
Multiple dates	Fixed multiple issues related to data validation, and added enhanced support for validating multibyte UTF-8 characters.

AWS Database Migration Service (AWS DMS) 2.4.3 Release Notes

The following tables show the features and bug fixes for version 2.4.3 of AWS Database Migration Service (AWS DMS).

New Feature or Enhancement	Description
Table metadata recreation on mismatch	<p>Added a new extra connect attribute for MySQL endpoints: <code>CleanSrcMetadataOnMismatch</code>.</p> <p>This is a Boolean attribute that cleans and recreates table metadata information on the replication instance when a mismatch occurs. An example is where running an alter statement in data definition language (DDL) on a table might result in different information about</p>

New Feature or Enhancement	Description
	the table cached in the replication instance. By default, this attribute is set to false.
Performance improvements for data validation	<p>Improvements include the following:</p> <ul style="list-style-type: none"> • Data validation now partitions data before it starts so it can compare like partitions and validate them. This version contains improvements to changes to fetch the bulk of partitions, which speeds up the partitioning time and makes data validation faster. • Improvements to handle collation differences automatically based on task settings. • Improvements to identify false positives during validation, which also reduces false positives during cached changes phase. • General logging improvements to data validation. • Improvements to queries used by data validation when tables have composite primary keys.

The issues resolved are as follows.

Date Reported	Description
February 12, 2018	Fixed an issue in ongoing replication using batch apply where AWS DMS was missing some inserts as a unique constraint in the table was being updated.
March 16, 2018	Fixed an issue where an Oracle to PostgreSQL migration task was crashing during the ongoing replication phase due to Multi-AZ failover on the source Amazon RDS for Oracle instance.

AWS Database Migration Service (AWS DMS) 2.4.2 Release Notes

The following tables show the features and bug fixes for version 2.4.2 of AWS Database Migration Service (AWS DMS).

New Feature or Enhancement	Description
Binary Reader support for Amazon RDS for Oracle during change data capture	Added support for using Binary Reader in change data capture (CDC) scenarios from an Amazon RDS for Oracle source during ongoing replication.
Additional COPY command parameters for Amazon Redshift as a target	<p>Introduced support for the following additional Amazon Redshift copy parameters using extra connection attributes. For more information, see Extra Connection Attributes When Using Amazon Redshift as a Target for AWS DMS (p. 166).</p> <ul style="list-style-type: none"> • TRUNCATECOLUMNS

New Feature or Enhancement	Description
	<ul style="list-style-type: none"> REMOVEQUOTES TRIMBLANKS
Option to fail a migration task when a table is truncated in a PostgreSQL source	Introduced support to fail a task when a truncate is encountered in a PostgreSQL source when using a new task setting. For more information, see the <code>ApplyErrorFailOnTruncationDdl</code> setting in the section Error Handling Task Settings (p. 234) .
Validation support for JSON/JSONB/HSTORE in PostgreSQL endpoints	Introduced data validation support for JSON, JSONB, and HSTORE columns for PostgreSQL as a source and target.
Improved logging for MySQL sources	Improved log visibility for issues when reading MySQL binary logs (binlogs) during change data capture (CDC). Logs now clearly show an error or warning if there are issues accessing MySQL source binlogs during CDC.
Additional data validation statistics	Added more replication table statistics. For more information, see Replication Task Statistics (p. 273) .

The issues resolved are as follows.

Date Reported	Description
January 14, 2018	Fixed all issues with respect to handling zero dates (0000-00-00) to MySQL targets during full load and CDC. MySQL doesn't accept 0000-00-00 (invalid in MySQL) although some engines do. All these dates become 0101-01-01 for a MySQL target.
January 21, 2018	Fixed an issue where migration fails when migrating a table with table name containing a \$ sign.
February 3, 2018	Fixed an issue where a JSON column from a PostgreSQL source was truncated when migrated to any supported target.
February 12, 2018	Fixed an issue where migration task was failing after a failover in Aurora MySQL target.
February 21, 2018	Fixed an issue where a migration task couldn't start its ongoing replication phase after a network connectivity issue.
February 23, 2018	Fixed an issue where certain transformation rules in table mappings were causing migration task crashes during ongoing replication to Amazon Redshift targets.
Reported on multiple dates	<p>Fixed various data validation issues:</p> <ul style="list-style-type: none"> Fixed wide string handling issues in Oracle source and target validation. Handle validation when a column is removed for a table in table mappings. Improved validation performance for sources with a high rate of change.

AWS Database Migration Service (AWS DMS) 2.4.1 Release Notes

The following tables show the features and bug fixes for version 2.4.1 of AWS Database Migration Service (AWS DMS).

New Feature or Enhancement	Description
JSONB support for PostgreSQL sources	Introduced support for JSONB migration from PostgreSQL as a source. JSONB is treated as a LOB data type and requires appropriate LOB settings to be used.
HSTORE support for PostgreSQL sources	Introduced support for HSTORE data type migration from PostgreSQL as a source. HSTORE is treated as a LOB data type and requires appropriate LOB settings to be used.
Additional COPY command parameters for Amazon Redshift as a target	Introduced support for the following additional copy parameters by using these extra connection attributes: <ul style="list-style-type: none"> • ACCEPTANYDATE • DATEFORMAT • TIMEFORMAT • EMPTYASNULL

The issues resolved are as follows.

Date Reported	Description
July 12, 2017	Fixed an issue where migration task hung before the full load phase starts when reading from an Oracle table with TDE column encryption enabled.
October 3, 2017	Fixed an issue where a JSON column from a PostgreSQL source didn't migrate as expected.
October 5, 2017	Fixed an issue when DMS migration task shows 0 source latency when an archive redo log file is not found on the source Oracle instance. This fix linearly increases source latency under such conditions.
November 20, 2017	Fixed an issue with LOB migration where a TEXT column in PostgreSQL was migrating to a CLOB column in Oracle with extra spaces after each character in the LOB entry.
November 20, 2017	Fixed an issue with a migration task not stopping as expected after an underlying replication instance upgrade from version 1.9.0 to 2.4.0.
November 30, 2017	Fixed an issue where a DMS migration task doesn't properly capture changes made by a copy command run on a source PostgreSQL instance.
December 11, 2017	Fixed an issue where a migration task failed when reading change data from a nonexistent binlog from a MySQL source.

Date Reported	Description
December 11, 2017	Fixed an issue where DMS is reading change data from a nonexistent table from a MySQL source.
December 20, 2017	Includes several fixes and enhancements for the data validation feature.
December 22, 2017	Fixed an issue with <code>maxFileSize</code> parameter for Amazon Redshift targets. This parameter was wrongly being interpreted as bytes instead of kilobytes.
January 4, 2018	Fixed a memory allocation bug for an Amazon DynamoDB as a target migration tasks. In certain conditions, AWS DMS didn't allocate enough memory if the object mapping being used contained a sort key.
January 10, 2018	Fixed an issue with Oracle 12.2 as a source where data manipulation language (DML) statements weren't captured as expected when <code>ROWDEPENDENCIES</code> are used.

AWS Database Migration Service (AWS DMS) 2.4.0 Release Notes

The following tables show the features and bug fixes for version 2.4.0 of AWS Database Migration Service (AWS DMS).

New Feature or Enhancement	Description
Replicating Oracle index tablespaces	Adds functionality to support replication of Oracle index tablespaces. More details about index tablespaces can be seen here .
Support for cross-account Amazon S3 access	<p>Adds functionality to support canned ACLs (predefined grants) to support cross-account access with S3 endpoints. Find more details about canned ACLs in Canned ACL in the <i>Amazon Simple Storage Service Developer Guide</i>.</p> <p>Usage: Set an extra connect attribute in S3 endpoint, that is <code>CannedAclForObjects=value</code>. Possible values are as follows:</p> <ul style="list-style-type: none"> • NONE • PRIVATE • PUBLIC_READ • PUBLIC_READ_WRITE • AUTHENTICATED_READ • AWS_EXEC_READ • BUCKET_OWNER_READ • BUCKET_OWNER_FULL_CONTROL

The issues resolved are as follows.

Date Reported	Description
July 19, 2017	Fixed an issue where replication task runs in a retry loop forever when a source PostgreSQL instance runs out of replication slots. With this fix, the task fails with an error reporting that DMS can't create a logical replication slot.
July 27, 2017	Fixed an issue in the replication engine where the enum MySQL data type caused task failure with a memory allocation error.
August 7, 2017	Fixed an issue that caused unexpected behavior with migration tasks with Oracle as a source when the source is down for more than five minutes. This issue caused the ongoing replication phase to hang even after the source became available.
August 24, 2017	Fixed an issue with PostgreSQL target where the fraction part in the TIME data type was handled incorrectly.
September 14, 2017	Fixed an issue where incorrect values were being written to TOAST fields in PostgreSQL-based targets during updates in the CDC phase.
October 8, 2017	Fixed an issue from version 2.3.0 where ongoing replication with MySQL 5.5 sources would not work as expected.
October 12, 2017	Fixed an issue with reading changes from a SQL Server 2016 source during the ongoing replication phase. This fix needs to be used with the following extra connect attribute in the source SQL Server endpoint: <code>IgnoreTxnCtxValidityCheck=true</code>

AWS Database Migration Service (AWS DMS) 2.3.0 Release Notes

The following tables show the features and bug fixes for version 2.3.0 of AWS Database Migration Service (AWS DMS).

New Feature or Enhancement	Description
S3 as a source	Using Amazon Simple Storage Service as a Source for AWS DMS (p. 138)
SQL Azure as a source	Using Microsoft Azure SQL Database as a Source for AWS DMS (p. 109)
Platform – AWS SDK update	Update to the AWS SDK in the replication instance to 1.0.113. The AWS SDK is used for certain endpoints (such as Amazon Redshift and S3) to upload data on customers' behalf into these endpoints. Usage is unrestricted.
Oracle source: Support replication of tablespace in Oracle	Ability to migrate tablespaces from an Oracle source eliminating the need to precreate tablespaces in the target before migration. Usage: Use the <code>ReadTableSpaceName</code> setting in the extra connect attributes in the Oracle source endpoint and set it to true to support tablespace replication. This option is set to false by default.

New Feature or Enhancement	Description
Oracle source: CDC support for Oracle Active Data Guard standby as a source	<p>Ability to use a standby instance for Oracle Active Data Guard as a source for replicating ongoing changes to a supported target. This change eliminates the need to connect to an active database that might be in production.</p> <p>Usage: Use the <code>StandbyDelayTime</code> setting in the extra connect attributes in the Oracle source endpoint and specify time in minutes to specify the delay in standby sync.</p>
PostgreSQL source: add WAL heartbeat	<p>Added a write-ahead log (WAL) heartbeat (that is, running dummy queries) for replication from a PostgreSQL source. This feature was added so that idle logical replication slots don't hold onto old WAL logs, which can result in storage full situations on the source. This heartbeat keeps <code>restart_lsn</code> moving and prevents storage full scenarios.</p> <p>Usage, wherever applicable, is as follows:</p> <ul style="list-style-type: none"> • <code>HeartbeatEnable</code> is set to true (default is false). • <code>HeartbeatSchema</code> is the schema for heartbeat artifacts (default is public). • <code>HeartbeatFrequency</code> is the heartbeat frequency in minutes (default is 5 and minimum value is 1)
All endpoints: Maintain homogeneous replication with transformation	<p>Ability to do like-to-like migrations for homogeneous migration tasks (from a table structure/data type perspective) came in 2.2.0. However, DMS still converted data types internally when a task was launched with table transformations. This feature maintains data types from the source on the target for homogeneous lift-and-shift migrations, even when transformations are used.</p> <p>Usage is unrestricted for all homogeneous migrations.</p>
All endpoints: Fail task when no tables are found	<p>Ability to force replication task failure when include transformation rules find no matches.</p> <p>Usage: Change the <code>FailOnNoTablesCaptured</code> task setting to true.</p>
Oracle source: Stop task when archive redo log is missing	<p>Ability to come out of a retry loop and stop a task when the archive redo log on the source is missing.</p> <p>Usage: Use the <code>RetryTimeoutInMinutes</code> extra connect attribute to specify the stop timeout in minutes.</p>

The issues resolved are as follows.

Date Reported	Description
January 5, 2017	Fixed a server ID collision issue when launching multiple DMS tasks to the same MySQL instance (version 5.6+)
February 21, 2017	Fixed an issue where table creation fails for <code>nestingLevel=ONE</code> when <code>_id</code> in MongoDB is a string in the document. Before this fix, <code>_id</code> (when

Date Reported	Description
	a string) was being created as a LONGTEXT (MySQL) or CLOB (Oracle), which causes a failure when it tries to make it a primary key.
May 5, 2017	Fixed an issue where NULL LOBs were migrating as empty when using full LOB mode with an Oracle source.
May 5, 2017	Fixed an issue where a task with MySQL as a source fails with a too many connections error when custom CDC start time is older than 24 hours.
May 24, 2017	Fixed an issue where task was in the starting status for too long when multiple tasks were launched on the replication instance at one time.
July 7, 2017	Fixed an issue that caused a PostgreSQL error message about using all available connection slots to appear. Now an error is logged in the default logging level when all available connection slots to PostgreSQL are used up and DMS can't get more slots to continue with replication.
July 19, 2017	Fixed an issue where updates and deletes from Oracle to DynamoDB were not being migrated correctly.
August 8, 2017	Fixed an issue that caused unexpected behavior during CDC when an Oracle source database instance went down for more than five minutes during a migration.
August 12, 2017	Fixed an issue where nulls from any source were being migrated as <code>amazon_null</code> , causing issues when inserted into data types other than <code>varchar</code> in Amazon Redshift.
August 27, 2017	For MongoDB, fixed an issue where a full load task crashes when <code>nestingLevel=NONE</code> and <code>_id</code> is not <code>ObjectId</code> .

Document History

The following table describes the important changes to the AWS Database Migration Service user guide documentation after January 2018.

You can subscribe to an RSS feed to be notified of updates to this documentation.

update-history-change	update-history-description	update-history-date
Support for Elasticsearch and Kinesis Data Streams as a target	Added support for Amazon Elasticsearch and Amazon Kinesis Data Streams as targets for data migration.	November 15, 2018
CDC native start support	Added support for native start points when using change data capture (CDC).	June 28, 2018
R4 support	Added support for R4 replication instance classes.	May 10, 2018
Db2 LUW support	Added support for IBM Db2 LUW as a source for data migration.	April 26, 2018
Task log support	Added support for seeing task log usage and purging task logs.	February 8, 2018
SQL Server as target support	Added support for Amazon RDS for Microsoft SQL Server as a source.	February 6, 2018

Earlier Updates

The following table describes the important changes to the AWS Database Migration Service user guide documentation prior to January 2018.

Change	Description	Date Changed
New feature	Added support for using AWS DMS with AWS Snowball to migrate large databases. For more information, see Migrating Large Data Stores Using AWS Database Migration Service and AWS Snowball (p. 285) .	November 17, 2017
New feature	Added support for task assessment report and data validation. For more information about the task assessment report, see Creating a Task Assessment Report (p. 215) . For more information about data validation, see Data Validation Task Settings (p. 234) .	November 17, 2017
New feature	Added support for AWS CloudFormation templates. For more information, see AWS DMS Support for AWS CloudFormation (p. 11) .	July 11, 2017

Change	Description	Date Changed
New feature	Added support for using Amazon Dynamo as a target. For more information, see Using an Amazon DynamoDB Database as a Target for AWS Database Migration Service (p. 175).	April 10, 2017
New feature	Added support for using MongoDB as a source. For more information, see Using MongoDB as a Source for AWS DMS (p. 132).	April 10, 2017
New feature	Added support for using Amazon S3 as a target. For more information, see Using Amazon Simple Storage Service as a Target for AWS Database Migration Service (p. 171).	March 27, 2017
New feature	Adds support for reloading database tables during a migration task. For more information, see Reloading Tables During a Task (p. 242).	March 7, 2017
New feature	Added support for events and event subscriptions. For more information, see Working with Events and Notifications in AWS Database Migration Service (p. 280).	January 26, 2017
New feature	Added support for SSL endpoints for Oracle. For more information, see SSL Support for an Oracle Endpoint (p. 50).	December 5, 2016
New feature	Added support for using change data capture (CDC) with an Amazon RDS PostgreSQL DB instance. For more information, see Setting Up an Amazon RDS PostgreSQL DB Instance as a Source (p. 115).	September 14, 2016
New region support	Added support for the Asia Pacific (Mumbai), Asia Pacific (Seoul), and South America (São Paulo) regions. For a list of supported regions, see What Is AWS Database Migration Service? (p. 1).	August 3, 2016
New feature	Added support for ongoing replication. For more information, see Ongoing Replication (p. 316).	July 13, 2016
New feature	Added support for secured connections using SSL. For more information, see Using SSL With AWS Database Migration Service (p. 47).	July 13, 2016
New feature	Added support for SAP Adaptive Server Enterprise (ASE) as a source or target endpoint. For more information, see Using an SAP ASE Database as a Source for AWS DMS (p. 129) and Using a SAP ASE Database as a Target for AWS Database Migration Service (p. 170).	July 13, 2016
New feature	Added support for filters to move a subset of rows from the source database to the target database. For more information, see Using Source Filters (p. 257).	May 2, 2016
New feature	Added support for Amazon Redshift as a target endpoint. For more information, see Using an Amazon Redshift Database as a Target for AWS Database Migration Service (p. 163).	May 2, 2016

Change	Description	Date Changed
General availability	Initial release of AWS Database Migration Service.	March 14, 2016
Public preview release	Released the preview documentation for AWS Database Migration Service.	January 21, 2016

AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the *AWS General Reference*.