

---

# AWS Database Migration Service

## User Guide

**API Version API Version 2016-01-01**



## AWS Database Migration Service: User Guide

Copyright © 2020 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 .....	3
High-level view of AWS DMS .....	3
Components .....	4
Sources .....	9
Targets .....	11
With other AWS services .....	12
Support for AWS CloudFormation .....	13
Setting up .....	14
Sign up for AWS .....	14
Create an IAM user .....	14
Constructing an ARN .....	16
Getting started .....	18
Start a database migration .....	18
Step 1: Welcome .....	19
Step 2: Create a replication instance .....	19
Step 3: Specify source and target endpoints .....	22
Step 4: Create a task .....	25
Monitor your task .....	29
Best practices .....	31
Migration planning for AWS Database Migration Service .....	31
Improving performance .....	32
Reducing load on your source database .....	33
Using the task log .....	33
Schema conversion .....	33
Migrating large binary objects (LOBs) .....	34
Using limited LOB mode .....	34
Ongoing replication .....	35
Improving performance when migrating large tables .....	35
Using your own on-premises name server .....	35
Using Amazon Route 53 Resolver with AWS DMS .....	37
Changing the user and schema for an Oracle target .....	37
Changing table and index tablespaces for an Oracle target .....	38
Working with replication instances .....	39
Choosing replication instance types .....	40
Deciding what instance class to use .....	41
Sizing a replication instance .....	42
Public and private replication instances .....	43
Replication engine versions .....	43
Upgrading the engine version using the console .....	44
Upgrading the engine version using the AWS CLI .....	44
Setting up a network for a replication instance .....	45
Network configurations for database migration .....	45
Creating a replication subnet group .....	50
Setting an encryption key .....	51
Creating a replication instance .....	52
Modifying a replication instance .....	54
Rebooting a replication instance .....	56
Deleting a replication instance .....	58
DMS maintenance window .....	59
Effect of maintenance on existing migration tasks .....	60
Changing the maintenance window setting .....	61
Endpoints .....	63

Creating source and target endpoints .....	63
Sources for data migration .....	64
Using Oracle as a source .....	65
Using SQL Server as a source .....	98
Using Azure SQL database as a source .....	111
Using PostgreSQL as a source .....	112
Using MySQL as a source .....	128
Using SAP ASE as a source .....	136
Using MongoDB as a source .....	141
Using Amazon S3 as a source .....	145
Using IBM Db2 LUW as a source .....	151
Targets for data migration .....	154
Using Oracle as a target .....	155
Using SQL Server as a target .....	161
Using PostgreSQL as a target .....	165
Using MySQL as a target .....	168
Using Amazon Redshift as a target .....	173
Using SAP ASE as a target .....	185
Using Amazon S3 as a target .....	187
Using Amazon DynamoDB as a target .....	207
Using Amazon Kinesis Data Streams as a target .....	221
Using Apache Kafka as a target .....	232
Using Amazon Elasticsearch Service as a target .....	242
Using Amazon DocumentDB as a target .....	246
Using Amazon Neptune as a target .....	259
Supported DDL statements .....	269
Tasks .....	271
Creating a task .....	273
Task settings .....	278
Setting LOB support .....	300
Creating multiple tasks .....	300
Continuous replication tasks .....	301
Replication starting from a CDC start point .....	302
Performing bidirectional replication .....	304
Modifying a task .....	307
Reloading tables during a task .....	307
AWS Management Console .....	307
Table mapping .....	309
Specifying table selection and transformations rules from the console .....	309
Specifying table selection and transformations rules using JSON .....	314
Selection rules and actions .....	314
Transformation rules and actions .....	318
Using transformation rule expressions to define column content .....	331
Table-settings rules and operations .....	333
Using source filters .....	349
Enabling and working with premigration assessments .....	353
Specifying, starting, and viewing assessment runs .....	354
Starting and viewing data type assessments .....	359
Specifying supplemental data .....	361
Monitoring tasks .....	362
Task status .....	363
Table state during tasks .....	364
Monitoring replication tasks using Amazon CloudWatch .....	365
AWS Database Migration Service metrics .....	367
Replication instance metrics .....	367
Replication task metrics .....	368
Viewing and managing AWS DMS logs .....	370

Logging AWS DMS API calls with AWS CloudTrail .....	371
AWS DMS information in CloudTrail .....	372
Understanding AWS DMS log file entries .....	372
Working with events and notifications .....	375
AWS DMS event categories and event messages .....	376
Subscribing to AWS DMS event notification .....	378
Using AWS Management Console .....	378
Using AWS DMS API and CLI .....	380
Data validation .....	381
Using JSON editor to modify validation rules .....	382
Replication task statistics .....	382
Replication task statistics with Amazon CloudWatch .....	384
Revalidating tables during a task .....	384
AWS Management Console .....	384
Troubleshooting .....	385
Limitations .....	386
Tagging resources .....	387
API .....	388
Security .....	390
Data protection .....	392
Data encryption .....	392
Internetwork traffic privacy .....	393
Identity and access management .....	394
Audience .....	394
Authenticating with identities .....	394
Managing access using policies .....	396
How AWS Database Migration Service works with IAM .....	397
Identity-based policy examples .....	402
Resource-based policy examples .....	407
Troubleshooting .....	410
Compliance validation .....	413
Resilience .....	414
Infrastructure security .....	415
IAM permissions required .....	416
IAM roles for the CLI and API .....	420
Fine-grained access control .....	424
Using resource names to control access .....	424
Using tags to control access .....	426
Setting an encryption key .....	431
Network security .....	433
Using SSL .....	435
Limitations on using SSL with AWS DMS .....	436
Managing certificates .....	436
Enabling SSL for a MySQL-compatible, PostgreSQL, or SQL Server endpoint .....	437
Changing the database password .....	439
Limits .....	440
Limits for AWS Database Migration Service .....	440
Troubleshooting and diagnostic support .....	441
Migration tasks run slowly .....	441
Task status bar doesn't move .....	442
Task completes but nothing was migrated .....	442
Foreign keys and secondary indexes are missing .....	442
Issues occur with connecting to Amazon RDS .....	443
Error message: Incorrect thread connection string: Incorrect thread value 0 .....	443
Networking issues occur .....	443
CDC is stuck after full load .....	444
Primary key violation errors occur when you restart a task .....	444

Initial load of a schema fails .....	444
Tasks fail with an unknown error .....	444
Task restart loads tables from the beginning .....	444
Number of tables per task causes issues .....	444
Tasks fail when a primary key is created on a LOB column .....	445
Duplicate records occur on a target table without a primary key .....	445
Source endpoints fall in the reserved IP range .....	445
Troubleshooting issues with Oracle .....	445
Pulling data from views .....	446
Migrating LOBs from Oracle 12c .....	446
Switching between Oracle LogMiner and Binary Reader .....	446
Error: Oracle CDC stopped 122301 oracle CDC maximum retry counter exceeded. ....	446
Automatically add supplemental logging to an Oracle source endpoint .....	447
LOB changes aren't being captured .....	447
Error: ORA-12899: Value too large for column <i>column-name</i> .....	447
NUMBER data type being misinterpreted .....	448
Records missing during full load .....	448
Troubleshooting issues with MySQL .....	448
CDC task failing for Amazon RDS DB instance endpoint because binary logging disabled .....	448
Connections to a target MySQL instance are disconnected during a task .....	449
Adding autocommit to a MySQL-compatible endpoint .....	449
Disable foreign keys on a target MySQL-compatible endpoint .....	449
Characters replaced with question mark .....	450
"Bad event" log entries .....	450
Change data capture with MySQL 5.5 .....	450
Increasing binary log retention for Amazon RDS DB instances .....	450
Log message: Some changes from the source database had no impact when applied to the target database. ....	451
Error: Identifier too long .....	451
Error: Unsupported character set causes field data conversion to fail .....	451
Error: Codepage 1252 to UTF8 [120112] a field data conversion failed .....	451
Troubleshooting issues with PostgreSQL .....	452
JSON data types being truncated .....	452
Columns of a user-defined data type not being migrated correctly .....	453
Error: No schema has been selected to create in .....	453
Deletes and updates to a table aren't being replicated using CDC .....	453
Truncate statements aren't being propagated .....	453
Preventing PostgreSQL from capturing DDL .....	453
Selecting the schema where database objects for capturing DDL are created .....	454
Oracle tables missing after migrating to PostgreSQL .....	454
Task using view as a source has no rows copied .....	454
Troubleshooting issues with Microsoft SQL Server .....	454
Special permissions for AWS DMS user account to use CDC .....	455
Errors capturing changes for SQL server database .....	455
Missing identity columns .....	455
Error: SQL Server doesn't support publications .....	455
Changes don't appear in your target .....	455
Non-uniform table mapped across partitions .....	456
Troubleshooting issues with Amazon Redshift .....	456
Loading in to an Amazon Redshift cluster in a different AWS Region .....	456
Error: Relation "awsdms_apply_exceptions" already exists .....	457
Errors with tables whose name begins with "awsdms_changes" .....	457
Seeing tables in clusters with names like dms.awsdms_changes00000000XXXX .....	457
Permissions required to work with Amazon Redshift .....	457
Troubleshooting issues with Amazon Aurora MySQL .....	457
Error: CHARACTER SET UTF8 fields terminated by ';' enclosed by "" lines terminated by '\n' .....	458
Working with diagnostic support scripts .....	458

Oracle support scripts .....	459
SQL Server support scripts .....	461
MySQL-compatible support scripts .....	463
PostgreSQL support scripts .....	464
Migrating large data stores with Snowball Edge .....	467
Process overview .....	468
Prerequisites .....	469
Migration checklist .....	469
Step-by-step procedures .....	471
Step 1: Create a Snowball Edge job .....	471
Step 2: Download and install the AWS Schema Conversion Tool (AWS SCT) .....	471
Step 3: Unlock the AWS Snowball Edge device .....	471
Step 4: Configure the AWS DMS agent host with ODBC drivers .....	473
Step 5: Install the AWS DMS Agent .....	475
Step 6: Create a new AWS SCT project .....	477
Step 7: Configure AWS SCT to use the AWS Snowball Edge .....	478
Step 8: Register the AWS DMS Agent in AWS SCT .....	480
Step 9: Create a local and AWS DMS task .....	481
Step 10: Run and monitor the task in SCT .....	484
Limitations .....	487
Reference .....	488
AWS DMS data types .....	488
Release notes .....	490
AWS DMS 3.4.1 release notes .....	490
AWS DMS 3.4.0 release notes .....	491
AWS DMS 3.3.4 release notes .....	492
AWS DMS 3.3.3 release notes .....	492
AWS DMS 3.3.2 release notes .....	493
AWS DMS 3.3.1 release notes .....	494
AWS DMS 3.3.0 release notes .....	496
AWS DMS 3.1.4 release notes .....	498
AWS DMS 3.1.3 release notes .....	499
AWS DMS 3.1.2 release notes .....	500
AWS DMS 3.1.1 release notes .....	501
AWS DMS 2.4.5 release notes .....	503
AWS DMS 2.4.4 release notes .....	504
AWS DMS 2.4.3 release notes .....	505
AWS DMS 2.4.2 release notes .....	506
AWS DMS 2.4.1 release notes .....	507
AWS DMS 2.4.0 release notes .....	509
AWS DMS 2.3.0 release notes .....	510
Document history .....	513
Earlier updates .....	513
AWS glossary .....	516

# 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 \(p. 39\)](#). 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 (Amazon S3). 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 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 \(p. 3\)](#)
- [Components of AWS DMS \(p. 4\)](#)
- [Sources for AWS DMS \(p. 9\)](#)
- [Targets for AWS DMS \(p. 11\)](#)
- [Using AWS DMS with other AWS services \(p. 12\)](#)

## 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 DMS

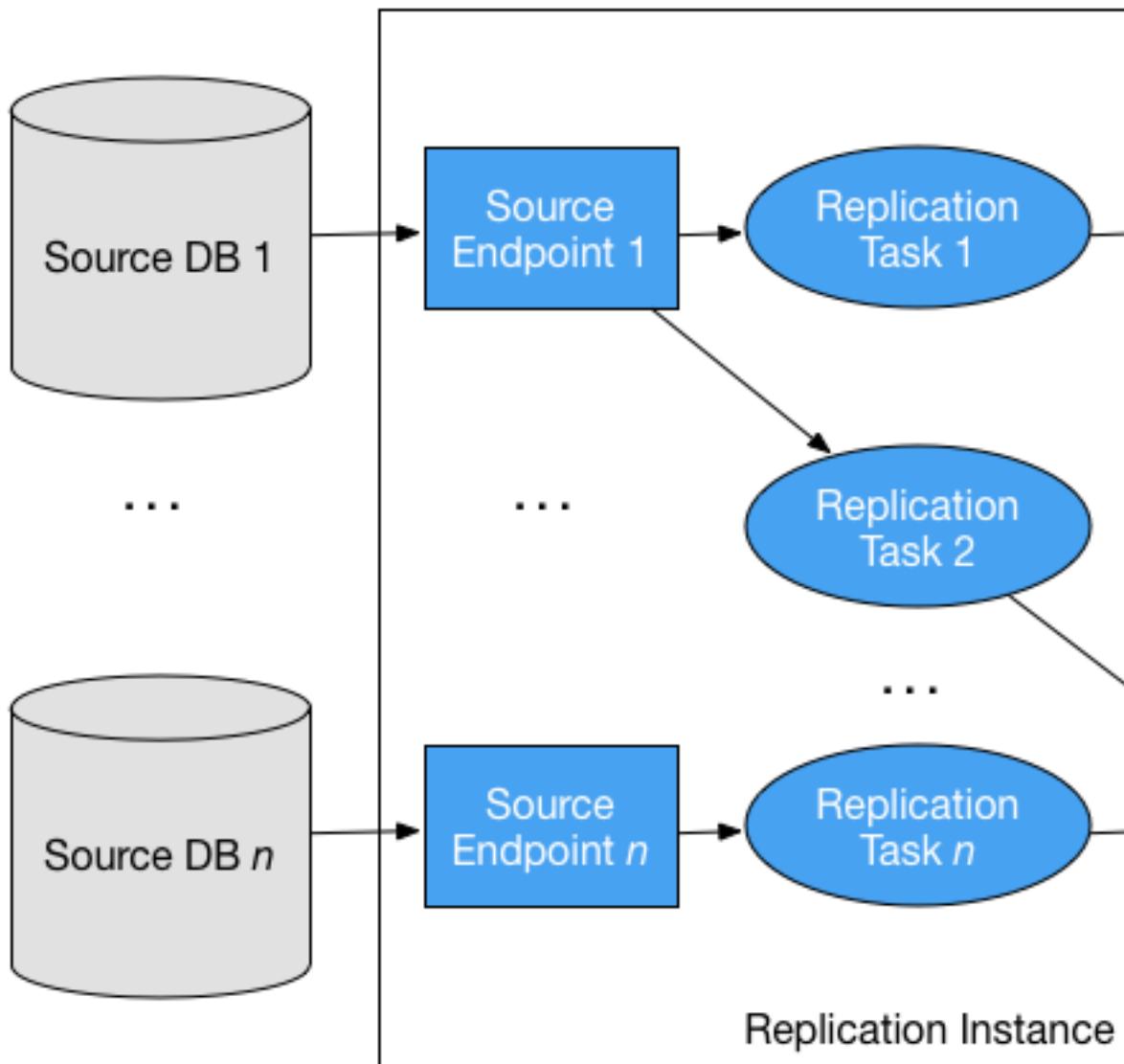
This section describes the internal components of AWS DMS and how they function together to accomplish your data migration. Understanding the underlying components of AWS DMS can help you migrate data more efficiently and provide better insight when troubleshooting or investigating issues.

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 [Choosing the right AWS DMS replication instance for your migration \(p. 40\)](#).

AWS DMS creates the replication instance on an 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 might want to increase the base storage allocation. Change data capture (CDC) might 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 \(p. 39\)](#).

## 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 or PostgreSQL..
- Server name – Server name or IP address that AWS DMS can reach.
- Port – Port number used for database server connections.
- Encryption – Secure Socket Layer (SSL) mode, if SSL is 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 to use later during task configuration. Schema information might include table definitions, primary key definitions, and unique key definitions, for example.

More than one replication task can use a single endpoint. For example, you might have two logically distinct applications hosted on the same source database that you want to migrate separately. In this case, you create two replication tasks, one for each set of application tables. You can use the same AWS DMS endpoint in both tasks.

You can customize the behavior of an endpoint by using extra connection attributes. *Extra connection 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 DMS \(p. 9\)](#) and [Targets for AWS DMS \(p. 11\)](#).

For more detailed information about AWS DMS endpoints, see [Working with AWS DMS endpoints \(p. 63\)](#).

## Replication tasks

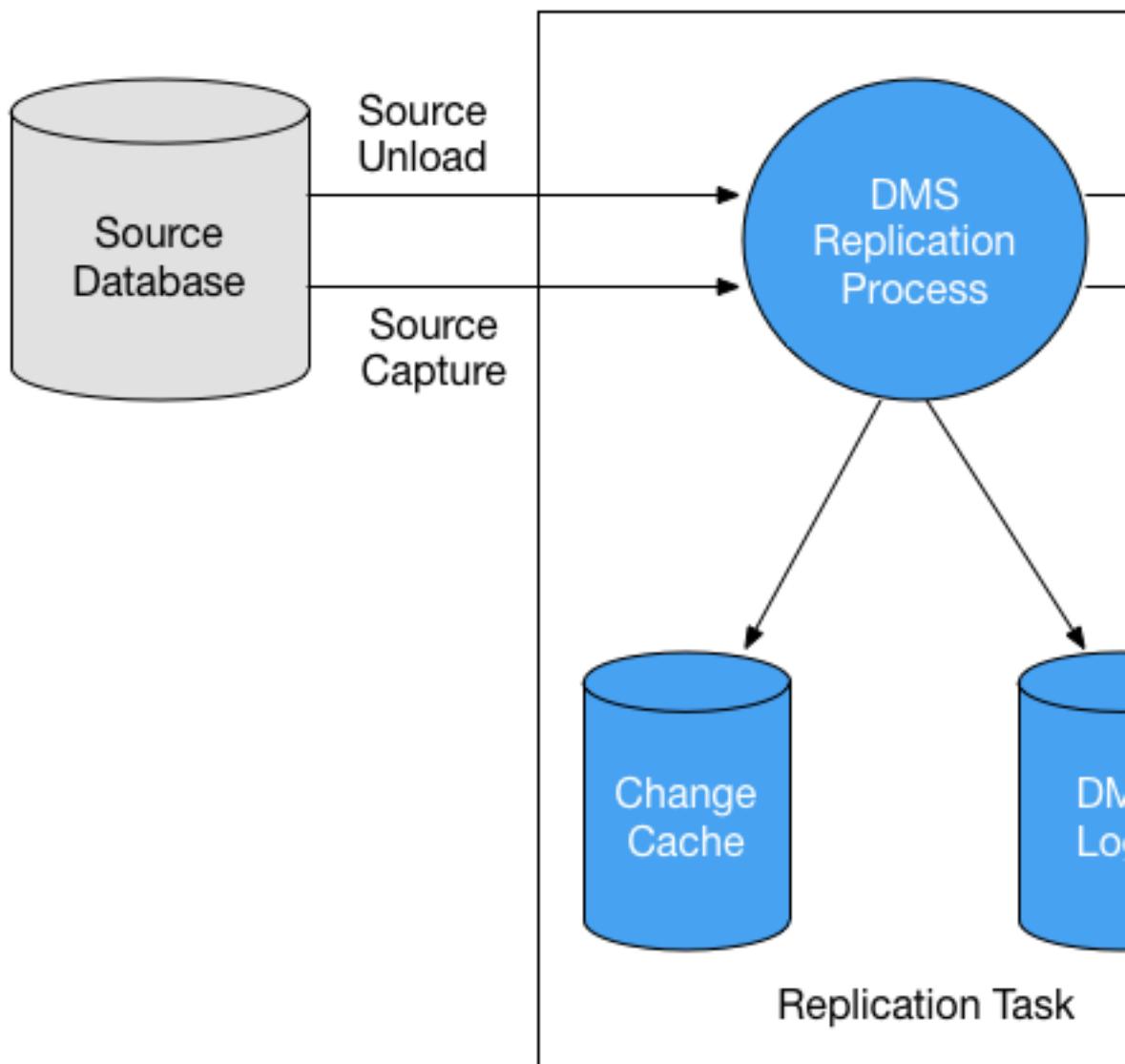
You use an AWS DMS replication task to move a set of data from the source endpoint to the target endpoint. Creating a replication task is the last step you need to take before you start a migration.

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

- Replication instance – the instance to host and run the task
- Source endpoint
- Target endpoint
- Migration type options, as listed following. For a full explanation of the migration type options, see [Creating a task \(p. 273\)](#).
  - 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. After 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 and import tools might be more efficient at loading 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.
- Target table preparation mode options, as listed following. For a full explanation of target table modes, see [Creating a task \(p. 273\)](#).
  - 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, as listed following. For a full explanation of LOB modes, see [Setting LOB support for source databases in an AWS DMS task \(p. 300\)](#).
  - 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 and how they are migrated. For more information, see [Using table mapping to specify task settings \(p. 309\)](#).
- Data transformations, as listed following. For more information on data transformations, see [Specifying table selection and transformations rules using JSON \(p. 314\)](#).
  - Changing schema, table, and column names.
  - Changing tablespace names (for Oracle target endpoints).
  - Defining primary keys and unique indexes on the target.
- Data validation
- Amazon 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 \(p. 271\)](#).

#### 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, and 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, with tables, indexes, views, and so on. You can also use AWS SCT to convert PL/SQL or TSQL to PgSQL 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 DMS migrate all objects to the specified database and schema. Or you can use these attributes to have DMS create each database and schema for you as it finds the schema on the source.

## Sources for AWS DMS

You can use the following data stores as source endpoints for data migration using AWS DMS.

### On-premises and EC2 instance databases

- Oracle versions 10.2 and later (for versions 10.x), 11g and up to 12.2, 18c, and 19c for the Enterprise, Standard, Standard One, and Standard Two editions

#### Note

- Support for Oracle version 19c as a source is available in AWS DMS versions 3.3.2 and later.
- Support for Oracle version 18c as a source is available in AWS DMS versions 3.3.1 and later.

- Microsoft SQL Server versions 2005, 2008, 2008R2, 2012, 2014, 2016, 2017, and 2019 for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.

#### Note

Support for Microsoft SQL Server version 2019 as a source is available in AWS DMS versions 3.3.2 and later.

- MySQL versions 5.5, 5.6, 5.7, and 8.0.

#### Note

Support for MySQL 8.0 as a source is available in AWS DMS versions 3.4.0 and later, except when the transaction payload is compressed.

- MariaDB (supported as a MySQL-compatible data source) versions 10.0.24 to 10.0.28, 10.1, 10.2, and 10.3 to 10.3.13.

**Note**

Support for MariaDB as a source is available in all AWS DMS versions where MySQL is supported.

- PostgreSQL version 9.4 and later (for versions 9.x), 10.x, 11.x, and 12.x.

**Note**

- PostgreSQL versions 12.x are supported as a source in AWS DMS versions 3.3.3 and later.
- PostgreSQL versions 11.x are supported as a source in AWS DMS versions 3.3.1 and later. You can use PostgreSQL version 9.4 and later (for versions 9.x) and 10.x as a source in any DMS version.

- MongoDB versions 3.x and 4.0.
- SAP Adaptive Server Enterprise (ASE) versions 12.5, 15, 15.5, 15.7, 16 and later.
- IBM Db2 for Linux, UNIX, and Windows (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 (Amazon S3)

- Oracle versions 11g (versions 11.2.0.4 and later) and up to 12.2, 18c, and 19c for the Enterprise, Standard, Standard One, and Standard Two editions.

**Note**

- Support for Oracle version 19c as a source is available in AWS DMS versions 3.3.2 and later.
- Support for Oracle version 18c as a source is available in AWS DMS versions 3.3.1 and later.

- Microsoft SQL Server versions 2008R2, 2012, 2014, 2016, 2017, and 2019 for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.

**Note**

Support for Microsoft SQL Server version 2019 as a source is available in AWS DMS versions 3.3.2 and later.

- MySQL versions 5.5, 5.6, 5.7, and 8.0.

**Note**

Support for MySQL 8.0 as a source is available in AWS DMS versions 3.4.0 and later, except when the transaction payload is compressed.

- MariaDB (supported as a MySQL-compatible data source) versions 10.0.24 to 10.0.28, 10.1, 10.2, and 10.3 to 10.3.13.

**Note**

Support for MariaDB as a source is available in all AWS DMS versions where MySQL is supported.

- PostgreSQL version 9.4 and later (for versions 9.x), 10.x, 11.x, and 12.x. Change data capture (CDC) is only supported for versions 9.4.9 and later, 9.5.4 and later, 10.x, 11.x, and 12.x. The `rds.logical_replication` parameter, which is required for CDC, is supported only in these versions and later.

**Note**

- PostgreSQL versions 12.x are supported as a source in AWS DMS versions 3.3.3 and later.

- PostgreSQL versions 11.x are supported as a source in AWS DMS versions 3.3.1 and later. You can use PostgreSQL version 9.4 and later (for versions 9.x) and 10.x as a source in any DMS version.
- Amazon Aurora with MySQL compatibility (supported as a MySQL-compatible data source).
- Amazon Aurora with PostgreSQL compatibility (supported as a PostgreSQL-compatible data source).
- Amazon S3.

For information about working with a specific source, see [Working with AWS DMS endpoints \(p. 63\)](#).

## Targets for AWS DMS

You can use the following data stores as target endpoints for data migration using AWS DMS.

### On-premises and Amazon EC2 instance databases

- Oracle versions 10g, 11g, 12c, 18c, and 19c for the Enterprise, Standard, Standard One, and Standard Two editions.

**Note**

Support for Oracle version 19c as a target is available in AWS DMS versions 3.3.2 and later.  
Support for Oracle version 18c as a target is available in AWS DMS versions 3.3.1 and later.

- Microsoft SQL Server versions 2005, 2008, 2008R2, 2012, 2014, 2016, 2017, and 2019 for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.

**Note**

Support for Microsoft SQL Server version 2019 as a target is available in AWS DMS versions 3.3.2 and later.

- MySQL versions 5.5, 5.6, 5.7, and 8.0.
- MariaDB (supported as a MySQL-compatible data target) versions 10.0.24 to 10.0.28, 10.1, 10.2 and 10.3.

**Note**

Support for MariaDB as a target is available in all AWS DMS versions where MySQL is supported.

- PostgreSQL version 9.4 and later (for versions 9.x), 10.x, 11.x, and 12.x.

**Note**

- PostgreSQL versions 12.x are supported as a target in AWS DMS versions 3.3.3 and later.
- PostgreSQL versions 11.x are supported as a target in AWS DMS versions 3.3.1 and later.  
You can use PostgreSQL version 9.4 and later (for versions 9.x) and 10.x as a source in any DMS version.
- SAP Adaptive Server Enterprise (ASE) versions 15, 15.5, 15.7, 16 and later .

### Amazon RDS instance databases, Amazon Redshift, Amazon DynamoDB, Amazon S3, Amazon Elasticsearch Service, Amazon Kinesis Data Streams, Amazon DocumentDB, Amazon Neptune, and Apache kafka

- Oracle versions 11g (versions 11.2.0.3.v1 and later), 12c, 18c, and 19c for the Enterprise, Standard, Standard One, and Standard Two editions.

**Note**

Support for Oracle version 19c as a target is available in AWS DMS versions 3.3.2 and later.

Support for Oracle version 18c as a target is available in AWS DMS versions 3.3.1 and later.

- Microsoft SQL Server versions 2008R2, 2012, 2014, 2016, 2017, and 2019 for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.

**Note**

Support for Microsoft SQL Server version 2019 as a target is available in AWS DMS versions 3.3.2 and later.

- MySQL versions 5.5, 5.6, 5.7, and 8.0.
- MariaDB (supported as a MySQL-compatible data target) versions 10.0.24 to 10.0.28, 10.1, 10.2 and 10.3.

**Note**

Support for MariaDB as a target is available in all AWS DMS versions where MySQL is supported.

- PostgreSQL version 9.4 and later (for versions 9.x), 10.x, 11.x, and 12.x.

**Note**

- PostgreSQL versions 12.x are supported as a target in AWS DMS versions 3.3.3 and later.
- PostgreSQL versions 11.x are supported as a target in AWS DMS versions 3.3.1 and later. You can use PostgreSQL version 9.4 and later (for versions 9.x) and 10.x as a source in any DMS version.

- Amazon Aurora with MySQL compatibility
- Amazon Aurora with PostgreSQL compatibility
- Amazon Aurora Serverless
- Amazon Redshift
- Amazon S3
- Amazon DynamoDB
- Amazon Elasticsearch Service
- Amazon Kinesis Data Streams
- Amazon DocumentDB (with MongoDB compatibility)
- Amazon Neptune
- Apache Kafka – [Amazon Managed Streaming for Apache Kafka \(Amazon MSK\)](#) and [self-managed Apache kafka](#)

For information about working with a specific target, see [Working with AWS DMS endpoints \(p. 63\)](#).

## 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 DMS 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](#)

# 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. [Constructing an Amazon Resource Name \(ARN\) for AWS DMS \(p. 16\)](#)

## 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, complete the following steps to create one.

### To sign up for an AWS account

1. Open <https://portal.aws.amazon.com/billing/signup>.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on 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 administrator user for yourself and add the user to an administrators group (console)

1. Sign in to the [IAM console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

#### 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, choose **Users** and then choose **Add user**.
3. For **User name**, enter **Administrator**.
4. Select the check box next to **AWS Management Console access**. Then select **Custom password**, and then enter your new password in the text box.
5. (Optional) By default, AWS requires the new user to create a new password when first signing in. You can clear the check box next to **User must create a new password at next sign-in** to allow the new user to reset their password after they sign in.
6. Choose **Next: Permissions**.
7. Under **Set permissions**, choose **Add user to group**.
8. Choose **Create group**.
9. In the **Create group** dialog box, for **Group name** enter **Administrators**.
10. Choose **Filter policies**, and then select **AWS managed -job function** to filter the table contents.
11. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.

#### Note

You must activate IAM user and role access to Billing before you can use the **AdministratorAccess** permissions to access the AWS Billing and Cost Management console. To do this, follow the instructions in [step 1 of the tutorial about delegating access to the billing console](#).

12. 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.
13. Choose **Next: Tags**.
14. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM Entities](#) in the *IAM User Guide*.
15. 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 that restrict user permissions to specific AWS resources, see [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.

## Constructing an Amazon Resource Name (ARN) for AWS DMS

If you use the AWS CLI or AWS DMS API to automate your database migration, then you work with Amazon Resource Name (ARNs). Each resource that is created in Amazon Web Services is 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 supply the ARN of the resource that 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, the following apply:

- `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 that 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
China (Beijing) Region	cn-north-1
China (Ningxia) Region	cn-northwest-1
Europe (Stockholm) Region	eu-north-1
EU (Frankfurt) Region	eu-central-1
Europe (Ireland) Region	eu-west-1
EU (London) Region	eu-west-2

Region	Name
EU (Paris) Region	eu-west-3
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	<code>arn:aws:dms:<i>region</i>:<i>account</i>:rep: <i>resourcename</i></code>
Endpoint	<code>arn:aws:dms:<i>region</i>:<i>account</i>:endpoint: <i>resourcename</i></code>
Replication task	<code>arn:aws:dms:<i>region</i>:<i>account</i>:task: <i>resourcename</i></code>
Subnet group	<code>arn:aws:dms:<i>region</i>:<i>account</i>:subgrp: <i>resourcename</i></code>

- **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:QLXQZ64MH7CXF4QCQOMGRVYVXAI</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>

# 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 \(p. 18\)](#)
- [Step 1: Welcome \(p. 19\)](#)
- [Step 2: Create a replication instance \(p. 19\)](#)
- [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** 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:

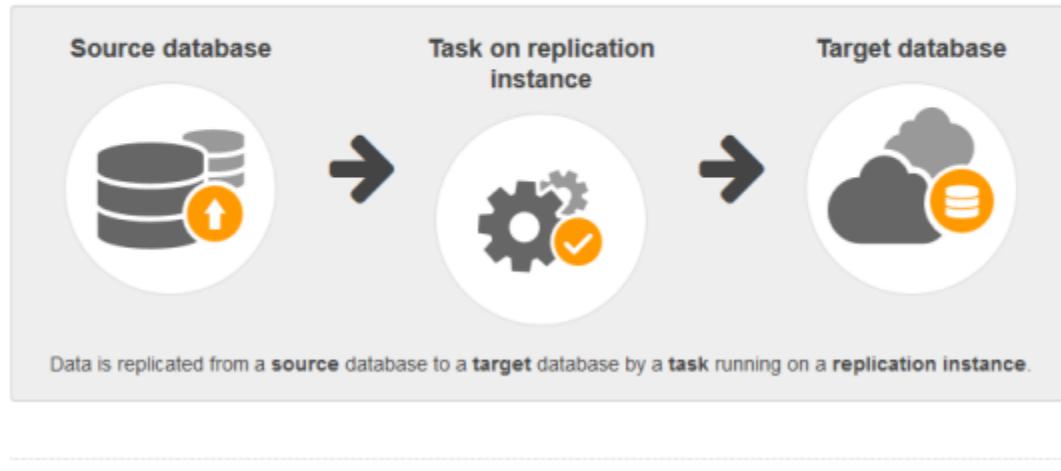
- Complete the tasks outlined in [Setting up for AWS Database Migration Service \(p. 14\)](#)
- Allocate a replication instance that performs all the processes for the migration
- Specify a source and a target database endpoint
- Create a task or set of tasks to define what tables and replication processes 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 \(p. 39\)](#).

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 console

- In the navigation pane, choose **Replication instances**.
- Choose **Create replication instance**.
- On the **Create replication instance** page, specify your replication instance information. The following table describes the settings.

For this option	Do this
<b>Name</b>	Type a name for the replication instance that contains up to 63 printable ASCII characters (excluding /, ", and @). The first character must be a letter, and the name cannot end with a hyphen. 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 <code>west2-mysql2mysql-instance1</code> .
<b>Description</b>	Type a brief description of the replication instance.
<b>Instance class</b>	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 <a href="#">Working with an AWS DMS replication instance (p. 39)</a> .
<b>Engine version</b>	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.
<b>Allocated storage (GiB)</b>	<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"> <li>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.</li> <li>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.</li> <li>Tasks configured with tables being loaded into Amazon Redshift. However, this configuration isn't an issue when Amazon Aurora is the target.</li> </ul> <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>

For this option	Do this
<b>VPC</b>	Choose the virtual private cloud (VPC) that you want to use. This VPC must be based on the Amazon Virtual Private Cloud (Amazon VPC) service. 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. Make sure that the replication instance can access the data in the source VPC. If neither your source nor your target database is in a VPC, choose a VPC where the replication instance is to be located.
<b>Multi-AZ</b>	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.
<b>Publicly accessible</b>	Choose this option if you want the replication instance to be accessible from the internet.

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

For this option	Do this
<b>Replication subnet group</b>	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 <a href="#">Creating a replication subnet group (p. 50)</a> .
<b>Availability zone</b>	Choose the Availability Zone where your source database is located.
<b>VPC security group(s)</b>	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.
<b>KMS master key</b>	Choose the encryption key to use to encrypt replication storage and connection information. If you choose <b>(Default) aws/dms</b> , 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 <a href="#">Setting an encryption key and specifying AWS KMS permissions (p. 431)</a> .

5. Specify the **Maintenance** settings. The following table describes the settings. For more information about maintenance settings, see [Working with the AWS DMS maintenance window \(p. 59\)](#)

For this option	Do this
<b>Maintenance window (UTC)</b>	Choose a weekly day and 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.
<b>Auto minor version upgrade</b>	Select to have minor engine upgrades applied automatically to the replication instance during the maintenance window.

6. Specify any **Tags** settings to help you organize your AWS DMS resources.
7. 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
Endpoint identifier* <input type="text" value="ProdEndpoint"/>	Endpoint identifier* <input type="text" value="TestEndpoint"/>
Source engine* <input type="text" value="- Select One -"/>	Target engine* <input type="text" value="- Select One -"/>
Server name* <input type="text"/>	Server name* <input type="text"/>
Port* <input type="text"/>	Port* <input type="text"/>
SSL mode* <input type="text" value="- Select One -"/>	SSL mode* <input type="text" value="- Select One -"/>
User name* <input type="text"/>	User name* <input type="text"/>
Password* <input type="text"/>	Password* <input type="text"/>
<span style="float: left; margin-right: 10px;">» Advanced</span> <span style="float: right;">» Advanced</span>	
<input type="button" value="Run test"/>	<input type="button" value="Run test"/>
<input type="button" value="Cancel"/> <input type="button" value="Previous"/> <input style="background-color: #0070C0; color: white;" type="button" value="Next"/>	

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 <b>oracle-source</b> or <b>PostgreSQL-target</b> . The name must be unique for all replication instances.

For this option	Do this
<b>Source engine and Target engine</b>	Choose the type of database engine that is the endpoint.
<b>Server name</b>	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>mysqlsvrinst.abcd12345678.us-west-2.rds.amazonaws.com</code> .
<b>Port</b>	Type the port used by the database.
<b>SSL mode</b>	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.
<b>User name</b>	Type the user name with the permissions required to allow data migration. For information about the permissions required by a particular source or target endpoint, see the appropriate endpoint section in <a href="#">Working with AWS DMS endpoints (p. 63)</a> .
<b>Password</b>	Type the password for the account with the required permissions.

- 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  ⓘ

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

Account

Key ARN

**Run test**

**Advanced**

Extra connection attributes

KMS master key  ⓘ

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

Account

Key ARN

**Run test**

---

**Cancel** **Previous** **Next**

For this option	Do this
<b>Extra connection attributes</b>	Type any additional connection parameters here. For more information about extra connection attributes, see the documentation section for your data store.
<b>KMS master key</b>	Choose the encryption key to use to encrypt replication storage and connection information. If you choose <b>(Default) aws/dms</b> , the default AWS Key Management Service (AWS KMS) key associated with your account and AWS Region is used. For more information on using the encryption key, see <a href="#">Setting an encryption key and specifying AWS KMS permissions (p. 431)</a> .

## 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.

The screenshot shows the 'Create Task' configuration page. It includes fields for 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 selected), and Start task on create (checked). Below the form are two expandable sections: 'Task Settings' and '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
<b>Replication instance</b>	Shows the replication instance that will be used.
<b>Migration type</b>	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.
<b>Start task on create</b>	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:

The screenshot shows the 'Task Settings' tab with the following configuration:

- Target table preparation mode:** Drop tables on target (selected)
- Include LOB columns in replication:** Limited LOB mode (selected)
- Max LOB size (kb):** 32
- Enable logging:** Unchecked

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

For this option	Do this
<b>Max LOB size (kb)</b>	In <b>Limited LOB Mode</b> , LOB columns which exceed the setting of <b>Max LOB Size</b> will be truncated to the specified Max LOB Size.
<b>Enable logging</b>	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 i  
 Drop tables on target i  
 Truncate i

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

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

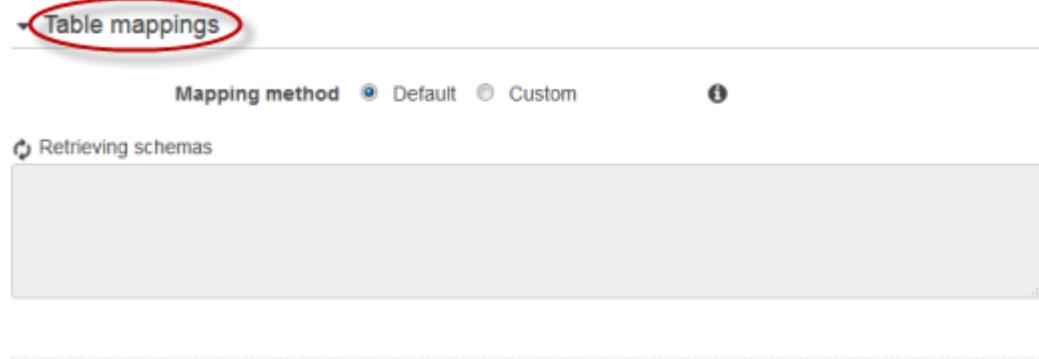
Max LOB size (kb)\*  ▲ i

Enable logging  i

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

For this option	Do this
<b>Stop task after full load completes</b>	<b>Don't stop</b> - Do not stop the task, immediately apply cached changes and continue on.  <b>Stop before applying cached changes</b> - 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.  <b>Stop after applying cached changes</b> - 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.
<b>Include LOB columns in replication</b>	<b>Don't include LOB columns</b> - LOB columns will be excluded from the migration.  <b>Full LOB mode</b> - 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.  <b>Limited LOB mode</b> - Truncate LOBs to 'Max LOB Size'. This method is faster than using Full LOB Mode.
<b>Max LOB size (kb)</b>	In <b>Limited LOB Mode</b> , LOB columns which exceed the setting of <b>Max LOB Size</b> will be truncated to the specified Max LOB Size.
<b>Enable logging</b>	Enables logging by Amazon CloudWatch.

3. 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 \(p. 309\)](#).



4. 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 \(p. 362\)](#)

task-fkvacppsulxnqyv

Overview Task monitoring **Table statistics** Logs

Filter:  Filter X

Schema	Table	State	Inserts	Deletes	Updates	DDLs	Full Load Rows	Total	Last
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

# 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

- [Migration planning for AWS Database Migration Service \(p. 31\)](#)
- [Improving the performance of an AWS DMS migration \(p. 32\)](#)
- [Reducing the load on your source database \(p. 33\)](#)
- [Using the task log to troubleshoot migration issues \(p. 33\)](#)
- [Converting schema \(p. 33\)](#)
- [Migrating large binary objects \(LOBs\) \(p. 34\)](#)
- [Ongoing replication \(p. 35\)](#)
- [Improving performance when migrating large tables \(p. 35\)](#)
- [Using your own on-premises name server \(p. 35\)](#)
- [Changing the user and schema for an Oracle target \(p. 37\)](#)
- [Changing table and index tablespaces for an Oracle target \(p. 38\)](#)

## 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 \(p. 45\)](#)
- **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 \(p. 64\)](#) and [Targets for data migration \(p. 154\)](#) 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 PgSQL 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.

# 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 12–13 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 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 systems other than Amazon RDS, disabling any logging on the target until after cutover 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. By doing this, you can 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 we recommend that you disable these during the migration process and enable them again as part of the cutover process.

## 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. In some cases, you might create the target tables using some other mechanism, such as import or export. In such cases, make sure 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:

1. **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.
2. **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.

## 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 rules from the console \(p. 309\)](#).

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.

## Using your own on-premises name server

Usually, an AWS DMS replication instance uses the Domain Name System (DNS) resolver in an Amazon EC2 instance to resolve domain endpoints. However, you can use your own on-premises name server to resolve certain endpoints if you use the Amazon Route 53 Resolver. With this tool, you can query between on-premises and AWS using inbound and outbound endpoints, forwarding rules, and a private

connection. The benefits of using an on-premises name server include improved security and ease of use behind a firewall.

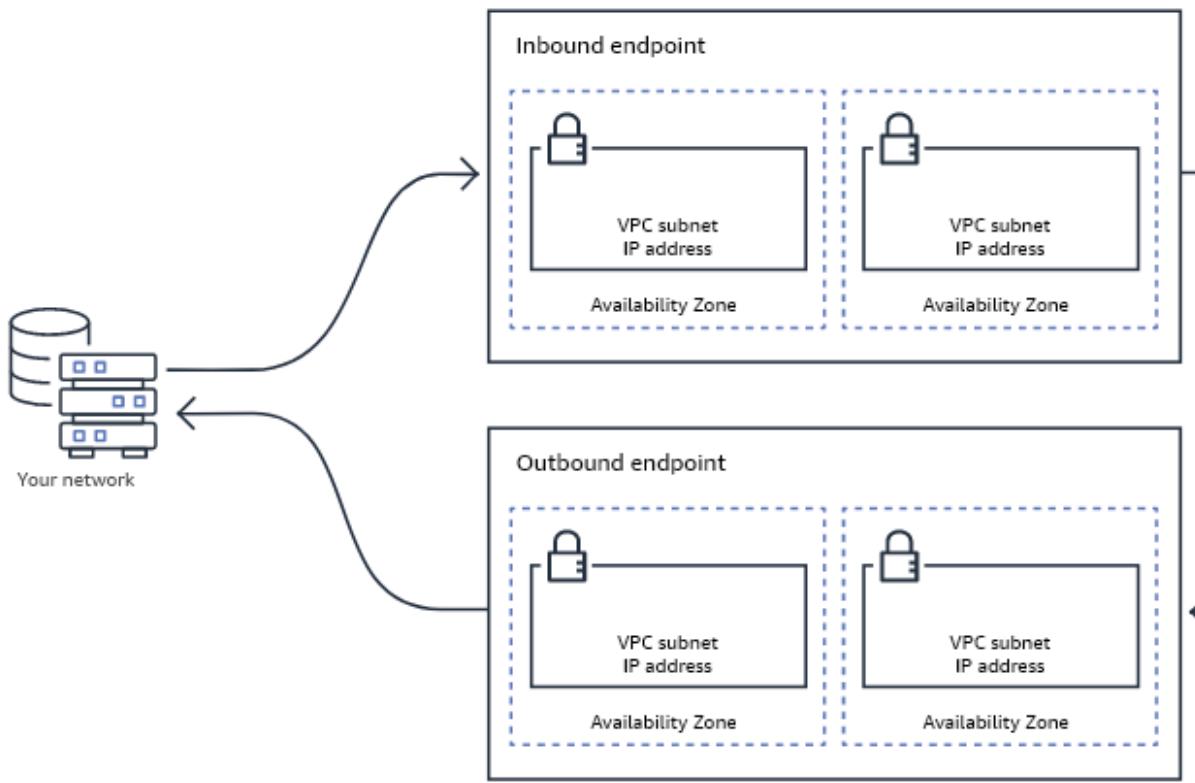
Inbound endpoints enable DNS queries that originate on-premises to resolve AWS hosted domains. To configure the endpoints, assign IP addresses in each subnet that you want to provide a resolver. To establish connectivity between your on-premises DNS infrastructure and AWS, use AWS Direct Connect or a virtual private network (VPN).

Outbound endpoints connect to your on-premises name server. The name server only grants access to IP addresses included in an allow list and set in an outbound endpoint. The IP address of your name server is the target IP address. When you choose a security group for an outbound endpoint, choose the same security group used by the replication instance.

To forward select domains to the name server, use forwarding rules. An outbound endpoint can handle multiple forwarding rules. The scope of the forwarding rule is your virtual private cloud (VPC). By using a forwarding rule associated with a VPC, you can provision a logically isolated section of the AWS Cloud. From this logically isolated section, you can launch AWS resources in a virtual network.

You can configure domains hosted within your on-premises DNS infrastructure as conditional forwarding rules that enable outbound DNS queries. When a query is made to one of those domains, rules trigger an attempt to forward DNS requests to servers that were configured with the rules. Again, a private connection over AWS Direct Connect or VPN is required.

The following diagram shows the Route 53 Resolver architecture.



For more information about AWS Route-53 DNS Resolver, see [Getting started with Route 53 Resolver](#) in the *Amazon Route 53 Developer Guide*.

## Using Amazon Route 53 Resolver with AWS DMS

You can create an on-premises name server for AWS DMS to resolve endpoints using [Amazon Route 53 Resolver](#). A description of the process follows.

### To create an on-premises name server for DMS based on Route 53

1. Sign in to the AWS Management Console and open the Route 53 console at <https://console.aws.amazon.com/route53/>.
2. On the Route 53 console, choose the AWS Region in which you want to configure your Route 53 Resolver. The Route 53 Resolver is specific to a Region.
3. Choose the query direction—inbound, outbound, or both.
4. Provide your inbound query configuration:
  - a. Enter an endpoint name and choose a VPC.
  - b. Assign one or more subnets from within the VPC (for example, choose two for availability).
  - c. Assign specific IP addresses to use as endpoints, or have Route 53 Resolver assign them automatically.
5. Create a rule for your on-premises domain so that workloads inside the VPC can route DNS queries to your DNS infrastructure.
6. Enter one or more IP addresses for your on-premises DNS servers.
7. Submit your rule.

When everything is created, your VPC is associated with your inbound and outbound rules and can start routing traffic.

For more information about Route 53 Resolver, see [Getting started with Route 53 Resolver](#) in the [Amazon Route 53 Developer Guide](#).

## Changing the user and schema for an Oracle target

When you use Oracle as a target, AWS DMS migrates the data to the schema owned by the target endpoint's user.

For example, suppose that you're migrating a schema named `PERFDATA` to an Oracle target endpoint, and that the target endpoint user name is `MASTER`. AWS DMS will connect to the Oracle target as `MASTER`, and populate the `MASTER` schema with database objects from `PERFDATA`.

To override this behavior, you need to provide a schema transformation. For example, if you want to migrate the `PERFDATA` schema objects to a `PERFDATA` schema at the target endpoint, you can use the following transformation:

```
{  
    "rule-type": "transformation",  
    "rule-id": "2",  
    "rule-name": "2",  
    "object-locator": {  
        "schema-name": "PERFDATA"  
    },  
    "rule-target": "schema",  
    "rule-action": "rename",  
    "value": "PERFDATA"  
}
```

For more information about transformations, see [Specifying table selection and transformations rules using JSON \(p. 314\)](#).

## Changing table and index tablespaces for an Oracle target

When using Oracle as a target, AWS DMS migrates all tables and indexes to default table and index tablespaces in the target. For example, suppose that your source is a database engine other than Oracle. All of the target tables and indexes are migrated to the same default tablespaces.

To override this behavior, you need to provide corresponding tablespace transformations. For example, suppose that you want to migrate tables and indexes to table and index tablespaces in the Oracle target that are named after the schema in the source. In this case, you can use transformations similar to the following. Here, the schema in the source is named INVENTORY and corresponding table and index tablespaces in the target are named INVENTORYTBL and INVENTORYIDX, respectively.

```
{  
    "rule-type": "transformation",  
    "rule-id": "3",  
    "rule-name": "3",  
    "rule-action": "rename",  
    "rule-target": "table-tablespace",  
    "object-locator": {  
        "schema-name": "INVENTORY",  
        "table-name": "%",  
        "table-tablespace-name": "%"  
    },  
    "value": "INVENTORYTBL"  
},  
{  
    "rule-type": "transformation",  
    "rule-id": "4",  
    "rule-name": "4",  
    "rule-action": "rename",  
    "rule-target": "index-tablespace",  
    "object-locator": {  
        "schema-name": "INVENTORY",  
        "table-name": "%",  
        "index-tablespace-name": "%"  
    },  
    "value": "INVENTORYIDX"  
}
```

For more information about transformations, see [Specifying table selection and transformations rules using JSON \(p. 314\)](#).

When Oracle is both source and target, you can preserve existing table or index tablespace assignments by setting the Oracle source extra connection attribute, enableHomogenousTablespace=true. For more information, see [Extra connection attributes when using Oracle as a source for AWS DMS \(p. 88\)](#)

# Working with an AWS DMS replication instance

When you create an AWS DMS replication instance, AWS DMS creates it on an Amazon EC2 instance in a virtual private cloud (VPC) based on the Amazon VPC service. You use this replication instance to perform your database migration. By using a replication instance, you can get high availability and failover support with a Multi-AZ deployment when you choose 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
China (Beijing) Region	cn-north-1
China (Ningxia) Region	cn-northwest-1
Europe (Stockholm) Region	eu-north-1
EU (Frankfurt) Region	eu-central-1
Europe (Ireland) Region	eu-west-1

Region	Name
EU (London) Region	eu-west-2
EU (Paris) Region	eu-west-3
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

AWS DMS supports a special AWS Region called AWS GovCloud (US) that is designed to allow US government agencies and customers to move 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

- [Choosing the right AWS DMS replication instance for your migration \(p. 40\)](#)
- [Choosing the optimum size for a replication instance \(p. 42\)](#)
- [Public and private replication instances \(p. 43\)](#)
- [Working with replication engine versions \(p. 43\)](#)
- [Setting up a network for a replication instance \(p. 45\)](#)
- [Setting an encryption key for a replication instance \(p. 51\)](#)
- [Creating a replication instance \(p. 52\)](#)
- [Modifying a replication instance \(p. 54\)](#)
- [Rebooting a replication instance \(p. 56\)](#)
- [Deleting a replication instance \(p. 58\)](#)
- [Working with the AWS DMS maintenance window \(p. 59\)](#)

## Choosing 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.
- C4 instances are optimized for compute-intensive workloads and deliver very cost-effective high performance at a low price per compute ratio. They achieve significantly higher packet per second (PPS) performance, lower network jitter, and lower network latency. AWS DMS can also 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.

- R4 instances are memory optimized for memory-intensive workloads. Ongoing migrations or replications of high-throughput transaction systems using AWS DMS can also consume large amounts of CPU and memory. R4 instances include more memory per vCPU than earlier generation instance types.

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 (GiB)
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

#### Topics

- [Deciding what instance class to use \(p. 41\)](#)

## Deciding what instance class to use

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.

## 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. (This process assumes batch optimized apply isn't 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 \(p. 288\)](#).

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, suppose that your load takes 24 hours and you produce 2 GB of transactions each hour. In this case, you might want to ensure that you have 48 GB of space for cached transactions. Also, the more storage space that 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.

## 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. You use a private instance when both source and target databases are in the same network that is connected to the replication instance's VPC. The network can be connected to the 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*.

## 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 \(p. 39\)](#).

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 about replication engine versions, see [AWS DMS release notes \(p. 490\)](#).

## 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 AWS 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:6EFQQ06U6EDPRCPKLNPL2SCEY`

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, note 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,ReplicationInstanceState]"
```

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 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.

Make sure that the elastic network interface allocated for your replication instance's VPC is associated with a security group. Also, make sure this security group's rules let all traffic on all ports leave (egress) the VPC. This approach allows communication from the replication instance to your source and target database endpoints, if 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. How you set this up depends on the network configuration that you use. 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.

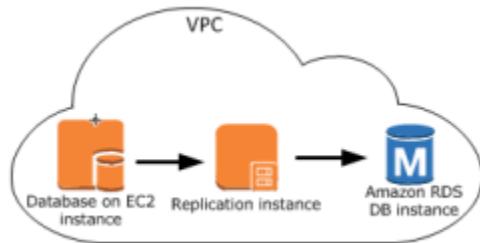
**Topics**

- [Configuration with all database migration components in one VPC \(p. 45\)](#)
- [Configuration with two VPCs \(p. 46\)](#)
- [Configuration for a network to a VPC using AWS Direct Connect or a VPN \(p. 46\)](#)
- [Configuration for a network to a VPC using the internet \(p. 47\)](#)
- [Configuration with an RDS DB instance not in a VPC to a DB instance in a VPC using ClassicLink \(p. 47\)](#)

## 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.



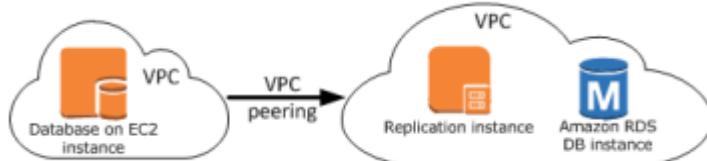
The VPC security group used in this configuration must allow ingress on the database port from the replication instance. You can do this in a couple of ways. You can ensure that the security group used by the replication instance has ingress to the endpoints. Or you can explicitly allow the private IP 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. You can 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. You can create a VPC peering connection between your own VPCs, with a VPC in another AWS account, or with a VPC in a different 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.

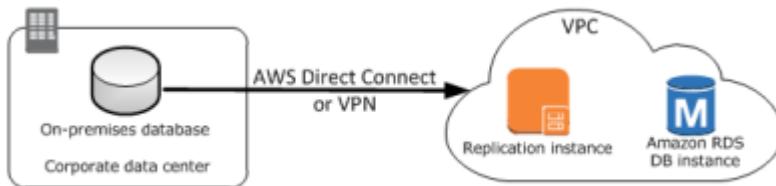


The VPC security groups used in this configuration must allow ingress on the database port from the replication instance.

## 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.

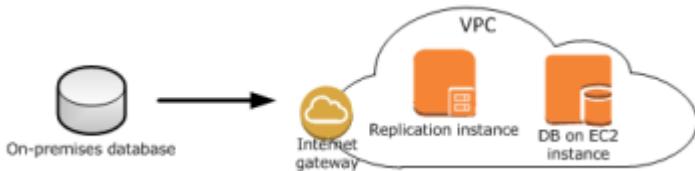
The following illustration shows a configuration where the source endpoint is an on-premises database in a corporate data center. It is connected by using AWS Direct Connect or a VPN to a VPC that contains the replication instance and a target database on an Amazon RDS DB instance.



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. These settings 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 your database. In this case, you can migrate to either an Amazon EC2 instance or an 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 route table 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. For more information, see [VPC Route Tables](#) in the *Amazon VPC User Guide*.

## Configuration with an RDS DB instance not in a VPC to a DB instance in a VPC using ClassicLink

To connect an Amazon RDS DB instance not in a VPC to a DMS replication server and DB instance in a VPC, you can use ClassicLink with a proxy server.

ClassicLink enables 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 can't directly access the source DB instance on the EC2-Classic platform using ClassicLink, you 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 connect an Amazon RDS instance that is not in a VPC to an AWS DMS replication server and DB instance that are in a VPC. To do so, you can use Amazon EC2 ClassicLink with a proxy server.

The following procedure shows how to use ClassicLink for this purpose. This procedure connects 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 VPCs.)
- 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. Create an AWS DMS replication instance and assign a VPC security group:
  - a. Sign in to the AWS Management Console and choose AWS Database Migration Service. If you're 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 \(p. 416\)](#).
  - b. On the **Dashboard** page, choose **Replication Instance**. Follow the instructions at [Step 2: Create a replication instance \(p. 19\)](#) to create a replication instance.
  - c. After you have created the AWS DMS replication instance, open the EC2 service console. Choose **Network Interfaces** from the navigation pane.
  - d. Choose the **DMSNetworkInterface**, and then choose **Change Security Groups** from the **Actions** menu.
  - e. Choose the security group you want to use for the replication instance and the target DB instance.
2. Associate the security group from the last step with the target DB instance:
  - a. Open the Amazon RDS service console. Choose **Instances** from the navigation pane.
  - b. Choose the target DB instance. For **Instance Actions**, choose **Modify**.
  - c. For the **Security Group** parameter, choose the security group you used in the previous step.
  - d. Choose **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. Create a ClassicLink connection between the proxy server and the target VPC that contains the target DB instance and the replication instance:
  - a. Open the EC2 console and choose the EC2 Classic instance that is running the proxy server.
  - b. For **Actions**, choose **ClassicLink**, then choose **Link to VPC**.
  - c. Choose the security group that you used earlier in this procedure.
  - d. Choose **Link to VPC**.
5. Step 5: Create AWS DMS endpoints using the procedure at [Step 3: Specify source and target endpoints \(p. 22\)](#). Make sure to use the internal EC2 DNS hostname of the proxy as the server name when specifying the source endpoint.
6. Create an AWS DMS task using the procedure in [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 virtual private cloud (VPC) that you plan to use. This VPC must be based on the Amazon VPC service. 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 choose, 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 \(p. 416\)](#).
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

Available Subnets			Subnet Group		
AZ	Subnet	CIDR	AZ	Subnet	CIDR
<input type="checkbox"/>	us-east-1a	subnet-4d802a3b	172.30.5.0/28	<b>Add</b>	No records found.
<input checked="" type="checkbox"/>	us-east-1a	subnet-6bec2f1d	172.30.0.0/24	<b>Remove</b>	
<input type="checkbox"/>	us-east-1b	subnet-37b3566f	172.30.1.0/24	<b>Reset</b>	
<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
<b>Identifier</b>	Enter 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 <b>DMS-default-VPC</b> .
<b>Description</b>	Enter a brief description of the replication subnet group.
<b>VPC</b>	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.
<b>Available Subnets</b>	Choose the subnets you want to include in the replication subnet group. You must choose subnets in at least two Availability Zones.

5. Choose **Add** to add the subnets to the replication subnet group.
6. 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 other supported AWS services. Supported AWS services include 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. This replication instance requires sufficient storage and processing power to perform the tasks that 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 \(p. 39\)](#).

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

For this option	Do this
<b>Name</b>	Enter 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 <b>west2-mysql2mysql-instance1</b> .
<b>Description</b>	Enter a brief description of the replication instance.
<b>Instance class</b>	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

For this option	Do this
	information on how to determine which instance class is best for your migration, see <a href="#">Working with an AWS DMS replication instance (p. 39)</a> .
<b>Replication engine version</b>	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.
<b>VPC</b>	Choose the VPC that 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. 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 or target database is in a VPC, choose a VPC where the replication instance is to be located.
<b>Multi-AZ</b>	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.
<b>Publicly accessible</b>	Choose this option if you want the replication instance to be accessible from the internet.

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

For this option	Do this
<b>Allocated storage (GiB)</b>	<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"> <li>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.</li> <li>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.</li> <li>Tasks configured with tables being loaded into Amazon Redshift. However, this configuration isn't an issue when Amazon Aurora is the target.</li> </ul> <p>In most cases, the default allocation of storage is sufficient. However, it's always a good idea to pay</p>

For this option	Do this
	attention to storage-related metrics. Make sure to scale up your storage if you find you are consuming more than the default allocation.
<b>Replication Subnet Group</b>	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 <a href="#">Creating a replication subnet group (p. 50)</a> .
<b>Availability zone</b>	Choose the Availability Zone where your source database is located.
<b>VPC Security group(s)</b>	The replication instance is created in a VPC. If your source database is in a VPC, choose the VPC security group that provides access to the DB instance where the database resides.
<b>KMS master key</b>	Choose the encryption key to use to encrypt replication storage and connection information. If you choose <b>(Default) aws/dms</b> , 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 <a href="#">Setting an encryption key and specifying AWS KMS permissions (p. 431)</a> .

- Specify the **Maintenance** settings. The following table describes the settings. For more information about maintenance settings, see [Working with the AWS DMS maintenance window \(p. 59\)](#).

For this option	Do this
<b>Auto minor version upgrade</b>	Choose to have minor engine upgrades applied automatically to the replication instance during the maintenance window.
<b>Maintenance window</b>	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.

- 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, choose the **Apply changes immediately** option in the AWS Management Console. Or use the `--apply-immediately` parameter when calling the AWS CLI, or set the `ApplyImmediately` parameter to `true` when using the 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 choose 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
<b>Name</b>	You can change the name of the replication instance. Enter 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 <code>west2-mysql2mysql-instance1</code> .
<b>Instance class</b>	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 choose the <b>Apply changes immediately</b> option.  For more information on how to determine which instance class is best for your migration, see <a href="#">Working with an AWS DMS replication instance (p. 39)</a> .
<b>Replication engine version</b>	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.
<b>Multi-AZ</b>	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.
<b>Allocated storage (GiB)</b>	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:

For this option	Do this
	<ul style="list-style-type: none"> <li>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.</li> <li>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.</li> <li>Tasks configured with tables being loaded into Amazon Redshift. However, this configuration isn't an issue when Amazon Aurora is the target.</li> </ul> <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>
<b>VPC Security Group(s)</b>	The replication instance is created in a VPC. If your source database is in a VPC, choose the VPC security group that provides access to the DB instance where the database resides.
<b>Auto minor version upgrade</b>	Choose this option to have minor engine upgrades applied automatically to the replication instance during the maintenance window or immediately if you choose the <b>Apply changes immediately</b> option.
<b>Maintenance window</b>	<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>
<b>Apply changes immediately</b>	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.

**Note**

After a reboot forces a failover from one Availability Zone to another, the Availability Zone change might not be reflected for several minutes. This lag appears in the AWS Management Console, and in calls to the AWS CLI and AWS DMS API.

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 choose 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 arn of my rep instance
```

### 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 arn of my rep instance \
--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 = *arn of my rep instance*

### Example Example simple reboot

The following code example reboots a replication instance.

```
https://dms.us-west-2.amazonaws.com/
?Action=RebootReplicationInstance
&DBInstanceArn=arn of my rep instance
&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=arn of my rep instance
&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 choose 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 arn of my rep instance
```

## Deleting a replication instance using the API

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

- `ReplicationInstanceArn = arn of my rep instance`

### Example Example delete

The following code example deletes a replication instance.

```
https://dms.us-west-2.amazonaws.com/
?Action=DeleteReplicationInstance
&DBInstanceArn=arn of my rep instance
&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 the 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
China (Beijing) Region	06:00–14:00 UTC
China (Ningxia) Region	06:00–14:00 UTC
Europe (Stockholm) Region	23:00–07:00 UTC
Europe (Frankfurt) Region	23:00–07:00 UTC
Europe (Ireland) Region	22:00–06:00 UTC
Europe (London) Region	06:00–14:00 UTC
Europe (Paris) Region	23:00–07:00 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** tab 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 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

- [Creating source and target endpoints \(p. 63\)](#)
- [Sources for data migration \(p. 64\)](#)
- [Targets for data migration \(p. 154\)](#)
- [DDL statements supported by AWS DMS \(p. 269\)](#)

## 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. (Note 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.)

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.

For this option	Do this
<b>Endpoint type</b>	Choose whether this endpoint is the source or target endpoint.
<b>Select RDS DB Instance</b>	Choose this option if the endpoint is an Amazon RDS DB instance.
<b>Endpoint identifier</b>	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 <b>oracle-source</b> or <b>PostgreSQL-</b>

For this option	Do this
	<b>target.</b> The name must be unique for all replication instances.
<b>Source engine and Target engine</b>	Choose the type of database engine that is the endpoint.
<b>Server name</b>	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>mysqlsvrinst.abcd12345678.us-west-2.rds.amazonaws.com</code> .
<b>Port</b>	Type the port used by the database.
<b>SSL mode</b>	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.
<b>User name</b>	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.
<b>Password</b>	Type the password for the account with the required permissions.
<b>Database name</b>	The name of the database you want to use as the endpoint.

2. Choose the **Advanced** tab to set values for **Extra connection attributes** and **KMS master key** if you need them. You can test the endpoint connection by choosing **Run test**. The following table describes the settings.

For this option	Do this
<b>Extra connection attributes</b>	Type any additional connection parameters here. For more information about extra connection attributes, see the documentation section for your data store.
<b>KMS master key</b>	Choose the encryption key to use to encrypt replication storage and connection information. If you choose <b>(Default) aws/dms</b> , the default AWS Key Management Service (AWS KMS) key associated with your account and AWS Region is used. For more information on using the encryption key, see <a href="#">Setting an encryption key and specifying AWS KMS permissions (p. 431)</a> .
<b>Test endpoint connection (optional)</b>	Add the VPC and replication instance name. To test the connection, choose <b>Run test</b> .

## 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 EC2

instance or an on-premises database. Or it can be a data source on an Amazon-managed service such as Amazon RDS or Amazon S3.

For a comprehensive list of valid sources, see [Sources for AWS DMS \(p. 1\)](#).

### Topics

- [Using an Oracle database as a source for AWS DMS \(p. 65\)](#)
- [Using a Microsoft SQL Server database as a source for AWS DMS \(p. 98\)](#)
- [Using Microsoft Azure SQL database as a source for AWS DMS \(p. 111\)](#)
- [Using a PostgreSQL database as a source for AWS DMS \(p. 112\)](#)
- [Using a MySQL-compatible database as a source for AWS DMS \(p. 128\)](#)
- [Using an SAP ASE database as a source for AWS DMS \(p. 136\)](#)
- [Using MongoDB as a source for AWS DMS \(p. 141\)](#)
- [Using Amazon S3 as a source for AWS DMS \(p. 145\)](#)
- [Using IBM Db2 for Linux, Unix, and Windows database \(Db2 LUW\) as a source for AWS DMS \(p. 151\)](#)

## 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.

DMS supports the following Oracle database editions:

- Oracle Enterprise Edition
- Oracle Standard Edition
- Oracle Express Edition
- Oracle Personal Edition

For self-managed Oracle databases, AWS DMS supports all Oracle database editions for versions 10.2 and later (for versions 10.x), 11g and up to 12.2, 18c, and 19c. For Amazon-managed Oracle databases provided by Amazon RDS, AWS DMS supports all Oracle database editions for versions 11g (versions 11.2.0.4 and later) and up to 12.2, 18c, and 19c.

### Note

- Support for Oracle version 19c as a source is available in AWS DMS versions 3.3.2 and later.
- Support for Oracle version 18c as a source is available in AWS DMS versions 3.3.1 and later.

You can use Secure Sockets Layer (SSL) to encrypt connections between your Oracle endpoint and your replication instance. For more information on using SSL with an Oracle endpoint, see [Using SSL with AWS Database Migration Service \(p. 435\)](#). AWS DMS also supports the use of Oracle transparent data encryption (TDE) to encrypt data at rest in the source database. For more information on using Oracle TDE with an Oracle source endpoint, see [Supported encryption methods for using Oracle as a source for AWS DMS \(p. 84\)](#).

Follow these steps to configure an Oracle database as an AWS DMS source endpoint:

1. Create an Oracle user with the appropriate permissions for AWS DMS to access your Oracle as a source database.
2. Create an Oracle as a source endpoint that conforms with your chosen Oracle database configuration. If you want to create a full-load-only task, no further configuration is needed.

3. If you want to create a task that handles change data capture (a CDC-only or full-load and CDC task), choose either Oracle LogMiner or AWS DMS Binary Reader to capture data changes. Choosing LogMiner or Binary Reader determines some of the later permissions and configuration options. For a comparison of LogMiner and Binary Reader, see the following section.

**Note**

For more information on full-load tasks, CDC-only tasks, and full-load and CDC tasks, see [Creating a task \(p. 273\)](#)

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

**Topics**

- [Using Oracle LogMiner or AWS DMS Binary Reader for change data capture \(CDC\) \(p. 66\)](#)
- [Working with a self-managed Oracle database as a source for AWS DMS \(p. 69\)](#)
- [Working with an Amazon-managed Oracle database as a source for AWS DMS \(p. 74\)](#)
- [Limitations on using Oracle as a source for AWS DMS \(p. 78\)](#)
- [SSL support for an Oracle endpoint \(p. 80\)](#)
- [Supported encryption methods for using Oracle as a source for AWS DMS \(p. 84\)](#)
- [Supported compression methods for using Oracle as a source for AWS DMS \(p. 86\)](#)
- [Replicating nested tables using Oracle as a source for AWS DMS \(p. 87\)](#)
- [Extra connection attributes when using Oracle as a source for AWS DMS \(p. 88\)](#)
- [Source data types for Oracle \(p. 95\)](#)

## Using Oracle LogMiner or AWS DMS Binary Reader for change data capture (CDC)

AWS DMS offers two methods for reading the redo logs when doing change data capture (CDC) for Oracle as a source: Oracle LogMiner and AWS DMS Binary Reader. LogMiner is an Oracle API to read the online redo logs and archived redo log files. Binary Reader is an AWS DMS native method that reads and parses the raw redo log files directly. Each method has the features following.

Feature	LogMiner	Binary Reader
Easy to configure	Yes	No
Better CDC performance and smaller impact on source system I/O and CPU	Yes	No
Supports Oracle table clusters	Yes	No
Supports all types of Oracle Hybrid Columnar Compression (HCC)	Yes	Partially <b>Note</b> Binary Reader doesn't support QUERY LOW for tasks with CDC. All

Feature	LogMiner	Binary Reader
		other HCC types are fully supported.
LOB column support in Oracle 12c	No	Yes
Supports UPDATE statements that affect only LOB columns	No	Yes
Supports Oracle transparent data encryption (TDE)	Yes	Partially <b>Note</b> Binary Reader supports TDE only for self-managed Oracle databases.
Supports all Oracle compression methods	Yes	No

**Note**

By default, AWS DMS uses Oracle LogMiner for (CDC).

The main 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, particularly compression and most options for encryption.
- LogMiner offers a simpler configuration, especially compared to Binary Reader's direct-access setup or when the redo logs are managed using Oracle Automatic Storage Management (ASM).
- LogMiner supports table clusters for use by AWS DMS. Binary Reader doesn't.

The main advantages of using Binary Reader with AWS DMS 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 doesn't.

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, or you have changes and are also using Oracle ASM.

**Note**

If you change between using Oracle LogMiner and AWS DMS Binary Reader, you must restart the CDC task.

## Configuration for change data capture (CDC) on an Oracle source database

For the Oracle as a source endpoint to connect to the database for a change data capture (CDC) task (either a full-load and CDC task or for a CDC-only task), you might need to specify additional extra connection attributes. The extra connection attributes you specify depends on the method you use to access the redo logs: Oracle LogMiner or AWS DMS Binary Reader. You specify extra connection attributes when you create the source endpoint. If you have multiple connection attribute settings, separate them from each other by semicolons with no additional white space (for example, oneSetting;thenAnother).

AWS DMS uses LogMiner by default. So, you don't have to specify additional extra connection attribute to use it.

To use Binary Reader to access the redo logs, add the following extra connection attributes.

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

### Note

Using an AWS DMS version prior to 3.3.1, you can configure a CDC task to use Binary Reader for an Amazon-managed Oracle database as a source only for Oracle versions 11.2.0.4.v11 and later, and 12.1.0.2.v7. To configure Binary Reader for an Amazon-managed Oracle as a source database for Oracle versions 12.2 and later, use AWS DMS version 3.3.1 or later. For more information, see [Working with an Amazon-managed Oracle database as a source for AWS DMS \(p. 74\)](#).

In AWS DMS versions earlier than 3.x where the Oracle source database uses Oracle ASM, using the Binary Reader requires additional attributes. In addition to the preceding, also make sure that you specify attributes for the ASM user name and ASM server address. When you create the source endpoint, the Password request parameter must specify both passwords: the Oracle source endpoint password and the ASM password.

Use the following format for the extra connection attributes to access a server that uses ASM with Binary Reader.

```
useLogMinerReader=N;useBfile=Y;asm_user=asm_username;asm_server=RAC_server_ip_address:port_number/  
+ASM;
```

Set the source endpoint Password request parameter to both the Oracle user password and the ASM password, separated by a comma as follows.

```
oracle_user_password,asm_user_password
```

In AWS DMS versions 3.x or later where the Oracle source uses ASM, you can work with high-performance options in Binary Reader for transaction processing at scale. These options include extra connection attributes to specify the number of parallel threads (parallelASMReadThreads) and the number of read-ahead buffers (readAheadBlocks). Setting these attributes together can significantly improve the performance of the CDC task. The settings shown following provide good results for most ASM configurations.

```
useLogMinerReader=N;useBfile=Y;asm_user=asm_username;asm_server=RAC_server_ip_address:port_number/  
+ASM;  
parallelASMReadThreads=6;readAheadBlocks=150000;
```

For more information on values that extra connection attributes support, see [Extra connection attributes when using Oracle as a source for AWS DMS \(p. 88\)](#).

In addition, the performance of a CDC task with an Oracle source that is using ASM depends on other settings that you choose. These settings include your AWS DMS extra connection attributes and the SQL settings to configure the Oracle source. If you are using AWS DMS versions 2.4.x, the above performance parameters do not apply. For these AWS DMS versions, see the [How to migrate from Oracle ASM to AWS using AWS DMS](#) post on the AWS Database Blog for more information on using ASM with your Oracle source endpoint.

Regardless of the method you chose for reading the redo logs and the performance of the CDC task, you also need to choose an appropriate CDC start point. Typically when choosing the CDC start point, you want to identify the point of transaction processing that captures the earliest open transaction from which to begin CDC. Otherwise, the CDC task can miss earlier open transactions. For an Oracle source database, you can choose a CDC native start point based on the Oracle system change number (SCN) to identify this earliest open transaction. For more information, see the description of identifying CDC native start points for Oracle source databases in [Performing replication starting from a CDC start point \(p. 302\)](#).

## 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 AWS DMS, grant the privileges following to the Oracle user specified in the Oracle endpoint connection settings.

#### Note

When granting privileges, use the actual name of objects, not the synonym for each object. For example, use V\_\$OBJECT including the underscore, not V\$OBJECT without the underscore.

- GRANT CREATE SESSION TO *db\_user*;
- GRANT SELECT ANY TRANSACTION TO *db\_user*;
- GRANT SELECT ON V\_\$ARCHIVED\_LOG TO *db\_user*;
- GRANT SELECT ON V\_\$LOG TO *db\_user*;
- GRANT SELECT ON V\_\$LOGFILE TO *db\_user*;
- GRANT SELECT ON V\_\$LOGMNR TO *db\_user*;
- GRANT SELECT ON V\_\$LOGMNR\_LOGS TO *db\_user*;
- GRANT SELECT ON V\_\$LOGMNR\_CONTENTS TO *db\_user*;
- GRANT SELECT ON V\_\$DATABASE TO *db\_user*;
- GRANT SELECT ON V\_\$THREAD TO *db\_user*;
- GRANT SELECT ON V\_\$PARAMETER TO *db\_user*;
- GRANT SELECT ON V\_\$NLS\_PARAMETERS TO *db\_user*;
- GRANT SELECT ON V\_\$TIMEZONE\_NAMES TO *db\_user*;
- GRANT SELECT ON V\_\$TRANSACTION TO *db\_user*;
- GRANT SELECT ON ALL\_INDEXES TO *db\_user*;
- GRANT SELECT ON ALL\_OBJECTS TO *db\_user*;

- GRANT SELECT ON ALL\_TABLES TO *db\_user*;
- GRANT SELECT ON ALL\_USERS TO *db\_user*;
- GRANT SELECT ON ALL\_CATALOG TO *db\_user*;
- GRANT SELECT ON ALL\_CONSTRAINTS TO *db\_user*;
- GRANT SELECT ON ALL\_CONS\_COLUMNS TO *db\_user*;
- GRANT SELECT ON ALL\_TAB\_COLS TO *db\_user*;
- GRANT SELECT ON ALL\_IND\_COLUMNS TO *db\_user*;
- GRANT SELECT ON ALL\_ENCRYPTED\_COLUMNS TO *db\_user*;
- GRANT SELECT ON ALL\_LOG\_GROUPS TO *db\_user*;
- GRANT SELECT ON ALL\_TAB\_PARTITIONS TO *db\_user*;
- GRANT SELECT ON SYS.DBA\_REGISTRY TO *db\_user*;
- GRANT SELECT ON SYS.OBJ\$ TO *db\_user*;
- GRANT SELECT ON DBA\_TABLESPACES TO *db\_user*;
- GRANT SELECT ON DBA\_OBJECTS TO *db\_user*; – Required if the Oracle version is earlier than 11.2.0.3.
- GRANT SELECT ON SYS.ENC\$ TO *db\_user*; – Required if transparent data encryption (TDE) is enabled. For more information on using Oracle TDE with AWS DMS, see [Supported encryption methods for using Oracle as a source for AWS DMS \(p. 84\)](#).

Grant the additional privilege following for each replicated table when you are using a specific table list.

```
SELECT on any-replicated-table;
```

Grant the additional privilege following for each replicated table when AWS DMS adds supplemental logging automatically (the default behavior) and you are using a specific table list. For information on how to turn off supplemental logging, see [Extra connection attributes when using Oracle as a source for AWS DMS \(p. 88\)](#).

```
ALTER on any-replicated-table;;
```

Grant the additional privilege following when AWS DMS adds supplemental logging automatically (the default behavior). For information on how to turn off supplemental logging, see [Extra connection attributes when using Oracle as a source for AWS DMS \(p. 88\)](#).

```
ALTER ANY TABLE;
```

When accessing an Oracle Standby database, grant the privilege following.

```
SELECT on V$STANDBY_LOG;
```

## Account privileges required when using Oracle LogMiner to access the redo logs

To access the redo logs using the Oracle LogMiner, grant the privileges following to the Oracle user specified in the Oracle endpoint connection settings:

- EXECUTE on DBMS\_LOGMNR
- SELECT on V\_\$LOGMNR\_LOGS
- SELECT on V\_\$LOGMNR\_CONTENTS

- GRANT LOGMINING – Required only if the Oracle version is 12c or later.

## Account privileges required when using AWS DMS binary reader to access the redo logs

To access the redo logs using the AWS DMS Binary Reader, grant the privileges following to the Oracle user specified in the Oracle endpoint connection settings:

- SELECT on v\_\$transportable\_platform – Grant this privilege if the redo logs are stored in Oracle Automatic Storage Management (ASM) and AWS DMS accesses them from ASM.
- CREATE ANY DIRECTORY – Grant this privilege to allow AWS DMS to use Oracle BFILE read file access in certain cases. This access is required when the replication instance doesn't have file-level access to the redo logs and the redo logs are on non-ASM storage.
- EXECUTE on DBMS\_FILE\_TRANSFER package – Grant this privilege to copy the redo log files to a temporary folder using the CopyToTempFolder method.
- EXECUTE on DBMS\_FILE\_GROUP

Binary Reader works with Oracle file features that include Oracle directories. Each Oracle directory object includes the name of the folder containing the redo log files to process. These Oracle directories aren't represented at the file system level. Instead, they are logical directories that are created at the Oracle database level. You can view them in the Oracle ALL\_DIRECTORIES view.

If you want AWS DMS to create these Oracle directories, grant the CREATE ANY DIRECTORY privilege specified preceding. AWS DMS creates the directory names with the DMS\_ prefix. If you don't grant the CREATE ANY DIRECTORY privilege, create the corresponding directories manually. In some cases when you create the Oracle directories manually, the Oracle user specified in the Oracle source endpoint isn't the user that created these directories. In these cases, also grant the READ on DIRECTORY privilege.

If the Oracle source endpoint is in Active Dataguard Standby (ADG), see the [How to use binary reader with ADG](#) post on the AWS Database Blog.

In some cases, you might use Oracle Managed Files (OMF) for storing the logs. Or your source endpoint is in ADG and the CREATE ANY DIRECTORY privilege can't be granted. In these cases, manually create the directories with all the possible log locations before starting the AWS DMS replication task. If AWS DMS doesn't find a precreated directory that it expects, the task stops. Also, AWS DMS doesn't delete the entries it has created in the ALL\_DIRECTORIES view, so manually delete them.

## Account privileges required when using binary reader with Oracle automatic storage management (ASM)

To access the redo logs in Automatic Storage Management (ASM) using Binary Reader, grant the privileges following to the Oracle user specified in the Oracle endpoint connection settings:

- SELECT ON v\_\$transportable\_platform
- SYSASM – To access the ASM account with Oracle 11g Release 2 (version 11.2.0.2) and later, grant the Oracle endpoint user the SYSASM privilege. For older supported Oracle versions, it's typically sufficient to grant the Oracle endpoint user the SYSDBA privilege.

You can validate ASM account access by opening a command prompt and invoking one of the statements following, depending on your Oracle version as specified preceding.

If you need the SYSDBA privilege, use the following.

```
sqlplus asmuser/asmpassword@+asmserver as sysdba
```

If you need the `SYSASM` privilege, use the following.

```
sqlplus asmuser/asmpassword@+asmserver as sysasm
```

## Configuring a self-managed Oracle source for replication with AWS DMS

Following are configuration requirements for using a self-managed Oracle database as a source in AWS DMS:

- Provide Oracle account access for the AWS DMS user. For more information, see [Providing Oracle account access \(p. 72\)](#)
- Set logs to the ARCHIVELOG mode. For more information, see [Setting logs to ARCHIVELOG mode \(p. 72\)](#)

You can run Oracle in two different modes: the ARCHIVELOG mode and the NOARCHIVELOG mode. To use the Oracle logs with AWS DMS, run the database in ARCHIVELOG mode.

- Set up and verify supplemental logging.

You can automatically set up supplemental logging using extra connection attributes for the Oracle source endpoint. For more information, see [Extra connection attributes when using Oracle as a source for AWS DMS \(p. 88\)](#). If you do not set these extra connection attributes, you can set up supplemental logging using a more fine-grained approach. For more information, see [Verifying and setting up supplemental logging \(p. 72\)](#)

### Providing Oracle account access

#### To provide Oracle account access to the AWS DMS user

- Make sure that the AWS DMS user has read/write privileges for the Oracle database. For information on setting up access to the Oracle account, see [User account privileges required on a self-managed Oracle source for AWS DMS \(p. 69\)](#).

### Setting logs to ARCHIVELOG mode

#### To set logs to ARCHIVELOG mode

- Run the command following.

```
ALTER database ARCHIVELOG;
```

#### Note

If your Oracle database instance is on Amazon RDS, run a different command. For more information, see [Working with an Amazon-managed Oracle database as a source for AWS DMS \(p. 74\)](#).

You can also set up and verify supplemental logging, as described following.

#### Verifying and setting up supplemental logging

To set up supplemental logging, take the following steps, described in more detail later in this section:

1. Verify that supplemental logging is enabled for the database.
2. Verify that the required supplemental logging is enabled for each table.
3. If a filter or transformation is defined for a table, enable additional logging as needed.

### To verify and, if needed, enable supplemental logging for the database

1. Run the sample query following to verify that the current version of the Oracle database is supported by AWS DMS. If the query runs without error, the returned database version is supported.

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

Here, `name`, `value`, and `description` are columns somewhere in the database that are being queried based on the value of `name`. As part of the query, an AWS DMS task checks the value of `v$parameter` against the returned version of the Oracle database. If there is a match, the query runs without error and AWS DMS supports this version of the database. If there isn't a match, the query raises an error and AWS DMS doesn't support this version of the database. In that case, to proceed with migration, first convert the Oracle database to a AWS DMS-supported version. Then, start again with configuring the database as described in [Configuring a self-managed Oracle source for replication with AWS DMS \(p. 72\)](#).

2. Run the query following to verify that supplemental logging is enabled for the database. If the returned result is `YES` or `IMPLICIT`, supplemental logging is enabled for the database.

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

3. If needed, enable supplemental logging for the database by running the command following.

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

#### Note

If your Oracle database instance is on Amazon RDS, run a different command. For more information, see [Working with an Amazon-managed Oracle database as a source for AWS DMS \(p. 74\)](#).

### To verify that the required supplemental logging is enabled for each database table

- Do one of the following:
  - If a primary key exists, add supplemental logging for the primary key. You can do this either by using the format to add supplemental logging on the primary key, or by adding supplemental logging on the primary key columns.
  - If no primary key exists, do one of the following:
    - If no primary key exists and the table has a single unique index, add all of the unique index's columns to the supplemental log.

Using `SUPPLEMENTAL LOG DATA (UNIQUE INDEX) COLUMNS` doesn't add the unique index columns to the log.

- If no primary key exists and the table has multiple unique indexes, add all of the columns for the unique index that AWS DMS selects to the supplemental log. AWS DMS selects the first unique index from an alphabetically ordered, ascending list.
- If no primary key exists and there is no unique index, add supplemental logging on all columns.

Using `SUPPLEMENTAL LOG DATA (UNIQUE INDEX) COLUMNS` doesn't add the selected unique index columns to the log.

#### Note

In some cases, the target table primary key or unique index are different than the source table primary key or unique index. In these cases, add supplemental logging manually on the source table columns that make up the target table primary key or unique index.

Also, if you change the target table primary key, add supplemental logging on the target unique index's columns instead of the columns of the source primary key or unique index.

If a filter or transformation is defined for a table, you might need to enable additional logging. Some considerations are as follows:

- If `ALL COLUMNS` supplemental logging has been added to the table, there is no need to add any additional logging.
- If the table has a unique index or a primary key, add supplemental logging on each column that is involved in a filter or transformation, but only if those columns are different from the primary key or unique index columns.
- If a transformation uses only one column, don't add this column to a supplemental logging group. For example, for a transformation `A+B`, add supplemental logging on both columns `A` and `B`. However, for a transformation `substring(A, 10)` don't add supplemental logging on column `A`.
- One method of setting up supplemental logging on both primary key or unique index columns and other specific columns that are filtered or transformed is to add `USER_LOG_GROUP` supplemental logging. Add this `USER_LOG_GROUP` supplemental logging on both the primary key/unique index columns and the other specific columns that are filtered or transformed.

For example, to replicate a table named `TEST.LOGGING` with primary key `ID` and a filter by the column `NAME`, you can run a command similar to the one following to create the log group supplemental logging.

```
ALTER TABLE TEST.LOGGING ADD SUPPLEMENTAL LOG GROUP TEST_LOG_GROUP (KEY, NAME) 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 S3. Following, you can find the privileges and configurations that 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

Grant the privileges following to the Oracle user account specified in the Oracle as a source endpoint definition.

#### Important

For all parameter values such as `db_user` and `any-replicated-table`, Oracle assumes the value is all uppercase unless you specify the value with a case-sensitive identifier. For example, if you create a `db_user` value without using quotes, as in `CREATE USER myuser` or `CREATE USER MYUSER`, Oracle identifies and stores the value as all uppercase (`MYUSER`). However, if you use quotes, as in `CREATE USER "MyUser"` or `CREATE USER 'MyUser'`, Oracle identifies and stores the case-sensitive value you specify (`MyUser`).

```
GRANT CREATE SESSION to db_user;
GRANT SELECT ANY TRANSACTION to db_user;
GRANT SELECT on DBA_TABLESPACES to db_user;
GRANT LOGMINING to db_user; (for Oracle 12c only)
GRANT SELECT ON any-replicated-table to db_user;
```

In addition, grant `SELECT` and `EXECUTE` permissions on `SYS` objects using the Amazon RDS procedure `rdsadmin.rdsadmin_util.grant_sys_object` as shown. For more information, see [Granting SELECT or EXECUTE privileges to SYS objects](#).

```
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_VIEWS', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_TAB_PARTITIONS', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_INDEXES', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_OBJECTS', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_TABLES', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_USERS', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_CATALOG', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_CONSTRAINTS', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_CONS_COLUMNS', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_TAB_COLS', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_IND_COLUMNS', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_LOG_GROUPS', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$ARCHIVED_LOG', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$LOG', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$LOGFILE', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$DATABASE', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$THREAD', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$PARAMETER', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$NLS_PARAMETERS', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$TIMEZONE_NAMES', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$TRANSACTION', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_REGISTRY', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('OBJ$', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_ENCRYPTED_COLUMNS', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$LOGMNR_LOGS', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$LOGMNR_CONTENTS', 'db_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_LOGMNR', 'db_user', 'EXECUTE');

-- (as of AWS DMS versions 3.3.1 and later and as of Oracle versions 12.1 and later)
exec rdsadmin.rdsadmin_util.grant_sys_object('REGISTRY$SQLPATCH', 'db_user', 'SELECT');

-- (for Amazon RDS Active Dataguard Standby (ADG))
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$STANDBY_LOG', 'db_user', 'SELECT');

-- (for transparent data encryption (TDE))
exec rdsadmin.rdsadmin_util.grant_sys_object('ENC$', 'db_user', 'SELECT');
```

### Note

- For more information on using Amazon RDS Active Dataguard Standby (ADG) with AWS DMS see [Using Amazon RDS Oracle Standby as a source with Binary Reader for CDC in AWS DMS \(p. 77\)](#).
- For more information on using Oracle TDE with AWS DMS, see [Supported encryption methods for using Oracle as a source for AWS DMS \(p. 84\)](#).

## Configuring an Amazon-managed Oracle source for AWS DMS

Before using an Amazon-managed Oracle database as a source for AWS DMS, perform the tasks following for the Oracle database:

- Set up supplemental logging.
- Enable automatic backups.
- Set up archiving. Archiving the redo logs for your Amazon RDS for Oracle DB instance allows AWS DMS to retrieve the log information using Oracle LogMiner or Binary Reader.

Each of the preceding steps is described in more detail following.

## To set up supplemental logging

1. Run the command following.

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

2. (Optional) Change supplemental logging attributes as needed by using the following commands.

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

## To enable automatic backups

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the **Management Settings** section for your Amazon RDS Oracle database instance, set the **Enabled Automatic Backups** option to **Yes**.

## To set up archiving

1. Run the `rdsadmin.rdsadmin_util.set_configuration` command to set up archiving.

For example, to retain the archived redo logs for 24 hours, run the command following.

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

2. Make sure that your storage has enough space for the archived redo logs during the specified retention period.

## 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 for CDC.

To use Oracle LogMiner, the minimum required user account privileges are sufficient. For more information, see [User account privileges required on an Amazon-managed Oracle source for AWS DMS \(p. 74\)](#).

To use AWS DMS Binary Reader, specify additional settings and extra connection attributes for the Oracle source endpoint, depending on your AWS DMS version. For Amazon RDS Oracle version 11g and 12.1 versions, see the post [AWS DMS now supports binary reader for Amazon RDS for Oracle and Oracle standby as a source](#) on the AWS Database Blog.

For AWS DMS versions 3.3.1 and later, Binary Reader support is available in Amazon RDS Oracle versions 12.2 and 18c.

## To configure CDC using AWS DMS Binary Reader

1. Log in to your Amazon RDS Oracle source database as the master user.
2. Run the stored procedures following to create the server-level directories. For more information, see [Accessing transaction logs](#) in the *Amazon RDS User Guide*.

```
exec rdsadmin.rdsadmin_master_util.create_archivelog_dir;
exec rdsadmin.rdsadmin_master_util.create_onlinelog_dir;
```

3. Set the extra connection attributes following on the Amazon RDS Oracle source endpoint. There can be no white space following the semicolon separator (;) for multiple attribute settings.

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

**Note**

There can be no white space following the semicolon separator (;) for multiple attribute settings (for example, oneSetting;thenAnother).

Regardless of the method you chose for reading the redo logs, you also need to choose an appropriate CDC start point. Typically when choosing the CDC start point, you want to identify the point of transaction processing that captures the earliest open transaction from which to begin CDC. For an Oracle source database, you can choose a CDC native start point based on the Oracle system change number (SCN) to identify this earliest open transaction. For more information, see the description of identifying CDC native start points for Oracle source databases in [Performing replication starting from a CDC start point \(p. 302\)](#).

## Using Amazon RDS Oracle Standby as a source with Binary Reader for CDC in AWS DMS

Verify the prerequisites following for using Amazon RDS Oracle Standby as a source when using Binary Reader for CDC in AWS DMS:

- Use the Oracle master user for setting up Binary Reader.
- Make sure that AWS DMS currently supports using only Oracle Active Data Guard Standby.

Complete the configuration steps following for using Amazon RDS Oracle Standby as a source when using Binary Reader for CDC in AWS DMS.

### To configure AWS DMS to use an Amazon RDS Oracle Standby as a source when using Binary Reader for CDC

1. Sign in to RDS for Oracle primary replica as the master user.
2. Execute the stored procedures following as documented in the Amazon RDS User Guide to create the server level directories.

```
exec rdsadmin.rdsadmin_master_util.create_archivelog_dir;
exec rdsadmin.rdsadmin_master_util.create_onlinelog_dir;
```

3. Identify the directories created in step 2.

```
SELECT directory_name, directory_path FROM all_directories
WHERE directory_name LIKE ( 'ARCHIVELOG_DIR_%' )
      OR directory_name LIKE ( 'ONLINELOG_DIR_%' )
```

For example, this displays a list of directories like the following.

DIRECTORY_NAME	DIRECTORY_PATH
ARCHIVELOG_DIR_A	/rdsdbdata/db/ORCL_A/arch
ARCHIVELOG_DIR_B	/rdsdbdata/db/ORCL_B/arch
ONLINELOG_DIR_A	/rdsdbdata/db/ORCL_A/onlineolog
ONLINELOG_DIR_B	/rdsdbdata/db/ORCL_B/onlineolog

- Grant the Read privilege on the above directories to the Oracle user account that is used to access the Standby.

```
GRANT READ ON DIRECTORY ARCHIVELOG_DIR_A TO db_user;
GRANT READ ON DIRECTORY ARCHIVELOG_DIR_B TO db_user;
GRANT READ ON DIRECTORY ONLINELOG_DIR_A TO db_user;
GRANT READ ON DIRECTORY ONLINELOG_DIR_B TO db_user;
```

- Perform an archive log switch on the primary instance. This makes sure that the changes to ALL\_DIRECTORIES are also ported to the Oracle Standby.
- Query ALL\_DIRECTORIES on the Oracle Standby to confirm that the changes were applied.
- Create a source endpoint for the Oracle Standby by using the AWS DMS Management Console or AWS CLI. While creating the endpoint, specify the extra connection attributes following.

```
useLogminerReader=N;useBfile=Y;archivedLogDestId=1;additionalArchivedLogDestId=2
```

- After creating the endpoint, use **Test endpoint connection** on the **Create endpoint** page of the AWS DMS Management Console or the AWS CLI `test-connection` command to verify that connectivity is established.

## 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 doesn't support long object names (over 30 bytes).
- AWS DMS doesn't support function-based indexes.
- If you manage supplemental logging and carry out transformations on any of the columns, make sure that supplemental logging is activated for all fields and columns. For more information on setting up supplemental logging for:
  - A self-managed Oracle as a source database, see [Verifying and setting up supplemental logging \(p. 72\)](#).
  - An Amazon-managed Oracle as a source database, see [Configuring an Amazon-managed Oracle source for AWS DMS \(p. 75\)](#).
- AWS DMS doesn't support multi-tenant container databases (CDB).
- AWS DMS doesn't support deferred constraints.
- AWS DMS supports the `rename table table-name to new-table-name` syntax for all supported Oracle versions 11 and later. This syntax is not supported for any Oracle version 10 source databases.
- AWS DMS doesn't replicate data changes that result from partition or subpartition operations (ADD, DROP, EXCHANGE, and TRUNCATE). Such updates might cause the errors following during replication:
  - For ADD operations, updates and deletes on the added data might raise a "0 rows affected" warning.
  - For DROP and TRUNCATE operations, new inserts might raise "duplicates" errors.

- EXCHANGE operations might raise both a "0 rows affected" warning and "duplicates" errors.

To replicate changes that result from partition or subpartition operations, reload the tables in question. After adding a new empty partition, operations on the newly added partition are replicated to the target as normal.

- AWS DMS doesn't support data changes on the target that result from running the `CREATE TABLE AS` statement on the source. However, the new table is created on the target.
- When you enable limited-size LOB mode, AWS DMS replicates LOBs from the Oracle source as `NULL` values on the target.
- AWS DMS doesn't capture changes made by the Oracle `DBMS_REDEFINITION` package, for example the table metadata and the `OBJECT_ID` field.
- AWS DMS maps empty BLOB and CLOB columns to `NULL` on the target.
- When capturing changes with Oracle 11 LogMiner, an update on a CLOB column with a string length greater than 1982 is lost, and the target is not updated.
- During change data capture (CDC), AWS DMS doesn't support batch updates to numeric columns defined as a primary key.
- AWS DMS doesn't support certain `UPDATE` commands. The example following is an unsupported `UPDATE` command.

```
UPDATE TEST_TABLE SET KEY=KEY+1;
```

Here, `TEST_TABLE` is the table name and `KEY` is a numeric column defined as a primary key.

- AWS DMS truncates any data in `LONG` or `LONG RAW` columns that is longer than 64 KB to 64 KB.
- AWS DMS doesn't replicate tables whose names contain apostrophes.
- AWS DMS doesn't support CDC from dynamic views.
- When you use AWS DMS Binary Reader to access the redo logs, AWS DMS doesn't support CDC for index-organized tables with an overflow segment. Alternatively, you can consider using LogMiner for such tables.
- When you use Oracle LogMiner to access the redo logs, AWS DMS has the following limitations:
  - For Oracle 12 only, AWS DMS doesn't replicate any changes to LOB columns.
  - For all Oracle versions, AWS DMS doesn't replicate the result of `UPDATE` operations on `XMLTYPE` and LOB columns.
- AWS DMS doesn't replicate results of the DDL statement `ALTER TABLE ADD column data_type DEFAULT default_value`. Instead of replicating `default_value` to the target, it sets the new column to `NULL`. Such a result can also happen even if the DDL statement that added the new column was run in a prior task.

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 but doesn't update the target. With the new column set to `NULL`, if the target table has no primary key or unique index subsequent updates raise a "zero rows affected" message.

- Oracle LogMiner doesn't support connections to a pluggable database (PDB). To connect to a PDB, access the redo logs using Binary Reader.
- When you use Binary Reader, AWS DMS has these limitations:
  - It doesn't support table clusters.
  - It supports only table-level `SHRINK SPACE` operations. This level includes the full table, partitions, and subpartitions.
  - It doesn't support changes to index-organized tables with key compression.
  - It doesn't support implementing online redo logs on raw devices.
- AWS DMS doesn't support connections to an Amazon RDS Oracle source using an Oracle Automatic Storage Management (ASM) proxy.

- AWS DMS doesn't support virtual columns.
- AWS DMS doesn't support the ROWID data type or materialized views based on a ROWID column.
- AWS DMS doesn't load or capture global temporary tables.
- For S3 targets using replication, enable supplemental logging on every column so source row updates can capture every column value. For example: `alter table yourtablename add supplemental log data (all) columns;`.

## SSL support for an Oracle endpoint

AWS DMS Oracle endpoints support SSL V3 for the none and verify-ca SSL modes. To use SSL with an Oracle endpoint, upload the Oracle wallet for the endpoint instead of .pem certificate files.

### Topics

- [Using an existing certificate for Oracle SSL \(p. 80\)](#)
- [Using a self-signed certificate for Oracle SSL \(p. 81\)](#)

## 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. If you use Amazon RDS, you can download the rds-ca-2015-root.pem root CA certificate file hosted by Amazon RDS. For more information about downloading this file, see [Using SSL/TLS to encrypt a connection to a DB instance](#) in the [Amazon RDS User Guide](#).
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 call 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
```

4. 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
```

5. Create an Oracle wallet directory for the Oracle database.

```
mkdir $ORACLE_HOME/self_signed_ssl_wallet
```

6. Create a new Oracle wallet.

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

7. 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
```

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

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

9. 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
```

10. Run the following command.

```
openssl pkcs12 -in $ORACLE_HOME/self_signed_ssl_wallet/ewallet.p12 -nodes -out nonoracle_wallet.pem
```

11. Put 'dms' as the common name.

```
openssl req -new -key nonoracle_wallet.pem -out self-signed-oracle.csr
```

12. Get the certificate signature.

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

13. If the output from step 12 is sha1WithRSAEncryption or 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
```

14. If the output from step 12 is md5WithRSAEncryption, 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
```

15. Add the certificate to the wallet.

```
orapki wallet add -wallet $ORACLE_HOME/self_signed_ssl_wallet -user_cert  
-cert self-signed-oracle.crt -pwd password
```

16. Configure the 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')  
-----
```

```
tcp$
```

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

```
cd SELF_SIGNED_CERT_DIRECTORY
```

25. Create a new client Oracle wallet for AWS DMS to 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 for AWS DMS to use. The list should include the self-signed root certificate.

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

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

## Supported encryption methods for using Oracle as a source for AWS DMS

In the following table, you can find the transparent data encryption (TDE) methods that AWS DMS supports when working with an Oracle source database.

**Note**

AWS DMS currently doesn't support the use of Oracle TDE in an Oracle version 19c database.

Redo logs access method	TDE tablespace	TDE column
Oracle LogMiner	Yes	Yes
Binary Reader	Yes	Yes (AWS DMS versions 3.x and later)

As of version 3.x, AWS DMS supports Oracle TDE (on both the column level and the tablespace level) when using Binary Reader. To use TDE encryption with AWS DMS, you need to identify the Oracle wallet location where the TDE encryption key and TDE password are stored, then identify the correct TDE encryption key and password for your Oracle source endpoint.

### To identify and specify the correct encryption key and password to use for both TDE tablespace-level and TDE column-level encryption

1. To identify the Oracle encryption wallet location on the Oracle database host, run the query following.

```
SQL> SELECT WRL_PARAMETER FROM V$ENCRYPTION_WALLET;
WRL_PARAMETER
-----
/u01/oracle/product/12.2.0/dbhome_1/data/wallet/
```

Here, /u01/oracle/product/12.2.0/dbhome\_1/data/wallet/ is the wallet location.

2. Retrieve the master key ID using one of the following encryption options, depending on which one returns this value.
  - a. For table/column-level encryption, run the queries following.

```
SQL> SELECT OBJECT_ID FROM ALL_OBJECTS
WHERE OWNER='DMS_USER' AND OBJECT_NAME='TEST_TDE_COLUMN' AND OBJECT_TYPE='TABLE';

OBJECT_ID
-----
81046
SQL> SELECT MKEYID FROM SYS.ENC$ WHERE OBJ#=81046;

MKEYID
-----
AWGDC9g1Sk8Xv+3bVveiVSgAAAAAAAAAAAAAAAAAAAAAA
```

Here, AWGDC9g1Sk8Xv+3bVveiVSg is the master key ID (**MKEYID**). If you get a value for **MKEYID**, you can continue with Step 3. Otherwise, continue with Step 2.2.

**Note**

The trailing string 'A' characters (AAA...) is not part of the value.

- b. For tablespace-level encryption, run the queries following.

```
SQL> SELECT TABLESPACE_NAME, ENCRYPTED FROM dba tablespaces;
TABLESPACE_NAME          ENC
-----
SYSTEM                  NO
SYSAUX                 NO
UNDOTBS1                NO
TEMP                   NO
USERS                  NO
TEST_ENCRYT              YES
SQL> SELECT name, utl_raw.cast_to_varchar2( utl_encode.base64_encode('01'||substr(mkeyid,1,4)) ||
       utl_raw.cast_to_varchar2( utl_encode.base64_encode(substr(mkeyid,5,length(mkeyid))) )
       masterkeyid_base64
FROM (SELECT t.name, RAWTOHEX(x.mkid) mkeyid FROM v$tablespace t, x$kcbtex x WHERE
      t.ts#=x.ts#)
WHERE name = 'TEST_ENCRYT';

NAME                  MASTERKEYID_BASE64
-----
TEST_ENCRYT            AWGDC9g1Sk8Xv+3bVveiVSg=
```

Here, AWGDC9g1Sk8Xv+3bVveiVSg is the master key ID (**TEST\_ENCRYT**). If both steps 2.1 and 2.2 return a value, they are always identical.

**Note**

The trailing '=' character is not part of the value.

3. From the command line, list the encryption wallet entries on the source Oracle database host.

```
$ mkstore -wrl /u01/oracle/product/12.2.0/dbhome_1/data/wallet/ -list
Oracle Secret Store entries:
ORACLE.SECURITY.DB.ENCRYPTION.AWGDC9g1Sk8Xv+3bVveiVSgAAAAAAAAAAAAAAAAAAAAAA
ORACLE.SECURITY.DB.ENCRYPTION.AY1mRA8OXU9Qvzo3idU4OH4AAAAAAAAAAAAAA
ORACLE.SECURITY.DB.ENCRYPTION.MASTERKEY
ORACLE.SECURITY.ID.ENCRYPTION.
ORACLE.SECURITY.KB.ENCRYPTION.
ORACLE.SECURITY.KM.ENCRYPTION.AY1mRA8OXU9Qvzo3idU4OH4AAAAAAAAAAAAAA
```

Find the entry containing the master key ID that you found in step 2 (AWGDC9g1Sk8Xv+3bVveiVSg). This entry is the TDE encryption key name.

- View the details of the entry that you found in the previous step.

```
$ mkstore -wrl /u01/oracle/product/12.2.0/dbhome_1/data/wallet/ -viewEntry ORACLE.SECURITY.DB.ENCRYPTION.AWGDC9g1Sk8Xv+3bVveiVSgAAAAAAAAAAAAAAAAAAAAA Oracle Secret Store Tool : Version 12.2.0.1.0 Copyright (c) 2004, 2016, Oracle and/or its affiliates. All rights reserved. Enter wallet password: ORACLE.SECURITY.DB.ENCRYPTION.AWGDC9g1Sk8Xv+3bVveiVSgAAAAAAAAAAAAAAAAAAAAA = AEMAASAASGYs0phWHfNt9J5mEMkkegGFid4LLfQszDojgDzbfoYDEACv0x3pJC+UGD/ PdtE2jLICBQcAeHgJChQGLA==
```

**Note**

Enter the wallet password to see the result.

Here, the value to the right of '=' is the TDE password.

- Specify the TDE encryption key name for the Oracle source endpoint by setting the `securityDbEncryptionName` extra connection attribute.

```
securityDbEncryptionName=ORACLE.SECURITY.DB.ENCRYPTION.AWGDC9g1Sk8Xv +3bVveiVSgAAAAAAAAAAAAAAAAAAAAA
```

- Provide the associated TDE password for this encryption key as part the **Password** field value for the Oracle source endpoint on the AWS DMS console. Use the order following to format the respective comma-separated password values, terminated by the TDE password value.

```
Oracle_db_password,ASM_Password,AEMAASAASGYs0phWHfNt9J5mEMkkegGFid4LLfQszDojgDzbfoYDEACv0x3pJC +UGD/PdtE2jLICBQcAeHgJChQGLA==
```

Specify the password values in this order regardless of your Oracle database configuration. For example, if you're using TDE but your Oracle database isn't using ASM, specify the relevant password values in the comma-separated order following.

```
Oracle_db_password,,AEMAASAASGYs0phWHfNt9J5mEMkkegGFid4LLfQszDojgDzbfoYDEACv0x3pJC+UGD/ PdtE2jLICBQcAeHgJChQGLA==
```

**Note**

- If the TDE credentials you specify are incorrect, the AWS DMS migration task doesn't fail. However, the task also doesn't read or apply ongoing replication changes to the target database. After starting the task, monitor **Table statistics** on the migration task page of the AWS DMS console to make sure changes are replicated.
- If a DBA changes the TDE credential values for the Oracle database while the task is running, the task then fails and the error message contains the new TDE encryption key name. Follow the above procedure to specify new values and restart the task.

## Supported compression methods for using Oracle as a source for AWS DMS

In the following table, you can find which compression methods AWS DMS supports when working with an Oracle source database. As the table shows, compression support depends both on your Oracle database version and whether DMS is configured to use Oracle LogMiner to access the redo logs.

Version	Basic	OLTP	HCC (from Oracle 11g R2 or newer)	Others
Oracle 10	No	N/A	N/A	No
Oracle 11 or newer – Oracle LogMiner	Yes	Yes	Yes	Yes – Any compression method supported by Oracle LogMiner.
Oracle 11 or newer – Binary Reader	Yes	Yes	Yes – See the note below.	Yes

#### Note

When the Oracle source endpoint is configured to use Binary Reader, the Query Low level of the HCC compression method is supported for full-load tasks only.

## Replicating nested tables using Oracle as a source for AWS DMS

As of version 3.3.1, AWS DMS supports the replication of Oracle tables containing columns that are nested tables or defined types. To enable this functionality, add the extra connection attribute setting following to the Oracle source endpoint.

```
allowSelectNestedTables=true;
```

AWS DMS creates the target tables from Oracle nested tables as regular parent and child tables on the target without a unique constraint. To access the correct data on the target, join the parent and child tables. To do this, first manually create a nonunique index on the `NESTED_TABLE_ID` column in the target child table. You can then use the `NESTED_TABLE_ID` column in the join `ON` clause together with the parent column that corresponds to the child table name. In addition, creating such an index improves performance when the target child table data is updated or deleted by AWS DMS. For an example, see [Example join for parent and child tables on the target \(p. 88\)](#).

We recommend that you configure the task to stop after a full load completes. Then, create these nonunique indexes for all the replicated child tables on the target and resume the task.

If a captured nested table is added to an existing parent table (captured or not captured), AWS DMS handles it correctly. However, the nonunique index for the corresponding target table isn't created. In this case, if the target child table becomes extremely large, performance might be affected. In such a case, we recommend that you stop the task, create the index, then resume the task.

After the nested tables are replicated to the target, have the DBA run a join on the parent and corresponding child tables to flatten the data.

### Prerequisites for replicating Oracle nested tables as a source

Ensure that you replicate parent tables for all the replicated nested tables. Include both the parent tables (the tables containing the nested table column) and the child (that is, nested) tables in the AWS DMS table mappings.

### Supported Oracle nested table types as a source

AWS DMS supports the following Oracle nested table types as a source:

- Data type
- User defined object

## Limitations of AWS DMS support for Oracle nested tables as a source

AWS DMS has the following limitations in its support of Oracle nested tables as a source:

- AWS DMS supports only one level of table nesting.
- AWS DMS table mapping doesn't check that both the parent and child table or tables are selected for replication. That is, it's possible to select a parent table without a child or a child table without a parent.

## How AWS DMS replicates Oracle nested tables as a source

AWS DMS replicates parent and nested tables to the target as follows:

- AWS DMS creates the parent table identical to the source. It then defines the nested column in the parent as RAW(16) and includes a reference to the parent's nested tables in its NESTED\_TABLE\_ID column.
- AWS DMS creates the child table identical to the nested source, but with an additional column named NESTED\_TABLE\_ID. This column has the same type and value as the corresponding parent nested column and has the same meaning.

## Example join for parent and child tables on the target

To flatten the parent table, run a join between the parent and child tables, as shown in the following example:

1. Create the Type table.

```
CREATE OR REPLACE TYPE NESTED_TEST_T AS TABLE OF VARCHAR(50);
```

2. Create the parent table with a column of type NESTED\_TEST\_T as defined preceding.

```
CREATE TABLE NESTED_PARENT_TEST (ID NUMBER(10,0) PRIMARY KEY, NAME NESTED_TEST_T) NESTED  
TABLE NAME STORE AS NAME_KEY;
```

3. Flatten the table NESTED\_PARENT\_TEST using a join with the NAME\_KEY child table where CHILD.NESTED\_TABLE\_ID matches PARENT.NAME.

```
SELECT ... FROM NESTED_PARENT_TEST PARENT, NAME_KEY CHILD WHERE CHILD.NESTED_  
TABLE_ID = PARENT.NAME;
```

## 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. If you have multiple connection attribute settings, separate them from each other by semicolons with no additional white space (for example, oneSetting;thenAnother).

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

Name	Description
addSupplementalLogging	<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: addSupplementalLogging=Y;</p> <p><b>Note</b> If you use this option, you still need to enable database-level supplemental logging as discussed previously.</p>
additionalArchivedLogDestId	<p>Set this attribute with <code>archivedLogDestId</code> in a primary/standby setup. This attribute is useful in the case of a switchover. In this case, AWS DMS needs to know which destination to get archive redo logs from to read changes. This need arises because the previous primary instance is now a standby instance after switchover.</p> <p>Note that although AWS DMS supports the use of Oracle's <code>RESETLOGS</code> command, never use <code>RESETLOGS</code> unless necessary. For additional information about the use of the Oracle <code>RESETLOGS</code> command, see Oracle's database recovery manager documentation.</p>
allowSelectNestedTables	<p>Set this attribute to true to enable replication of Oracle tables containing columns that are nested tables or defined types. For more information, see <a href="#">Replicating nested tables using Oracle as a source for AWS DMS (p. 87)</a>.</p> <p>Default value: false</p> <p>Valid values: true/false</p> <p>Example: allowSelectNestedTables=true;</p>
useLogminerReader	<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 you set this option to N, also add the setting <code>useBfile=Y</code>. For more information on this setting and using Oracle Automatic Storage Management (ASM), see <a href="#">Using Oracle LogMiner or AWS DMS Binary Reader for change data capture (CDC) (p. 66)</a>.</p> <p>Default value: Y</p> <p>Valid values: Y/N</p> <p>Example: useLogminerReader=N;useBfile=Y;</p>
useBfile	Set this attribute to Y in order to capture change data using the Binary Reader utility. Set <code>useLogminerReader</code> to N

Name	Description
	<p>to set this attribute to Y. To use the Binary Reader with an Amazon RDS for Oracle as the source, you set additional attributes. For more information on this setting and using Oracle Automatic Storage Management (ASM), see <a href="#">Using Oracle LogMiner or AWS DMS Binary Reader for change data capture (CDC) (p. 66)</a>.</p> <p>Default value: N</p> <p>Valid values: Y/N</p> <p>Example: <code>useLogminerReader=N;useBfile=Y;</code></p>
<code>parallelASMReadThreads</code>	<p>Set this attribute to change the number of threads that DMS configures to perform a Change Data Capture (CDC) load using Oracle Automatic Storage Management (ASM). You can specify an integer value between 2 (the default) and 8 (the maximum). Use this attribute together with the <code>readAheadBlocks</code> attribute. For more information, see <a href="#">Configuring change data capture (CDC) for an Amazon RDS for Oracle source for AWS DMS (p. 76)</a>.</p> <p>Default value: 2</p> <p>Valid values: An integer from 2 to 8</p> <p>Example: <code>parallelASMReadThreads=6;readAheadBlocks=150000;</code></p>
<code>readAheadBlocks</code>	<p>Set this attribute to change the number of read-ahead blocks that DMS configures to perform a Change Data Capture (CDC) load using Oracle Automatic Storage Management (ASM). You can specify an integer value between 1000 (the default) and 200,000 (the maximum). Use this attribute together with the <code>readAheadBlocks</code> attribute. For more information, see <a href="#">Configuring change data capture (CDC) for an Amazon RDS for Oracle source for AWS DMS (p. 76)</a>.</p> <p>Default value: 1000</p> <p>Valid values: An integer from 1000 to 200,000</p> <p>Example: <code>parallelASMReadThreads=6;readAheadBlocks=150000;</code></p>

Name	Description
accessAlternateDirectly	<p>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 <a href="#">Configuring change data capture (CDC) for an Amazon RDS for Oracle source for AWS DMS (p. 76)</a>.</p> <p>Default value: true</p> <p>Valid values: true/false</p> <p>Example: <code>useLogminerReader=N;useBfile=Y;accessAlternateDirectly=false;</code></p>
useAlternateFolderForOnline	<p>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 <a href="#">Configuring change data capture (CDC) for an Amazon RDS for Oracle source for AWS DMS (p. 76)</a>.</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>
oraclePathPrefix	<p>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 <a href="#">Configuring change data capture (CDC) for an Amazon RDS for Oracle source for AWS DMS (p. 76)</a>.</p> <p>Default value: none</p> <p>Valid value: /rdsdbdata/db/ORCL_A/</p> <p>Example: <code>useLogminerReader=N;useBfile=Y;accessAlternateDirectly=false;useAlternateFolderForOnline=true;oraclePathPrefix=/rdsdbdata/db/ORCL_A/;</code></p>

Name	Description
usePathPrefix	<p>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 <a href="#">Configuring change data capture (CDC) for an Amazon RDS for Oracle source for AWS DMS (p. 76)</a>.</p> <p>Default value: none</p> <p>Valid value: /rdsdbdata/log/</p> <p>Example:</p> <pre>useLogminerReader=N;useBfile=Y;accessAlternateDirectly=false; useAlternateFolderForOnline=true;oraclePathPrefix=/rdsdbdata/db/ORCL_A/; usePathPrefix=/rdsdbdata/log/;</pre>
replacePathPrefix	<p>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 <a href="#">Configuring change data capture (CDC) for an Amazon RDS for Oracle source for AWS DMS (p. 76)</a>.</p> <p>Default value: false</p> <p>Valid values: true/false</p> <p>Example:</p> <pre>useLogminerReader=N;useBfile=Y;accessAlternateDirectly=false; useAlternateFolderForOnline=true;oraclePathPrefix=/rdsdbdata/db/ORCL_A/; usePathPrefix=/rdsdbdata/log/;replacePathPrefix=true;</pre>
retryInterval	<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: retryInterval=6;</p>

Name	Description
archivedLogDestId	<p>Specifies the destination of the archived redo logs. The value should be the same as the DEST_ID number in the v\$archived_log table. When working with multiple log destinations (DEST_ID), we recommend that you 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: archivedLogDestId=1;</p>
archivedLogsOnly	<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: archivedLogsOnly=Y;</p>
numberDataTypeScale	<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: numberDataTypeScale=12</p>
afterConnectScript	<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: afterConnectScript=ALTER SESSION SET CURRENT_SCHEMA = system;</p>
failTasksOnLobTruncation	<p>When set to true, this attribute causes a task to fail if the actual size of an LOB column is greater than the specified LobMaxSize.</p> <p>If a task is set to limited LOB mode and this option is set to true, 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>

Name	Description
readTableSpaceName	<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>
standbyDelayTime	<p>Use this attribute to specify a time in minutes for the delay in standby sync.</p> <p>In AWS DMS, you can create an Oracle CDC task that uses an Active Data Guard standby instance as a source for replicating ongoing changes. Doing this eliminates the need to connect to an active database that might be in production.</p> <p>Default value: <code>0</code></p> <p>Valid values: Number</p> <p>Example: <code>standbyDelayTime=1;</code></p>
securityDbEncryptionName	<p>Specifies the name of a key used for the transparent data encryption (TDE) of the columns and tablespace in the Oracle source database. For more information on setting this attribute and its associated password on the Oracle source endpoint, see <a href="#">Supported encryption methods for using Oracle as a source for AWS DMS (p. 84)</a>.</p> <p>Default value: <code>""</code></p> <p>Valid values: String</p> <p><code>securityDbEncryptionName=ORACLE.SECURITY.DB.ENCRYPTION.Ad m5QUaaNJEAAAAAAAAAAAAAAA</code></p>
spatialSdo2GeoJsonFunctionName	<p>For Oracle version 12.1 or earlier sources migrating to PostgreSQL targets, use this attribute to convert <code>SDO_GEOGRAPHY</code> to <code>GEOJSON</code> format.</p> <p>By default, AWS DMS calls the <code>SDO2GEOJSON</code> custom function which must be present and accessible to the AWS DMS user. Alternatively, you can create your own custom function that mimics the operation of <code>SDOGEOJSON</code> and set <code>spatialSdo2GeoJsonFunctionName</code> to call it instead.</p> <p>Default value: <code>SDO2GEOJSON</code></p> <p>Valid values: String</p> <p>Example: <code>spatialSdo2GeoJsonFunctionName=myCustomSDO2GEOJSONFunction</code></p>

Name	Description
enableHomogenousTablespace	<p>Set this attribute to enable homogenous tablespace replication and create existing tables or indexes under the same tablespace on the target</p> <p>Default value: false</p> <p>Valid values: true/false</p> <p>Example: enableHomogenousTablespace=true</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 \(p. 488\)](#).

Oracle data type	AWS DMS data type
BINARY_FLOAT	REAL4
BINARY_DOUBLE	REAL8
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.	<p>When scale is 0:</p> <ul style="list-style-type: none"> <li>And precision is 0, use REAL8.</li> <li>And precision is greater than or equal to 2, use INT1.</li> <li>And precision is greater than 2 and less than or equal to 4, use INT2.</li> <li>And precision is greater than 4 and less than or equal to 9, use INT4.</li> <li>And precision is greater than 9, use NUMERIC.</li> <li>And precision is greater than or equal to scale, use NUMERIC.</li> </ul> <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

Oracle data type	AWS DMS data type
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  CLOB  Starting with AWS DMS version 3.3.3, an Oracle VARCHAR2 data type with a declared size greater than 4,000 bytes maps to a CLOB data type. For more information, see <a href="#">Migrating Oracle extended data types (p. 97)</a> .
NCHAR	WSTRING
NVARCHAR2	WSTRING  CLOB  Starting with AWS DMS version 3.3.3, an Oracle NVARCHAR2 data type with a declared size greater than 4,000 bytes maps to a CLOB data type. For more information, see <a href="#">Migrating Oracle extended data types (p. 97)</a> .
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.
CLOB	CLOB  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.
NCLOB	NCLOB  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.

Oracle data type	AWS DMS data type
LONG	CLOB  The LONG data type isn't 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.
LONG RAW	BLOB  The LONG RAW data type isn't 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.
XMLTYPE	CLOB  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).
SDO_Geometry	BLOB (when an Oracle to Oracle migration)  CLOB (when an Oracle to PostgreSQL migration)

Oracle tables used as a source with columns of the following data types aren't supported and can't be replicated. Replicating columns with these data types result in a null column.

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

**Note**

Virtual columns aren't supported.

## Migrating Oracle extended data types

Starting with release of AWS DMS 3.3.3, Oracle extended data types for both source and targets are supported. An Oracle VARCHAR2 or NVARCHAR2 data type with a declared size greater than 4,000 bytes is an *extended data type*.

The Oracle initialization parameter `MAX_STRING_SIZE` controls the maximum size of a string in an Oracle database. When `MAX_STRING_SIZE = EXTENDED`, AWS DMS maps Oracle VARCHAR2 and NVARCHAR2 data types to a CLOB data type during migration. Similarly, an Oracle VARCHAR2 or NVARCHAR2 data type with a declared size greater than 4,000 bytes maps to an AWS DMS CLOB data type during migration.

## Migrating Oracle spatial data types

*Spatial data* identifies the geometry information for an object or location in space. In an Oracle database, the geometric description of a spatial object is stored in an object of type SDO\_GEOMETRY. Within this object, the geometric description is stored in a single row in a single column of a user-defined table.

AWS DMS supports migrating the Oracle type SDO\_GEOMETRY from an Oracle source to either an Oracle or PostgreSQL target.

When you migrate Oracle spatial data types using AWS DMS, be aware of these considerations:

- When migrating to an Oracle target, make sure to manually transfer USER\_SDO\_GEOM\_METADATA entries that include type information.
- When migrating from an Oracle source endpoint to a PostgreSQL target endpoint, AWS DMS creates target columns. These columns have default geometry and geography type information with a 2D dimension and a spatial reference identifier (SRID) equal to zero (0). An example is GEOMETRY, 2, 0.
- For Oracle version 12.1 or earlier sources migrating to PostgreSQL targets, convert SDO\_GEOMETRY objects to GEOJSON format by using the SDO2GEOJSON function, or the spatialSdo2GeoJsonFunctionName extra connection attribute. For more information, see [Extra connection attributes when using Oracle as a source for AWS DMS \(p. 88\)](#).

## Using a Microsoft SQL Server database as a source for AWS DMS

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

AWS DMS supports, as a source, Microsoft SQL Server versions 2005, 2008, 2008R2, 2012, 2014, 2016, 2017, and 2019 on-premise databases and Amazon EC2 instance databases. The Enterprise, Standard, Workgroup, Developer, and Web editions are supported. Ongoing replication (CDC) is supported for all versions of Enterprise Edition, and Standard Edition version 2016 SP1 and later.

AWS DMS supports, as a source, Amazon RDS DB instance databases for SQL Server versions 2008R2, 2012, 2014, 2016, 2017, and 2019. The Enterprise and Standard editions are supported. Ongoing replication (CDC) is supported for all versions of Enterprise Edition, and Standard Edition version 2016 SP1 and later.

### Note

Support for Microsoft SQL Server version 2019 as a source is available in AWS DMS versions 3.3.2 and later.

The source SQL Server database can be installed on any computer in your network. A SQL Server account with appropriate access privileges to the source database for the type of task you chose is 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
```

Also, obtain the port number that your named instance of SQL Server listens on, and use it to configure your AWS DMS source endpoint.

**Note**

Port 1433 is the default for Microsoft SQL Server. But, dynamic ports that change each time SQL Server is started, and specific static port numbers used to connect to SQL Server through a firewall are also often used. So, know the actual port number of your named instance of SQL Server when you create the AWS DMS source endpoint.

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 \(p. 435\)](#).

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**

- [Permissions for full load only tasks \(p. 99\)](#)
- [Limitations on using SQL Server as a source for AWS DMS \(p. 99\)](#)
- [Using ongoing replication \(CDC\) from a SQL Server source \(p. 101\)](#)
- [Supported compression methods \(p. 106\)](#)
- [Working with SQL Server AlwaysOn availability groups \(p. 106\)](#)
- [Configuring a SQL Server database as a replication source for AWS DMS \(p. 106\)](#)
- [Extra connection attributes when using SQL Server as a source for AWS DMS \(p. 107\)](#)
- [Source data types for SQL Server \(p. 108\)](#)

## Permissions for full load only tasks

The permissions following are required to perform full load only tasks.

```
USE db_name;

CREATE USER dms_user FOR LOGIN dms_user;
ALTER ROLE [db_datareader] ADD MEMBER dms_user;
GRANT VIEW DATABASE STATE to dms_user ;
```

```
USE master;

GRANT VIEW SERVER STATE TO dms_user;
```

## 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:

- You can't use an SQL Server database as an AWS DMS source if SQL Server Change Tracking (CT) is enabled on the database.

**Note**

This limitation no longer applies in AWS DMS versions 3.3.1 and later.

- The identity property for a column isn't migrated to a target database column.
- The SQL Server endpoint doesn't support the use of sparse tables.
- Windows Authentication isn't supported.
- Changes to computed fields in a SQL Server aren't replicated.
- Temporal tables aren't supported.
- SQL Server partition switching isn't supported.
- When using the WRITETEXT and UPDATERTEXT utilities, AWS DMS doesn't capture events applied on the source database.
- The following data manipulation language (DML) pattern isn't supported:

```
SELECT * INTO new_table FROM existing_table
```

- When using SQL Server as a source, column-level encryption isn't 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 or SQL Server 2008 R2 as sources. 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
```

- Geometry columns are not supported in full lob mode when using SQL Server as a source. Instead, use limited lob mode or set the `InlineLobMaxSize` task setting to utilize inline lob mode.
- A secondary SQL Server database isn't supported as a source database.
- When using a Microsoft SQL Server source database in a replication task, the SQL Server Replication Publisher definitions are not removed if you remove the task. A Microsoft SQL Server system administrator must delete those definitions from Microsoft SQL Server.
- Replicating data from indexed views isn't supported.
- Renaming tables using `sp_rename` isn't supported (for example, `sp_rename 'Sales.SalesRegion', 'SalesReg';`)
- Renaming columns using `sp_rename` isn't supported (for example, `sp_rename 'Sales.Sales.Region', 'RegID', 'COLUMN';`)
- `TRUNCATE` events aren't captured.

The following limitations apply when accessing the backup transaction logs:

- Encrypted backups aren't supported.
- Backups stored at a URL or on Windows Azure aren't supported.

The following limitations apply when accessing the backup transaction logs at file level:

- The backup transaction logs must reside in a shared folder with the appropriate permissions and access rights.
- Active transaction logs are accessed through the Microsoft SQL Server API (and not at file-level).
- AWS DMS and Microsoft SQL Server machines must reside in the same domain.

- Compressed backup transaction logs aren't supported.
- Transparent Data Encryption (TDE) isn't supported. Note that when accessing the backup transaction logs using SQL Server's native functionality (not using file-level access), TDE encryption is supported
- UNIX platforms aren't supported.
- Reading the backup logs from multiple stripes isn't supported.
- Microsoft SQL Server backup to multiple disks isn't supported.
- When inserting a value into SQL Server spatial data types (GEOGRAPHY and GEOMETRY), you can either ignore the SRID (Spatial Reference System Identifier) property or specify a different number. When replicating tables with spatial data types, AWS DMS replaces the SRID with the default SRID (0 for GEOMETRY and 4326 for GEOGRAPHY).
- If your database isn't configured for MS-REPLICATION or MS-CDC, you can still capture tables that do not have a Primary Key, but only INSERT/DELETE DML events are captured. UPDATE and TRUNCATE TABLE events are ignored.
- Columnstore indexes aren't supported.
- Memory-optimized tables (using In-Memory OLTP) aren't supported.
- When replicating a table with a Primary Key that consists of multiple columns, updating the Primary Key columns during Full Load isn't supported.
- Delayed durability isn't supported.

## 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 Edition, Standard Edition version 2016 SP1 and later, 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 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. You do so by taking 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 SQL Server transaction log backups created using third-party backup software that *aren't* in native format. To support transaction log backups that *are* in native format and created

using third-party backup software, add the `use3rdPartyBackupDevice=Y` connection attribute to the source endpoint.

- 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 full logging to be turned on in SQL Server. To turn on full logging 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

AWS DMS on-going replication for SQL Server uses Native SQL Server Replication for tables with primary keys, and change data capture (CDC) for tables without primary keys.

For tables with primary keys, AWS DMS can configure the required artifacts on the source. But for self-managed SQL Server source database instances, the SQL Server *Distribution* must first be configured manually. Then, AWS DMS source endpoint users with sysadmin permission can automatically create the *Publication* for tables with primary keys.

To check if distribution has already been configured, run the following command.

```
sp_get_distributor
```

If the result is `NULL` for column distribution, distribution is not configured. To enable distribution, follow these steps:

1. Connect to the SQL Server source database using the SQL Server Management Studio (SSMS) tool.

2. Right click on the **Replication** folder, and select **Configure Distribution**. The Configure Distribution Wizard appears.
3. Follow the wizard to enter the default values and create the *Distribution*.

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.

### Note

You can perform this procedure while the DMS task is running. If the DMS task is stopped, you can perform this procedure only if there are no transaction log or database backups in progress. This is because SQL Server requires the SYSADMIN privilege in order to query the backups for the LSN position.

### 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 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, see [Creating a SQL Server publication for ongoing replication \(p. 104\)](#).

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, create a publication for each table that is participating in ongoing replication.

### To create a publication for SQL Server ongoing replication

1. Log in to SSMS using the SYSADMIN user account.
2. Expand **Replication**.
3. Open the context (right-click) menu for **Local Publications**.
4. In the **New Publication Wizard**, choose **Next**.
5. Choose the database where you want to create the publication.
6. Choose **Transactional publication**, and then choose **Next**.
7. Expand **Tables** and choose the tables with PK (also the tables you want to publish). Choose **Next**.
8. Choose **Next**, because you don't need to create a filter.
9. In the **Snapshot Agent** screen, choose the first option to **Create a snapshot immediately and keep the snapshot available to initialize subscriptions**. 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 format **AR\_PUBLICATION\_000DBID**.

For example, if your **DBID** is less than 10, you need to name the publication **AR\_PUBLICATION\_0000DBID>** (4 zeros). If your **DBID** is greater than or equal to 10, you need to name the publication **AR\_PUBLICATION\_000DBID** (3 zeros). 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 doesn't 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_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'schema_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'schema_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'schema_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 commands.

```
use dbname
EXEC sys.sp_cdc_change_job @job_type = 'capture' ,@pollinginterval = 3599
exec sp_cdc_start_job 'capture'
```

The parameter `@pollinginterval` is measured in seconds with a maximum length of 3599. This means that the transaction log retains changes for 3599 seconds (nearly one hour) when `@pollinginterval = 3599`. The procedure `exec sp_cdc_start_job 'capture'` initiates the settings.

If an AWS DMS replication task that captures ongoing changes to your SQL Server source stops for more than one hour, use the following procedure.

### To maintain the retention period during an AWS DMS replication task

1. Stop the job truncating the transaction logs (TLogs) using this command:

```
exec sp_cdc_stop_job 'capture'
```

2. Navigate to your task on the AWS DMS Console and resume the task.
3. Open the Monitoring Tab from the AWS DMS Console and check the **CDCLatencySource** metric.
4. Once the **CDCLatencySource** metric equals 0 (zero) and stays there, re-start the job truncating the TLogs using the following command:

```
exec sp_cdc_start_job 'capture'
```

**Note**

Remember to start the job that truncates SQL Server TLogs, otherwise storage on the SQL Server instance might fill up.

## 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 aren't 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. This slowness is because the creation of the table articles is being duplicated by the Availability Groups Server.

**Note**

In AWS DMS versions 3.3.1 and later, you can migrate changes from a single AlwaysOn replica.

## 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. 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.

## 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. If you have multiple connection attribute settings, separate them from each other by semicolons with no additional white space (for example, oneSetting;thenAnother).

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

Name	Description
alwaysOnSharedSyncedBackupIsEnabled	<p>This attribute adjusts the behavior of AWS DMS when migrating from an SQL Server source database that is hosted as part of an Always On availability group cluster. Starting with AWS DMS version 3.3.x, AWS DMS has enhanced support for SQL Server source databases that are configured to run in an Always On cluster. In this case, AWS DMS attempts to track if a transaction backups are happening from nodes in the Always On cluster other than the node where the source database instance is hosted. At migration task start up, AWS DMS tries to connect to each node in the cluster, but fails if it cannot connect to any one of the nodes. In version 3.1.x, this behavior does not occur because AWS DMS only connects to the node hosting the SQL Server instance configured as the source endpoint. Essentially, version 3.1.x treats an Always On cluster node as a standalone node.</p> <p>If this attribute is <code>true</code>, AWS DMS behaves in an Always On cluster like the 3.1.x version. If you need to have AWS DMS poll all the nodes in the Always On cluster for transaction backups, set this attribute to <code>false</code>.</p> <p>Default value: <code>true</code></p> <p>Valid values: <code>true</code> or <code>false</code></p> <p>Example:  <code>alwaysOnSharedSyncedBackupIsEnabled=false;</code></p>
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"> <li><b>Start transactions in the database:</b> 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.</li> </ol>

Name	Description
	<p><b>2. Exclusively use sp_repldone within a single task:</b> 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 doesn't involve any transactional activities, it can only be used when Microsoft Replication isn't 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> <p>Default value: <code>RELY_ON_SQL_SERVER_REPLICATION_AGENT</code></p> <p>Valid values: {<code>EXCLUSIVE_AUTOMATIC_TRUNCATION</code>, <code>RELY_ON_SQL_SERVER_REPLICATION_AGENT</code>}</p> <p>Example: <code>safeguardPolicy=RELY_ON_SQL_SERVER_REPLICATION_AGENT;</code></p>
<code>readBackupOnly</code>	<p>When this attribute is set to <code>Y</code>, AWS DMS only reads changes from transaction log backups and doesn't read from the active transaction log file during ongoing replication. Setting this parameter to <code>Y</code> enables you to control active transaction log file growth during full load and ongoing replication tasks. However, it can add some source latency to ongoing replication.</p> <p>Valid values: <code>N</code> or <code>Y</code>. The default is <code>N</code>.</p> <p>Example: <code>readBackupOnly=Y;</code></p>
<code>use3rdPartyBackupDevice</code>	<p>When this attribute is set to <code>Y</code>, AWS DMS processes third party transaction log backups if they are created in native format.</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 \(p. 488\)](#).

SQL Server data types	AWS DMS data types
<code>BIGINT</code>	<code>INT8</code>
<code>BIT</code>	<code>BOOLEAN</code>
<code>DECIMAL</code>	<code>NUMERIC</code>
<code>INT</code>	<code>INT4</code>

SQL Server data types	AWS DMS data types
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
VARCHAR (max)	CLOB TEXT  To use this data type with AWS DMS, you must enable the use of CLOB data types for a specific task.  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.  During CDC, AWS DMS supports CLOB data types only in tables that include a primary key.
NCHAR	WSTRING
NVARCHAR (length)	WSTRING

SQL Server data types	AWS DMS data types
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. For more information about enabling data types for a specific task, see <a href="#">Transformation rules and actions (p. 318)</a>.</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
UNIQUEIDENTIFIER	STRING
HIERARCHYID	<p>Use HIERARCHYID when replicating to a SQL Server target endpoint.</p> <p>Use WSTRING (250) when replicating to all other target endpoints.</p>

SQL Server data types	AWS DMS data types
XML	<p>NCLOB</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 NCLOB data types for a specific task.</p> <p>During CDC, AWS DMS supports NCLOB data types only in tables that include a primary key.</p>
GEOMETRY	<p>Use GEOMETRY when replicating to target endpoints that support this data type.</p> <p>Use CLOB when replicating to target endpoints that don't support this data type.</p>
GEOGRAPHY	<p>Use GEOGRAPHY when replicating to target endpoints that support this data type.</p> <p>Use CLOB when replicating to target endpoints that don't support this data type.</p>

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 \(p. 98\)](#).

**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 (for versions 9.x), 10.x, 11.x, and 12.x database as a source for these types of databases:

- On-premises databases
- Databases on an EC2 instance
- Databases on an Amazon RDS DB instance
- Databases on an Amazon Aurora DB instance with PostgreSQL compatibility

## Note

- AWS DMS does not work with Amazon RDS for PostgreSQL 10.4 or Amazon Aurora (PostgreSQL 10.4) either as source or target.
- PostgreSQL versions 12.x are supported as a source in AWS DMS versions 3.3.3 and later.
- PostgreSQL versions 11.x are supported as a source only in AWS DMS versions 3.3.1 and later. You can use PostgreSQL version 9.4 and later (versions 9.x) and 10.x as a source in any DMS version.
- PostgreSQL versions 10.x contain numerous changes in function names and folder names from previous versions.

In some cases, you might use a PostgreSQL version 10.x database as a source and an AWS DMS version earlier than 3.3.1. In these cases, see [Using PostgreSQL version 10.x as a source for AWS DMS \(p. 122\)](#) for information on preparing your database as a source for AWS DMS.

## Note

If you use a PostgreSQL 10.x database as a source with AWS DMS versions 3.3.1 or later, *don't* perform these preparations for source 10.x databases required for earlier AWS DMS versions.

For a summary of the AWS DMS version requirements to use the supported PostgreSQL source versions, see the following table.

PostgreSQL source version	AWS DMS version to use
9.x	Use any available AWS DMS version.
10.x	If you use a AWS DMS version earlier than 3.3.1, prepare the PostgreSQL source using the wrapper functions described in <a href="#">Using PostgreSQL version 10.x as a source for AWS DMS (p. 122)</a> .  If you use a AWS DMS version 3.3.1 or later, don't create these wrapper functions. You can use the PostgreSQL source without any additional preparation.
11.x	Use AWS DMS version 3.3.1.
12.x	Use AWS DMS version 3.3.3.

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 \(p. 435\)](#).

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. Amazon RDS also supports logical replication for an Amazon Aurora DB instance using versions 2.2.0 and 2.2.1, with PostgreSQL 10.6 compatibility.

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

### Topics

- [Migrating from PostgreSQL to PostgreSQL using AWS DMS \(p. 113\)](#)
- [Prerequisites for using a PostgreSQL database as a source for AWS DMS \(p. 116\)](#)
- [Security requirements when using a PostgreSQL database as a source for AWS DMS \(p. 117\)](#)
- [Limitations on using a PostgreSQL database as a source for AWS DMS \(p. 117\)](#)
- [Setting up an Amazon RDS PostgreSQL DB instance as a source \(p. 119\)](#)
- [Removing AWS DMS artifacts from a PostgreSQL source database \(p. 122\)](#)
- [Additional configuration settings when using a PostgreSQL database as a source for AWS DMS \(p. 122\)](#)
- [Using PostgreSQL version 10.x as a source for AWS DMS \(p. 122\)](#)
- [Extra connection attributes when using PostgreSQL as a source for AWS DMS \(p. 124\)](#)
- [Source data types for PostgreSQL \(p. 125\)](#)

## 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 with PostgreSQL compatibility, 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 Aurora PostgreSQL instance. Core or basic PostgreSQL data types most often migrate successfully.

Data types that are supported on the source database but aren't supported on the target might 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	Partially migrates	Doesn't migrate	Comments
INTEGER	X			
SMALLINT	X			
BIGINT	X			
NUMERIC/DECIMAL(p,s)		X		Where 0<p<39 and 0<s
NUMERIC/DECIMAL		X		Where p>38 or p=s=0
REAL	X			
DOUBLE	X			
SMALLSERIAL	X			
SERIAL	X			
BIGSERIAL	X			
MONEY	X			
CHAR		X		Without specified precision
CHAR(n)	X			
VARCHAR		X		Without specified precision

Data type	Migrates successfully	Partially migrates	Doesn't migrate	Comments
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			PostGIS spatial data type
LINE			X	
LSEG			X	
BOX			X	
PATH			X	
POLYGON	X			PostGIS spatial data type
CIRCLE			X	
JSON		X		
ARRAY			X	
COMPOSITE			X	
RANGE			X	
LINESTRING	X			PostGIS spatial data type

Data type	Migrates successfully	Partially migrates	Doesn't migrate	Comments
MULTIPOINT	X			PostGIS spatial data type
MULTILINESTRING	X			PostGIS spatial data type
MULTIPOLYGON	X			PostGIS spatial data type
GEOMETRYCOLLECTION	X			PostGIS spatial data type

#### Note

If the PostgreSQL NUMERIC(p,s) data type doesn't specify any precision and scale, AWS DMS uses a precision of 28 and a scale of 6 by default, NUMERIC(28,6). For example, the value 0.61111104488373 from the source is converted to 0.611111 on the PostgreSQL target.

## Migrating PostGIS spatial data types

*Spatial data* identifies the geometry information of an object or location in space. PostgreSQL object-relational databases support PostGIS spatial data types.

Before migrating PostgreSQL spatial data objects, ensure the PostGIS plugin is enabled at the global level. Doing this ensures AWS DMS creates the exact source spatial data columns for the PostgreSQL target database instance.

For PostgreSQL to PostgreSQL homogeneous migrations, AWS DMS supports the migration of PostGIS geometric and geographic (geodetic coordinates) data object types and subtypes such as the following:

- POINT
- LINESTRING
- POLYGON
- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

## Prerequisites for using a PostgreSQL database as a source for AWS DMS

For a PostgreSQL database to be a source for AWS DMS, do the following:

- Use a PostgreSQL database that is version 9.4.x or later.
- For full-load plus Change Data Capture (CDC) tasks or CDC-only tasks, grant superuser permissions for the user account specified for the PostgreSQL source database. Superuser permissions are needed to access replication-specific functions in the source. For full-load only tasks, SELECT permissions are needed on tables to migrate them.
- Add the IP address of the AWS DMS replication server to the pg\_hba.conf configuration file and enable replication and socket connections. For example:

```
# Replication Instance
host all all 12.3.4.56/00 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
host replication dms 12.3.4.56/00 md5
```

PostgreSQL's `pg_hba.conf` configuration file controls client authentication. (HBA stands for host-based authentication.) The file is traditionally stored in the database cluster's data directory.

- 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.

**Note**

Some parameters can only be set at server start; any changes to their entries in the configuration file will be ignored until the server is restarted. See PostgreSQL database documentation for more information.

- 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. Don't 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:

- AWS DMS doesn't work with Amazon RDS for PostgreSQL 10.4 or Amazon Aurora PostgreSQL 10.4 either as source or target.
- 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 isn't 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 \(p. 119\)](#).
- Replication of multiple tables with the same name where each name has a different case (for example, table1, TABLE1, and Table1) can cause unpredictable behavior. Because of this issue, AWS DMS doesn't support this type of replication.
- In most cases, AWS DMS supports change processing of CREATE, ALTER, and DROP DDL statements for tables. AWS DMS doesn't support this change processing if the tables are held in an inner function or procedure body block or in other nested constructs.

For example, the following change isn't captured:

```
CREATE OR REPLACE FUNCTION attu.create_distributors1() RETURNS void
LANGUAGE plpgsql
AS $$$
BEGIN
create table attu.distributors1(did serial PRIMARY KEY, name
varchar(40) NOT NULL);
END;
$$;
```

- Currently, boolean datatypes in a PostgreSQL source are migrated to a SQLServer target as bit datatype with inconsistent values. As a workaround, pre-create the table with a VARCHAR(1) datatype for the column (or let AWS DMS create the table), and then have downstream processing treat an "F" as False and a "T" as True.
- AWS DMS doesn't support change processing of TRUNCATE operations.
- The OID LOB data type isn't migrated to the target.
- If your source is a PostgreSQL database that is on-premises or on an Amazon EC2 instance, ensure that the `test_decoding` output plugin is installed on your source endpoint. You can find this plugin in the Postgres contrib package. For more information about the test-decoding plugin, see the [PostgreSQL documentation](#).
- AWS DMS doesn't support change processing to set and unset 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 data type isn't 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 0.61111104488373 from the source is 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. You can use a DB instance for both full-load tasks and for change data capture (CDC) for ongoing replication. You can use a Read Replica only for full-load tasks and not 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 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 \(p. 120\)](#).

### 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 for 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

You can't use Amazon RDS for PostgreSQL Read Replicas for CDC (ongoing replication).

#### To enable logical replication for an RDS PostgreSQL DB instance

1. 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 \(p. 120\)](#).
2. Set the `rds.logical_replication` parameter in your DB parameter group to 1. This static parameter requires a reboot of the DB instance 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 write ahead log (WAL) generation, so only set `rds.logical_replication` when you use logical replication slots.
3. Set the `wal_sender_timeout` parameter to 0, as a best practice. Setting this parameter to 0 prevents PostgreSQL from terminating replication connections that are inactive longer than the specified timeout. When AWS DMS migrates data, replication connections need to be able to last longer than the specified timeout.
4. Enable native CDC start points with PostgreSQL as a source by setting the `slotName` extra connection attribute to the name of an existing logical replication slot when you create the endpoint. This logical replication slot holds ongoing changes from the time of endpoint creation, so it supports replication from a previous point in time.

PostgreSQL writes the database changes to WAL files that are discarded only after AWS DMS successfully reads changes from the logical replication slot. Using logical replication slots can protect logged changes from being deleted before they are consumed by the replication engine.

However, depending on rate of change and consumption, changes being held in a logical replication slot can cause elevated disk usage. We recommend that you set space usage alarms in the source PostgreSQL instance when logical replication slots are used. For more information on setting the `slotName` extra connection attribute, see [Extra connection attributes when using PostgreSQL as a source for AWS DMS \(p. 124\)](#).

The following procedure walks through this approach in more detail.

### To use a native CDC start point to set up a CDC load of a PostgreSQL source endpoint

1. Identify the logical replication slot used by an earlier replication task (a parent task) that you want to use as a start point. Then query the `pg_replication_slots` view on your source database to make sure that this slot doesn't have any active connections. If it does, resolve and terminate them before proceeding.

For the following steps, assume that your logical replication slot is `abc1d2efghijk_34567890_z0yx98w7_6v54_32ut_1srq_1a2b34c5d67ef`.

2. Create a new source endpoint that includes the following extra connection attribute setting.

```
slotName=abc1d2efghijk_34567890_z0yx98w7_6v54_32ut_1srq_1a2b34c5d67ef;
```

3. Create a new CDC only task using the AWS DMS API or CLI. For example, using the CLI you might run the `create-replication-task` command following.

AWS DMS doesn't currently support creating a CDC task with a native start point using the console.

```
aws dms create-replication-task --replication-task-identifier postgresql-slot-name-test
--source-endpoint-arn arn:aws:dms:us-west-2:012345678901:endpoint:ABCD1EFGHIJK2LMNOPOQRST3UV4
--target-endpoint-arn arn:aws:dms:us-west-2:012345678901:endpoint:ZYX9WVUTSRQONM8LKJIHGF7ED6
--replication-instance-arn arn:aws:dms:us-west-2:012345678901:rep:AAAAAAAAA5BB4CCC3DDDD2EE
--migration-type cdc --table-mappings "file://mappings.json" --cdc-start-position "4AF/B00000D0"
--replication-task-settings "file://task-pg.json"
```

In the preceding command, the following options are set:

- `source-endpoint-arn` is set to the new value that you created in step 2.
- `replication-instance-arn` is set to the same value as for the parent task from step 1.
- `table-mappings` and `replication-task-settings` are set to the same values as for the parent task from step 1.
- `cdc-start-position` is set to a start position value. To find this start position, either query the `pg_replication_slots` view on your source database or view the console details for the parent task in step 1. For more information, see [Determining a CDC native start point \(p. 302\)](#).

When this CDC task runs, AWS DMS raises an error if the specified logical replication slot doesn't exist or the task isn't created with a valid setting for `cdc-start-position`.

### Migrating an Amazon RDS for PostgreSQL database without using the master user account

In some cases, you might not use the master user account for the Amazon RDS PostgreSQL DB instance that you are using as a source. In these cases, 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 `captureDDLs` extra connection attribute 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 \(p. 124\)](#).
- 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.
- The table-level parameter `REPLICATE IDENTITY` in PostgreSQL versions 9.4 and later lets you control information written to WAL (write-ahead log) that identify rows which are updated or deleted. `REPLICATE IDENTITY FULL` records the old values of all columns in the row. Use `REPLICATE IDENTITY FULL` carefully for each table as `FULL` generates an extra amount of WAL that may not be necessary.

## Using PostgreSQL version 10.x as a source for AWS DMS

PostgreSQL version 10.x databases have numerous changes in function names and folder names from previous PostgreSQL versions. These changes make certain migration actions not backward compatible when using AWS DMS versions earlier than 3.3.1.

**Note**

If you use a PostgreSQL 10.x database as a source in for AWS DMS 3.3.1 or later, *don't* perform the preparations described following. You can use the PostgreSQL source without any additional preparation.

Because most of the name changes are superficial, AWS DMS has created wrapper functions that let AWS DMS work with PostgreSQL versions 10.x. 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, use the same AWS DMS user account ([user\\_name](#)) that you used to create the source endpoint. To define and associate these wrapper functions with this account, run the SQL code following on the source PostgreSQL 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 don't run this preparatory code on a source PostgreSQL 10.x database for AWS DMS versions earlier than 3.3.1, you see an error like the following.

```
2018-10-29T02:57:50 [SOURCE_CAPTURE ]E: RetCode: SQL_ERROR SqlState: 42703
NativeError: 1 Message:
ERROR: column "xlog_position" does not exist;,
No query has been executed with that handle [1022502] (ar_odb(stmt.c:3647)
```

After you upgrade your AWS DMS version to 3.3.1 or later, follow these steps:

1. Remove the `fnRenames` reference from the `ALTER USER` statement you use to set your source PostgreSQL 10.x configuration search path.
2. Delete the `fnRenames` schema from your PostgreSQL database.

If you don't follow these steps after upgrading, you see the following error in the log when the `fnRenames` schema is accessed.

```
RetCode: SQL_ERROR SqlState: 42703 NativeError: 1 Message: ERROR: column "lsn" does not exist;
```

If AWS DMS is using a non-master user account for the database, you also need to set certain permissions to access these wrapper functions with source PostgreSQL 10.x databases. To set these permissions, perform the grants following.

```
GRANT USAGE ON SCHEMA fnRenames TO dms_superuser;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA fnRenames TO dms_superuser;
```

For more information on using non-master user accounts with a source PostgreSQL 10.x database, see [Migrating an Amazon RDS for PostgreSQL database without using the master user account \(p. 120\)](#).

## 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. If you have multiple connection attribute settings, separate them from each other by semicolons with no additional white space (for example, `oneSetting;thenAnother`).

The following table shows the extra connection attributes that you can use when using PostgreSQL as a source for AWS DMS.

Attribute name	Description
<code>captureDDLS</code>	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 <a href="#">Removing AWS DMS artifacts from a PostgreSQL source database (p. 122)</a> .  If this value is set to N, you don't have to create tables or triggers on the source database. For more information, see <a href="#">Migrating an Amazon RDS for PostgreSQL database without using the master user account (p. 120)</a> .  Streamed DDL events are captured.  Default value: Y  Valid values: Y/N  Example: <code>captureDDLS=Y;</code>
<code>ddlArtifactsSchema</code>	The schema in which the operational DDL database artifacts are created.  Default value: public

Attribute name	Description
	<p>Valid values: String</p> <p>Example: <code>ddlArtifactsSchema=xyzddlschema;</code></p>
<code>failTasksOnLobTruncation</code>	<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: <code>false</code></p> <p>Valid values: Boolean</p> <p>Example: <code>failTasksOnLobTruncation=true;</code></p>
<code>executeTimeout</code>	<p>Sets the client statement timeout for the PostgreSQL instance, in seconds. The default value is 60 seconds.</p> <p>Example: <code>executeTimeout=100;</code></p>
<code>slotName</code>	<p>Sets the name of a previously created logical replication slot for a CDC load of the PostgreSQL source instance.</p> <p>When used with the AWS DMS API <code>CdcStartPosition</code> request parameter, this attribute also enables using native CDC start points. DMS verifies that the specified logical replication slot exists before starting the CDC load task. It also verifies that the task was created with a valid setting of <code>CdcStartPosition</code>. If the specified slot doesn't exist or the task doesn't have a valid <code>CdcStartPosition</code> setting, DMS raises an error.</p> <p>For more information about how DMS uses logical replication slots to start CDC loads for PostgreSQL, see <a href="#">Using CDC with an RDS for PostgreSQL DB instance (p. 119)</a>. For more information about setting the <code>CdcStartPosition</code> request parameter, see <a href="#">Determining a CDC native start point (p. 302)</a>. For more information about using <code>CdcStartPosition</code>, see the documentation for the <code>CreateReplicationTask</code>, <code>StartReplicationTask</code>, and <code>ModifyReplicationTask</code> API operations in the <a href="#">AWS Database Migration Service API Reference</a>.</p> <p>Valid values: String</p> <p>Example: <code>slotName=abc1d2efghijk_34567890_z0yx98w7_6v54_32ut_1srq_1</code></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 \(p. 488\)](#).

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)  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
TIME	TIME
TIME (z)	TIME

PostgreSQL data types	AWS DMS data types
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)

PostgreSQL column sizes affect the conversion of PostgreSQL LOB data types to AWS DMS data types. To work with this, take the following steps for the following AWS DMS data types:

- BLOB – Set **Limit LOB size to the Maximum LOB size (KB)** value at task creation.
- CLOB – Replication handles each character as a UTF8 character. Therefore, find the length of the longest character text in the column, shown here as `max_num_chars_text` and use it to specify the value for **Limit LOB size to**. If the data includes 4-byte characters, multiply by 2 to specify the **Limit LOB size to** value, which is in bytes. In this case, **Limit LOB size to** is equal to `max_num_chars_text` multiplied by 2.
- NCLOB – Replication handles each character as a double-byte character. Therefore, find the length of the longest character text in the column (`max_num_chars_text`) and multiply by 2 to specify the value for **Limit LOB size to**. In this case, **Limit LOB size to** is equal to `max_num_chars_text` multiplied by 2. If the data includes 4-byte characters, multiply by 2 again. In this case, **Limit LOB size to** is equal to `max_num_chars_text` multiplied by 4.

## 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, 5.7, and 8.0. MariaDB versions 10.0.24 to 10.0.28, 10.1, 10.2, and 10.3 to 10.3.13, and also Amazon Aurora MySQL, are supported for on-premises.

### Note

Support for MySQL 8.0 as a source is available in AWS DMS versions 3.4.0 and later, except when the transaction payload is compressed.

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 \(p. 435\)](#).

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 S3.

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 \(p. 128\)](#)
- [Using any MySQL-compatible database as a source for AWS DMS \(p. 130\)](#)
- [Using a self-managed MySQL-compatible database as a source for AWS DMS \(p. 131\)](#)
- [Using an Amazon-managed MySQL-compatible database as a source for AWS DMS \(p. 132\)](#)
- [Limitations on using a MySQL database as a source for AWS DMS \(p. 132\)](#)
- [Extra connection attributes when using MySQL as a source for AWS DMS \(p. 133\)](#)
- [Source data types for MySQL \(p. 134\)](#)

## 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 with MySQL compatibility, 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 Aurora MySQL instance. Core or basic MySQL data types most often migrate successfully.

Data types that are supported on the source database but aren't 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, 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			

Data type	Migrates successfully	Will partially migrate	Will not migrate	Comments
YEAR	X			
DOUBLE	X			
FLOAT		X		
VARCHAR(N)	X			
VARBINARY(N)	X			
CHAR(N)	X			
TEXT	X			
LONGTEXT	X			
MEDIUMTEXT	X			
TINYTEXT	X			
JSON	X			Supported in AWS DMS versions 3.3.1 and later
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 UPDATE statements as INSERT 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 an 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 with MySQL compatibility

When using an Amazon-managed MySQL-compatible database as a source for AWS DMS, make sure that you have the following prerequisites:

- Enable automatic backups. For more information on setting up automatic backups, see [Working with automated backups](#) in the *Amazon RDS User Guide*.
- 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*.
- 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);
```

- Set the `binlog_format` parameter to "ROW".
- Set the `binlog_checksum` parameter 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, enable backups on the read replica.

## Limitations on using a MySQL database as a source for AWS DMS

When using a MySQL database as a source, consider the following:

- Change data capture (CDC) isn't supported for Amazon RDS MySQL 5.5 or lower. For Amazon RDS MySQL, you must use version 5.6 or 5.7 to enable CDC. Note that CDC is supported for self-managed MySQL 5.5 sources.
- The data definition language (DDL) statements `DROP TABLE` and `RENAME TABLE` aren't supported. Additionally, all DDL statements for partitioned tables aren't 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) isn't supported. Columns are always added to the end of the table.
- CDC isn't 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 AUTO\_INCREMENT attribute on a column isn't migrated to a target database column.
- Capturing changes when the binary logs aren't stored on standard block storage isn't supported. For example, CDC doesn't work when the binary logs are stored on Amazon S3.
- 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 chosen 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 aren't affected and are migrated to the target.
- Temporal data tables or system-versioned tables are not supported on MariaDB source and target databases.
- If migrating between two Amazon RDS Aurora MySQL clusters, the RDS Aurora MySQL source endpoint must be a read/write instance, not a read replica instance.

## 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. If you have multiple connection attribute settings, separate them from each other by semicolons with no additional white space (for example, oneSetting;thenAnother).

The following table shows the extra connection attributes available when using Amazon RDS MySQL as a source for AWS DMS.

Name	Description
eventsPollInterval	<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: eventsPollInterval=5;</p> <p>In the example, AWS DMS checks for changes in the binary logs every five seconds.</p>
serverTimezone	<p>Specifies the time zone for the source MySQL database.</p> <p>Example: serverTimezone=US/Pacific;</p> <p>Note: Do not enclose time zones in single quotes.</p>
afterConnectScript	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.

Name	Description
	<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>
CleanSrcMetadataOnMismatch	<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.

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 \(p. 488\)](#).

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

MySQL data types	AWS DMS data types
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.
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)
JSON	NCLOB
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.
- AWS DMS supports the JSON data type in versions 3.3.1 and later.

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—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.

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

## Topics

- Prerequisites for using an SAP ASE database as a source for AWS DMS (p. 136)
  - Limitations on using SAP ASE as a source for AWS DMS (p. 136)
  - Permissions required for using SAP ASE as a source for AWS DMS (p. 137)
  - Removing the truncation point (p. 137)
  - Extra connection attributes when using SAP ASE as a source for AWS DMS (p. 137)
  - Source data types for SAP ASE (p. 140)

## Prerequisites for using an SAP ASE database as a source for AWS DMS

For an SAP ASE database to be a source for AWS DMS, do the following:

- Enable SAP ASE replication for tables by using the `sp_setreptable` command.
  - Disable RepAgent on the SAP ASE database.
  - To replicate to SAP ASE version 15.7 on an Amazon EC2 instance on Microsoft Windows configured for non-Latin characters (for example, Chinese), install SAP ASE 15.7 SP121 on the target computer.

## Limitations on using SAP ASE as a source for AWS DMS

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

- Only one AWS DMS task can be run for each SAP ASE database.
  - You can't rename a table. For example, the following command fails:

```
sp_rename 'Sales.SalesRegion', 'SalesReg';
```

- You can't rename a column. For example, the following command fails:

```
sp_rename 'Sales.Sales.Region', 'RegID', 'COLUMN'
```

- If the database default is set not to allow NULL values, AWS DMS creates the target table with columns that don't allow NULL values. Consequently, if a full load or change data capture (CDC) replication task contains empty values, AWS DMS throws an error. You can prevent these errors by allowing NULL values in the source database by using the following commands.

```
sp_dboption database_name, 'allow nulls by default', 'true'  
go  
use database_name  
CHECKPOINT  
go
```

- The `reorg rebuild` index command isn't supported.
- Clusters aren't supported.

## 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`
- By default, where you need to have permission to run the `sp_setreptable` stored procedure, AWS DMS enables the SAP ASE replication option. If you want to run `sp_setreptable` on a table directly from the database endpoint and not through AWS DMS itself, you can use the `enableReplication` extra connection attribute. For more information, see [Extra connection attributes when using SAP ASE as a source for AWS DMS \(p. 137\)](#).

## 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).

## Extra connection attributes when using SAP ASE as a source for AWS DMS

You can use extra connection attributes to configure an SAP ASE source. You specify these settings when you create the source endpoint. You must separate multiple extra connection attribute settings from each other by semicolons and no additional white space.

The table following shows the extra connection attributes available when using SAP ASE as a source for AWS DMS.

Name	Description
charset	<p>Set this attribute to the SAP ASE name that corresponds to the international character set.</p> <p>Default value: <code>iso_1</code></p> <p>Example: <code>charset=utf8;</code></p> <p>Valid values:</p> <ul style="list-style-type: none"> <li>• <code>acsii_8</code></li> <li>• <code>big5hk</code></li> <li>• <code>cp437</code></li> <li>• <code>cp850</code></li> <li>• <code>cp852</code></li> <li>• <code>cp852</code></li> <li>• <code>cp855</code></li> <li>• <code>cp857</code></li> <li>• <code>cp858</code></li> <li>• <code>cp860</code></li> <li>• <code>cp864</code></li> <li>• <code>cp866</code></li> <li>• <code>cp869</code></li> <li>• <code>cp874</code></li> <li>• <code>cp932</code></li> <li>• <code>cp936</code></li> <li>• <code>cp950</code></li> <li>• <code>cp1250</code></li> <li>• <code>cp1251</code></li> <li>• <code>cp1252</code></li> <li>• <code>cp1253</code></li> <li>• <code>cp1254</code></li> <li>• <code>cp1255</code></li> <li>• <code>cp1256</code></li> <li>• <code>cp1257</code></li> <li>• <code>cp1258</code></li> <li>• <code>deckanji</code></li> <li>• <code>euccns</code></li> <li>• <code>eucgb</code></li> <li>• <code>eucjis</code></li> <li>• <code>eucksc</code></li> <li>• <code>gb18030</code></li> <li>• <code>greek8</code></li> <li>• <code>iso_1</code></li> <li>• <code>iso88592</code></li> <li>• <code>iso88595</code></li> <li>• <code>iso88596</code></li> </ul>

Name	Description
	<ul style="list-style-type: none"> <li>• iso88597</li> <li>• iso88598</li> <li>• iso88599</li> <li>• iso15</li> <li>• kz1048</li> <li>• koi8</li> <li>• roman8</li> <li>• iso88599</li> <li>• sjis</li> <li>• tis620</li> <li>• turkish8</li> <li>• utf8</li> </ul> <p>For any further questions about supported character sets in a SAP ASE database, see <a href="#">Adaptive Server Enterprise: Supported character sets</a>.</p>
enableReplication	<p>Set this attribute if you want to enable <code>sp_setreptable</code> on tables from the database end and not through AWS DMS.</p> <p>Default value: <code>true</code></p> <p>Example: <code>enableReplication=false;</code></p> <p>Valid values: <code>true</code> or <code>false</code></p>
encryptPassword	<p>Set this attribute if you have enabled "net password encryption reqd" at the source database.</p> <p>Default value: 0</p> <p>Example: <code>encryptPassword=1;</code></p> <p>Valid values: 0, 1, or 2</p> <p>For more information on these parameter values, see <a href="#">Adaptive Server Enterprise: Using the EncryptPassword Connection string property</a>.</p>
provider	<p>Set this attribute if you want to use TLS for versions of ASE 15.7 and later.</p> <p>Default value: <code>Adaptive Server Enterprise</code></p> <p>Example: <code>provider=Adaptive Server Enterprise 16.03.06;</code></p> <p>Valid values: <code>Adaptive Server Enterprise 16.03.06</code></p> <p><b>Note</b> AWS DMS supports this extra connection attribute in versions 3.3.2 and later.</p>

## 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 \(p. 154\)](#) section for your target endpoint.

For additional information about AWS DMS data types, see [Data types for AWS Database Migration Service \(p. 488\)](#).

SAP ASE data types	AWS DMS data types
BIGINT	INT8
BINARY	BYTES
BIT	BOOLEAN
CHAR	STRING
DATE	DATE
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
UNIEXT	NCLOB

SAP ASE data types	AWS DMS data types
UNIVARCHAR	UNICODE
VARBINARY	BYTES
VARCHAR	STRING

## Using MongoDB as a source for AWS DMS

AWS DMS supports MongoDB versions 3.x and 4.0 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* except when using Amazon DocumentDB as target.

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
5a94815f40bd44d1b02b4fe0		2	3
5a94815f40bd44d1b02b4fe1		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 isn't 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:

- **MONGODB-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 isn't 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. If you have multiple connection attribute settings, separate them from each other by semicolons with no additional white space (for example, `oneSetting;thenAnother`).

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
<code>authType</code>	"no" "password"	"password" – When "no" is specified, user name and password parameters aren't used and can be empty.
<code>authMechanism</code>	"default" "mongodb_cr" "scram_sha_1"	For the default value, in MongoDB version 2.x, "default" is "mongodb_cr". For MongoDB version 3.x or later, "default" is "scram_sha_1". This setting isn't used when <code>authType</code> is set to "no".
<code>nestingLevel</code>	"none" "one"	"none" – Specify "none" to use document mode. Specify "one" to use table mode.
<code>extractDocID</code>	"true" "false"	"false" – Use this attribute when <code>nestingLevel</code> is set to "none".
<code>docsToInvestigate</code>	A positive integer greater than 0.	1000 – Use this attribute when <code>nestingLevel</code> is set to "one".
<code>authSource</code>	A valid MongoDB database name.	"admin" – This attribute isn't used when <code>authType</code> is set to "no".

**Note**

If the target endpoint is DocumentDB, ensure that the following extra connection attributes for the MongoDB source are set as follows:

- `nestingLevel="none"`
- `extractDocID="false"`

For more information, see [Using Amazon DocumentDB as a target for AWS Database Migration Service \(p. 246\)](#).

## 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 \(p. 488\)](#).

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 S3 as a source for AWS DMS

You can migrate data from an Amazon S3 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 \(p. 150\)](#).

## 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,2014-06-04>New York  
102,Smith,Bob,2015-10-08,Los Angeles  
103,Smith,Bob,2017-03-13,Dallas  
104,Smith,Bob,2017-03-13,Dallas
```

Following is an example external table definition for this data.

```
{  
    "TableCount": "1",  
    "Tables": [  
        {  
            "TableName": "employee",  
            "TablePath": "hr/employee/",  
            "TableOwner": "hr",  
            "TableColumns": [  
                {  
                    "Column": "id",  
                    "Type": "NUMBER",  
                    "Length": 10,  
                    "Scale": 0  
                },  
                {  
                    "Column": "name",  
                    "Type": "STRING",  
                    "Length": 50  
                },  
                {  
                    "Column": "hireDate",  
                    "Type": "DATE",  
                    "Format": "YYYY-MM-DD"  
                },  
                {  
                    "Column": "location",  
                    "Type": "STRING",  
                    "Length": 50  
                }  
            ]  
        }  
    ]  
}
```

```
{
    "TableName": "Employees",
    "TablePath": "s3://mybucket/employees.csv",
    "TableOwner": "myuser",
    "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 S3 \(p. 151\)](#).
  - **ColumnLength** – the number of bytes in this column. Maximum column length is limited to 2147483647 Bytes (2,047 MegaBytes) since an S3 source doesn't support FULL LOB mode.
  - **ColumnNullable** – a Boolean value that is **true** if this column can contain NULL values (**default=false**).
  - **ColumnIsPk** – a Boolean value that is **true** if this column is part of the primary key (**default=false**).
- **TableColumnsTotal** – the total number of columns. This number must match the number of elements in the **TableColumns** array.

**ColumnLength** applies for the following data types:

- **BYTE**

- STRING

If you don't specify otherwise, AWS DMS assumes that `ColumnLength` is zero.

**Note**

In supported versions of AWS DMS, the S3 source data can also contain an optional operation column as the first column before the `TableName` column value. This operation column identifies the operation (`INSERT`) used to migrate the data to an S3 target endpoint during a full load.

If present, the value of this column is the initial character of the `INSERT` operation keyword (`I`). If specified, this column generally indicates that the S3 source was created by DMS as an S3 target during a previous migration.

In previous DMS versions, this column wasn't present in S3 source data created from a previous DMS full load. Adding this column to S3 target data allows the format of all rows written to the S3 target to be consistent whether they are written during a full load or during a CDC load. For more information on the options for formatting S3 target data, see [Indicating source DB operations in migrated S3 data \(p. 206\)](#).

For a column of the `NUMERIC` type, 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
...

```

**Note**

To replicate CDC files in the change data folder successfully upload them in a lexical (sequential) order. For example, upload the file `CDC00002.csv` before the file `CDC00003.csv`. Otherwise, `CDC00002.csv` is skipped and isn't replicated if you load it after `CDC00003.csv`. But the file `CDC00004.csv` replicates successfully if loaded after `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` or `I`, `UPDATE` or `U`, or `DELETE` or `D`. These keyword and character values are case-insensitive.

**Note**

In supported AWS DMS versions, AWS DMS can identify the operation to perform for each load record in two ways. AWS DMS can do this from the record's keyword value (for example, `INSERT`) or from its keyword initial character (for example, `I`). In prior versions, AWS DMS recognized the load operation only from the full keyword value.

In prior versions of AWS DMS, the full keyword value was written to log the CDC data. Also, prior versions wrote the operation value to any S3 target using only the keyword initial.

Recognizing both formats allows AWS DMS to handle the operation regardless of how the operation column is written to create the S3 source data. This approach supports using S3 target data as the source for a later migration. With this approach, you don't need to change the format of any keyword initial value that appears in the operation column of the later S3 source.

- 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,2014-06-04,New York
UPDATE,employee,hr,101,Smith,Bob,2015-10-08,Los Angeles
UPDATE,employee,hr,101,Smith,Bob,2017-03-13,Dallas
DELETE,employee,hr,101,Smith,Bob,2017-03-13,Dallas
```

## Prerequisites when using Amazon S3 as a source for AWS DMS

To use Amazon S3 as a source for AWS DMS, your source S3 bucket must be in the same AWS Region as the DMS replication instance that migrates 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 isn't specified, then the path used is <code>schemaName/tableName/</code> . An example follows.  <code>bucketFolder=testFolder;</code>
bucketName	The name of the S3 bucket. An example follows.  <code>bucketName=buckettest;</code>
cdcPath	The location of change data capture (CDC) files. This attribute is required if a task captures change data; otherwise, it's optional. If <code>cdcPath</code> is present, then AWS DMS reads CDC files from this path and replicates the data changes to the target endpoint. For more information, see <a href="#">Using CDC with Amazon S3 as a source for AWS DMS (p. 148)</a> . An example follows.  <code>cdcPath=dataChanges;</code>
csvDelimiter	The delimiter used to separate columns in the source files. The default is a comma. An example follows.  <code>csvDelimiter=,;</code>
csvRowDelimiter	The delimiter used to separate rows in the source files. The default is a newline ( <code>\n</code> ). An example follows.  <code>csvRowDelimiter=\n;</code>
externalTableDefinition	An <code>JSON</code> object that describes how AWS DMS should interpret the data in the Amazon S3 bucket during the migration. For more information, see <a href="#">Defining external tables for Amazon S3 as a source for AWS DMS (p. 146)</a> . An example follows.  <code>externalTableDefinition=json_object;</code>
ignoreHeaderRows	When this value is set to 1, AWS DMS ignores the first row header in a <code>.csv</code> file. A value of 1 enables the feature, a value of 0 disables the feature. The default is 0.  <code>ignoreHeaderRows=1;</code>
rfc4180	When this value is set to true or y, each leading double quotation mark has to be followed by an ending double quotation mark. This formatting complies with RFC 4180. When this value is set to false, string literals are copied to the target as is. In this case, a delimiter (row or column) signals the end of the field. Thus, you can't use a delimiter as part of the string, because it signals the end of the value.  The default is true.  Valid values: true, false, y, n

Option	Description
	Example: <code>rfc4180=false;</code>

## Source data types for Amazon S3

Data migration that uses Amazon S3 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 \(p. 146\)](#).

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 \(p. 488\)](#).

The following AWS DMS data types are used with Amazon S3 as a source:

- BYTE – Requires `ColumnLength`. For more information, see [Defining external tables for Amazon S3 as a source for AWS DMS \(p. 146\)](#).
- DATE
- TIME
- DATETIME
- TIMESTAMP
- INT1
- INT2
- INT4
- INT8
- NUMERIC – Requires `ColumnPrecision` and `ColumnScale`. For more information, see [Defining external tables for Amazon S3 as a source for AWS DMS \(p. 146\)](#).
- REAL4
- REAL8
- STRING – Requires `ColumnLength`. For more information, see [Defining external tables for Amazon S3 as a source for AWS DMS \(p. 146\)](#).
- UINT1
- UINT2
- UINT4
- UINT8
- BLOB
- CLOB
- BOOLEAN

## Using 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 Database Migration Service (AWS DMS). AWS DMS supports the following versions of Db2 LUW as migration sources:

- Version 9.7, with all Fix Packs supported
- Version 10.1, with all Fix Packs supported
- Version 10.5, with all Fix Packs supported except for Fix Pack 5
- Version 11.1, with all Fix Packs supported

You can use Secure Sockets Layer (SSL) to encrypt connections between your Db2 LUW endpoint and the replication instance. For more information on using SSL with a Db2 LUW endpoint, see [Using SSL with AWS Database Migration Service \(p. 435\)](#).

## 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), do the following:

- Set the database to be recoverable, which AWS DMS requires to capture changes. A database is recoverable if either or both of the database configuration parameters LOGARCHMETH1 and LOGARCHMETH2 are set to ON.
- Grant the user account the following permissions:

SYSADM or DBADM

DATAACCESS

## Limitations when using Db2 LUW as a source for AWS DMS

AWS DMS doesn't support clustered databases. However, you can define a separate Db2 LUW for each of the endpoints of a cluster.

When using ongoing replication (CDC), the following limitations apply:

- When a table with multiple partitions is truncated, the number of DDL events shown in the AWS DMS console is equal to the number of partitions. This is because Db2 LUW records a separate DDL for each partition.
- The following DDL actions aren't supported on partitioned tables:
  - ALTER TABLE ADD PARTITION
  - ALTER TABLE DETACH PARTITION
  - ALTER TABLE ATTACH PARTITION
- AWS DMS doesn't support an ongoing replication migration from a DB2 high availability disaster recovery (HADR) standby instance. The standby is inaccessible.
- The DECFLOAT data type isn't supported. Consequently, changes to DECFLOAT columns are ignored during ongoing replication.
- The RENAME COLUMN statement isn't supported.
- When performing updates to Multi-Dimensional Clustering (MDC) tables, each update is shown in the AWS DMS console as INSERT + DELETE.
- When the task setting **Include LOB columns in replication** isn't enabled, 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 is NULL.
- When the change table option is enabled, the first timestamp record in the table is zero (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 are ignored. This limitation only applies 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. If you have multiple connection attribute settings, separate them from each other by semicolons with no additional white space (for example, `oneSetting;thenAnother`).

The following table shows the extra connection attributes that you can use with Db2 LUW as a source.

Name	Description
<code>CurrentLSN</code>	For ongoing replication (CDC), use <code>CurrentLSN</code> to specify a log sequence number (LSN) where you want the replication to start.
<code>MaxKBytesPerRead</code>	Maximum number of bytes per read, as a NUMBER value. The default is 64 KB.
<code>SetDataCaptureChanges</code>	Enables ongoing replication (CDC) 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 that you're using.

For additional information about AWS DMS data types, see [Data types for AWS Database Migration Service \(p. 488\)](#).

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 is 16, then REAL8; if precision is 34, then STRING
GRAPHIC (n)	WSTRING, for fixed-length graphic strings of double byte chars with a length greater than 0 and less than or equal to 127

Db2 LUW data types	AWS DMS data types
VARGRAPHIC (n)	WSTRING, for varying-length graphic strings with a length greater than 0 and less than or equal to 16,352 double byte chars
LONG VARGRAPHIC (n)	CLOB, for varying-length graphic strings with a length greater than 0 and less than or equal to 16,352 double byte chars
CHARACTER (n)	STRING, for fixed-length strings of double byte chars with a length greater than 0 and less than or equal to 255
VARCHAR (n)	STRING, for varying-length strings of double byte chars with a length greater than 0 and less than or equal to 32,704
LONG VARCHAR (n)	CLOB, for varying-length strings of double byte chars with a length greater than 0 and less than or equal to 32,704
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 (n)	BLOB  Maximum length is 2,147,483,647 bytes
CLOB (n)	CLOB  Maximum length is 2,147,483,647 bytes
DBCLOB (n)	CLOB  Maximum length is 1,073,741,824 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.

For a comprehensive list of valid targets, see [Targets for AWS DMS \(p. 11\)](#).

**Note**

AWS DMS doesn't support migration across AWS Regions for the following target endpoint types:

- Amazon DynamoDB
- Amazon Elasticsearch Service
- Amazon Kinesis Data Streams

### Topics

- [Using an Oracle database as a target for AWS Database Migration Service \(p. 155\)](#)
- [Using a Microsoft SQL Server database as a target for AWS Database Migration Service \(p. 161\)](#)
- [Using a PostgreSQL database as a target for AWS Database Migration Service \(p. 165\)](#)
- [Using a MySQL-compatible database as a target for AWS Database Migration Service \(p. 168\)](#)
- [Using an Amazon Redshift database as a target for AWS Database Migration Service \(p. 173\)](#)
- [Using a SAP ASE database as a target for AWS Database Migration Service \(p. 185\)](#)
- [Using Amazon S3 as a target for AWS Database Migration Service \(p. 187\)](#)
- [Using an Amazon DynamoDB database as a target for AWS Database Migration Service \(p. 207\)](#)
- [Using Amazon Kinesis Data Streams as a target for AWS Database Migration Service \(p. 221\)](#)
- [Using Apache Kafka as a target for AWS Database Migration Service \(p. 232\)](#)
- [Using an Amazon Elasticsearch Service cluster as a target for AWS Database Migration Service \(p. 242\)](#)
- [Using Amazon DocumentDB as a target for AWS Database Migration Service \(p. 246\)](#)
- [Using Amazon Neptune as a target for AWS Database Migration Service \(p. 259\)](#)

## 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 \(p. 435\)](#). AWS DMS also supports the use of Oracle transparent data encryption (TDE) to encrypt data at rest in the target database because Oracle TDE does not require an encryption key or password to write to the database.

AWS DMS supports Oracle versions 10g, 11g, 12c, 18c, and 19c for on-premises and EC2 instances for the Enterprise, Standard, Standard One, and Standard Two editions as targets. AWS DMS supports Oracle versions 11g (version 11.2.0.3.v1 and later), 12c, 18c, and 19c for Amazon RDS instance databases for the Enterprise, Standard, Standard One, and Standard Two editions.

### Note

Support for Oracle version 19c as a target is available in AWS DMS versions 3.3.2 and later.  
Support for Oracle version 18c as a target is available in AWS DMS versions 3.3.1 and later.

When you use Oracle as a target, we assume that the data is to be migrated into the schema or user that is used for the target connection. If you want to migrate data to a different schema, use a schema transformation to do so. For example, suppose that your target endpoint connects to the user RDMASTER and you want to migrate from the user PERFDATA to PERFDATA. In this case, create a transformation like the following.

```
{  
    "rule-type": "transformation",  
    "rule-id": "2",  
    "rule-name": "2",  
}
```

```
"rule-action": "rename",
"rule-target": "schema",
"object-locator": {
  "schema-name": "PERFDATA"
},
"value": "PERFDATA"
}
```

When using Oracle as a target, AWS DMS migrates all tables and indexes to default table and index tablespaces in the target. If you want to migrate tables and indexes to different table and index namespaces, use a tablespace transformation to do so. For example, suppose that you have a set of tables in the `INVENTORY` schema assigned to some tablespaces in the Oracle source. For the migration, you want to assign all of these tables to a single `INVENTORYSPACE` tablespace in the target. In this case, create a transformation like the following.

```
{
  "rule-type": "transformation",
  "rule-id": "3",
  "rule-name": "3",
  "rule-action": "rename",
  "rule-target": "table-tablespace",
  "object-locator": {
    "schema-name": "INVENTORY",
    "table-name": "%",
    "table-tablespace-name": "%"
  },
  "value": "INVENTORYSPACE"
}
```

For more information about transformations, see [Specifying table selection and transformations rules using JSON \(p. 314\)](#).

If Oracle is both source and target, you can preserve existing table or index tablespace assignments by setting the Oracle source extra connection attribute, `enableHomogenousTablespace=true`. For more information, see [Extra connection attributes when using Oracle as a source for AWS DMS \(p. 88\)](#)

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 \(p. 156\)](#)
- [User account privileges required for using Oracle as a target \(p. 157\)](#)
- [Configuring an Oracle database as a target for AWS Database Migration Service \(p. 158\)](#)
- [Extra connection attributes when using Oracle as a target for AWS DMS \(p. 158\)](#)
- [Target data types for Oracle \(p. 159\)](#)

## 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 doesn't 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 the user or schema, which AWS DMS uses to connect to the target instance. To migrate multiple schemas, create multiple replication tasks.

- 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.
- With the batch optimized apply option, loading into the net changes table uses a direct path, which doesn't support XML type. As a workaround, you can use transactional apply mode.
- Empty strings migrated from source databases can be treated differently by the Oracle target (converted to one-space strings, for example). This can result in AWS DMS validation reporting a mismatch.

## User account privileges required for using Oracle as a target

To use an Oracle target in an AWS Database Migration Service task, grant the following privileges in the Oracle database. You grant these to the user account specified in the Oracle database definitions for AWS DMS.

- `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 these additional privileges:

- 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 a 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`.

- For some full load scenarios, you might choose the "DROP and CREATE table" or "TRUNCATE before loading" option where a target table schema is different from the DMS user's. In this case, grant DROP ANY TABLE.
- To store changes in change tables or an audit table where the target table schema is different from the DMS user's, grant 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 [User account privileges required for using Oracle as a target \(p. 157\)](#).

## 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. If you have multiple connection attribute settings, separate them from each other by semicolons with no additional white space.

The following table shows the extra connection attributes available when using Oracle as a target.

Name	Description
useDirectPathFullLoad	When set to Y, AWS DMS uses a direct path full load. Specify this value to enable the direct path protocol in the Oracle Call Interface (OCI). This OCI protocol enables the bulk loading of Oracle target tables during a full load.  Default value: Y  Valid values: Y/N  Example: useDirectPathFullLoad=N;

Name	Description
directPathParallelLoad	<p>When set to <code>true</code>, this attribute specifies a parallel load when <code>useDirectPathFullLoad</code> is set to <code>Y</code>. This attribute also only applies when you use the AWS DMS parallel load feature. For more information, see the description of the parallel-load operation in <a href="#">Table-settings rules and operations (p. 333)</a>.</p> <p>A limitation on specifying this parallel load setting is that the target table cannot have any constraints or indexes. For more information on this limitation, see <a href="#">Enabling Constraints After a Parallel Direct Path Load</a>. If constraints or indexes are enabled, setting this attribute to <code>true</code> has no effect.</p> <p>Default value: <code>false</code></p> <p>Valid values: <code>true/false</code></p> <p>Example: <code>directPathParallelLoad=true;</code></p>
directPathNoLog	<p>When set to <code>true</code>, this attribute helps to increase the commit rate on the Oracle target database by writing directly to tables and not writing a trail to database logs. For more information, see <a href="#">Direct-Load INSERT</a>. This attribute also only applies when you set <code>useDirectPathFullLoad</code> to <code>Y</code>.</p> <p>Default value: <code>false</code></p> <p>Valid values: <code>true/false</code></p> <p>Example: <code>directPathNoLog=true;</code></p>
charLengthSemantics	<p>Specifies whether the length of a character column is in bytes or in characters. To indicate that the character column length is in characters, set this attribute to <code>CHAR</code>. Otherwise, the character column length is in bytes.</p> <p>Default value: Not set to <code>CHAR</code></p> <p>Valid values: <code>CHAR</code></p> <p>Example: <code>charLengthSemantics=CHAR;</code></p>

## 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)

AWS DMS data type	Oracle data type
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	<p>With date indication: DATE</p> <p>With time indication: TIMESTAMP</p> <p>With timestamp indication: TIMESTAMP</p> <p>With timestamp_with_timezone indication: TIMESTAMP WITH TIMEZONE</p> <p>With timestamp_with_local_timezone indication: TIMESTAMP WITH LOCAL TIMEZONE With interval_year_to_month indication: INTERVAL YEAR TO MONTH</p> <p>With interval_day_to_second indication: INTERVAL DAY TO SECOND</p> <p>If length &gt; 4000: CLOB</p> <p>In all other cases: VARCHAR2 (length)</p>
UINT1	NUMBER (3)
UINT2	NUMBER (5)
UINT4	NUMBER (10)
UINT8	NUMBER (19)
WSTRING	<p>If length &gt; 2000: NCLOB</p> <p>In all other cases: NVARCHAR2 (length)</p>
BLOB	<p>BLOB</p> <p>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</p>

AWS DMS data type	Oracle data type
CLOB	<p>CLOB</p> <p>To use this data type with AWS DMS, you must enable the use of CLOBS for a specific task. During change data capture (CDC), CLOB data types are supported only in tables that include a primary key.</p> <p>STRING</p> <p>Beginning with AWS DMS 3.3.3, an Oracle VARCHAR2 data type on the source with a declared size greater than 4000 bytes maps through the AWS DMS CLOB to a STRING on the Oracle target.</p>
NCLOB	<p>NCLOB</p> <p>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.</p> <p>WSTRING</p> <p>Beginning with AWS DMS 3.3.3, an Oracle VARCHAR2 data type on the source with a declared size greater than 4000 bytes maps through the AWS DMS NCLOB to a WSTRING on the Oracle target.</p>
XMLTYPE	<p>The XMLTYPE target data type is only relevant in Oracle-to-Oracle replication tasks.</p> <p>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.</p>

## 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, 2016, 2017, and 2019. The Enterprise, Standard, Workgroup, Developer, and Web editions are supported by AWS DMS.

For Amazon RDS instance databases, AWS DMS supports as a target SQL Server versions 2008R2, 2012, 2014, 2016, 2017, and 2019. The Enterprise, Standard, Workgroup, Developer, and Web editions are supported by AWS DMS.

### Note

Support for Microsoft SQL Server version 2019 as a target is available in AWS DMS versions 3.3.2 and later.

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

## 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 on the **Advanced** tab on the AWS Management Console. 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.
- Currently, boolean data types in a PostgreSQL source are migrated to a SQLServer target as the bit data type with inconsistent values. As a workaround, precreate the table with a VARCHAR(1) data type for the column (or let AWS DMS create the table). Then have downstream processing treat an "F" as False and a "T" as True.

## 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:

- The AWS DMS user account must have at least the db\_owner user role on the SQL Server database that you are connecting to.
- A SQL Server system administrator must provide this permission to all AWS DMS user accounts.

## Extra connection attributes when using SQL Server 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. If you have multiple connection attribute settings, separate them from each other by semicolons with no additional white space.

The following table shows the extra connection attributes that you can use when SQL Server is the target.

Name	Description
useBCPFullLoad	<p>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 <b>use BCP for loading table</b> option.</p> <p>Default value: Y</p> <p>Valid values: Y/N</p> <p>Example: useBCPFullLoad=Y</p>

Name	Description
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 \(p. 488\)](#).

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>
TIME	For SQL Server 2008 and later, use DATETIME2 (%d).

AWS DMS data type	SQL Server data type
	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	NUMERIC (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"><li>• For SQL Server 2008 and later, use DATETIME2.</li><li>• For earlier versions, if the scale is 3 or less use DATETIME. In all other cases, use VARCHAR (37).</li></ul> 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 change data capture (CDC), AWS DMS supports CLOB data types only in tables that include a primary key.
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 (for 9.x), 10.x, 11.x, and 12.x as a target are supported for these types of databases:

- On-premises databases
- Databases on an EC2 instance
- Databases on an Amazon RDS DB instance
- Databases on an Amazon Aurora DB instance with PostgreSQL compatibility

## Note

- PostgreSQL versions 12.x are supported as a target in AWS DMS versions 3.3.3 and later.
- PostgreSQL versions 11.x are supported as a target in AWS DMS versions 3.3.1 and later. You can use PostgreSQL version 9.4 and later (for versions 9.x) and 10.x as a source in any DMS version.

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 \(p. 166\)](#)
- [Security requirements when using a PostgreSQL database as a target for AWS Database Migration Service \(p. 166\)](#)
- [Extra connection attributes when using PostgreSQL as a target for AWS DMS \(p. 166\)](#)
- [Target data types for PostgreSQL \(p. 167\)](#)

## 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.
- Amazon Aurora Serverless is available as a target for Amazon Aurora with PostgreSQL version 10.7 compatibility. For more information about Amazon Aurora Serverless, see [Using Amazon Aurora Serverless](#) in the *Amazon Aurora User Guide*.

## 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.

Your PostgreSQL target endpoint requires minimum user permissions to run an AWS DMS migration, see the following examples.

```
CREATE USER newuser WITH PASSWORD 'your-password';
ALTER SCHEMA schema_name OWNER TO newuser;
```

Or,

```
GRANT USAGE ON SCHEMA schema_name TO myuser;
GRANT CONNECT ON DATABASE postgres TO myuser;
GRANT CREATE ON DATABASE postgres TO myuser;
GRANT CREATE ON SCHEMA schema_name TO myuser;
GRANT UPDATE, INSERT, SELECT, DELETE, TRUNCATE ON ALL TABLES IN SCHEMA schema_name TO
myuser;
GRANT TRUNCATE ON schema_name."BasicFeed" TO myuser;
```

## 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. If you have multiple connection attribute settings, separate them from each other by semicolons with no additional white space.

The following table shows the extra connection attributes you can use to configure PostgreSQL as a target for AWS DMS.

Name	Description
maxFileSize	Specifies the maximum size (in KB) of any .csv file used to transfer data to PostgreSQL.

Name	Description
	<p>Default value: 32,768 KB (32 MB)</p> <p>Valid values: 1–1,048,576 KB (up to 1.1 GB)</p> <p>Example: <code>maxFileSize=512</code></p>
<code>executeTimeout</code>	<p>Sets the client statement timeout for the PostgreSQL instance, in seconds. The default value is 60 seconds.</p> <p>Example: <code>executeTimeout=100</code></p>
<code>afterConnectScript=SET session_replication_role='replica'</code>	<p>For use with change data capture (CDC) only, this attribute has AWS DMS bypass foreign keys and user triggers to reduce the time it takes to bulk load data.</p> <p><b>Note</b> This attribute is only effective in change data capture mode.</p>

## 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 \(p. 488\)](#).

AWS DMS data type	PostgreSQL data type
BOOL	BOOL
BYTES	BYTEA
DATE	DATE
TIME	TIME
DATETIME	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

AWS DMS data type	PostgreSQL data type
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).
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 S3. 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 \(p. 435\)](#).

AWS DMS supports versions 5.5, 5.6, 5.7, and 8.0 of MySQL and Aurora MySQL. In addition, AWS DMS supports MariaDB versions 10.0.24 to 10.0.28, 10.1, 10.2 and 10.3.

**Note**

Support for MySQL 8.0 as a target is available in AWS DMS versions 3.3.1 and later.

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 a table in 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** option. For more information, see [Full-load task settings \(p. 283\)](#).

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 \(p. 169\)](#)
- [Limitations on using a MySQL-compatible database as a target for AWS Database Migration Service \(p. 169\)](#)
- [Extra connection attributes when using a MySQL-compatible database as a target for AWS DMS \(p. 170\)](#)
- [Target data types for MySQL \(p. 171\)](#)

## 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 completed the following prerequisites:

- 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
```

- Set the database parameter `local_infile = 1` to enable AWS DMS to load data into the target database.

## 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.

- Aurora Serverless is available as a target for Amazon Aurora version 1, compatible with MySQL version 5.6. Aurora Serverless is available as a target for Amazon Aurora version 2, compatible with MySQL version 5.7. (Select Aurora MySQL version 2.07.1 to be able to use Aurora Serverless with MySQL 5.7 compatibility.) For more information about Aurora Serverless, see [Using Amazon Aurora Serverless](#) in the *Amazon Aurora User Guide*.

## 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. If you have multiple connection attribute settings, separate them from each other by semicolons with no additional white space.

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: MULTIPLE_DATABASES</p> <p>Valid values: {SPECIFIC_DATABASE, MULTIPLE_DATABASES}</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>

Name	Description
	Valid values: 1–5  Example: <code>parallelLoadThreads=1</code>
<code>initstmt=SET FOREIGN_KEY_CHECKS=0</code>	Disables foreign key checks.
<code>initstmt=SET time_zone</code>	Specifies the time zone for the target MySQL-compatible database.  Default value: UTC  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.  Example: <code>initstmt=SET time_zone=UTC</code>
<code>afterConnectScript=SET character_set_connection='latin1'</code>	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.
<code>maxFileSize</code>	Specifies the maximum size (in KB) of any .csv file used to transfer data to a MySQL-compatible database.  Default value: 32,768 KB (32 MB)  Valid values: 1–1,048,576  Example: <code>maxFileSize=512</code>
<code>CleanSrcMetadataOnMismatch</code>	Cleans and recreates table metadata information on the replication instance when a mismatch occurs. An example is a situation where running an alter DDL statement on a table might result in different information about the table cached in the replication instance. Boolean.  Default value: <code>false</code>  Example: <code>CleanSrcMetadataOnMismatch=false</code>

## 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 \(p. 488\)](#).

AWS DMS data types	MySQL data types
BOOLEAN	BOOLEAN

AWS DMS data types	MySQL data types
BYTES	If the length is from 1 through 65,535, then use VARBINARY (length).  If the length is from 65,536 through 2,147,483,647, then use LONGLOB.
DATE	DATE
TIME	TIME
TIMESTAMP	"If scale is => 0 and =< 6, then: DATETIME (Scale)  If scale is => 7 and =< 9, then: VARCHAR (37)"
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 VARCHAR (length).  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).

AWS DMS data types	MySQL data types
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. When 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 AWS Region .

If you use the AWS CLI or DMS API to migrate data to Amazon Redshift, 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 \(p. 420\)](#).

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 S3. When the files are uploaded to Amazon S3, 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 S3 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 \(p. 174\)](#)
- [Limitations on using Amazon Redshift as a target for AWS Database Migration Service \(p. 174\)](#)
- [Configuring an Amazon Redshift database as a target for AWS Database Migration Service \(p. 175\)](#)
- [Using enhanced VPC routing with an Amazon Redshift as a target for AWS Database Migration Service \(p. 175\)](#)
- [Creating and using AWS KMS keys to encrypt Amazon Redshift target data \(p. 176\)](#)
- [Endpoint settings when using Amazon Redshift as a target for AWS DMS \(p. 179\)](#)
- [Extra connection attributes when using Amazon Redshift as a target for AWS DMS \(p. 180\)](#)
- [Target data types for Amazon Redshift \(p. 183\)](#)

## 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. 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 S3 bucket to transfer data to the Amazon Redshift database. For AWS DMS to create the bucket, the console uses an 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 \(p. 420\)](#).
- 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.
- Set the target metadata task setting [BatchApplyEnabled \(p. 288\)](#) to `true` for AWS DMS to handle changes to Amazon Redshift target tables during CDC. A Primary Key on both the source and target table is required. Without a Primary Key, changes are applied statement by statement. And that can adversely affect task performance during CDC by causing target latency and impacting the cluster commit queue.

## 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.
- Applying a DELETE statement to a table with a multi-column primary key is not supported when any of the primary key column names use a reserved word. Go [here](#) to see a list of Amazon Redshift reserved words.

## 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 \(p. 180\)](#).

## Using enhanced VPC routing with an Amazon Redshift as a target for AWS Database Migration Service

If you use Enhanced VPC Routing with your Amazon Redshift target, all COPY traffic between your Amazon Redshift cluster and your data repositories goes through your 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.

## Creating and using AWS KMS keys to encrypt Amazon Redshift target data

You can encrypt your target data pushed to Amazon S3 before it is copied to Amazon Redshift. To do so, you can create and use custom AWS KMS keys. You can use the key you created to encrypt your target data using one of the following mechanisms when you create the Amazon Redshift target endpoint:

- Use the following option when you run the `create-endpoint` command using the AWS CLI.

```
--redshift-settings '{"EncryptionMode": "SSE_KMS", "ServerSideEncryptionKmsKeyId": "your-kms-key-ARN"}'
```

Here, `your-kms-key-ARN` is the Amazon Resource Name (ARN) for your KMS key. For more information, see [Endpoint settings when using Amazon Redshift as a target for AWS DMS \(p. 179\)](#).

- Set the extra connection attribute `encryptionMode` to the value `SSE_KMS` and the extra connection attribute `serverSideEncryptionKmsKeyId` to the ARN for your KMS key. For more information, see [Extra connection attributes when using Amazon Redshift as a target for AWS DMS \(p. 180\)](#).

To encrypt Amazon Redshift target data using a KMS key, you need an AWS Identity and Access Management (IAM) role that has permissions to access Amazon Redshift data. This IAM role is then accessed in a policy (a key policy) attached to the encryption key that you create. You can do this in your IAM console by creating the following:

- An IAM role with an Amazon-managed policy.
- A KMS encryption key with a key policy that references this role.

The following procedures describe how to do this.

### To create an IAM role with the required Amazon-managed policy

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**. The **Roles** page opens.
3. Choose **Create role**. The **Create role** page opens.
4. With **AWS service** chosen as the trusted entity, choose **DMS** as the service to use the role.
5. Choose **Next: Permissions**. The **Attach permissions policies** page appears.
6. Find and select the `AmazonDMSRedshiftS3Role` policy.
7. Choose **Next: Tags**. The **Add tags** page appears. Here, you can add any tags you want.
8. Choose **Next: Review** and review your results.
9. If the settings are what you need, enter a name for the role (for example, `DMS-Redshift-endpoint-access-role`), and any additional description, then choose **Create role**. The **Roles** page opens with a message indicating that your role has been created.

You have now created the new role to access Amazon Redshift resources for encryption with a specified name, for example `DMS-Redshift-endpoint-access-role`.

### To create an AWS KMS encryption key with a key policy that references your IAM role

#### Note

For more information about how AWS DMS works with AWS KMS encryption keys, see [Setting an encryption key and specifying AWS KMS permissions \(p. 431\)](#).

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose **Create key**. The **Configure key** page opens.
5. For **Key type**, choose **Symmetric**.

#### Note

When you create this key, you can only create a symmetric key, because all AWS services, such as Amazon Redshift, only work with symmetric encryption keys.

6. Choose **Advanced Options**. For **Key material origin**, make sure that **KMS** is chosen, then choose **Next**. The **Add labels** page opens.
7. For **Create alias and description**, enter an alias for the key (for example, `DMS-Redshift-endpoint-encryption-key`) and any additional description.
8. For **Tags**, add any tags that you want to help identify the key and track its usage, then choose **Next**. The **Define key administrative permissions** page opens showing a list of users and roles that you can choose from.
9. Add the users and roles that you want to manage the key. Make sure that these users and roles have the required permissions to manage the key.
10. For **Key deletion**, choose whether key administrators can delete the key, then choose **Next**. The **Define key usage permissions** page opens showing an additional list of users and roles that you can choose from.
11. For **This account**, choose the available users you want to perform cryptographic operations on Amazon Redshift targets. Also choose the role that you previously created in **Roles** to enable access to encrypt Amazon Redshift target objects, for example `DMS-Redshift-endpoint-access-role`.
12. If you want to add other accounts not listed to have this same access, for **Other AWS accounts**, choose **Add another AWS account**, then choose **Next**. The **Review and edit key policy** page opens, showing the JSON for the key policy that you can review and edit by typing into the existing JSON. Here, you can see where the key policy references the role and users (for example, `Admin` and `User1`) that you chose in the previous step. You can also see the different key actions permitted for the different principals (users and roles), as shown in the example following.

```
{  
  "Id": "key-consolepolicy-3",  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "Enable IAM User Permissions",  
      "Effect": "Allow",  
      "Principal": {  
        "AWS": [  
          "arn:aws:iam::111122223333:root"  
        ]  
      },  
      "Action": "kms:*",  
      "Resource": "*"
```

```
},
{
  "Sid": "Allow access for Key Administrators",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:iam::111122223333:role/Admin"
    ]
  },
  "Action": [
    "kms>Create*",
    "kms>Describe*",
    "kms>Enable*",
    "kms>List*",
    "kms>Put*",
    "kms>Update*",
    "kms>Revoke*",
    "kms>Disable*",
    "kms>Get*",
    "kms>Delete*",
    "kms>TagResource",
    "kms>UntagResource",
    "kms>ScheduleKeyDeletion",
    "kms>CancelKeyDeletion"
  ],
  "Resource": "*"
},
{
  "Sid": "Allow use of the key",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:iam::111122223333:role/DMS-Redshift-endpoint-access-role",
      "arn:aws:iam::111122223333:role/Admin",
      "arn:aws:iam::111122223333:role/User1"
    ]
  },
  "Action": [
    "kms>Encrypt",
    "kms>Decrypt",
    "kms>ReEncrypt*",
    "kms>GenerateDataKey*",
    "kms>DescribeKey"
  ],
  "Resource": "*"
},
{
  "Sid": "Allow attachment of persistent resources",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:iam::111122223333:role/DMS-Redshift-endpoint-access-role",
      "arn:aws:iam::111122223333:role/Admin",
      "arn:aws:iam::111122223333:role/User1"
    ]
  },
  "Action": [
    "kms>CreateGrant",
    "kms>ListGrants",
    "kms>RevokeGrant"
  ],
  "Resource": "*",
  "Condition": {
    "Bool": {
      "kms:GrantIsForAWSResource": true
    }
  }
}
```

```
        }  
    ]
```

13. Choose **Finish**. The **Encryption keys** page opens with a message indicating that your master encryption key has been created.

You have now created a new KMS key with a specified alias (for example, `DMS-Redshift-endpoint-encryption-key`). This key enables AWS DMS to encrypt Amazon Redshift target data.

## Endpoint settings when using Amazon Redshift as a target for AWS DMS

You can use endpoint settings to configure your Amazon Redshift target similar to using extra connection attributes. You can specify these settings when you create the target endpoint using the `create-endpoint` command in the AWS CLI, with the `--redshift-settings "json-settings"` option. Here, `json-settings` is a JSON object containing parameters to specify the settings. You can also specify a .json file containing the same `json-settings` object, for example, as in the following: `--redshift-settings file:///your-file-path/my_redshift_settings.json`. Here, `my_redshift_settings.json` is the name of a .json file that contains the same `json-settings` object.

The parameter names for endpoint settings are the same as the names for equivalent extra connections attributes, except that the parameter names for endpoint settings have initial caps. Also, not all Amazon Redshift target endpoint settings using extra connection attributes are available using the `--redshift-settings` option of the `create-endpoint` command. For more information about the available settings in an AWS CLI call to `create-endpoint`, see [create-endpoint](#) in the *AWS CLI Command Reference* for AWS DMS. For more information on these settings, see the equivalent extra connection attributes in [Extra connection attributes when using Amazon Redshift as a target for AWS DMS \(p. 180\)](#).

You can use Amazon Redshift target endpoint settings to configure the following:

- A custom AWS KMS data encryption key. You can then use this key to encrypt your data pushed to Amazon S3 before it is copied to Amazon Redshift.
- A custom S3 bucket as intermediate storage for data migrated to Amazon Redshift.

## KMS Key settings for data encryption

The following examples show configuring a custom KMS key to encrypt your data pushed to S3. To start, you might make the following `create-endpoint` call using the AWS CLI.

```
aws dms create-endpoint --endpoint-identifier redshift-target-endpoint --endpoint-type target  
--engine-name redshift --username your-username --password your-password  
--server-name your-server-name --port 5439 --database-name your-db-name  
--redshift-settings '{"EncryptionMode": "SSE_KMS",  
"ServerSideEncryptionKmsKeyId": "arn:aws:kms:us-east-1:111122223333:key/24c3c5a1-f34a-4519-a85b-2debbeff226d1"}'
```

Here, the JSON object specified by `--redshift-settings` option defines two parameters. One is an `EncryptionMode` parameter with the value `SSE_KMS`. The other is an `ServerSideEncryptionKmsKeyId` parameter with the value `arn:aws:kms:us-east-1:111122223333:key/24c3c5a1-f34a-4519-a85b-2debbeff226d1`. This value is an Amazon Resource Name (ARN) for your custom KMS key.

By default, S3 data encryption occurs using S3 server-side encryption. For the previous example's Amazon Redshift target, this is also equivalent of specifying its endpoint settings, as in the following example.

```
aws dms create-endpoint --endpoint-identifier redshift-target-endpoint --endpoint-type target  
--engine-name redshift --username your-username --password your-password  
--server-name your-server-name --port 5439 --database-name your-db-name  
--redshift-settings '{"EncryptionMode": "SSE_S3"}'
```

For more information about working with S3 server-side encryption, see [Protecting data using server-side encryption](#) in the *Amazon Simple Storage Service Developer Guide*.

**Note**

You can also use the CLI `modify-endpoint` command to change the value of the `EncryptionMode` parameter for an existing endpoint from `SSE_KMS` to `SSE_S3`. But you can't change the `EncryptionMode` value from `SSE_S3` to `SSE_KMS`.

## Amazon S3 bucket settings

When you migrate data to an Amazon Redshift target endpoint, AWS DMS uses a default Amazon S3 bucket as intermediate task storage before copying the migrated data to Amazon Redshift. For example, the examples shown for creating an Amazon Redshift target endpoint with a KMS data encryption key use this default S3 bucket (see [KMS Key settings for data encryption \(p. 179\)](#)).

You can instead specify a custom S3 bucket for this intermediate storage by including the following parameters in the value of your `--redshift-settings` option on the AWS CLI `create-endpoint` command:

- `BucketName` – A string you specify as the name of the S3 bucket storage.
- `BucketFolder` – (Optional) A string you can specify as the name of the storage folder in the specified S3 bucket.
- `ServiceAccessRoleArn` – The ARN of an IAM role that permits administrative access to the S3 bucket. Typically, you create this role based on the `AmazonDMSRedshiftS3Role` policy. For an example, see the procedure to create an IAM role with the required Amazon-managed policy in [Creating and using AWS KMS keys to encrypt Amazon Redshift target data \(p. 176\)](#).

**Note**

If you specify the ARN of a different IAM role using the `--service-access-role-arn` option of the `create-endpoint` command, this IAM role option takes precedence.

The following example shows how you might use these parameters to specify a custom Amazon S3 bucket in the following `create-endpoint` call using the AWS CLI.

```
aws dms create-endpoint --endpoint-identifier redshift-target-endpoint --endpoint-type target  
--engine-name redshift --username your-username --password your-password  
--server-name your-server-name --port 5439 --database-name your-db-name  
--redshift-settings '{"ServiceAccessRoleArn": "your-service-access-ARN",  
"BucketName": "your-bucket-name", "BucketFolder": "your-bucket-folder-name"}'
```

## 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 target endpoint. If you have multiple connection attribute settings, separate them from each other by semicolons with no additional white space.

The following table shows the extra connection attributes available when Amazon Redshift is the target.

Name	Description
<code>maxFileSize</code>	<p>Specifies the maximum size (in KB) of any .csv file used to transfer data to Amazon Redshift.</p> <p>Default value: 32768 KB (32 MB)</p> <p>Valid values: 1–1,048,576</p> <p>Example: <code>maxFileSize=512</code></p>
<code>fileTransferUploadStreams</code>	<p>Specifies the number of threads used to upload a single file.</p> <p>Default value: 10</p> <p>Valid values: 1–64</p> <p>Example: <code>fileTransferUploadStreams=20</code></p>
<code>acceptanydate</code>	<p>Specifies if any date format is accepted, including invalid dates formats such as 0000-00-00. Boolean value.</p> <p>Default value: false</p> <p>Valid values: true   false</p> <p>Example: <code>acceptanydate=true</code></p>
<code>dateformat</code>	<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 auto argument with the <code>dateformat</code> parameter. The auto 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: '<code>dateformat_string</code>' or <code>auto</code></p> <p>Example: <code>dateformat=auto</code></p>
<code>timeformat</code>	<p>Specifies the time format. This is a string input and is empty by default. The <code>auto</code> argument recognizes several formats that aren't 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: '<code>timeformat_string</code>'   <code>'auto'</code>   <code>'epochsecs'</code>   <code>'epochmillisecs'</code></p> <p>Example: <code>timeformat=auto</code></p>

Name	Description
<code>emptyasnull</code>	<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>
<code>truncateColumns</code>	<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><code>truncateColumns=true;</code></pre>
<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 <a href="#">Amazon Redshift Database Developer Guide</a>.</p> <p>Default value: false</p> <p>Valid values: true   false</p> <p>Example:</p> <pre><code>removeQuotes=true;</code></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><code>trimBlanks=false;</code></pre>

Name	Description
encryptionMode	<p>Specifies the server-side encryption mode that you want to use to push your data to S3 before it is copied to Amazon Redshift. The valid values are SSE_S3 (S3 server-side encryption) or SSE_KMS (KMS key encryption). If you choose SSE_KMS, set the serverSideEncryptionKmsKeyId parameter to the Amazon Resource Name (ARN) for the KMS key to be used for encryption.</p> <p><b>Note</b> You can also use the CLI modify-endpoint command to change the value of the encryptionMode attribute for an existing endpoint from SSE_KMS to SSE_S3. But you can't change the encryptionMode value from SSE_S3 to SSE_KMS.</p> <p>Default value: SSE_S3</p> <p>Valid values: SSE_S3 or SSE_KMS</p> <p>Example: encryptionMode=SSE_S3;</p>
serverSideEncryptionKmsKeyId	<p>If you set encryptionMode to SSE_KMS, set this parameter to the ARN for the KMS key. You can find this ARN by selecting the key alias in the list of KMS keys created for your account. When you create the key, you must associate specific policies and roles with it. For more information, see <a href="#">Creating and using AWS KMS keys to encrypt Amazon Redshift target data (p. 176)</a>.</p> <p>Example:</p> <pre>serverSideEncryptionKmsKeyId=arn:aws:kms:us-east-1:111122223333:key/24c3c5a1-f34a-4519-a85b-2debef226d1;</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 \(p. 488\)](#).

AWS DMS data types	Amazon Redshift data types
BOOLEAN	BOOL
BYTES	VARCHAR (Length)
DATE	DATE
TIME	VARCHAR(20)
DATETIME	If the scale is => 0 and =< 6, then:

AWS DMS data types	Amazon Redshift data types
	TIMESTAMP (s)  If the scale is => 7 and =< 9, then:  VARCHAR (37)
INT1	INT2
INT2	INT2
INT4	INT4
INT8	INT8
NUMERIC	If the scale is => 0 and =< 37, then:  NUMERIC (p,s)  If the scale is => 38 and =< 127, then:  VARCHAR (Length)
REAL4	FLOAT4
REAL8	FLOAT8
STRING	If the length is 1–65,535, then use VARCHAR (length in bytes)  If the length is 65,536–2,147,483,647, then use VARCHAR (65535)
UINT1	INT2
UINT2	INT2
UINT4	INT4
UINT8	NUMERIC (20,0)
WSTRING	If the length is 1–65,535, then use NVARCHAR (length in bytes)  If the length is 65,536–2,147,483,647, then use NVARCHAR (65535)
BLOB	VARCHAR (maximum LOB size *2)  The maximum LOB size cannot exceed 31 KB. Amazon Redshift doesn't support VARCHARs larger than 64 KB.
NCLOB	NVARCHAR (maximum LOB size)  The maximum LOB size cannot exceed 63 KB. Amazon Redshift doesn't support VARCHARs larger than 64 KB.

AWS DMS data types	Amazon Redshift data types
CLOB	VARCHAR (maximum LOB size)  The maximum LOB size cannot exceed 63 KB. Amazon Redshift doesn't support VARCHARs larger than 64 KB.

## 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:

- Provide SAP ASE account access to the AWS DMS user. This user must have read/write privileges in the SAP ASE database.
- In some cases, you might replicate to SAP ASE version 15.7 installed on an Amazon EC2 instance on Microsoft Windows that is configured with non-Latin characters (for example, Chinese). In such cases, 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. If you have multiple connection attribute settings, separate them from each other by semicolons with no additional white space.

The following table shows the extra connection attributes available when using SAP ASE as a target:

Name	Description
driver	<p>Set this attribute if you want to use TLS for versions of ASE 15.7 and later.</p> <p>Default value: <code>Adaptive Server Enterprise</code></p> <p>Example: <code>driver=Adaptive Server Enterprise 16.03.06;</code></p> <p>Valid values: <code>Adaptive Server Enterprise 16.03.06</code></p> <p><b>Note</b> AWS DMS supports this extra connection attribute in versions 3.3.2 and later.</p>

Name	Description
additionalConnectionProperties	Any additional ODBC connection parameters that you want to specify.

## 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 \(p. 488\)](#).

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 S3 as a target for AWS Database Migration Service

You can migrate data to Amazon S3 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 value (.csv) format by default. For more compact storage and faster query options, you also have the option to have the data written to Apache Parquet (.parquet) 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 for .csv files. AWS DMS names CDC files using timestamps, for example 20141029-1134010000.csv. For each source table that contains records, AWS DMS creates a folder under the specified target folder (if the source table is not empty). AWS DMS writes all full load and CDC files to the specified Amazon S3 bucket.

The parameter `bucketFolder` contains the location where the .csv or .parquet files are stored before being uploaded to the S3 bucket. With .csv files, table data is stored in the following format in the S3 bucket.

```
<schema_name>/<table_name>/LOAD0000001.csv
<schema_name>/<table_name>/LOAD0000002.csv
...
<schema_name>/<table_name>/LOAD0000009.csv
<schema_name>/<table_name>/LOAD000000A.csv
<schema_name>/<table_name>/LOAD000000B.csv
...
<schema_name>/<table_name>/LOAD000000F.csv
<schema_name>/<table_name>/LOAD0000010.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 \(p. 200\)](#) at the end of this section.

When you use AWS DMS to replicate data changes, the first column of the .csv or .parquet output file indicates how the data was changed as shown for the following .csv file.

```
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 or .parquet 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 or .parquet 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 or .parquet 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 or .parquet 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 or .parquet 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 S3 as a target

Before using Amazon S3 as a target, check that the following are true:

- The S3 bucket that you're using as a target is in the same AWS Region as the DMS replication instance you are using to migrate your data.
- The AWS account that you use for the migration has an IAM role with write and delete access to the S3 bucket you are using as a target.
- This role has tagging access so you can tag any S3 objects written to the target bucket.
- The IAM role has DMS (dms.amazonaws.com) added as *trusted entity*.

To set up this account access, ensure that the role assigned to the user account used to create the migration task has the following set of permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:DeleteObject",
                "s3:PutObjectTagging"
            ],
            "Resource": [
                "arn:aws:s3:::buckettest2/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3>ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::buckettest2"
            ]
        }
    ]
}
```

## Limitations to using Amazon S3 as a target

The following limitations apply when using Amazon S3 as a target:

- A VPCE-enabled (gateway VPC) S3 bucket isn't currently supported.
- The following data definition language (DDL) commands are supported for change data capture (CDC): Truncate Table, Drop Table, Create Table, Rename Table, Add Column, Drop Column, Rename Column, and Change Column Data Type.
- 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 S3 as a target, the account used for the migration must have write and delete access to the Amazon S3 bucket that is used as the target. Specify the Amazon Resource Name (ARN) of an IAM role that has the permissions required to access Amazon S3.

AWS DMS supports a set of predefined grants for Amazon S3, known as canned access control lists (ACLs). Each canned ACL has a set of grantees and permissions that you can use to set permissions for the Amazon S3 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 \(p. 200\)](#). For more information about Amazon S3 canned ACLs, see [Canned ACL](#).

The IAM role that you use for the migration must be able to perform the `s3:PutObjectAcl` API operation.

## Using Apache parquet to store Amazon S3 objects

The comma-separated value (.csv) format is the default storage format for Amazon S3 target objects. For more compact storage and faster queries, you can instead use Apache Parquet (.parquet) as the storage format.

Apache Parquet is an open-source file storage format originally designed for Hadoop. For more information on Apache Parquet, see <https://parquet.apache.org/>.

To set .parquet as the storage format for your migrated S3 target objects, you can use the following mechanisms:

- Endpoint settings that you provide as parameters of a JSON object when you create the endpoint using the AWS CLI or the API for AWS DMS. For more information, see [Endpoint settings when using Amazon S3 as a target for AWS DMS \(p. 198\)](#).
- Extra connection attributes that you provide as a semicolon-separated list when you create the endpoint. For more information, see [Extra connection attributes when using Amazon S3 as a target for AWS DMS \(p. 200\)](#).

## Amazon S3 object tagging

You can tag Amazon S3 objects that a replication instance creates by specifying appropriate JSON objects as part of task-table mapping rules. For more information about requirements and options for S3 object tagging, including valid tag names, see [Object tagging](#) in the *Amazon Simple Storage Service Developer Guide*. For more information about table mapping using JSON, see [Specifying table selection and transformations rules using JSON \(p. 314\)](#).

You tag S3 objects created for specified tables and schemas by using one or more JSON objects of the selection rule type. You then follow this selection object (or objects) by one or more JSON objects of the post-processing rule type with add-tag action. These post-processing rules identify the S3 objects that you want to tag and specify the names and values of the tags that you want to add to these S3 objects.

You can find the parameters to specify in JSON objects of the post-processing rule type in the following table.

Parameter	Possible values	Description
rule-type	post-processing	A value that applies post-processing actions to the generated target objects. You can specify one or more post-processing rules to tag selected S3 objects.
rule-id	A numeric value.	A unique numeric value to identify the rule.
rule-name	An alphanumeric value.	A unique name to identify the rule.
rule-action	add-tag	The post-processing action that you want to apply to the S3 object. You can add one or more tags using a single JSON post-processing object for the add-tag action.
object-locator	schema-name – The name of the table schema.  table-name – The name of the table.	The name of each schema and table to which the rule applies. You can use the "%" percent sign as a wildcard for all or part of the value of each object-locator parameter. Thus, you can match these items: <ul style="list-style-type: none"> <li>• A single table in a single schema</li> <li>• A single table in some or all schemas</li> <li>• Some or all tables in a single schema</li> <li>• Some or all tables in some or all schemas</li> </ul>
tag-set	key – Any valid name for a single tag.  value – Any valid JSON value for this tag.	The names and values for one or more tags that you want to set on each created S3 object that matches the specified object-locator. You can specify up to 10 key-value pairs in a single tag-set parameter object. For more information on S3 object tagging, see <a href="#">Object tagging</a> in the <i>Amazon Simple Storage Service Developer Guide</i> . <p>You can also specify a dynamic value for all or part of the value for both the key and value parameters of a tag using \${dyn-value}. Here, \${dyn-value} can be either \${schema-name} or \${table-name}. Thus, you can insert the name of the currently selected schema or table as the whole or any part of the parameter value.</p> <p><b>Note</b></p> <p><b>Important</b> If you insert a dynamic value for the key parameter, you can</p>

Parameter	Possible values	Description
		generate tags with duplicate names for an S3 object, depending on how you use it. In this case, only one of the duplicate tag settings is added to the object.

When you specify multiple post-processing rule types to tag a selection of S3 objects, each S3 object is tagged using only one tag-set object from one post-processing rule. The particular tag set used to tag a given S3 object is the one from the post-processing rule whose associated object locator best matches that S3 object.

For example, suppose that two post-processing rules identify the same S3 object. Suppose also that the object locator from one rule uses wildcards and the object locator from the other rule uses an exact match to identify the S3 object (without wildcards). In this case, the tag set associated with the post-processing rule with the exact match is used to tag the S3 object. If multiple post-processing rules match a given S3 object equally well, the tag set associated with the first such post-processing rule is used to tag the object.

#### Example Adding static tags to an S3 object created for a single table and schema

The following selection and post-processing rules add three tags (`tag_1`, `tag_2`, and `tag_3` with corresponding static values `value_1`, `value_2`, and `value_3`) to a created S3 object. This S3 object corresponds to a single table in the source named `STOCK` with a schema named `aat2`.

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "5",
      "rule-name": "5",
      "object-locator": {
        "schema-name": "aat2",
        "table-name": "STOCK"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "post-processing",
      "rule-id": "41",
      "rule-name": "41",
      "rule-action": "add-tag",
      "object-locator": {
        "schema-name": "aat2",
        "table-name": "STOCK",
      },
      "tag-set": [
        {
          "key": "tag_1",
          "value": "value_1"
        },
        {
          "key": "tag_2",
          "value": "value_2"
        },
        {
          "key": "tag_3",
          "value": "value_3"
        }
      ]
    }
  ]
}
```

```
        "value": "value_3"
    }
]
}
}
```

### Example Adding static and dynamic tags to S3 objects created for multiple tables and schemas

The following example has one selection and two post-processing rules, where input from the source includes all tables and all of their schemas.

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "%",
        "table-name": "%"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "post-processing",
      "rule-id": "21",
      "rule-name": "21",
      "rule-action": "add-tag",
      "object-locator": {
        "schema-name": "%",
        "table-name": "%"
      },
      "tag-set": [
        {
          "key": "dw-schema-name",
          "value": "${schema-name}"
        },
        {
          "key": "dw-schema-table",
          "value": "my_prefix_${table-name}"
        }
      ]
    },
    {
      "rule-type": "post-processing",
      "rule-id": "41",
      "rule-name": "41",
      "rule-action": "add-tag",
      "object-locator": {
        "schema-name": "aat",
        "table-name": "ITEM",
      },
      "tag-set": [
        {
          "key": "tag_1",
          "value": "value_1"
        },
        {
          "key": "tag_2",
          "value": "value_2"
        }
      ]
    }
}
```

```
    ]  
}
```

The first post-processing rule adds two tags (`dw-schema-name` and `dw-schema-table`) with corresponding dynamic values (`#{schema-name}` and `my_prefix_#{table-name}`) to almost all S3 objects created in the target. The exception is the S3 object identified and tagged with the second post-processing rule. Thus, each target S3 object identified by the wildcard object locator is created with tags that identify the schema and table to which it corresponds in the source.

The second post-processing rule adds `tag_1` and `tag_2` with corresponding static values `value_1` and `value_2` to a created S3 object that is identified by an exact-match object locator. This created S3 object thus corresponds to the single table in the source named `ITEM` with a schema named `aat`. Because of the exact match, these tags replace any tags on this object added from the first post-processing rule, which matches S3 objects by wildcard only.

### Example Adding both dynamic tag names and values to S3 objects

The following example has two selection rules and one post-processing rule. Here, input from the source includes just the `ITEM` table in either the `retail` or `wholesale` schema.

```
{  
  "rules": [  
    {  
      "rule-type": "selection",  
      "rule-id": "1",  
      "rule-name": "1",  
      "object-locator": {  
        "schema-name": "retail",  
        "table-name": "ITEM"  
      },  
      "rule-action": "include"  
    },  
    {  
      "rule-type": "selection",  
      "rule-id": "1",  
      "rule-name": "1",  
      "object-locator": {  
        "schema-name": "wholesale",  
        "table-name": "ITEM"  
      },  
      "rule-action": "include"  
    },  
    {  
      "rule-type": "post-processing",  
      "rule-id": "21",  
      "rule-name": "21",  
      "rule-action": "add-tag",  
      "object-locator": {  
        "schema-name": "%",  
        "table-name": "ITEM",  
      },  
      "tag-set": [  
        {  
          "key": "dw-schema-name",  
          "value": "#{schema-name}"  
        },  
        {  
          "key": "dw-schema-table",  
          "value": "my_prefix_ITEM"  
        },  
        {  
          "key": "#{schema-name}_ITEM_tag_1",  
          "value": "value_1"  
        }  
      ]  
    }  
  ]  
}
```

```
        },
        {
            "key": "${schema-name}_ITEM_tag_2",
            "value": "value_2"
        }
    ]
}
```

The tag set for the post-processing rule adds two tags (`dw-schema-name` and `dw-schema-table`) to all S3 objects created for the `ITEM` table in the target. The first tag has the dynamic value `"${schema-name}"` and the second tag has a static value, `"my_prefix_ITEM"`. Thus, each target S3 object is created with tags that identify the schema and table to which it corresponds in the source.

In addition, the tag set adds two additional tags with dynamic names (`-${schema-name}_ITEM_tag_1` and `-${schema-name}_ITEM_tag_2`). These have the corresponding static values `value_1` and `value_2`. Thus, these tags are each named for the current schema, `retail` or `wholesale`. You can't create a duplicate dynamic tag name in this object, because each object is created for a single unique schema name. The schema name is used to create an otherwise unique tag name.

## Creating AWS KMS keys to encrypt Amazon S3 target objects

You can create and use custom AWS KMS keys to encrypt your Amazon S3 target objects. After you create a KMS key, you can use it to encrypt objects using one of the following approaches when you create the S3 target endpoint:

- Use the following options for S3 target objects (with the default .csv file storage format) when you run the `create-endpoint` command using the AWS CLI.

```
--s3-settings '{"ServiceAccessRoleArn": "your-service-access-ARN",
"CsvRowDelimiter": "\n", "CsvDelimiter": ",", "BucketFolder": "your-bucket-folder",
"BucketName": "your-bucket-name", "EncryptionMode": "SSE_KMS",
"ServerSideEncryptionKmsKeyId": "your-KMS-key-ARN"}'
```

Here, `your-KMS-key-ARN` is the Amazon Resource Name (ARN) for your AWS KMS key. For more information, see [Endpoint settings when using Amazon S3 as a target for AWS DMS \(p. 198\)](#).

- Set the extra connection attribute `encryptionMode` to the value `SSE_KMS` and the extra connection attribute `serverSideEncryptionKmsKeyId` to the ARN for your AWS KMS key. For more information, see [Extra connection attributes when using Amazon S3 as a target for AWS DMS \(p. 200\)](#).

To encrypt Amazon S3 target objects using an AWS KMS key, you need an IAM role that has permissions to access the Amazon S3 bucket. This IAM role is then accessed in a policy (a key policy) attached to the encryption key that you create. You can do this in your IAM console by creating the following:

- A policy with permissions to access the Amazon S3 bucket.
- An IAM role with this policy.
- An AWS KMS encryption key with a key policy that references this role.

The following procedures describe how to do this.

### To create an IAM policy with permissions to access the Amazon S3 bucket

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies** in the navigation pane. The **Policies** page opens.

3. Choose **Create policy**. The **Create policy** page opens.
  4. Choose **Service** and choose **S3**. A list of action permissions appears.
  5. Choose **Expand all** to expand the list and choose the following permissions at a minimum:
    - **ListBucket**
    - **PutObject**
    - **DeleteObject**
- Choose any other permissions you need, and then choose **Collapse all** to collapse the list.
6. Choose **Resources** to specify the resources that you want to access. At a minimum, choose **All resources** to provide general Amazon S3 resource access.
  7. Add any other conditions or permissions you need, then choose **Review policy**. Check your results on the **Review policy** page.
  8. If the settings are what you need, enter a name for the policy (for example, **DMS-S3-endpoint-access**), and any description, then choose **Create policy**. The **Policies** page opens with a message indicating that your policy has been created.
  9. Search for and choose the policy name in the **Policies** list. The **Summary** page appears displaying JSON for the policy similar to the following.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "VisualEditor0",  
            "Effect": "Allow",  
            "Action": [  
                "s3:PutObject",  
                "s3>ListBucket",  
                "s3>DeleteObject"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

You have now created the new policy to access Amazon S3 resources for encryption with a specified name, for example **DMS-S3-endpoint-access**.

#### To create an IAM role with this policy

1. On your IAM console, choose **Roles** in the navigation pane. The **Roles** detail page opens.
2. Choose **Create role**. The **Create role** page opens.
3. With AWS service selected as the trusted entity, choose **DMS** as the service to use the IAM role.
4. Choose **Next: Permissions**. The **Attach permissions policies** view appears in the **Create role** page.
5. Find and select the IAM policy for the IAM role that you created in the previous procedure (**DMS-S3-endpoint-access**).
6. Choose **Next: Tags**. The **Add tags** view appears in the **Create role** page. Here, you can add any tags you want.
7. Choose **Next: Review**. The **Review** view appears in the **Create role** page. Here, you can verify the results.
8. If the settings are what you need, enter a name for the role (required, for example, **DMS-S3-endpoint-access-role**), and any additional description, then choose **Create role**. The **Roles** detail page opens with a message indicating that your role has been created.

You have now created the new role to access Amazon S3 resources for encryption with a specified name, for example, DMS-S3-endpoint-access-role.

### To create an AWS KMS encryption key with a key policy that references your IAM role

#### Note

For more information about how AWS DMS works with AWS KMS encryption keys, see [Setting an encryption key and specifying AWS KMS permissions \(p. 431\)](#).

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose **Create key**. The **Configure key** page opens.
5. For **Key type**, choose **Symmetric**.

#### Note

When you create this key, you can only create a symmetric key, because all AWS services, such as Amazon S3, only work with symmetric encryption keys.

6. Choose **Advanced Options**. For **Key material origin**, make sure that **KMS** is chosen, then choose **Next**. The **Add labels** page opens.
7. For **Create alias and description**, enter an alias for the key (for example, DMS-S3-endpoint-encryption-key) and any additional description.
8. For **Tags**, add any tags that you want to help identify the key and track its usage, then choose **Next**. The **Define key administrative permissions** page opens showing a list of users and roles that you can choose from.
9. Add the users and roles that you want to manage the key. Make sure that these users and roles have the required permissions to manage the key.
10. For **Key deletion**, choose whether key administrators can delete the key, then choose **Next**. The **Define key usage permissions** page opens showing an additional list of users and roles that you can choose from.
11. For **This account**, choose the available users you want to perform cryptographic operations on Amazon S3 targets. Also choose the role that you previously created in **Roles** to enable access to encrypt Amazon S3 target objects, for example DMS-S3-endpoint-access-role).
12. If you want to add other accounts not listed to have this same access, for **Other AWS accounts**, choose **Add another AWS account**, then choose **Next**. The **Review and edit key policy** page opens, showing the JSON for the key policy that you can review and edit by typing into the existing JSON. Here, you can see where the key policy references the role and users (for example, Admin and User1) that you chose in the previous step. You can also see the different key actions permitted for the different principals (users and roles), as shown in the example following.

```
{  
  "Id": "key-consolepolicy-3",  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "Enable IAM User Permissions",  
      "Effect": "Allow",  
      "Principal": {  
        "AWS": [  
          "arn:aws:iam::111122223333:root"  
        ]  
      },  
      "Action": "kms:*",  
      "Resource": "*"  
    },  
  ],  
  "SignatureVersion": "2012-10-17",  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "Enable IAM User Permissions",  
      "Effect": "Allow",  
      "Principal": {  
        "AWS": [  
          "arn:aws:iam::111122223333:root"  
        ]  
      },  
      "Action": "kms:*",  
      "Resource": "*"  
    },  
  ]  
}
```

```
{  
    "Sid": "Allow access for Key Administrators",  
    "Effect": "Allow",  
    "Principal": {  
        "AWS": [  
            "arn:aws:iam::111122223333:role/Admin"  
        ]  
    },  
    "Action": [  
        "kms>Create*",  
        "kmsDescribe*",  
        "kmsEnable*",  
        "kmsList*",  
        "kmsPut*",  
        "kmsUpdate*",  
        "kmsRevoke*",  
        "kmsDisable*",  
        "kmsGet*",  
        "kmsDelete*",  
        "kmsTagResource",  
        "kmsUntagResource",  
        "kmsScheduleKeyDeletion",  
        "kmsCancelKeyDeletion"  
    ],  
    "Resource": "*"  
},  
{  
    "Sid": "Allow use of the key",  
    "Effect": "Allow",  
    "Principal": {  
        "AWS": [  
            "arn:aws:iam::111122223333:role/DMS-S3-endpoint-access-role",  
            "arn:aws:iam::111122223333:role/Admin",  
            "arn:aws:iam::111122223333:role/User1"  
        ]  
    },  
    "Action": [  
        "kmsEncrypt",  
        "kmsDecrypt",  
        "kmsReEncrypt*",  
        "kmsGenerateDataKey*",  
        "kmsDescribeKey"  
    ],  
    "Resource": "*"  
},  
{  
    "Sid": "Allow attachment of persistent resources",  
    "Effect": "Allow",  
    "Principal": {  
        "AWS": [  
            "arn:aws:iam::111122223333:role/DMS-S3-endpoint-access-role",  
            "arn:aws:iam::111122223333:role/Admin",  
            "arn:aws:iam::111122223333:role/User1"  
        ]  
    },  
    "Action": [  
        "kmsCreateGrant",  
        "kmsListGrants",  
        "kmsRevokeGrant"  
    ],  
    "Resource": "*",  
    "Condition": {  
        "Bool": {  
            "kmsGrantIsForAWSResource": true  
        }  
    }  
}
```

```
    }  
]
```

13. Choose **Finish**. The **Encryption keys** page opens with a message indicating that your master encryption key has been created.

You have now created a new AWS KMS key with a specified alias (for example, `DMS-S3-endpoint-encryption-key`). This key enables AWS DMS to encrypt Amazon S3 target objects.

## Endpoint settings when using Amazon S3 as a target for AWS DMS

You can use endpoint settings to configure your Amazon S3 target similar to using extra connection attributes. You can specify these settings when you create the target endpoint using the `create-endpoint` command in the AWS CLI, with the `--s3-settings` '`json-settings`' option. Here, `json-settings` is a JSON object containing parameters to specify the settings.

You can also specify a .json file containing the same `json-settings` object, for example, as in the following: `--s3-settings file:///your-file-path/my_s3_settings.json`. Here, `my_s3_settings.json` is the name of a .json file that contains the same `json-settings` object.

The parameter names for endpoint settings are the same as the names for equivalent extra connections attributes, except that the parameter names for endpoint settings have initial caps. However, not all S3 target endpoint settings using extra connection attributes are available using the `--s3-settings` option of the `create-endpoint` command. For more information about the available settings for the `create-endpoint` CLI command, see [create-endpoint](#) in the *AWS CLI Command Reference* for AWS DMS. For general information about these settings, see the equivalent extra connection attributes in [Extra connection attributes when using Amazon S3 as a target for AWS DMS \(p. 200\)](#).

You can use S3 target endpoint settings to configure the following:

- A custom AWS KMS key to encrypt your S3 target objects
- Parquet files as the storage format for S3 target objects

### AWS KMS key settings for data encryption

The following examples show configuring a custom AWS KMS key to encrypt your S3 target objects. To start, you might run the following `create-endpoint` CLI command.

```
aws dms create-endpoint --endpoint-identifier s3-target-endpoint --engine-name s3 --  
endpoint-type target  
--s3-settings '{\"ServiceAccessRoleArn\": \"your-service-access-ARN\", \"CsvRowDelimiter\":  
\"\\n\",  
\"CsvDelimiter\": \",\", \"BucketFolder\": \"your-bucket-folder\",  
\"BucketName\": \"your-bucket-name\",  
\"EncryptionMode\": \"SSE_KMS\",  
\"ServerSideEncryptionKmsKeyId\": \"arn:aws:kms:us-  
east-1:111122223333:key/72abb6fb-1e49-4ac1-9aed-c803dfcc0480\"}'
```

Here, the JSON object specified by `--s3-settings` option defines two parameters. One is an `EncryptionMode` parameter with the value `SSE_KMS`. The other is an `ServerSideEncryptionKmsKeyId` parameter with the value of `arn:aws:kms:us-east-1:111122223333:key/72abb6fb-1e49-4ac1-9aed-c803dfcc0480`. This value is an Amazon Resource Name (ARN) for your custom AWS KMS key. For an S3 target, you also specify additional settings. These identify the server access role, provide delimiters for the default CSV object storage format, and give the bucket location and name to store S3 target objects.

By default, S3 data encryption occurs using S3 server-side encryption. For the previous example's S3 target, this is also equivalent to specifying its endpoint settings as in the following example.

```
aws dms create-endpoint --endpoint-identifier s3-target-endpoint --engine-name s3 --  
endpoint-type target  
--s3-settings '{"ServiceAccessRoleArn": "your-service-access-ARN", "CsvRowDelimiter":  
"\n",  
"CsvDelimiter": ",", "BucketFolder": "your-bucket-folder",  
"BucketName": "your-bucket-name",  
"EncryptionMode": "SSE_S3"}'
```

For more information about working with S3 server-side encryption, see [Protecting data using server-side encryption](#).

**Note**

You can also use the CLI `modify-endpoint` command to change the value of the `EncryptionMode` parameter for an existing endpoint from `SSE_KMS` to `SSE_S3`. But you can't change the `EncryptionMode` value from `SSE_S3` to `SSE_KMS`.

## Settings for using .parquet files to store S3 target objects

The default format for creating S3 target objects is .csv files. The following examples show some endpoint settings for specifying .parquet files as the format for creating S3 target objects. You can specify the .parquet files format with all the defaults, as in the following example.

```
aws dms create-endpoint --endpoint-identifier s3-target-endpoint --engine-name s3 --  
endpoint-type target  
--s3-settings '{"ServiceAccessRoleArn": "your-service-access-ARN", "DataFormat":  
"parquet"}'
```

Here, the `DataFormat` parameter is set to `parquet` to enable the format with all the S3 defaults. These defaults include a dictionary encoding (`EncodingType: "rle-dictionary"`) that uses a combination of bit-packing and run-length encoding to more efficiently store repeating values.

You can add additional settings for options other than the defaults as in the following example.

```
aws dms create-endpoint --endpoint-identifier s3-target-endpoint --engine-name s3 --  
endpoint-type target  
--s3-settings '{"ServiceAccessRoleArn": "your-service-access-ARN", "BucketFolder": "your-  
bucket-folder",  
"BucketName": "your-bucket-name", "CompressionType": "GZIP", "DataFormat": "parquet",  
"EncodingType": "plain-dictionary", "dictPageSizeLimit": 3,072,000,  
"EnableStatistics": false }'
```

Here, in addition to parameters for several standard S3 bucket options and the `DataFormat` parameter, the following additional .parquet file parameters are set:

- `EncodingType` – Set to a dictionary encoding (`plain-dictionary`) that stores values encountered in each column in a per-column chunk of the dictionary page.
- `dictPageSizeLimit` – Set to a maximum dictionary page size of 3 MB.
- `EnableStatistics` – Disables the default that enables the collection of statistics about Parquet file pages and row groups.

## Extra connection attributes when using Amazon S3 as a target for AWS DMS

You can specify the following options as extra connection attributes. If you have multiple connection attribute settings, separate them from each other by semicolons with no additional white space.

Option	Description
addColumnName	<p>An optional parameter that when set to <code>true</code> or <code>y</code> you can use to add column name information to the .csv output file.</p> <p>Default value: <code>false</code></p> <p>Valid values: <code>true</code>, <code>false</code>, <code>y</code>, <code>n</code></p> <p>Example:</p> <pre>addColumnName=true;</pre>
bucketFolder	<p>An optional parameter to set a folder name in the S3 bucket. If provided, target objects are created as .csv or .parquet files in the path <code>bucketFolder/schema_name/table_name/</code>. If this parameter isn't specified, then the path used is <code>schema_name/table_name/</code>.</p> <p>Example:</p> <pre>bucketFolder=testFolder;</pre>
bucketName	<p>The name of the S3 bucket where S3 target objects are created as .csv or .parquet files.</p> <p>Example:</p> <pre>bucketName=buckettest;</pre>
cannedAclForObjects	<p>A value that enables AWS DMS to specify a predefined (canned) access control list for objects created in the S3 bucket as .csv or .parquet files. For more information about Amazon S3 canned ACLs, see <a href="#">Canned ACL</a> in the <a href="#">Amazon S3 Developer Guide</a>.</p> <p>Default value: <code>NONE</code></p> <p>Valid values for this attribute are: <code>NONE</code>; <code>PRIVATE</code>; <code>PUBLIC_READ</code>; <code>PUBLIC_READ_WRITE</code>; <code>AUTHENTICATED_READ</code>; <code>AWS_EXEC_READ</code>; <code>BUCKET_OWNER_READ</code>; <code>BUCKET_OWNER_FULL_CONTROL</code>.</p> <p>Example:</p> <pre>cannedAclForObjects=PUBLIC_READ;</pre>
cdcInsertsOnly	<p>An optional parameter during a change data capture (CDC) load to write only <code>INSERT</code> operations to the comma-separated value (.csv) or columnar storage (.parquet) output files. By default (the <code>false</code> setting), the first field in a .csv or .parquet record contains the letter <code>I</code> (<code>INSERT</code>), <code>U</code> (<code>UPDATE</code>), or <code>D</code> (<code>DELETE</code>). This letter indicates whether the row was inserted, updated, or deleted at the source database for a CDC load to the target. If <code>cdcInsertsOnly</code> is set to <code>true</code> or <code>y</code>, only <code>INSERTS</code> from the source database are migrated to the .csv or .parquet file.</p>

Option	Description
	<p>For .csv format only, how these INSERTS are recorded depends on the value of <code>includeOpForFullLoad</code>. If <code>includeOpForFullLoad</code> is set to <code>true</code>, the first field of every CDC record is set to <code>I</code> to indicate the INSERT operation at the source. If <code>includeOpForFullLoad</code> is set to <code>false</code>, every CDC record is written without a first field to indicate the INSERT operation at the source. For more information about how these parameters work together, see <a href="#">Indicating source DB operations in migrated S3 data (p. 206)</a>.</p> <p>Default value: <code>false</code></p> <p>Valid values: <code>true, false, y, n</code></p> <p>Example:</p> <pre>cdcInsertsOnly=true;</pre>
<code>cdcInsertsAndUpdates</code>	<p>Enables a change data capture (CDC) load to write INSERT and UPDATE operations to .csv or .parquet (columnar storage) output files. The default setting is <code>false</code>, but when <code>cdcInsertsAndUpdates</code> is set to <code>true</code> or <code>y</code>, INSERTs and UPDATEs from the source database are migrated to the .csv or .parquet file.</p> <p>For .csv file format only, how these INSERTs and UPDATEs are recorded depends on the value of the <code>includeOpForFullLoad</code> parameter. If <code>includeOpForFullLoad</code> is set to <code>true</code>, the first field of every CDC record is set to either <code>I</code> or <code>U</code> to indicate INSERT and UPDATE operations at the source. But if <code>includeOpForFullLoad</code> is set to <code>false</code>, CDC records are written without an indication of INSERT or UPDATE operations at the source.</p> <p>For more information about how these parameters work together, see <a href="#">Indicating source DB operations in migrated S3 data (p. 206)</a>.</p> <p><b>Note</b>  AWS DMS supports the use of the <code>cdcInsertsAndUpdates</code> parameter in versions 3.3.1 and later.  <code>cdcInsertsOnly</code> and <code>cdcInsertsAndUpdates</code> can't both be set to <code>true</code> for the same endpoint. Set either <code>cdcInsertsOnly</code> or <code>cdcInsertsAndUpdates</code> to <code>true</code> for the same endpoint, but not both.</p> <p>Default value: <code>false</code></p> <p>Valid values: <code>true, false, y, n</code></p> <p>Example:</p> <pre>cdcInsertsAndUpdates=true;</pre>

Option	Description
<code>includeOpForFullLoad</code>	<p>An optional parameter during a full load to write the INSERT operations to the comma-separated value (.csv) output files only.</p> <p>For full load, records can only be inserted. By default (the <code>false</code> setting), there is no information recorded in these output files for a full load to indicate that the rows were inserted at the source database. If <code>includeOpForFullLoad</code> is set to <code>true</code> or <code>y</code>, the INSERT is recorded as an <code>I</code> annotation in the first field of the .csv file.</p> <p><b>Note</b>  This parameter works together with <code>cdcInsertsOnly</code> or <code>cdcInsertsAndUpdates</code> for output to .csv files only. For more information about how these parameters work together, see <a href="#">Indicating source DB operations in migrated S3 data (p. 206)</a>.</p> <p>Default value: <code>false</code></p> <p>Valid values: <code>true</code>, <code>false</code>, <code>y</code>, <code>n</code></p> <p>Example:</p> <pre>includeOpForFullLoad=true;</pre>
<code>compressionType</code>	<p>An optional parameter when set to <code>GZIP</code> uses GZIP to compress the target .csv or .parquet files. When this parameter is set to the default, it leaves the files uncompressed.</p> <p>Default value: <code>NONE</code></p> <p>Valid values: <code>GZIP</code> or <code>NONE</code></p> <p>Example:</p> <pre>compressionType=GZIP;</pre>
<code>csvDelimiter</code>	<p>The delimiter used to separate columns in .csv source files. The default is a comma (,).</p> <p>Example:</p> <pre>csvDelimiter=,;</pre>
<code>csvRowDelimiter</code>	<p>The delimiter used to separate rows in the .csv source files. The default is a newline (\n).</p> <p>Example:</p> <pre>csvRowDelimiter=\n;</pre>
<code>maxFileSize</code>	<p>A value that 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: 1,048,576 KB (1 GB)</p> <p>Valid values: 1–1,048,576</p> <p>Example:</p> <pre>maxFileSize=512</pre>

Option	Description
rfc4180	<p>An optional parameter used to set behavior to comply with RFC for data migrated to Amazon S3 using .csv file format only. When this value is set to <code>true</code> or <code>y</code> using Amazon S3 as a target, if the data has quotation marks or newline characters in it, AWS DMS encloses the entire column with an additional pair of double quotation marks (""). Every quotation mark within the data is repeated twice. This formatting complies with RFC 4180.</p> <p>Default value: <code>true</code></p> <p>Valid values: <code>true</code>, <code>false</code>, <code>y</code>, <code>n</code></p> <p>Example:</p> <pre>rfc4180=false;</pre>
encryptionMode	<p>The server-side encryption mode that you want to encrypt your .csv or .parquet object files copied to S3. The valid values are <code>SSE_S3</code> (S3 server-side encryption) or <code>SSE_KMS</code> (AWS KMS key encryption). If you choose <code>SSE_KMS</code>, set the <code>serverSideEncryptionKmsKeyId</code> parameter to the Amazon Resource Name (ARN) for the AWS KMS key to be used for encryption.</p> <p><b>Note</b>            You can also use the CLI <code>modify-endpoint</code> command to change the value of the <code>encryptionMode</code> attribute for an existing endpoint from <code>SSE_KMS</code> to <code>SSE_S3</code>. But you can't change the <code>encryptionMode</code> value from <code>SSE_S3</code> to <code>SSE_KMS</code>.</p> <p>Default value: <code>SSE_S3</code></p> <p>Valid values: <code>SSE_S3</code> or <code>SSE_KMS</code></p> <p>Example:</p> <pre>encryptionMode=SSE_S3;</pre>
serverSideEncryptionKmsKeyId	<p>If you set <code>encryptionMode</code> to <code>SSE_KMS</code>, set this parameter to the Amazon Resource Name (ARN) for the AWS KMS key. You can find this ARN by selecting the key alias in the list of AWS KMS keys created for your account. When you create the key, you must associate specific policies and roles associated with this AWS KMS key. For more information, see <a href="#">Creating AWS KMS keys to encrypt Amazon S3 target objects (p. 194)</a>.</p> <p>Example:</p> <pre>serverSideEncryptionKmsKeyId=arn:aws:kms:us-east-1:111122223333:key/72abb6fb-1e49-4ac1-9aed-c803dfcc0480;</pre>

Option	Description
<code>dataFormat</code>	<p>The output format for the files that AWS DMS uses to create S3 objects. For Amazon S3 targets, AWS DMS supports either .csv or .parquet files. The .parquet files have a binary columnar storage format with efficient compression options and faster query performance. For more information about .parquet files, see <a href="https://parquet.apache.org/">https://parquet.apache.org/</a>.</p> <p>Default value: csv</p> <p>Valid values: csv or parquet</p> <p>Example:</p> <pre>dataFormat=parquet;</pre>
<code>encodingType</code>	<p>The Parquet encoding type. The encoding type options include the following:</p> <ul style="list-style-type: none"> <li>• <code>rle-dictionary</code> – This dictionary encoding uses a combination of bit-packing and run-length encoding to more efficiently store repeating values.</li> <li>• <code>plain</code> – No encoding.</li> <li>• <code>plain-dictionary</code> – This dictionary encoding builds a dictionary of values encountered in a given column. The dictionary is stored in a dictionary page for each column chunk.</li> </ul> <p>Default value: rle-dictionary</p> <p>Valid values: rle-dictionary, plain, or plain-dictionary</p> <p>Example:</p> <pre>encodingType=plain-dictionary;</pre>
<code>dictPageSizeLimit</code>	<p>The maximum allowed size, in bytes, for a dictionary page in a .parquet file. If a dictionary page exceeds this value, the page uses plain encoding.</p> <p>Default value: 1,024,000 (1 MB)</p> <p>Valid values: Any valid integer value</p> <p>Example:</p> <pre>dictPageSizeLimit=2,048,000;</pre>
<code>rowGroupLength</code>	<p>The number of rows in one row group of a .parquet file.</p> <p>Default value: 10,024 (10 KB)</p> <p>Valid values: Any valid integer value</p> <p>Example:</p> <pre>rowGroupLength=20,048;</pre>

Option	Description
<code>dataPageSize</code>	<p>The maximum allowed size, in bytes, for a data page in a .parquet file.</p> <p>Default value: 1,024,000 (1 MB)</p> <p>Valid values: Any valid integer value</p> <p>Example:</p> <pre>dataPageSize=2,048,000;</pre>
<code>parquetVersion</code>	<p>The version of the .parquet file format.</p> <p>Default value: PARQUET_1_0</p> <p>Valid values: PARQUET_1_0 or PARQUET_2_0</p> <p>Example:</p> <pre>parquetVersion=PARQUET_2_0;</pre>
<code>enableStatistics</code>	<p>Set to true or y to enable statistics about .parquet file pages and row groups.</p> <p>Default value: true</p> <p>Valid values: true, false, y, n</p> <p>Example:</p> <pre>enableStatistics=false;</pre>
<code>timestampColumnName</code>	<p>An optional parameter to include a timestamp column in the S3 target endpoint data.</p> <p>AWS DMS includes an additional STRING column in the .csv or .parquet object files of your migrated data when you set <code>timestampColumnName</code> to a nonblank value.</p> <p>For a full load, each row of this timestamp column contains a timestamp for when the data was transferred from the source to the target by DMS.</p> <p>For a CDC load, each row of the timestamp column contains the timestamp for the commit of that row in the source database.</p> <p>The string format for this timestamp column value is yyyy-MM-dd HH:mm:ss.ssssss. By default, the precision of this value is in microseconds. For a CDC load, the rounding of the precision depends on the commit timestamp supported by DMS for the source database.</p> <p>When the <code>addColumnName</code> parameter is set to true, DMS also includes the name for the timestamp column that you set as the nonblank value of <code>timestampColumnName</code>.</p> <p>Example:</p> <pre>timestampColumnName=TIMESTAMP;</pre>

Option	Description
parquetTimestampInMillisecond	<p>An optional parameter that specifies the precision of any <b>TIMESTAMP</b> column values written to an S3 object file in .parquet format.</p> <p>When this attribute is set to <code>true</code> or <code>y</code>, AWS DMS writes all <b>TIMESTAMP</b> columns in a .parquet formatted file with millisecond precision. Otherwise, DMS writes them with microsecond precision.</p> <p>Currently, Amazon Athena and AWS Glue can handle only millisecond precision for <b>TIMESTAMP</b> values. Set this attribute to true for .parquet formatted S3 endpoint object files only if you plan to query or process the data with Athena or AWS Glue.</p> <p><b>Note</b></p> <ul style="list-style-type: none"> <li>• AWS DMS writes any <b>TIMESTAMP</b> column values written to an S3 file in .csv format with microsecond precision.</li> <li>• The setting of this attribute has no effect on the string format of the timestamp column value inserted by setting the <code>timestampColumnName</code> attribute.</li> </ul> <p>Default value: <code>false</code></p> <p>Valid values: <code>true</code>, <code>false</code>, <code>y</code>, <code>n</code></p> <p>Example:</p> <pre>parquetTimestampInMillisecond=true;</pre>

## Indicating source DB operations in migrated S3 data

When AWS DMS migrates records to an S3 target, it can create an additional field in each migrated record. This additional field indicates the operation applied to the record at the source database.

For a full load when `includeOpForFullLoad` is `true` and the output format is .csv, DMS always creates an additional first field in each .csv record. This field contains the letter I (INSERT) to indicate that the row was inserted at the source database. For a CDC load when `cdcInsertsOnly` is `false` (the default), DMS also always creates an additional first field in each .csv or .parquet record. This field contains the letter I (INSERT), U (UPDATE), or D (DELETE) to indicate whether the row was inserted, updated, or deleted at the source database.

When the output format is .csv only, if and how DMS creates and sets this first field also depends on the settings of `includeOpForFullLoad` and `cdcInsertsOnly` or `cdcInsertsAndUpdates`.

### Note

AWS DMS supports the interaction between `includeOpForFullLoad` and `cdcInsertAndUpdates` in versions 3.3.1 and later.

In the following table, you can see how the settings of the `includeOpForFullLoad` and `cdcInsertsOnly` attributes work together to affect the setting of migrated records in this format.

With these parameter settings		DMS sets target records as follows for .csv output	
<code>true</code>	<code>true</code>	Added first field value set to I	Added first field value set to I

With these parameter settings		DMS sets target records as follows for .csv output	
false	false	No added field	Added first field value set to I, U, or D
false	true	No added field	No added field
true	false	Added first field value set to I	Added first field value set to I, U, or D

When `includeOpForFullLoad` and `cdcInsertsOnly` are set to the same value, the target records are set according to the attribute that controls record settings for the current migration type. That attribute is `includeOpForFullLoad` for full load and `cdcInsertsOnly` for CDC load.

When `includeOpForFullLoad` and `cdcInsertsOnly` are set to different values, AWS DMS makes the target record settings consistent for both CDC and full load. It does this by making the record settings for a CDC load conform to the record settings for any earlier full load specified by `includeOpForFullLoad`.

In other words, suppose that a full load is set to add a first field to indicate an inserted record. In this case, a following CDC load is set to add a first field that indicates an inserted, updated, or deleted record as appropriate at the source. In contrast, suppose that a full load is set to *not* add a first field to indicate an inserted record. In this case, a CDC load is also set to not add a first field to each record regardless of its corresponding record operation at the source.

Similarly, how DMS creates and sets an additional first field depends on the settings of `includeOpForFullLoad` and `cdcInsertsAndUpdates`. In the following table, you can see how the settings of the `includeOpForFullLoad` and `cdcInsertsAndUpdates` attributes work together to affect the setting of migrated records in this format.

With these parameter settings		DMS sets target records as follows for .csv output	
true	true	Added first field value set to I	Added first field value set to I or U
false	false	No added field	Added first field value set to I, U, or D
false	true	No added field	Added first field value set to I or U
true	false	Added first field value set to I	Added first field value set to I, U, or D

## 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 enables you to determine where the source data is located in the target.

When AWS DMS creates tables on an DynamoDB target endpoint, it creates as many tables as in the source database endpoint. AWS DMS also sets several DynamoDB parameter values. The cost for the table creation depends on the amount of data and the number of tables to be migrated.

To help increase the speed of the transfer, AWS DMS supports a multithreaded full load to a DynamoDB target instance. DMS supports this multithreading with task settings that include the following:

- **MaxFullLoadSubTasks** – Use this option to indicate the maximum number of source tables to load in parallel. DMS loads each table into its corresponding DynamoDB target table using a dedicated subtask. The default value is 8. The maximum value is 49.
- **ParallelLoadThreads** – Use this option to specify the number of threads that AWS DMS uses to load each table into its DynamoDB target table. The default value is 0 (single-threaded). The maximum value is 200. You can ask to have this maximum limit increased.
- **ParallelLoadBufferSize** – Use this option to specify the maximum number of records to store in the buffer that the parallel load threads use to load data to the DynamoDB target. The default value is 50. The maximum value is 1,000. Use this setting with **ParallelLoadThreads**. **ParallelLoadBufferSize** is valid only when there is more than one thread.
- **Table-mapping settings for individual tables** – Use table-settings rules to identify individual tables from the source that you want to load in parallel. Also use these rules to specify how to segment the rows of each table for multithreaded loading. For more information, see [Table-settings rules and operations \(p. 333\)](#).

**Note**

DMS assigns each segment of a table to its own thread for loading. Therefore, set **ParallelLoadThreads** to the maximum number of segments that you specify for a table in the source.

When AWS DMS sets 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** task setting is set greater than 1 (the 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 40,000. If the AWS Region is us-east-1 and the WCU parameter value is greater than 40,000, the WCU parameter value is set to 40,000.
- In AWS Regions other than us-east-1, the largest possible WCU parameter value is 10,000. For any AWS Region other than us-east-1, if the WCU parameter value is set greater than 10,000 the WCU parameter value is set to 10,000.

## Migrating from a relational database to a DynamoDB table

AWS DMS supports migrating data to DynamoDB 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 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. This IAM role should allow 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 IAM 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",  
                "dynamodb:UpdateItem"  
            ],  
            "Resource": [  
                "arn:aws:dynamodb:us-west-2:account-id:table/name1",  
                "arn:aws:dynamodb:us-west-2:account-id:table/OtherName*",  
                "arn:aws:dynamodb:us-west-2:account-id:table/awsdms_apply_exceptions",  
                "arn:aws:dynamodb:us-west-2:account-id:table/awsdms_full_load_exceptions"  
            ]  
        },  
    ]  
}
```

```
{  
    "Effect": "Allow",  
    "Action": [  
        "dynamodb>ListTables"  
    ],  
    "Resource": "*"  
}  
]  
}
```

## Limitations when using DynamoDB as a target for AWS Database Migration Service

The following limitations apply when using 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 DynamoDB doesn't have a Date data type, data using the Date data type are converted to strings.
- 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 do one of two things. It can either fail or insert a new item with the updated primary key and incomplete data.
- AWS DMS only supports replication of tables with noncomposite primary keys. The exception is if 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 the data.
- When you use DynamoDB as target, only the Apply Exceptions control table (`dmslogs.awsdms_apply_exceptions`) is supported. For more information about control tables, see [Control table task settings \(p. 285\)](#).

## 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.

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. These values 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.

- You can use `map-record-to-record` when migrating from a relational database to DynamoDB. It uses the primary key from the relational database as the partition key in 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. It does so regardless of whether that source column is used in an attribute mapping.
- You use `map-record-to-document` to put source columns into a single, flat DynamoDB map on the target using the attribute name `_doc`. When using `map-record-to-document`, AWS DMS places the data into a single, flat, DynamoDB map attribute on the source. This attribute is called `_doc`. This placement applies to any column in the source table not listed in the `exclude-columns` attribute list.

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": "2",
            "rule-name": "TransformToDDB",
            "rule-action": "map-record-to-record",
            "object-locator": {
                "schema-name": "test",
                "table-name": "customer"
            },
            "rule-mapping": {
                "source-column": "income",
                "target-column": "CustomerIncome"
            }
        }
    ]
}

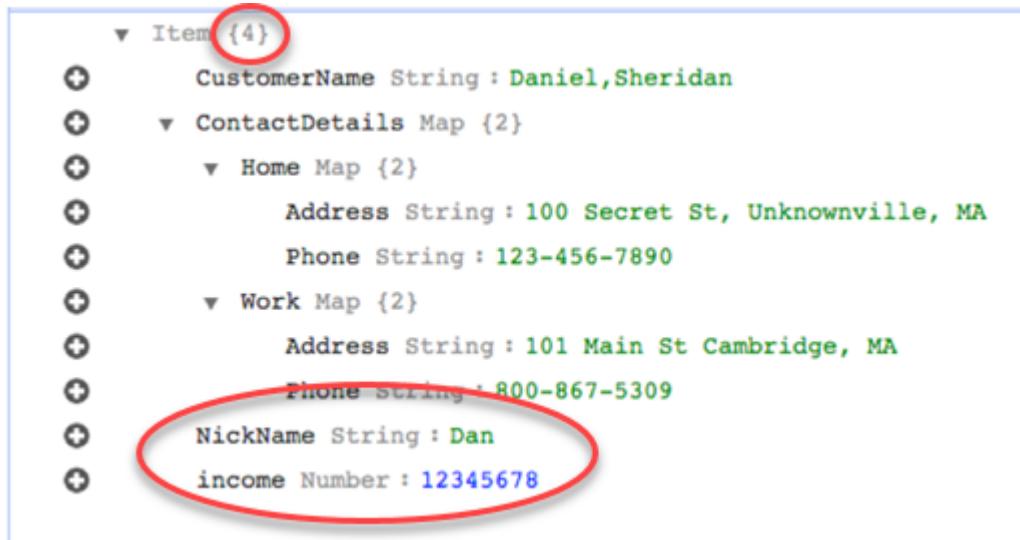
```

```

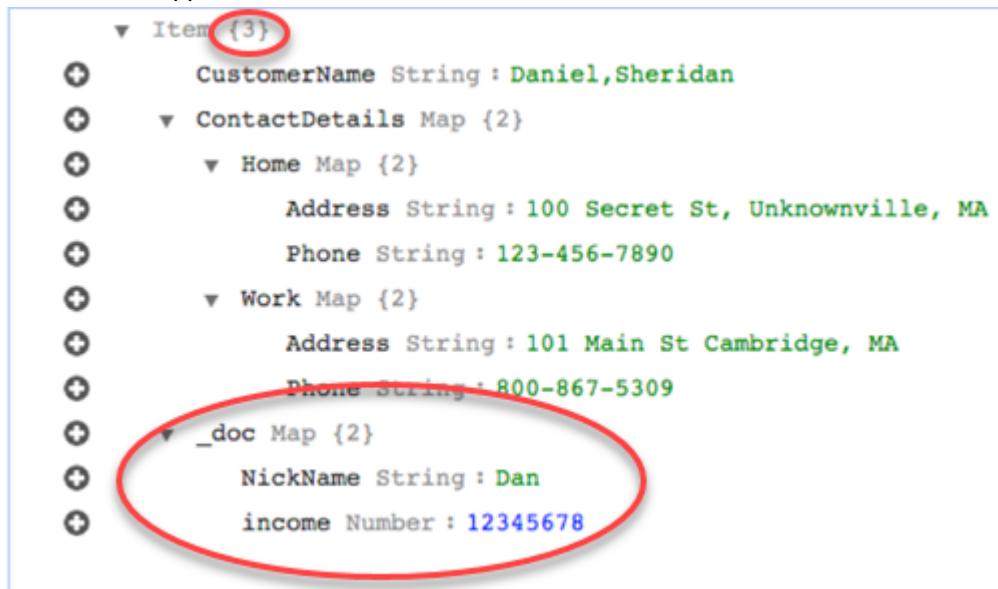
"target-table-name": "customer_t",
"mapping-parameters": {
    "partition-key-name": "CustomerName",
    "exclude-columns": [
        "FirstName",
        "LastName",
        "HomeAddress",
        "HomePhone",
        "WorkAddress",
        "WorkPhone"
    ],
    "attribute-mappings": [
        {
            "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.



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.



## Using custom condition expressions with object mapping

You can use a feature of 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}"
        }
      ]
    }
  ]
}
```

Object mapping section defines name, rule-action, and object locator information

Condition expression

Options

## 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	Stn	HomeA	HomeP	WorkAddress	Work	DateOfBirth
Primary Key				N/A			
Randy	Mars	15	221B Baker Street	1234567890	Spooner Street, Quahog		02/29/1988

The structure of the DynamoDB data is as follows:

Customer	StoreId	ContactDetails	DateOfBirth
Partition Key	Sort Key	N/A	
Randy	Mars	<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",
            "transform": {
                "source": {
                    "columns": [
                        "First", "Last", "Stn", "HomeA", "HomeP", "WorkAddress", "Work", "DateOfBirth"
                    ]
                },
                "target": {
                    "columns": [
                        "Customer", "StoreId", "ContactDetails", "DateOfBirth"
                    ],
                    "mappings": [
                        {
                            "source": "First", "target": "Customer"
                        },
                        {
                            "source": "Last", "target": "Customer"
                        },
                        {
                            "source": "Stn", "target": "StoreId"
                        },
                        {
                            "source": "HomeA", "target": "ContactDetails"
                        },
                        {
                            "source": "HomeP", "target": "ContactDetails"
                        },
                        {
                            "source": "WorkAddress", "target": "ContactDetails"
                        },
                        {
                            "source": "Work", "target": "ContactDetails"
                        },
                        {
                            "source": "DateOfBirth", "target": "DateOfBirth"
                        }
                    ]
                }
            }
        }
    ]
}
```

```

    "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": "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": "${StoreId}"
        },
        {
            "target-attribute-name": "ContactDetails",
            "attribute-type": "document",
            "attribute-sub-type": "dynamodb-map",
            "value": {
                "M": {
                    "Name": {
                        "S": "${FirstName}"
                    },
                    "Home": {
                        "M": {
                            "Address": {
                                "S": "${HomeAddress}"
                            },
                            "Phone": {
                                "S": "${HomePhone}"
                            }
                        }
                    },
                    "Work": {
                        "M": {
                            "Address": {
                                "S": "${WorkAddress}"
                            },
                            "Phone": {
                                "S": "${WorkPhone}"
                            }
                        }
                    }
                }
            }
        }
    ]
}
```

```
    ]
}
```

## Example 1: Using attribute mapping with object mapping

The following example migrates data from two MySQL database tables, *nfl\_data* and *sport\_team*, to two DynamoDB table called *NFLTeams* and *SportTeams*. The structure of the tables and the JSON used to map the data from the MySQL database tables to the DynamoDB tables are shown following.

The structure of the MySQL database table *nfl\_data* is shown below:

```
mysql> desc nfl_data;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| Position | varchar(5) | YES | | NULL | |
| player_number | smallint(6) | YES | | NULL | |
| Name | varchar(40) | YES | | NULL | |
| status | varchar(10) | YES | | NULL | |
| stat1 | varchar(10) | YES | | NULL | |
| stat1_val | varchar(10) | YES | | NULL | |
| stat2 | varchar(10) | YES | | NULL | |
| stat2_val | varchar(10) | YES | | NULL | |
| stat3 | varchar(10) | YES | | NULL | |
| stat3_val | varchar(10) | YES | | NULL | |
| stat4 | varchar(10) | YES | | NULL | |
| stat4_val | varchar(10) | YES | | NULL | |
| 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}\", \"Status\": \"${status}\", \"Stats\": {\"Stat1\": \"${stat1}:${stat1_val}\", \"Stat2\": \"${stat2}:${stat2_val}\", \"Stat3\": \"${stat3}:${stat3_val}\", \"Stat4\": \"${stat4}:${stat4_val}\"}"
            }
        ]
    }
},
{
    "rule-type": "object-mapping",
    "rule-id": "4",
    "rule-name": "MapSportTeam",
    "rule-action": "map-record-to-record",
    "object-locator": {
        "schema-name": "dms_sample",
        "table-name": "sport_team"
    },
    "target-table-name": "SportTeams",
    "mapping-parameters": {

```

```
"partition-key-name": "TeamName",
"exclude-columns": [
    "name", "id"
],
"attribute-mappings": [
    {
        "target-attribute-name": "TeamName",
        "attribute-type": "scalar",
        "attribute-sub-type": "string",
        "value": "${name}"
    },
    {
        "target-attribute-name": "TeamInfo",
        "attribute-type": "scalar",
        "attribute-sub-type": "string",
        "value": "{\"League\": \"${sport_league_short_name}\", \"Division\": \"${sport_division_short_name}\"}"
    }
]
}
```

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 DynamoDB

The DynamoDB endpoint for AWS DMS supports most 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 \(p. 488\)](#).

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
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 or a JSON\_UNFORMATTED message format. A JSON\_UNFORMATTED message format is a single line JSON string with new line delimiter. It allows Amazon Kinesis Data Firehose to deliver Kinesis data to an Amazon S3 destination, and then query it using various query engines including Amazon Athena.

**Note**

Support for the JSON\_UNFORMATTED Kinesis message format is available in AWS DMS versions 3.3.1 and later.

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.

When AWS DMS creates tables on an Kinesis Data Streams target endpoint, it creates as many tables as in the source database endpoint. AWS DMS also sets several Kinesis Data Streams parameter values. The cost for the table creation depends on the amount of data and the number of tables to be migrated.

### Kinesis Data Streams endpoint settings

When you use Kinesis Data Streams target endpoints, you can get transaction and control details using the `KinesisSettings` option in the AWS DMS API. In the CLI, use the request parameters of the `--kinesis-settings` option following:

**Note**

Support for the following endpoint settings for Kinesis Data Streams target endpoints is available in AWS DMS versions 3.3.1 and higher.

- `IncludeControlDetails` – Shows detailed control information for table definition, column definition, and table and column changes in the Kinesis message output. The default is `false`.
- `IncludeNullAndEmpty` – Include NULL and empty columns in the target. The default is `false`.
- `IncludePartitionValue` – Shows the partition value within the Kinesis message output, unless the partition type is `schema-table-type`. The default is `false`.
- `IncludeTableAlterOperations` – Includes any data definition language (DDL) operations that change the table in the control data, such as `rename-table`, `drop-table`, `add-column`, `drop-column`, and `rename-column`. The default is `false`.
- `IncludeTransactionDetails` – Provides detailed transaction information from the source database. This information includes a commit timestamp, a log position, and values for `transaction_id`, `previous_transaction_id`, and `transaction_record_id` (the record offset within a transaction). The default is `false`.
- `PartitionIncludeSchemaTable` – Prefixes schema and table names to partition values, when the partition type is `primary-key-type`. Doing this increases data distribution among Kinesis shards. For example, suppose that a SysBench schema has thousands of tables and each table has only limited range for a primary key. In this case, the same primary key is sent from thousands of tables to the same shard, which causes throttling. The default is `false`.

The following example shows the `kinesis-settings` option in use with an example `create-endpoint` command issued using the AWS CLI.

```
aws dms create-endpoint --endpoint-identifier=$target_name --engine-name kinesis --  
endpoint-type target  
--region us-east-1 --kinesis-settings ServiceAccessRoleArn=arn:aws:iam::333333333333:role/  
dms-kinesis-role,
```

```
StreamArn=arn:aws:kinesis:us-east-1:333333333333:stream/dms-kinesis-target-
doc,MessageFormat=json-unformatted,
IncludeControlDetails=true,IncludeTransactionDetails=true,IncludePartitionValue=true,PartitionIncludeSc
IncludeTableAlterOperations=true
```

## Multithreaded full load task settings

To help increase the speed of the transfer, AWS DMS supports a multithreaded full load to a Kinesis Data Streams target instance. DMS supports this multithreading with task settings that include the following:

- **MaxFullLoadSubTasks** – Use this option to indicate the maximum number of source tables to load in parallel. DMS loads each table into its corresponding Kinesis target table using a dedicated subtask. The default is 8; the maximum value is 49.
- **ParallelLoadThreads** – Use this option to specify the number of threads that AWS DMS uses to load each table into its Kinesis target table. The maximum value for a Kinesis Data Streams target is 32. You can ask to have this maximum limit increased.
- **ParallelLoadBufferSize** – Use this option to specify the maximum number of records to store in the buffer that the parallel load threads use to load data to the Kinesis target. The default value is 50. The maximum value is 1,000. Use this setting with **ParallelLoadThreads**. **ParallelLoadBufferSize** is valid only when there is more than one thread.
- **ParallelLoadQueuesPerThread** – Use this option to specify the number of queues each concurrent thread accesses to take data records out of queues and generate a batch load for the target. The default is 1. However, for Kinesis targets of various payload sizes, the valid range is 5–512 queues per thread.
- **Table-mapping settings for individual tables** – Use **table-settings** rules to identify the individual tables from the source that you want to load in parallel. Also use these rules to specify how to segment the rows of each table for multithreaded loading. For more information, see [Table-settings rules and operations \(p. 333\)](#).

### Note

DMS assigns each segment of a table to its own thread for loading. Therefore, set **ParallelLoadThreads** to the maximum number of segments that you specify for a table in the source.

## Multithreaded CDC load task settings

You can improve the performance of change data capture (CDC) for real-time data streaming target endpoints like Kinesis using task settings to modify the behavior of the **PutRecords** API call. To do this, you can specify the number of concurrent threads, queues per thread, and the number of records to store in a buffer using **ParallelApply\*** task settings. For example, suppose you want to perform a CDC load and apply 128 threads in parallel. You also want to access 64 queues per thread, with 50 records stored per buffer.

### Note

Support for the use of **ParallelApply\*** task settings during CDC to Kinesis Data Streams target endpoints is available in AWS DMS versions 3.3.1 and later.

To promote CDC performance, AWS DMS supports these task settings:

- **ParallelApplyThreads** – Specifies the number of concurrent threads that AWS DMS uses during a CDC load to push data records to a Kinesis target endpoint. The default value is zero (0) and the maximum value is 32.
- **ParallelApplyBufferSize** – Specifies the maximum number of records to store in each buffer queue for concurrent threads to push to a Kinesis target endpoint during a CDC load. The default value is 50 and the maximum value is 1,000. Use this option when **ParallelApplyThreads** specifies more than one thread.

- `ParallelApplyQueuesPerThread` – Specifies the number of queues that each thread accesses to take data records out of queues and generate a batch load for a Kinesis endpoint during CDC.

When using `ParallelApply*` task settings, the `partition-key-type` default is the `primary-key` of the table, not `schema-name.table-name`.

## Using a before image to view original values of CDC rows for a Kinesis data stream as a target

When writing CDC updates to a data-streaming target like Kinesis, you can view a source database row's original values before change by an update. To make this possible, AWS DMS populates a *before image* of update events based on data supplied by the source database engine.

### Note

Support for the use of before image task settings during CDC to Amazon Kinesis Data Streams target endpoints is available in AWS DMS versions 3.3.1 and later.

Different source database engines provide different amounts of information for a before image:

- Oracle provides updates to columns only if they change.
- PostgreSQL provides only data for columns that are part of the primary key (changed or not).
- MySQL generally provides data for all columns (changed or not).

To enable before imaging to add original values from the source database to the AWS DMS output, use either the `BeforeImageSettings` task setting or the `add-before-image-columns` parameter. This parameter applies a column transformation rule.

`BeforeImageSettings` adds a new JSON attribute to every update operation with values collected from the source database system, as shown following.

```
"BeforeImageSettings": {  
    "EnableBeforeImage": boolean,  
    "FieldName": string,  
    "ColumnFilter": pk-only (default) / non-lob / all (but only one)  
}
```

### Note

Apply `BeforeImageSettings` to full load plus CDC tasks (which migrate existing data and replicate ongoing changes), or to CDC only tasks (which replicate data changes only). Don't apply `BeforeImageSettings` to tasks that are full load only.

For `BeforeImageSettings` options, the following applies:

- Set the `EnableBeforeImage` option to `true` to enable before imaging. The default is `false`.
- Use the `FieldName` option to assign a name to the new JSON attribute. When `EnableBeforeImage` is `true`, `FieldName` is required and can't be empty.
- The `ColumnFilter` option specifies a column to add by using before imaging. To add only columns that are part of the table's primary keys, use the default value, `pk-only`. To add only columns that are not of LOB type, use `non-lob`. To add any column that has a before image value, use `all`.

```
"BeforeImageSettings": {  
    "EnableBeforeImage": true,  
    "FieldName": "before-image",  
    "ColumnFilter": "pk-only"
```

}

**Note**

Amazon S3 targets don't support `BeforeImageSettings`. For S3 targets, use only the `add-before-image-columns` transformation rule to perform before imaging during CDC.

## Using a before image transformation rule

As an alternative to task settings, you can use the `add-before-image-columns` parameter, which applies a column transformation rule. With this parameter, you can enable before imaging during CDC on data streaming targets like Kinesis.

By using `add-before-image-columns` in a transformation rule, you can apply more fine-grained control of the before image results. Transformation rules enable you to use an object locator that gives you control over tables selected for the rule. Also, you can chain transformation rules together, which allows different rules to be applied to different tables. You can then manipulate the columns produced by using other rules.

**Note**

Don't use the `add-before-image-columns` parameter together with the `BeforeImageSettings` task setting within the same task. Instead, use either the parameter or the setting, but not both, for a single task.

A transformation rule type with the `add-before-image-columns` parameter for a column must provide a `before-image-def` section. The following shows an example.

```
{  
    "rule-type": "transformation",  
    ...  
    "rule-target": "column",  
    "rule-action": "add-before-image-columns",  
    "before-image-def":{  
        "column-filter": one-of (pk-only / non-lob / all),  
        "column-prefix": string,  
        "column-suffix": string,  
    }  
}
```

The value of `column-prefix` is prepended to a column name, and the default value of `column-prefix` is `BI_`. The value of `column-suffix` is appended to the column name, and the default is empty. Don't set both `column-prefix` and `column-suffix` to empty strings.

Choose one value for `column-filter`. To add only columns that are part of table primary keys, choose `pk-only`. Choose `non-lob` to only add columns that are not of LOB type. Or choose `all` to add any column that has a before-image value.

## Example for a before image transformation rule

The transformation rule in the following example adds a new column called `BI_emp_no` in the target. So a statement like `UPDATE employees SET emp_no = 3 WHERE emp_no = 1;` populates the `BI_emp_no` field with 1. When you write CDC updates to Amazon S3 targets, the `BI_emp_no` column makes it possible to tell which original row was updated.

```
{  
    "rules": [  
        {  
            "rule-type": "selection",  
            "rule-id": "1",  
            "rule-name": "1",  
            "rule-condition": "emp_no = 1"  
        },  
        {  
            "rule-type": "transformation",  
            "rule-id": "2",  
            "rule-name": "2",  
            "rule-condition": "emp_no = 1",  
            "rule-action": "add-before-image-columns",  
            "before-image-def": {  
                "column-filter": "all",  
                "column-prefix": "BI_",  
                "column-suffix": ""  
            }  
        }  
    ]  
}
```

```

    "object-locator": {
        "schema-name": "%",
        "table-name": "%"
    },
    "rule-action": "include"
},
{
    "rule-type": "transformation",
    "rule-id": "2",
    "rule-name": "2",
    "rule-target": "column",
    "object-locator": {
        "schema-name": "%",
        "table-name": "employees"
    },
    "rule-action": "add-before-image-columns",
    "before-image-def": {
        "column-prefix": "BI_",
        "column-suffix": "",
        "column-filter": "pk-only"
    }
}
]
}

```

For information on using the `add-before-image-columns` rule action, see [Transformation rules and actions \(p. 318\)](#).

## 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 IAM 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:DescribeStream",
        "kinesis:PutRecord",
        "kinesis:PutRecords"
    ],
    "Resource": "arn:aws:kinesis:region:accountID:stream/streamName"
}
}
```

## 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 supports a maximum message size of 1 MiB for a Kinesis Data Streams target.
- AWS DMS publishes each update to a single record in the source database as one data record in a given Kinesis data stream regardless of transactions. However, you can include transaction details for each data record by using relevant parameters of the `KinesisSettings` API.
- 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.
- For information about encrypting your data at rest within Kinesis Data Streams, see [Data protection in Kinesis Data Streams](#) in the *AWS Key Management Service Developer Guide*.

## 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. In an object mapping rule, the possible values of a `partition-key-type` for data records are `schema-table`, `transaction-id`, `primary-key`, `constant`, and `attribute-name`.

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. The `map-record-to-record` and `map-record-to-document` values 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.

Use `map-record-to-document` to put source columns into a single, flat document in the appropriate target stream using the attribute name `_doc`. AWS DMS places the data into a single, flat map on the source called `_doc`. This placement applies to any column in the source table not listed in the `exclude-columns` attribute list.

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	987654321002	29/1988

To migrate this information from a schema named `Test` 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",
            "rule-action": "include",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "%"
            }
        },
        {
            "rule-type": "object-mapping",
            "rule-id": "2",
            "rule-name": "DefaultMapToKinesis",
            "rule-action": "map-record-to-record",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "Customers"
            }
        }
    ]
}
```

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"  
}
```

However, suppose that you use the same rules but change the `rule-action` parameter to `map-record-to-document` and exclude certain columns. The following rule illustrates the mapping.

```
{  
    "rules": [  
        {  
            "rule-type": "selection",  
            "rule-id": "1",  
            "rule-name": "1",  
            "rule-action": "include",  
            "object-locator": {  
                "schema-name": "Test",  
                "table-name": "%"  
            }  
        },  
        {  
            "rule-type": "object-mapping",  
            "rule-id": "2",  
            "rule-name": "DefaultMapToKinesis",  
            "rule-action": "map-record-to-document",  
            "object-locator": {  
                "schema-name": "Test",  
                "table-name": "Customers"  
            },  
            "mapping-parameters": {  
                "exclude-columns": [  
                    "homeaddress",  
                    "homephone",  
                    "workaddress",  
                    "workphone"  
                ]  
            }  
        }  
    ]  
}
```

In this case, the columns not listed in the `exclude-columns` parameter, `FirstName`, `LastName`, `StoreId` and `DateOfBirth`, are mapped to `_doc`. The following illustrates the resulting record format.

```
{
```

```
"data":{  
    "_doc":{  
        "FirstName": "Randy",  
        "LastName": "Marsh",  
        "StoreId": "5",  
        "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",  
            "rule-action": "include",  
            "object-locator": {  
                "schema-name": "Test",  
                "table-name": "%"  
            }  
        },  
        {  
            "rule-type": "object-mapping",  
            "rule-id": "2",  
            "rule-name": "TransformToKinesis",  
            "rule-action": "map-record-to-record",  
            "target-table-name": "CustomerData",  
            "object-locator": {  
                "schema-name": "Test",  
                "table-name": "Customers"  
            },  
            "mapping-parameters": {  
                "partition-key-type": "attribute-name",  
                "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": "${lastname}, ${firstname}"  
                    },  
                    {  
                        "target-attribute-name": "ContactDetails",  
                        "attribute-type": "document",  
                        "attribute-sub-type": "json",  
                        "value": {  
                            "Home": {  
                                "Address": "${homeaddress}"  
                            }  
                        }  
                    }  
                ]  
            }  
        }  
    ]  
}
```

```
        "Phone": "${homephone}"  
    },  
    "Work": {  
        "Address": "${workaddress}",  
        "Phone": "${workphone}"  
    }  
}  
}  
]  
}  
]  
}
```

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}"
              }
            }
          }
        ]
      }
    }
  ]
}
```

```
        "Address": "${WorkAddress}",
        "Phone": "${WorkPhone}"
    }
},
{
    "attribute-name": "DateOfBirth",
    "value": "${DateOfBirth}"
}
]
}
]
```

**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.

## Message format for Kinesis Data Streams

The JSON output is simply a list of key-value pairs. A `JSON_UNFORMATTED` message format is a single line JSON string with new line delimiter.

**Note**

Support for `JSON_UNFORMATTED` Kinesis message format is available in AWS DMS versions 3.3.1 and later.

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.

## Using Apache Kafka as a target for AWS Database Migration Service

You can use AWS DMS to migrate data to an Apache Kafka cluster. Apache Kafka is a distributed streaming platform. You can use Apache Kafka for ingesting and processing streaming data in real-time.

AWS also offers Amazon Managed Streaming for Apache Kafka (Amazon MSK) to use as an AWS DMS target. Amazon MSK is a fully managed Apache Kafka streaming service that simplifies the implementation and management of Apache Kafka instances. It works with open-source Apache Kafka versions, and you access Amazon MSK instances as AWS DMS targets exactly like any Apache Kafka instance. For more information, see [What is Amazon MSK?](#) in the *Amazon Managed Streaming for Apache Kafka Developer Guide*.

A Kafka cluster stores streams of records in categories called topics that are divided into partitions. *Partitions* are uniquely identified sequences of data records (messages) in a topic. Partitions can be distributed across multiple brokers in a cluster to enable parallel processing of a topic's records. For more information on topics and partitions and their distribution in Apache Kafka, see [Topics and logs](#) and [Distribution](#).

AWS Database Migration Service publishes records to a Kafka topic using JSON. During conversion, AWS DMS serializes each record from the source database into an attribute-value pair in JSON format.

To migrate your data from any supported data source to a target Kafka cluster, you use object mapping. With object mapping, you determine how to structure the data records in the target topic. You also define a partition key for each table, which Apache Kafka uses to group the data into its partitions.

When AWS DMS creates tables on an Apache Kafka target endpoint, it creates as many tables as in the source database endpoint. AWS DMS also sets several Apache Kafka parameter values. The cost for the table creation depends on the amount of data and the number of tables to be migrated.

## Apache Kafka endpoint settings

You can specify connection details through endpoint settings in the AWS DMS console, or the `--kafka-settings` option in the CLI. The requirements for each setting follow:

- **Broker** – Specify the broker location in the form `broker-hostname:port`. For example, `"ec2-12-345-678-901.compute-1.amazonaws.com:2345"`. This can be the location of any broker in the cluster. The cluster brokers all communicate to handle the partitioning of data records migrated to the topic.
- **Topic** – (Optional) Specify the topic name with a maximum length of 255 letters and symbols. You can use period (.), underscore (\_), and minus (-). Topic names with a period (.) or underscore (\_) can collide in internal data structures. Use either one, but not both of these symbols in the topic name. If you don't specify a topic name, AWS DMS uses "kafka-default-topic" as the migration topic.

### Note

To have AWS DMS create either a migration topic you specify or the default topic, set `auto.create.topics.enable = true` as part of your Kafka cluster configuration. For more information, see [Limitations when using Apache Kafka as a target for AWS Database Migration Service \(p. 237\)](#)

- **MessageFormat** – The output format for the records created on the endpoint. The message format is `JSON` (default) or `JSON_UNFORMATTED` (a single line with no tab).
- **MessageMaxBytes** – The maximum size in bytes for records created on the endpoint. The default is 1,000,000.
- **IncludeTransactionDetails** – Provides detailed transaction information from the source database. This information includes a commit timestamp, a log position, and values for `transaction_id`, `previous_transaction_id`, and `transaction_record_id` (the record offset within a transaction). The default is `false`.
- **IncludePartitionValue** – Shows the partition value within the Kafka message output, unless the partition type is `schema-table-type`. The default is `false`.
- **PartitionIncludeSchemaTable** – Prefixes schema and table names to partition values, when the partition type is `primary-key-type`. Doing this increases data distribution among Kafka partitions. For example, suppose that a SysBench schema has thousands of tables and each table has only

limited range for a primary key. In this case, the same primary key is sent from thousands of tables to the same partition, which causes throttling. The default is `false`.

- `IncludeTableAlterOperations` – Includes any data definition language (DDL) operations that change the table in the control data, such as `rename-table`, `drop-table`, `add-column`, `drop-column`, and `rename-column`. The default is `false`.
- `IncludeControlDetails` – Shows detailed control information for table definition, column definition, and table and column changes in the Kafka message output. The default is `false`.
- `IncludeNullAndEmpty` – Include NULL and empty columns in the target. The default is `false`.

### Multithreaded full load task settings

You can use settings to help increase the speed of your transfer. To do so, AWS DMS supports a multithreaded full load to an Apache Kafka target cluster. AWS DMS supports this multithreading with task settings that include the following:

- `MaxFullLoadSubTasks` – Use this option to indicate the maximum number of source tables to load in parallel. AWS DMS loads each table into its corresponding Kafka target table using a dedicated subtask. The default is 8; the maximum value is 49.
- `ParallelLoadThreads` – Use this option to specify the number of threads that AWS DMS uses to load each table into its Kafka target table. The maximum value for an Apache Kafka target is 32. You can ask to have this maximum limit increased.
- `ParallelLoadBufferSize` – Use this option to specify the maximum number of records to store in the buffer that the parallel load threads use to load data to the Kafka target. The default value is 50. The maximum value is 1,000. Use this setting with `ParallelLoadThreads`. `ParallelLoadBufferSize` is valid only when there is more than one thread.
- `ParallelLoadQueuesPerThread` – Use this option to specify the number of queues each concurrent thread accesses to take data records out of queues and generate a batch load for the target. The default is 1. However, for Kafka targets of various payload sizes, the valid range is 5–512 queues per thread.
- `Table-mapping` settings for individual tables – Use `table-settings` rules to identify the individual tables from the source that you want to load in parallel. Also use these rules to specify how to segment the rows of each table for multithreaded loading. For more information, see [Table-settings rules and operations \(p. 333\)](#).

#### Note

DMS assigns each segment of a table to its own thread for loading. Therefore, set `ParallelLoadThreads` to the maximum number of segments that you specify for a table in the source.

### Multithreaded CDC load task settings

You can improve the performance of change data capture (CDC) for real-time data streaming target endpoints like Kafka using task settings to modify the behaviour of the `PutRecords` API call. To do this, you can specify the number of concurrent threads, queues per thread, and the number of records to store in a buffer using `ParallelApply*` task settings. For example, suppose you want to perform a CDC load and apply 128 threads in parallel. You also want to access 64 queues per thread, with 50 records stored per buffer.

To promote CDC performance, AWS DMS supports these task settings:

- `ParallelApplyThreads` – Specifies the number of concurrent threads that AWS DMS uses during a CDC load to push data records to a Kafka target endpoint. The default value is zero (0) and the maximum value is 32.
- `ParallelApplyBufferSize` – Specifies the maximum number of records to store in each buffer queue for concurrent threads to push to a Kafka target endpoint during a CDC load. The default value

is 50 and the maximum value is 1,000. Use this option when `ParallelApplyThreads` specifies more than one thread.

- `ParallelApplyQueuesPerThread` – Specifies the number of queues that each thread accesses to take data records out of queues and generate a batch load for a Kafka endpoint during CDC.

When using `ParallelApply*` task settings, the `partition-key-type` default is the `primary-key` of the table, not `schema-name.table-name`.

## Using a before image to view original values of CDC rows for Apache Kafka as a target

When writing CDC updates to a data-streaming target like Kafka you can view a source database row's original values before change by an update. To make this possible, AWS DMS populates a *before image* of update events based on data supplied by the source database engine.

Different source database engines provide different amounts of information for a before image:

- Oracle provides updates to columns only if they change.
- PostgreSQL provides only data for columns that are part of the primary key (changed or not).
- MySQL generally provides data for all columns (changed or not).

To enable before imaging to add original values from the source database to the AWS DMS output, use either the `BeforeImageSettings` task setting or the `add-before-image-columns` parameter. This parameter applies a column transformation rule.

`BeforeImageSettings` adds a new JSON attribute to every update operation with values collected from the source database system, as shown following.

```
"BeforeImageSettings": {  
    "EnableBeforeImage": boolean,  
    "FieldName": string,  
    "ColumnFilter": pk-only (default) / non-lob / all (but only one)  
}
```

### Note

Apply `BeforeImageSettings` to full load plus CDC tasks (which migrate existing data and replicate ongoing changes), or to CDC only tasks (which replicate data changes only). Don't apply `BeforeImageSettings` to tasks that are full load only.

For `BeforeImageSettings` options, the following applies:

- Set the `EnableBeforeImage` option to `true` to enable before imaging. The default is `false`.
- Use the `FieldName` option to assign a name to the new JSON attribute. When `EnableBeforeImage` is `true`, `FieldName` is required and can't be empty.
- The `ColumnFilter` option specifies a column to add by using before imaging. To add only columns that are part of the table's primary keys, use the default value, `pk-only`. To add only columns that are not of LOB type, use `non-lob`. To add any column that has a before image value, use `all`.

```
"BeforeImageSettings": {  
    "EnableBeforeImage": true,  
    "FieldName": "before-image",  
    "ColumnFilter": "pk-only"
```

}

## Using a before image transformation rule

As an alternative to task settings, you can use the `add-before-image-columns` parameter, which applies a column transformation rule. With this parameter, you can enable before imaging during CDC on data streaming targets like Kafka.

By using `add-before-image-columns` in a transformation rule, you can apply more fine-grained control of the before image results. Transformation rules enable you to use an object locator that gives you control over tables selected for the rule. Also, you can chain transformation rules together, which allows different rules to be applied to different tables. You can then manipulate the columns produced by using other rules.

### Note

Don't use the `add-before-image-columns` parameter together with the `BeforeImageSettings` task setting within the same task. Instead, use either the parameter or the setting, but not both, for a single task.

A transformation rule type with the `add-before-image-columns` parameter for a column must provide a `before-image-def` section. The following shows an example.

```
{  
    "rule-type": "transformation",  
    ...  
    "rule-target": "column",  
    "rule-action": "add-before-image-columns",  
    "before-image-def":{  
        "column-filter": one-of (pk-only / non-lob / all),  
        "column-prefix": string,  
        "column-suffix": string,  
    }  
}
```

The value of `column-prefix` is prepended to a column name, and the default value of `column-prefix` is `BI_`. The value of `column-suffix` is appended to the column name, and the default is empty. Don't set both `column-prefix` and `column-suffix` to empty strings.

Choose one value for `column-filter`. To add only columns that are part of table primary keys, choose `pk-only`. Choose `non-lob` to only add columns that are not of LOB type. Or choose `all` to add any column that has a before-image value.

## Example for a before image transformation rule

The transformation rule in the following example adds a new column called `BI_emp_no` in the target. So a statement like `UPDATE employees SET emp_no = 3 WHERE emp_no = 1;` populates the `BI_emp_no` field with 1. When you write CDC updates to Amazon S3 targets, the `BI_emp_no` column makes it possible to tell which original row was updated.

```
{  
    "rules": [  
        {  
            "rule-type": "selection",  
            "rule-id": "1",  
            "rule-name": "1",  
            "object-locator": {  
                "schema-name": "%",  
                "table-name": "%"}}
```

```

        },
        "rule-action": "include"
    },
    {
        "rule-type": "transformation",
        "rule-id": "2",
        "rule-name": "2",
        "rule-target": "column",
        "object-locator": {
            "schema-name": "%",
            "table-name": "employees"
        },
        "rule-action": "add-before-image-columns",
        "before-image-def": {
            "column-prefix": "BI_",
            "column-suffix": "",
            "column-filter": "pk-only"
        }
    }
]
}

```

For information on using the add-before-image-columns rule action, see [Transformation rules and actions \(p. 318\)](#).

## Limitations when using Apache Kafka as a target for AWS Database Migration Service

The following limitations apply when using Apache Kafka as a target:

- AWS DMS supports a maximum message size of 1 MiB for a Kafka target.
- Configure both your AWS DMS replication instance and your Kafka cluster in the same virtual private cloud (VPC) based on Amazon VPC and in the same security group. The Kafka cluster can either be an Amazon MSK instance or your own Kafka instance running on Amazon EC2. For more information, see [Setting up a network for a replication instance \(p. 45\)](#).

### Note

To specify a security group for Amazon MSK, on the [Create cluster](#) page, choose **Advanced settings**, select **Customize settings**, and select the security group or accept the default if it is the same as for your replication instance.

- Specify a Kafka configuration file for your cluster with properties that allow AWS DMS to automatically create new topics. Include the setting, `auto.create.topics.enable = true`. If you are using Amazon MSK, you can specify the default configuration when you create your Kafka cluster, then change the `auto.create.topics.enable` setting to `true`. For more information about the default configuration settings, see [The default Amazon MSK configuration](#) in the *Amazon Managed Streaming for Apache Kafka Developer Guide*. If you need to modify an existing Kafka cluster created using Amazon MSK, run the AWS CLI command `aws kafka create-configuration` to update your Kafka configuration, as in the example following:

```

14:38:41 $ aws kafka create-configuration --name "kafka-configuration" --kafka-versions
           "2.2.1" --server-properties file://~/kafka_configuration
{
    "LatestRevision": {
        "Revision": 1,
        "CreationTime": "2019-09-06T14:39:37.708Z"
    },
    "CreationTime": "2019-09-06T14:39:37.708Z",
    "Name": "kafka-configuration",
    "Arn": "arn:aws:kafka:us-east-1:111122223333:configuration/kafka-
           configuration/7e008070-6a08-445f-9fe5-36ccf630ecfd-3"
}

```

```
}
```

Here, `//~/kafka_configuration` is the configuration file you have created with the required property settings.

If you are using your own Kafka instance installed on Amazon EC2, modify the Kafka cluster configuration with similar property settings, including `auto.create.topics.enable = true`, using the options provided with your instance.

- AWS DMS publishes each update to a single record in the source database as one data record (message) in a given Kafka topic regardless of transactions.
- 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 Kafka topic

AWS DMS uses table-mapping rules to map data from the source to the target Kafka topic. To map data to a target topic, 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 a Kafka topic.

Kafka topics don't have a preset structure other than having a partition key.

To create an object-mapping rule, 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. The `map-record-to-record` and `map-record-to-document` values 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 Kafka topic. This rule type uses the `taskResourceId.schemaName.tableName` value from the relational database as the partition key in the Kafka topic 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 topic. 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	31 Spooner Street, Quahog	987654321002	02/29/1988

To migrate this information from a schema named `Test` to a Kafka topic, you create rules to map the data to the target topic. The following rule illustrates the mapping.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "rule-action": "include",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "%"
            }
        },
        {
            "rule-type": "object-mapping",
            "rule-id": "2",
            "rule-name": "DefaultMapToKafka",
            "rule-action": "map-record-to-record",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "Customers"
            }
        }
    ]
}
```

Given a Kafka topic and a partition key (in this case, `taskResourceId.schemaName.tableName`), the following illustrates the resulting record format using our sample data in the Kafka target topic:

```
{
    "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 Kafka topic 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",
            "rule-action": "include",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "%"
            }
        },
        {
            "rule-type": "object-mapping",
            "rule-id": "2",
            "rule-name": "DefaultMapToKafka",
            "rule-action": "map-record-to-record",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "Customers"
            }
        }
    ]
}
```

```

        "rule-type": "selection",
        "rule-id": "1",
        "rule-name": "1",
        "rule-action": "include",
        "object-locator": {
            "schema-name": "Test",
            "table-name": "%"
        }
    },
    {
        "rule-type": "object-mapping",
        "rule-id": "2",
        "rule-name": "TransformToKafka",
        "rule-action": "map-record-to-record",
        "target-table-name": "CustomerData",
        "object-locator": {
            "schema-name": "Test",
            "table-name": "Customers"
        },
        "mapping-parameters": {
            "partition-key-type": "attribute-name",
            "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": "${lastname}, ${firstname}"
                },
                {
                    "target-attribute-name": "ContactDetails",
                    "attribute-type": "document",
                    "attribute-sub-type": "json",
                    "value": {
                        "Home": {
                            "Address": "${homeaddress}",
                            "Phone": "${homephone}"
                        },
                        "Work": {
                            "Address": "${workaddress}",
                            "Phone": "${workphone}"
                        }
                    }
                }
            ]
        }
    }
}

```

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 partition. 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": "TransformToKafka",
    "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}"
            }
        ]
    }
}
}
```

### 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.

## Message format for Apache Kafka

The JSON output is simply a list of key-value pairs.

### 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.

## 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 Amazon ES cluster.

In Amazon ES, 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 ES, it creates one index for each table from the source endpoint. The cost for creating an Amazon ES 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.

Configure your Amazon ES 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 document count.

### Multithreaded full load task settings

To help increase the speed of the transfer, AWS DMS supports a multithreaded full load to an Amazon ES target cluster. AWS DMS supports this multithreading with task settings that include the following:

- **MaxFullLoadSubTasks** – Use this option to indicate the maximum number of source tables to load in parallel. DMS loads each table into its corresponding Amazon ES target index using a dedicated subtask. The default is 8; the maximum value is 49.
- **ParallelLoadThreads** – Use this option to specify the number of threads that AWS DMS uses to load each table into its Amazon ES target index. The maximum value for an Amazon ES target is 32. You can ask to have this maximum limit increased.

**Note**

If you don't change `ParallelLoadThreads` from its default (0), AWS DMS transfers a single record at a time. This approach puts undue load on your Amazon ES cluster. Make sure that you set this option to 1 or more.

- **ParallelLoadBufferSize** – Use this option to specify the maximum number of records to store in the buffer that the parallel load threads use to load data to the Amazon ES target. The default value is 50. The maximum value is 1,000. Use this setting with `ParallelLoadThreads`. `ParallelLoadBufferSize` is valid only when there is more than one thread.
- **Table-mapping settings for individual tables** – Use `table-settings` rules to identify individual tables from the source that you want to load in parallel. Also use these rules to specify how to segment the rows of each table for multithreaded loading. For more information, see [Table-settings rules and operations \(p. 333\)](#).

**Note**

DMS assigns each segment of a table to its own thread for loading. Therefore, set `ParallelLoadThreads` to the maximum number of segments that you specify for a table in the source.

For more information on how DMS loads an Amazon ES cluster using multithreading, see the AWS blog post [Scale Amazon Elasticsearch Service for AWS Database Migration Service migrations](#).

### Multithreaded CDC load task settings

You can improve the performance of change data capture (CDC) for an Amazon ES target cluster using task settings to modify the behavior of the `PutRecords` API call. To do this, you can specify the number of concurrent threads, queues per thread, and the number of records to store in a buffer using `ParallelApply*` task settings. For example, suppose you want to perform a CDC load and apply 128 threads in parallel. You also want to access 64 queues per thread, with 50 records stored per buffer.

**Note**

Support for the use of `ParallelApply*` task settings during CDC to Amazon Elasticsearch Service target endpoints is available in AWS DMS versions 3.4.0 and later.

To promote CDC performance, AWS DMS supports these task settings:

- **ParallelApplyThreads** – Specifies the number of concurrent threads that AWS DMS uses during a CDC load to push data records to a Amazon ES target endpoint. The default value is zero (0) and the maximum value is 32.
- **ParallelApplyBufferSize** – Specifies the maximum number of records to store in each buffer queue for concurrent threads to push to a Amazon ES target endpoint during a CDC load. The default value is 50 and the maximum value is 1,000. Use this option when `ParallelApplyThreads` specifies more than one thread.
- **ParallelApplyQueuesPerThread** – Specifies the number of queues that each thread accesses to take data records out of queues and generate a batch load for a Amazon ES endpoint during CDC.

When using `ParallelApply*` task settings, the `partition-key-type` default is the `primary-key` of the table, not `schema-name.table-name`.

## Migrating from a relational database table to an Amazon ES index

AWS DMS supports migrating data to Amazon ES's scalar data types. When migrating from a relational database like Oracle or MySQL to Amazon ES, you might want to restructure how you store this data.

AWS DMS supports the following Amazon ES 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 Amazon ES 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 Amazon ES indexes at the target endpoint. The minimum set of access permissions is shown in the following IAM 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 Amazon ES must have the following permissions.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "es:ESHttpDelete",  
                "es:ESHttpGet",  
                "es:ESHttpHead",  
                "es:ESHttpPost",  
                "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 Amazon ES as a target for AWS DMS

When you set up your Amazon ES 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 Amazon ES instance as an AWS DMS source.

Attribute name	Valid values	Default value and description
<code>fullLoadErrorPercentage</code>	A 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.
<code>errorRetryDuration</code>	A 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:

- Amazon ES uses dynamic mapping (auto guess) to determine the data types to use for migrated data.
- Amazon ES 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 Amazon ES 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 Amazon ES 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 Amazon ES.

AWS DMS data type	Amazon ES data type
Boolean	boolean
Date	string
Time	date
Timestamp	date
INT4	integer
Real4	float
UINT4	integer

For additional information about AWS DMS data types, see [Data types for AWS Database Migration Service \(p. 488\)](#).

## 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 \(p. 144\)](#).

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 S3 as a source for AWS DMS \(p. 145\)](#).

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 \(p. 247\)](#)
- [Ongoing replication with Amazon DocumentDB as a target \(p. 250\)](#)
- [Limitations to using Amazon DocumentDB as a target \(p. 251\)](#)
- [Target data types for Amazon DocumentDB \(p. 252\)](#)
- [Walkthrough: Migrating from MongoDB to Amazon DocumentDB \(p. 253\)](#)

## 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\": \"11111111\"}, \"Work\": {\"Address\": \"Boston\", \"Phone\": \"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",
    "ContactDetails": "{\"Home\": {\"Address\": \"Boston\", \"Phone\": \"11111111\"}, \"Work\": {\"Address\": \"Boston\", \"Phone\": \"2222222222\"}}"
}
```

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
<code>ContactAddress</code>	<code>array_ContactAddress</code>
<code>ContactPhoneNumbers</code>	<code>array_ContactPhoneNumbers</code>

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 retrieve the public key for Amazon DocumentDB by downloading the file, `rds-combined-ca-bundle.pem`, from an AWS-hosted Amazon S3 bucket. For more information on downloading this file, see [Encrypting connections using TLS](#) in the *Amazon DocumentDB Developer Guide*

After you download this .pem file, you can import the public key that it contains 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
CREATE TABLE	Creates an empty collection.
Statement that renames a table (RENAME TABLE, ALTER TABLE...RENAME, and similar)	Renames the collection.
TRUNCATE TABLE	Removes all the documents from the collection, but only if <code>HandleSourceTableTruncated</code> is true. For more information, see <a href="#">Task settings for change processing DDL handling (p. 291)</a> .
DROP TABLE	Deletes the collection, but only if <code>HandleSourceTableDropped</code> is true. For more information, see <a href="#">Task settings for change processing DDL handling (p. 291)</a> .
Statement that adds a column to a table (ALTER TABLE...ADD 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.
ALTER TABLE...RENAME COLUMN	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.
ALTER TABLE...DROP COLUMN	The DDL statement is ignored, and a warning is issued.
Statement that changes the column data type (ALTER COLUMN...MODIFY 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 \(p. 488\)](#).

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).

AWS DMS data type	Amazon DocumentDB data type
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 \(p. 253\)](#)
- [Step 2: Install and configure MongoDB community edition \(p. 254\)](#)
- [Step 3: Create an AWS DMS replication instance \(p. 255\)](#)
- [Step 4: Create source and target endpoints \(p. 256\)](#)
- [Step 5: Create and run a migration task \(p. 258\)](#)

### 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 TPC port 22. If this isn't the case for your VPC, add this rule; for more information, see [What is Amazon VPC?](#) in the *Amazon VPC User Guide*.
  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 \(p. 253\)](#).

### 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 `mongoRep`. API Version API Version 2016-01-01

- 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. Here, *aws-api-domain* completes the Amazon S3 domain in your AWS Region required to access the specified S3 bucket and the *rds-combined-ca-bundle.pem* file that it provides.

```
wget https://s3.aws-api-domain/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**, do one of the following to attach the SSL certificate to your endpoint:
    - If available, choose the existing **rds-combined-ca-bundle** certificate from the **Choose a certificate** drop down.
    - Choose **Add new CA certificate**. Then, for **Certificate identifier**, enter **rds-combined-ca-bundle**. For **Import certificate file**, choose **Choose file** and navigate to the *rds-combined-ca-bundle.pem* file that you previously downloaded. Select and open the file. Choose **Import certificate**, then choose **rds-combined-ca-bundle** from the **Choose a certificate** drop down.
  - For **User name**, enter the master user name of your Amazon DocumentDB cluster.
  - For **Password**, enter the master password of your Amazon DocumentDB cluster.
  - For **Database name**, enter *zips-db*.

When the settings are as you want them, choose **Create endpoint**.

Now that you've created the source and target endpoints, test them to ensure that they work correctly. Also, to ensure that AWS DMS can access the database objects at each endpoint, refresh the endpoints' schemas.

## To test an endpoint

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 \(p. 255\)](#).
  - For **Source endpoint**, choose the source endpoint that you created in [Step 4: Create source and target endpoints \(p. 256\)](#).
  - For **Target endpoint**, choose the target endpoint that you created in [Step 4: Create source and target endpoints \(p. 256\)](#).
  - 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*.

# Using Amazon Neptune as a target for AWS Database Migration Service

Amazon Neptune is a fast, reliable, fully managed graph database service that makes it easy to build and run applications that work with highly connected datasets. The core of Neptune is a purpose-built, high-performance graph database engine. This engine is optimized for storing billions of relationships and querying the graph with milliseconds latency. Neptune supports the popular graph query languages Apache TinkerPop Gremlin and W3C's SPARQL. For more information on Amazon Neptune, see [What is Amazon Neptune?](#) in the *Amazon Neptune User Guide*.

Without a graph database such as Neptune, you probably model highly connected data in a relational database. Because the data has potentially dynamic connections, applications that use such data sources have to model connected data queries in SQL. This approach requires you to write an extra layer to convert graph queries into SQL. Also, relational databases come with schema rigidity. Any changes in the schema to model changing connections require downtime and additional maintenance of the query conversion to support the new schema. The query performance is also another big constraint to consider while designing your applications.

Graph databases can greatly simplify such situations. Free from a schema, a rich graph query layer (Gremlin or SPARQL) and indexes optimized for graph queries increase flexibility and performance. The Amazon Neptune graph database also has enterprise features such as encryption at rest, a secure authorization layer, default backups, Multi-AZ support, read replica support, and others.

Using AWS DMS, you can migrate relational data that models a highly connected graph to a Neptune target endpoint from a DMS source endpoint for any supported SQL database.

For more details, see the following.

## Topics

- [Overview of migrating to Amazon Neptune as a target \(p. 259\)](#)
- [Specifying endpoint settings for Amazon Neptune as a target \(p. 260\)](#)
- [Creating an IAM service role for accessing Amazon Neptune as a target \(p. 261\)](#)
- [Specifying graph-mapping rules using Gremlin and R2RML for Amazon Neptune as a target \(p. 263\)](#)
- [Data types for Gremlin and R2RML migration to Amazon Neptune as a target \(p. 266\)](#)
- [Limitations of using Amazon Neptune as a target \(p. 268\)](#)

## Overview of migrating to Amazon Neptune as a target

Before starting a migration to a Neptune target, create the following resources in your AWS account:

- A Neptune cluster for the target endpoint.
- A SQL relational database supported by AWS DMS for the source endpoint.
- An Amazon S3 bucket for the target endpoint. Create this S3 bucket in the same AWS Region as your Neptune cluster. AWS DMS uses this S3 bucket as intermediate file storage for the target data that it bulk loads to the Neptune database. For more information on creating an S3 bucket, see [Creating a bucket](#) in the *Amazon Simple Storage Service Getting Started Guide*.
- A virtual private cloud (VPC) endpoint for S3 in the same VPC as the Neptune cluster.
- An AWS Identity and Access Management (IAM) role that includes an IAM policy. This policy should specify the `GetObject`, `PutObject`, `DeleteObject` and `ListObject` permissions to the S3 bucket for your target endpoint. This role is assumed by both AWS DMS and Neptune with IAM access to both the target S3 bucket and the Neptune database. For more information, see [Creating an IAM service role for accessing Amazon Neptune as a target \(p. 261\)](#).

After you have these resources, setting up and starting a migration to a Neptune target is similar to any full load migration using the console or DMS API. However, a migration to a Neptune target requires some unique steps.

### To migrate an AWS DMS relational database to Neptune

1. Create a replication instance as described in [Creating a replication instance \(p. 52\)](#).
2. Create and test a SQL relational database supported by AWS DMS for the source endpoint.
3. Create and test the target endpoint for your Neptune database.

To connect the target endpoint to the Neptune database, specify the server name for either the Neptune cluster endpoint or the Neptune writer instance endpoint. Also, specify the S3 bucket folder for AWS DMS to store its intermediate files for bulk load to the Neptune database.

During migration, AWS DMS stores all migrated target data in this S3 bucket folder up to a maximum file size that you specify. When this file storage reaches this maximum size, AWS DMS bulk loads the stored S3 data into the target database. It clears the folder to enable storage of any additional target data for subsequent loading to the target database. For more information on specifying these settings, see [Specifying endpoint settings for Amazon Neptune as a target \(p. 260\)](#).

4. Create a full-load replication task with the resources created in steps 1–3 and do the following:
  - a. Use task table mapping as usual to identify specific source schemas, tables, and views to migrate from your relational database using appropriate selection and transformation rules. For more information, see [Using table mapping to specify task settings \(p. 309\)](#).
  - b. Specify target mappings by choosing one of the following to specify mapping rules from source tables and views to your Neptune target database graph:
    - Gremlin JSON – For information on using Gremlin JSON to load a Neptune database, see [Gremlin load data format](#) in the *Amazon Neptune User Guide*.
    - SPARQL RDB to Resource Description Framework Mapping Language (R2RML) – For information on using SPARQL R2RML, see the W3C specification [R2RML: RDB to RDF mapping language](#).
  - c. Do one of the following:
    - Using the AWS DMS console, specify graph-mapping options using **Graph mapping rules** on the **Create database migration task** page.
    - Using the AWS DMS API, specify these options using the `TaskData` request parameter of the `CreateReplicationTask` API call.

For more information and examples using Gremlin JSON and SPARQL R2RML to specify graph-mapping rules, see [Specifying graph-mapping rules using Gremlin and R2RML for Amazon Neptune as a target \(p. 263\)](#).

5. Start the replication for your migration task.

## Specifying endpoint settings for Amazon Neptune as a target

To create or modify a target endpoint, you can use the console or the `CreateEndpoint` or `ModifyEndpoint` API operations.

For a Neptune target in the AWS DMS console, specify **Endpoint-specific settings** on the **Create endpoint** or **Modify endpoint** console page. For `CreateEndpoint` and `ModifyEndpoint`, specify request parameters for the `NeptuneSettings` option. The example following shows how to do this using the CLI.

```
dms create-endpoint --endpoint-identifier my-neptune-target-endpoint
--endpoint-type target --engine-name neptune
--server-name my-neptune-db.cluster-cspckvklbvgf.us-east-1.neptune.amazonaws.com
--port 8192
--neptune-settings
  '{"ServiceAccessRoleArn":"arn:aws:iam::123456789012:role/myNeptuneRole",
   "S3BucketName":"my-bucket",
   "S3BucketFolder":"my-bucket-folder",
   "ErrorRetryDuration":57,
   "MaxFileSize":100,
   "MaxRetryCount": 10,
   "IAMAuthEnabled":false}'
```

Here, the CLI `--server-name` option specifies the server name for the Neptune cluster writer endpoint. Or you can specify the server name for a Neptune writer instance endpoint.

The `--neptune-settings` option request parameters follow:

- `ServiceAccessRoleArn` – (Required) The Amazon Resource Name (ARN) of the service role that you created for the Neptune target endpoint. For more information, see [Creating an IAM service role for accessing Amazon Neptune as a target \(p. 261\)](#).
- `S3BucketName` – (Required) The name of the S3 bucket where DMS can temporarily store migrated graph data in .csv files before bulk loading it to the Neptune target database. DMS maps the SQL source data to graph data before storing it in these .csv files.
- `S3BucketFolder` – (Required) A folder path where you want DMS to store migrated graph data in the S3 bucket specified by `S3BucketName`.
- `ErrorRetryDuration` – (Optional) The number of milliseconds for DMS to wait to retry a bulk load of migrated graph data to the Neptune target database before raising an error. The default is 250.
- `MaxFileSize` – (Optional) The maximum size in KB of migrated graph data stored in a .csv file before DMS bulk loads the data to the Neptune target database. The default is 1,048,576 KB (1 GB). If successful, DMS clears the bucket, ready to store the next batch of migrated graph data.
- `MaxRetryCount` – (Optional) The number of times for DMS to retry a bulk load of migrated graph data to the Neptune target database before raising an error. The default is 5.
- `IAMAuthEnabled` – (Optional) If you want IAM authorization enabled for this endpoint, set this parameter to `true` and attach the appropriate IAM policy document to your service role specified by `ServiceAccessRoleArn`. The default is `false`.

## Creating an IAM service role for accessing Amazon Neptune as a target

To access Neptune as a target, create a service role using IAM. Depending on your Neptune endpoint configuration, attach to this role some or all of the IAM policy and trust documents described following. When you create the Neptune endpoint, you provide the ARN of this service role. Doing so enables AWS DMS and Amazon Neptune to assume permissions to access both Neptune and its associated Amazon S3 bucket.

If you set the `IAMAuthEnabled` parameter in `NeptuneSettings` to `true` in your Neptune endpoint configuration, attach an IAM policy like the following to your service role. If you set `IAMAuthEnabled` to `false`, you can ignore this policy.

```
// Policy to access Neptune
{
  "Version": "2012-10-17",
  "Statement": [
```

```
{
    "Sid": "VisualEditor0",
    "Effect": "Allow",
    "Action": "neptune-db:*",
    "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-CLG7H7FHK54AZGHEH6MNS55JKM/*"
}
]
```

The preceding IAM policy allows full access to the Neptune target cluster specified by `Resource`.

Attach an IAM policy like the following to your service role. This policy allows DMS to temporarily store migrated graph data in the S3 bucket that you created for bulk loading to the Neptune target database.

```
//Policy to access S3 bucket

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListObjectsInBucket0",
            "Effect": "Allow",
            "Action": "s3>ListBucket",
            "Resource": [
                "arn:aws:s3:::my-bucket"
            ]
        },
        {
            "Sid": "AllObjectActions",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3>DeleteObject",
                "s3:DeleteObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket/"
            ],
            {
                "Sid": "ListObjectsInBucket1",
                "Effect": "Allow",
                "Action": "s3>ListBucket",
                "Resource": [
                    "arn:aws:s3:::my-bucket",
                    "arn:aws:s3:::my-bucket/"
                ]
            }
        ]
    }
}
```

The preceding IAM policy allows your account to query the contents of the S3 bucket (`arn:aws:s3:::my-bucket`) created for your Neptune target. It also allows your account to fully operate on the contents of all bucket files and folders (`arn:aws:s3:::my-bucket/`).

Edit the trust relationship and attach the following IAM role to your service role to allow both AWS DMS and Amazon Neptune database service to assume the role.

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

```
    "Effect": "Allow",
    "Principal": {
        "Service": "dms.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
},
{
    "Sid": "neptune",
    "Effect": "Allow",
    "Principal": {
        "Service": "rds.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
}
]
}
```

For information about specifying this service role for your Neptune target endpoint, see [Specifying endpoint settings for Amazon Neptune as a target \(p. 260\)](#).

## Specifying graph-mapping rules using Gremlin and R2RML for Amazon Neptune as a target

The graph-mapping rules that you create specify how data extracted from an SQL relational database source is loaded into a Neptune database cluster target. The format of these mapping rules differs depending on whether the rules are for loading property-graph data using Apache TinkerPop Gremlin or Resource Description Framework (RDF) data using R2RML. Following, you can find information about these formats and where to learn more.

You can specify these mapping rules when you create the migration task using either the console or DMS API.

Using the console, specify these mapping rules using **Graph mapping rules** on the **Create database migration task** page. In **Graph mapping rules**, you can enter and edit the mapping rules directly using the editor provided. Or you can browse for a file that contains the mapping rules in the appropriate graph-mapping format.

Using the API, specify these options using the `TaskData` request parameter of the `CreateReplicationTask` API call. Set `TaskData` to the path of a file containing the mapping rules in the appropriate graph-mapping format.

### Graph-mapping rules for generating property-graph data using Gremlin

Using Gremlin to generate the property-graph data, specify a JSON object with a mapping rule for each graph entity to be generated from the source data. The format of this JSON is defined specifically for bulk loading Amazon Neptune. The template following shows what each rule in this object looks like.

```
{
    "rules": [
        {
            "rule_id": "(an identifier for this rule)",
            "rule_name": "(a name for this rule)",
            "table_name": "(the name of the table or view being loaded)",
            "vertex_definitions": [
                {
                    "vertex_id_template": "{col1}",
                    "vertex_label": "(the vertex to create)",
                    "vertex_definition_id": "(an identifier for this vertex)",
                    "vertex_properties": [
                        ...
                    ]
                }
            ]
        }
    ]
}
```

```

        {
            "property_name": "(name of the property)",
            "property_value_template": "{col2} or text",
            "property_value_type": "(data type of the property)"
        }
    ]
}
{
    "rule_id": "(an identifier for this rule)",
    "rule_name": "(a name for this rule)",
    "table_name": "(the name of the table or view being loaded)",
    "edge_definitions": [
        {
            "from_vertex": {
                "vertex_id_template": "{col1}",
                "vertex_definition_id": "(an identifier for the vertex referenced above)"
            },
            "to_vertex": {
                "vertex_id_template": "{col3}",
                "vertex_definition_id": "(an identifier for the vertex referenced above)"
            },
            "edge_id_template": {
                "label": "(the edge label to add)",
                "template": "{col1}_{col3}"
            },
            "edge_properties": [
                {
                    "property_name": "(the property to add)",
                    "property_value_template": "{col4} or text",
                    "property_value_type": "(data type like String, int, double)"
                }
            ]
        }
    ]
}

```

The presence of a vertex label implies that the vertex is being created here. Its absence implies that the vertex is created by a different source, and this definition is only adding vertex properties. Specify as many vertex and edge definitions as required to specify the mappings for your entire relational database source.

A sample rule for an employee table follows.

```
{
    "rules": [
        {
            "rule_id": "1",
            "rule_name": "vertex_mapping_rule_from_nodes",
            "table_name": "nodes",
            "vertex_definitions": [
                {
                    "vertex_id_template": "{emp_id}",
                    "vertex_label": "employee",
                    "vertex_definition_id": "1",
                    "vertex_properties": [

```

```

        "property_name": "name",
        "property_value_template": "{emp_name}",
        "property_value_type": "String"
    }
]
}
],
{
"rule_id": "2",
"rule_name": "edge_mapping_rule_from_emp",
"table_name": "nodes",
"edge_definitions": [
{
"from_vertex": {
"vertex_id_template": "{emp_id}",
"vertex_definition_id": "1"
},
"to_vertex": {
"vertex_id_template": "{mgr_id}",
"vertex_definition_id": "1"
},
"edge_id_template": {
"label": "reportsTo",
"template": "{emp_id}_{mgr_id}"
},
"edge_properties": [
{
"property_name": "team",
"property_value_template": "{team}",
"property_value_type": "String"
}
]
}
]
}
]
```

Here, the vertex and edge definitions map a reporting relationship from an employee node with employee ID (`EmpID`) and an employee node with a manager ID (`managerId`).

For more information about creating graph-mapping rules using Gremlin JSON, see [Gremlin load data format](#) in the *Amazon Neptune User Guide*.

## Graph-mapping rules for generating RDF/SPARQL data

If you are loading RDF data to be queried using SPARQL, write the graph-mapping rules in R2RML. R2RML is a standard W3C language for mapping relational data to RDF. In an R2RML file, a *triples map* (for example, `<#TriplesMap1>` following) specifies a rule for translating each row of a logical table to zero or more RDF triples. A *subject map* (for example, any `rr:subjectMap` following) specifies a rule for generating the subjects of the RDF triples generated by a triples map. A *predicate-object map* (for example, any `rr:predicateObjectMap` following) is a function that creates one or more predicate-object pairs for each logical table row of a logical table.

A simple example for a `nodes` table follows.

```

@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ex: <http://example.com/ns#>.

<#TriplesMap1>
  rr:logicalTable [ rr:tableName "nodes" ];
```

```
rr:subjectMap [
    rr:template "http://data.example.com/employee/{id}";
    rr:class ex:Employee;
];
rr:predicateObjectMap [
    rr:predicate ex:name;
    rr:objectMap [ rr:column "label" ];
]
```

In the previous example, the mapping defines graph nodes mapped from a table of employees.

Another simple example for a Student table follows.

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ex: <http://example.com/#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

<#TriplesMap2>
rr:logicalTable [ rr:tableName "Student" ];
rr:subjectMap [ rr:template "http://example.com/{ID}{Name}";
    rr:class foaf:Person ];
rr:predicateObjectMap [
    rr:predicate ex:id ;
    rr:objectMap [ rr:column "ID";
        rr:datatype xsd:integer ]
];
rr:predicateObjectMap [
    rr:predicate foaf:name ;
    rr:objectMap [ rr:column "Name" ]
].
```

In the previous example, the mapping defines graph nodes mapping friend-of-a-friend relationships between persons in a Student table.

For more information about creating graph-mapping rules using SPARQL R2RML, see the W3C specification [R2RML: RDB to RDF mapping language](#).

## Data types for Gremlin and R2RML migration to Amazon Neptune as a target

AWS DMS performs data type mapping from your SQL source endpoint to your Neptune target in one of two ways. Which way you use depends on the graph mapping format that you're using to load the Neptune database:

- Apache TinkerPop Gremlin, using a JSON representation of the migration data.
- W3C's SPARQL, using an R2RML representation of the migration data.

For more information on these two graph mapping formats, see [Specifying graph-mapping rules using Gremlin and R2RML for Amazon Neptune as a target \(p. 263\)](#).

Following, you can find descriptions of the data type mappings for each format.

### SQL source to Gremlin target data type mappings

The following table shows the data type mappings from a SQL source to a Gremlin formatted target.

AWS DMS maps any unlisted SQL source data type to a Gremlin String.

SQL source data types	Gremlin target data types
NUMERIC (and variants)	Double
DECIMAL	
TINYINT	Byte
SMALLINT	Short
INT, INTEGER	Int
BIGINT	Long
FLOAT	Float
DOUBLE PRECISION	
REAL	Double
BIT	Boolean
BOOLEAN	
DATE	Date
TIME	
TIMESTAMP	
CHARACTER (and variants)	String

For more information on the Gremlin data types for loading Neptune, see [Gremlin data types in the Neptune User Guide](#).

### SQL source to R2RML (RDF) target data type mappings

The following table shows the data type mappings from a SQL source to an R2RML formatted target.

All listed RDF data types are case-sensitive, except RDF literal. AWS DMS maps any unlisted SQL source data type to an RDF literal.

An *RDF literal* is one of a variety of literal lexical forms and data types. For more information, see [RDF literals](#) in the W3C specification *Resource Description Framework (RDF): Concepts and Abstract Syntax*.

SQL source data types	R2RML (RDF) target data types
BINARY (and variants)	xsd:hexBinary
NUMERIC (and variants)	xsd:decimal
DECIMAL	
TINYINT	xsd:integer
SMALLINT	
INT, INTEGER	
BIGINT	

SQL source data types	R2RML (RDF) target data types
FLOAT	xsd:double
DOUBLE PRECISION	
REAL	
BIT	xsd:boolean
BOOLEAN	
DATE	xsd:date
TIME	xsd:time
TIMESTAMP	xsd:dateTime
CHARACTER (and variants)	RDF literal

For more information on the RDF data types for loading Neptune and their mappings to SQL source data types, see [Datatype conversions](#) in the W3C specification *R2RML: RDB to RDF Mapping Language*.

## Limitations of using Amazon Neptune as a target

The following limitations apply when using Neptune as a target:

- AWS DMS currently supports full load tasks only for migration to a Neptune target. Change data capture (CDC) migration to a Neptune target isn't supported.
- Make sure that your target Neptune database is manually cleared of all data before starting the migration task, as in the examples following.

To drop all data (vertices and edges) within the graph, run the Gremlin command following.

```
gremlin> g.V().drop().iterate()
```

To drop vertices that have the label 'customer', run the Gremlin command following.

```
gremlin> g.V().hasLabel('customer').drop()
```

### Note

It can take some time to drop a large dataset. You might want to iterate `drop()` with a limit, for example, `limit(1000)`.

To drop edges that have the label 'rated', run the Gremlin command following.

```
gremlin> g.E().hasLabel('rated').drop()
```

### Note

It can take some time to drop a large dataset. You might want to iterate `drop()` with a limit, for example `limit(1000)`.

- The DMS API operation `DescribeTableStatistics` can return inaccurate results about a given table because of the nature of Neptune graph data structures.

During migration, AWS DMS scans each source table and uses graph mapping to convert the source data into a Neptune graph. The converted data is first stored in the S3 bucket folder specified for

the target endpoint. If the source is scanned and this intermediate S3 data is generated successfully, `DescribeTableStatistics` assumes that the data was successfully loaded into the Neptune target database. But this isn't always true. To verify that the data was loaded correctly for a given table, compare `count()` return values at both ends of the migration for that table.

In the example following, AWS DMS has loaded a `customer` table from the source database, which is assigned the label '`customer`' in the target Neptune database graph. You can make sure that this label is written to the target database. To do this, compare the number of `customer` rows available from the source database with the number of '`customer`' labeled rows loaded in the Neptune target database after the task completes.

To get the number of `customer` rows available from the source database using SQL, run the following.

```
select count(*) from customer;
```

To get the number of '`customer`' labeled rows loaded into the target database graph using Gremlin, run the following.

```
gremlin> g.V().hasLabel('customer').count()
```

- Currently, if any single table fails to load, the whole task fails. Unlike in a relational database target, data in Neptune is highly connected, which makes it impossible in many cases to resume a task. If a task can't be resumed successfully because of this type of data load failure, create a new task to load the table that failed to load. Before running this new task, manually clear the partially loaded table from the Neptune target.

**Note**

You can resume a task that fails migration to a Neptune target if the failure is recoverable (for example, a network transit error).

- AWS DMS supports most standards for R2RML. However, AWS DMS doesn't support certain R2RML standards, including inverse expressions, joins, and views. A work-around for an R2RML view is to create a corresponding custom SQL view in the source database. In the migration task, use table mapping to choose the view as input. Then map the view to a table that is then consumed by R2RML to generate graph data.
- When you migrate source data with unsupported SQL data types, the resulting target data can have a loss of precision. For more information, see [Data types for Gremlin and R2RML migration to Amazon Neptune as a target \(p. 266\)](#).

## 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 tasks

An AWS Database Migration Service (AWS DMS) task is where all the work happens. You specify what tables (or views) 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, make sure that you 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 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
<b>Creating a task</b>  When you create a task, you specify the source, target, and replication instance, along with any migration settings.	<a href="#">Creating a task (p. 273)</a>
<b>Creating an ongoing replication Task</b>  You can set up a task to provide continuous replication between the source and target.	<a href="#">Creating tasks for ongoing replication using AWS DMS (p. 301)</a>
<b>Applying task settings</b>  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.	<a href="#">Specifying task settings for AWS Database Migration Service tasks (p. 278)</a>
<b>Using table mapping</b>  Table mapping specifies additional task settings for tables using several types of rules. These rules allow you to specify the data source, source schema, tables and views, data, any table and data transformations that are to occur during	Selection rules <a href="#">Selection rules and actions (p. 314)</a>  Transformation rules <a href="#">Transformation rules and actions (p. 318)</a>  Table-settings rules <a href="#">Table-settings rules and operations (p. 333)</a>

Task	Relevant documentation
the task, and settings for how these tables and columns are migrated from the source to the target.	
<p><b>Running premigration task assessments</b></p> <p>You can enable and run premigration task assessments showing issues with a supported source and target database that can cause problems during a migration. This can include issues such as unsupported data types, mismatched indexes and primary keys, and other conflicting task settings. These premigration assessments run before you run the task to identify potential issues before they occur during a migration.</p>	<a href="#">Enabling and working with premigration assessments for a task (p. 353)</a>
<p><b>Data Validation</b></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>	<a href="#">AWS DMS data validation (p. 381).</a>
<p><b>Modifying a Task</b></p> <p>When a task is stopped, you can modify the settings for the task.</p>	<a href="#">Modifying a task (p. 307)</a>
<p><b>Reloading Tables During a Task</b></p> <p>You can reload a table during a task if an error occurs during the task.</p>	<a href="#">Reloading tables during a task (p. 307)</a>
<p><b>Applying Filters</b></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>	<a href="#">Using source filters (p. 349)</a>
<p><b>Monitoring a Task</b></p> <p>There are several ways to get information on the performance of a task and the tables used by the task.</p>	<a href="#">Monitoring AWS DMS tasks (p. 362)</a>

Task	Relevant documentation
<b>Managing Task Logs</b>  You can view and delete task logs using the AWS DMS API or AWS CLI.	<a href="#">Viewing and managing AWS DMS task logs (p. 370)</a>

### Topics

- [Creating a task \(p. 273\)](#)
- [Creating tasks for ongoing replication using AWS DMS \(p. 301\)](#)
- [Modifying a task \(p. 307\)](#)
- [Reloading tables during a task \(p. 307\)](#)
- [Using table mapping to specify task settings \(p. 309\)](#)
- [Enabling and working with premigration assessments for a task \(p. 353\)](#)
- [Specifying supplemental data for task settings \(p. 361\)](#)

## Creating a task

To create an AWS DMS migration task, you do the following:

- Create a source endpoint, a target endpoint, and a replication instance before you create a migration task.
- Choose 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 Management 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 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 isn't 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 an AWS DMS task \(p. 300\)](#).
- 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 \(p. 278\)](#).
- 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 \(p. 309\)](#). Before

you specify your mapping, make sure that you review the documentation section on data type mapping for your source and your target database.

- Enable and run premigration task assessments before you run the task. For more information about premigration assessments, see [Enabling and working with premigration assessments for a task \(p. 353\)](#).
- Specify any required supplemental data for the task to migrate your data. For more information, see [Specifying supplemental data for task settings \(p. 361\)](#).

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 following procedure assumes that you have already specified replication instance information and endpoints. For more information on setting up endpoints, see [Creating source and target endpoints \(p. 63\)](#).

### To create a migration task

1. Sign in to the AWS Management Console and open the AWS DMS console at <https://console.aws.amazon.com/dms/v2/>.

If you are signed in as an AWS Identity and Access Management (IAM) user, make sure that you have the appropriate permissions to access AWS DMS. For more information on the permissions required, see [IAM permissions needed to use AWS DMS \(p. 416\)](#).

2. On the navigation pane, choose **Tasks**, and then choose **Create task**.
3. 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.

<b>Task name</b>	<input type="text" value="ProdEndpoint-TestEndpoint"/> <span style="font-size: small;">i</span>
<b>Task description</b>	<input type="text" value="e.g. migrate customer tables to RD"/> <span style="font-size: small;">i</span>
<b>Source endpoint</b>	rdsoracle-endpoint
<b>Target endpoint</b>	rdspostgres-endpoint
<b>Replication instance</b>	replinstance1-26
<b>Migration type</b>	<input type="button" value="Migrate existing data"/> <span style="font-size: small;">i</span>
<input checked="" type="checkbox"/> <b>Start task on create</b>	
 <span style="font-size: small;">» Task Settings</span>	
 <span style="font-size: small;">» Table mappings</span>	

For this option	Do this
<b>Task name</b>	Enter a name for the task.
<b>Task description</b>	Enter a description for the task.
<b>Source endpoint</b>	Shows the source endpoint to be used.
<b>Target endpoint</b>	Shows the target endpoint to be used.
<b>Replication instance</b>	Shows the replication instance to be used.
<b>Migration type</b>	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.
<b>Start task on create</b>	When this option is selected, the task begins as soon as it is created.

4. 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 that you choose. For example, when you choose **Migrate existing data**, the following options appear.

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)\*

32

Enable logging

For this option	Do this
<b>Target table preparation mode</b>	<b>Do nothing</b> – In <b>Do nothing</b> mode, AWS DMS assumes that the target tables have been precreated on the target. If the migration is a full load or full load plus CDC, make sure that the target tables are empty before starting the migration. <b>Do nothing</b> mode is appropriate for CDC-only tasks when the target tables have been backfilled from the source and ongoing replication is applied to keep the source and target in sync. To precreate tables, you can use the AWS Schema Conversion Tool (AWS SCT). For more information, see <a href="#">Installing AWS SCT</a> .

For this option	Do this
	<p><b>Drop tables on target</b> – In <b>Drop tables on target</b> mode, AWS DMS drops the target tables and recreates them before starting the migration. This approach 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, nonprimary 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 at this point. Then, 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 <b>Drop tables on target</b> mode. For example, for an Oracle target, AWS DMS can't create a schema (database user) for security reasons. In this case, you precreate 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><b>Truncate</b> – In <b>Truncate</b> mode, AWS DMS truncates all target tables before the migration starts. <b>Truncate</b> mode is appropriate for full load or full load plus CDC migrations where the target schema has been precreated before the migration starts. To precreate tables, you can use AWS SCT. For more information, see <a href="#">Installing AWS SCT</a>.</p>
<b>Include LOB columns in replication</b>	<p><b>Don't include LOB columns</b> – LOB columns are excluded from the migration.</p> <p><b>Full LOB mode</b> – Migrate complete LOBs regardless of size. AWS DMS migrates LOBs piecemece in chunks controlled by the <b>Max LOB size</b> parameter. This mode is slower than using Limited LOB mode.</p> <p><b>Limited LOB mode</b> – Truncate LOBs to the value of the <b>Max LOB size</b> parameter. This mode is faster than using Full LOB mode.</p>
<b>Max LOB size (kb)</b>	In <b>Limited LOB Mode</b> , LOB columns that exceed the setting of <b>Max LOB size</b> are truncated to the specified <b>Max LOB Size</b> value.
<b>Enable validation</b>	Enables data validation, to verify that the data is migrated accurately from the source to the target. For more information, see <a href="#">AWS DMS data validation (p. 381)</a> .
<b>Enable logging</b>	Enables logging by Amazon CloudWatch.

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

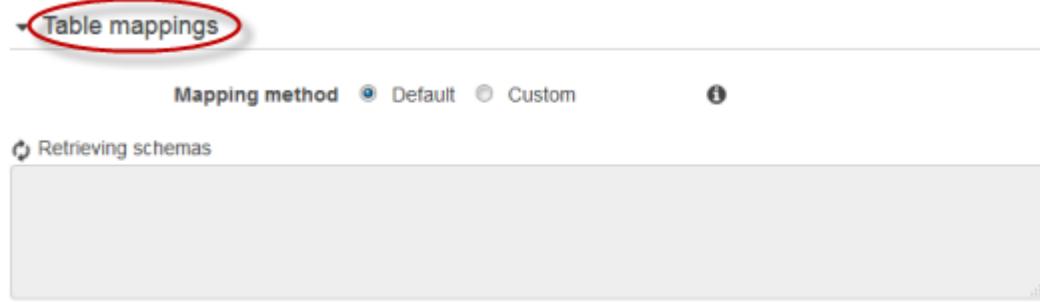
**Task Settings**

<b>Target table preparation mode</b>	<input type="radio"/> Do nothing	<small>i</small>
	<input checked="" type="radio"/> Drop tables on target	<small>i</small>
	<input type="radio"/> Truncate	<small>i</small>
<b>Stop task after full load completes</b>	<input checked="" type="radio"/> Don't stop	<small>i</small>
	<input type="radio"/> Stop Before Applying Cached Changes	<small>i</small>
	<input type="radio"/> Stop After Applying Cached Changes	<small>i</small>
<b>Include LOB columns in replication</b>	<input type="radio"/> Don't include LOB columns	<small>i</small>
	<input type="radio"/> Full LOB mode	<small>i</small>
	<input checked="" type="radio"/> Limited LOB mode	<small>i</small>
<b>Max LOB size (kb)*</b>	<input type="text" value="32"/> <span style="border: 1px solid #ccc; padding: 2px 5px;">▼</span>	<small>i</small>
<b>Enable logging</b>	<input type="checkbox"/>	<small>i</small>

For this option	Do this
<b>Target table preparation mode</b>	<p><b>Do nothing</b> – Data and metadata of the target tables aren't changed.</p> <p><b>Drop tables on target</b> – The tables are dropped and new tables are created in their place.</p> <p><b>Truncate</b> – Tables are truncated without affecting table metadata.</p>
<b>Stop task after full load completes</b>	<p><b>Don't stop</b> – Don't stop the task but immediately apply cached changes and continue on.</p> <p><b>Stop before applying cached changes</b> - 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><b>Stop after applying cached changes</b> - 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>

For this option	Do this
<b>Include LOB columns in replication</b>	<b>Don't include LOB columns</b> – LOB columns is excluded from the migration.  <b>Full LOB mode</b> – 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.  <b>Limited LOB mode</b> – Truncate LOBs to 'Max LOB Size'. This method is faster than using Full LOB Mode.
<b>Max LOB size (KB)</b>	In <b>Limited LOB Mode</b> , LOB columns that exceed the setting of <b>Max LOB Size</b> are truncated to the specified Max LOB Size.
<b>Enable validation</b>	Enables data validation, to verify that the data is migrated accurately from the source to the target. For more information, see <a href="#">AWS DMS data validation (p. 381)</a> .
<b>Enable logging</b>	Enables logging by 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 \(p. 309\)](#).



- If necessary, specify supplemental task data in the appropriate tab. For example, if your target endpoint is for an Amazon Neptune graph database, choose the **Graph mapping rules** tab. Then either browse to choose the appropriate graph mapping configuration file or enter the mapping rules directly using the editor. For more information on specifying supplemental task data, see [Specifying supplemental data for task settings \(p. 361\)](#).
- 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 \(p. 282\)](#)
- [Full-load task settings \(p. 283\)](#)
- [Logging task settings \(p. 284\)](#)
- [Control table task settings \(p. 285\)](#)
- [Stream buffer task settings \(p. 288\)](#)
- [Change processing tuning settings \(p. 288\)](#)
- [Data validation task settings \(p. 289\)](#)
- [Task settings for change processing DDL handling \(p. 291\)](#)
- [Character substitution task settings \(p. 291\)](#)
- [Before image task settings \(p. 296\)](#)
- [Error handling task settings \(p. 296\)](#)
- [Saving task settings \(p. 298\)](#)

Task settings	Relevant documentation
<b>Creating a Task Assessment Report</b>  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.	<a href="#">Enabling and working with premigration assessments for a task (p. 353)</a>
<b>Creating a Task</b>  When you create a task, you specify the source, target, and replication instance, along with any migration settings.	<a href="#">Creating a task (p. 273)</a>
<b>Creating an Ongoing Replication Task</b>  You can set up a task to provide continuous replication between the source and target.	<a href="#">Creating tasks for ongoing replication using AWS DMS (p. 301)</a>
<b>Applying Task Settings</b>  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.	<a href="#">Specifying task settings for AWS Database Migration Service tasks (p. 278)</a>
<b>Data Validation</b>  Use data validation to have AWS DMS compare the data on your target data store with the data from your source data store.	<a href="#">AWS DMS data validation (p. 381)</a>
<b>Modifying a Task</b>	<a href="#">Modifying a task (p. 307)</a>

Task settings	Relevant documentation
When a task is stopped, you can modify the settings for the task.	
<b>Reloading Tables During a Task</b>  You can reload a table during a task if an error occurs during the task.	<a href="#">Reloading tables during a task (p. 307)</a>
<b>Using Table Mapping</b>  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.	Selection Rules <a href="#">Selection rules and actions (p. 314)</a>  Transformation Rules <a href="#">Transformation rules and actions (p. 318)</a>
<b>Applying Filters</b>  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.	<a href="#">Using source filters (p. 349)</a>
<b>Monitoring a Task</b>  There are several ways to get information on the performance of a task and the tables used by the task.	<a href="#">Monitoring AWS DMS tasks (p. 362)</a>
<b>Managing Task Logs</b>  You can view and delete task logs using the AWS DMS API or AWS CLI.	<a href="#">Viewing and managing AWS DMS task logs (p. 370)</a>

A task settings JSON file can look like the following.

```
{
    "TargetMetadata": {
        "TargetSchema": "",
        "SupportLobs": true,
        "FullLobMode": false,
        "LobChunkSize": 64,
        "LimitedSizeLobMode": true,
        "LobMaxSize": 32,
        "InlineLobMaxSize": 0,
        "LoadMaxFileSize": 0,
        "ParallelLoadThreads": 0,
        "ParallelLoadBufferSize": 0,
        "ParallelLoadQueuesPerThread": 1,
        "ParallelApplyThreads": 0,
        "ParallelApplyBufferSize": 50,
        "ParallelApplyQueuesPerThread": 1,
        "BatchApplyEnabled": false,
    }
}
```

```

        "TaskRecoveryTableEnabled": false
    },
    "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
    },
    "LoopbackPreventionSettings": {
        "EnableLoopbackPrevention": true,
        "SourceSchema": "LOOP-DATA",
        "TargetSchema": "loop-data"
    },
    "ValidationSettings": {
        "EnableValidation": true,
        "FailureMaxCount": 10000,
        "HandleCollationDiff": false,
        "RecordFailureDelayLimitInMinutes": 0,
        "SkipLobColumns": false,
        "TableFailureMaxCount": 1000,
        "ThreadCount": 5,
        "ValidationOnly": false,
        "ValidationPartialLobSize": 0
    },
    "CharacterSetSettings": {
        "CharacterReplacements": [
            {
                "SourceCharacterCodePoint": 35,
                "TargetCharacterCodePoint": 52
            },
            {
                "SourceCharacterCodePoint": 37,
                "TargetCharacterCodePoint": 103
            }
        ]
    }
}

```

```

        ],
        "CharacterSetSupport": {
            "CharacterSet": "UTF16_PlatformEndian",
            "ReplaceWithCharacterCodePoint": 0
        }
    },
    "BeforeImageSettings": {
        "EnableBeforeImage": false,
        "FieldName": "",
        "ColumnFilter": "pk-only"
    },
    "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 DMS 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 DMS checks all LOBs and performs an inline transfer for the LOBs that are smaller than `InlineLobMaxSize`. AWS DMS 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.

For more information on the criteria for using these task LOB settings, see [Setting LOB support for source databases in an AWS DMS task \(p. 300\)](#). You can also control the management of LOBs for individual tables. For more information, see [Table-settings rules and operations \(p. 333\)](#).

- `LoadMaxFileSize` – An option for 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, you can use `BatchApplyEnabled` only in limited LOB mode.

For more information on using these settings for a change data capture (CDC) load, see [Change processing tuning settings \(p. 288\)](#).

- `ParallelLoadThreads` – Specifies the number of threads that AWS DMS uses to load each table into the target database. This parameter has maximum values for non-RDBMS targets. The maximum value for a DynamoDB target is 200. The maximum value for an Amazon Kinesis Data Streams, Apache Kafka, or Amazon Elasticsearch Service target is 32. You can ask to have this maximum limit increased. For information on the settings for parallel load of individual tables, see [Table-settings rules and operations \(p. 333\)](#).
- `ParallelLoadBufferSize` – Specifies the maximum number of records to store in the buffer that the parallel load threads use to load data to the target. The default value is 50. The maximum value is 1,000. This setting is currently only valid when DynamoDB, Kinesis, Apache Kafka, or Elasticsearch is the target. Use this parameter with `ParallelLoadThreads`. `ParallelLoadBufferSize` is valid only when there is more than one thread. For information on the settings for parallel load of individual tables, see [Table-settings rules and operations \(p. 333\)](#).
- `ParallelLoadQueuesPerThread` – Specifies the number of queues that each concurrent thread accesses to take data records out of queues and generate a batch load for the target. The default is 1. This setting is currently only valid when Kinesis or Apache Kafka is the target.
- `ParallelApplyThreads` – Specifies the number of concurrent threads that AWS DMS uses during a CDC load to push data records to a Kinesis, Apache Kafka, or Elasticsearch target endpoint. The default is zero (0).
- `ParallelApplyBufferSize` – Specifies the maximum number of records to store in each buffer queue for concurrent threads to push to a Kinesis, Apache Kafka, or Elasticsearch target endpoint during a CDC load. The default value is 50. Use this option when `ParallelApplyThreads` specifies more than one thread.
- `ParallelApplyQueuesPerThread` – Specifies the number of queues that each thread accesses to take data records out of queues and generate a batch load for a Kinesis, Apache Kafka, or Elasticsearch endpoint during CDC. The default value is 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 aren't 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 a full load completes, set the `CreatePkAfterFullLoad` option to `true`.
- For full-load and CDC-enabled tasks, you can set the following options for `Stop task after full load completes`:
  - `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.
- To indicate the maximum number of tables to load in parallel, set the `MaxFullLoadSubTasks` option. The default is 8; the maximum value is 49.
- You can set the number of seconds that AWS DMS waits for transactions to close before beginning a full-load operation. To do so, 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. The default value is 10000, and the maximum value is 50000.

## Logging task settings

Logging uses Amazon CloudWatch to log information during the migration process. Using logging task settings, you can specify which component activities are logged and what amount of information is written to the log. Logging task settings are written to a JSON file.

You can enable CloudWatch logging in several ways. You can select the `EnableLogging` option on the AWS Management Console when you create a migration task. Or, you can 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.

CloudWatch integrates with AWS Identity and Access Management (IAM), and you can specify which CloudWatch actions a user in your AWS account can perform. For more information on working with IAM in CloudWatch, see [Identity and access management for amazon CloudWatch](#) and [Logging Amazon CloudWatch API calls in the Amazon CloudWatch User Guide](#).

To delete the task logs, you can set `DeleteTaskLogs` to `true` in the JSON of the logging section of the task settings.

You can specify logging for the following actions:

- `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 one of the preceding, you can then specify the amount of information that is logged, as shown in the following list.

The levels of severity are in order from lowest to highest level of information. The higher levels always include information from the lower levels.

- **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.

The following JSON example shows task settings for logging all actions and levels of severity.

```
...
  "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"
      }
    ]
  },
...
}
```

## Control table task settings

Control tables provide information about an AWS DMS task. They also provide 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 by choosing **Advanced Settings** on the **Create task** page in the AWS DMS console. The Apply Exceptions table (`dmslogs.awsdms_apply_exceptions`) is always created.

For Full load + CDC (Migrate existing data and replicate ongoing changes) and CDC only (Replicate data changes only) tasks, you can also 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. Also, it indicates if the task is in the full load phase 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 table (`dmslogs.awsdms_apply_exceptions`) 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 table (`dmslogs.awsdms_history`) 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	varchar	One of the following values: <ul style="list-style-type: none"> <li>• FULL LOAD</li> <li>• CHANGE PROCESSING (CDC)</li> </ul> If the task is running both full load and CDC, two history records are written to the time slot.
TIMESLOT	timestamp	The ending timestamp of the time slot.
TIMESLOT_DURATION	int	The duration of the time slot, in minutes.
TIMESLOT_LATENCY	int	The target latency at the end of the time slot, in seconds. This value only applies 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 table (`dmslogs.awsdms_status`) 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	varchar	<p>One of the following values:</p> <ul style="list-style-type: none"> <li>• FULL LOAD</li> <li>• CHANGE PROCESSING (CDC)</li> </ul> <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 weren't applied to the target.
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 isn't 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 isn't 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:

- `HistoryTimeslotInMinutes` – Use this option to indicate the length of each time slot in the Replication History table. The default is 5 minutes.
- `ControlSchema` – Use this option to indicate the database schema name for the control tables for the AWS DMS target. If you don't enter any information for this option, then the tables are copied to the default location in the database as listed following:
  - PostgreSQL, Public
  - Oracle, the target schema
  - Microsoft SQL Server, dbo in the target database
  - MySQL, awsdms\_control
  - MariaDB, awsdms\_control
  - Amazon Redshift, Public
  - DynamoDB, created as individual tables in the database

## Stream buffer task settings

You can set stream buffer settings using the AWS CLI, including 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:  $[\text{Max LOB size (or LOB chunk size)}] * [\text{number of LOB columns}] * [\text{number of stream buffers}] * [\text{number of tables loading in parallel per task}(\text{MaxFullLoadSubTasks})] * 3$
- `CtrlStreamBufferSizeInMB` – Use this option to set the size of the control stream buffer. The 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 \(p. 282\)](#).

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.

The following setting applies when the target metadata parameter `BatchApplyEnabled` is set to either `true` or `false`.

- `HandleSourceTableAltered` – Set this option to `true` to alter the target table when the source table is altered.

The following settings apply only when `BatchApplyEnabled` is set to `false`.

- `LoopbackPreventionSettings` – These settings provide loopback prevention for each ongoing replication task in any pair of tasks involved in bidirectional replication. *Loopback prevention* prevents identical changes from being applied in both directions of the bidirectional replication, which can corrupt data. For more information on bidirectional replication, see [Performing bidirectional replication \(p. 304\)](#).

AWS DMS attempts to keep transaction data in memory until the transaction is fully committed to the source, the target, or both. However, transactions that are larger than the allocated memory or that aren't 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 validation for a task, AWS DMS begins comparing the source and target data immediately after a full load is performed for a table. For more information about task data validation, its requirements, the scope of its database support, and the metrics it reports, see [AWS DMS data validation \(p. 381\)](#).

The data validation settings and their values include the following:

- `EnableValidation` – Enables data validation when set to true. Otherwise, validation is disabled for the task. The default value is false.
- `FailureMaxCount` – Specifies the maximum number of records that can fail validation before validation is suspended for the task. The default value is 10,000. If you want the validation to continue regardless of the number of records that fail validation, set this value higher than the number of records in the source.
- `HandleCollationDiff` – When this option is set to `true`, the validation accounts for column collation differences in PostgreSQL endpoints when identifying source and target records to compare. Otherwise, any such differences in column collation are ignored for validation. In PostgreSQL endpoints, column collations can dictate the order of rows, which is important for data validation. Setting `HandleCollationDiff` to `true` resolves those collation differences automatically and prevents false positives in data validation. The default value is `false`.
- `RecordFailureDelayLimitInMinutes` – Specifies the delay before reporting any validation failure details. Normally, AWS DMS uses the task latency to recognize actual delay for changes to make it to the target in order to prevent false positives. This setting overrides the actual delay value and enables you to set a higher delay before reporting any validation metrics. The default value is 0.
- `SkipLobColumns` – When this option is set to `true`, AWS DMS skips data validation for all the LOB columns in the tables part of the task validation. The default value is `false`.
- `TableFailureMaxCount` – Specifies the maximum number of tables that can fail validation before validation is suspended for the task. The default value is 1,000. If you want the validation to continue regardless of the number of tables that fail validation, set this value higher than the number of tables in the source.
- `ThreadCount` – Specifies the number of execution threads that AWS DMS uses during validation. Each thread selects not-yet-validated data from the source and target to compare and validate. The default value is 5. If you set `ThreadCount` to a higher number, AWS DMS can complete the validation faster. However, AWS DMS then runs more simultaneous queries, consuming more resources on the source and the target.
- `ValidationOnly` – When this option is set to `true`, the task previews data validation without performing any migration or replication of data. The default value is `false`. But `ValidationOnly` set to `true` is particularly useful for the following use cases:
  - After data was migrated in a separate full load task without validation enabled, you can validate data without performing another full load. Just create and run another task with `ValidationOnly` set to `true`.
  - During a full load task, validation only begins once the initial load is complete. But, by creating a separate `ValidationOnly` task, you can see early validation results and resolve any failures prior to actually moving *all* of the data. This approach can be more efficient than waiting to resolve failures after all the source data has been migrated to the target.

To be able to set this option, set the task **Migration type** to **Replicate data changes only** in the AWS DMS console. Alternatively, in the AWS DMS API set the migration type to `cdc`.

- `ValidationPartialLobSize` – Specifies if you want to do partial validation for LOB columns instead of validating all of the data stored in the column. This is something you might find useful when you are migrating just part of the LOB data and not the whole LOB data set. The value is in KB units. Default value is 0, which means AWS DMS validates all the LOB column data. For example, `"ValidationPartialLobSize": 32` means that AWS DMS only validates the first 32KB of the column data in both the source and target.

For example, the following JSON enables data validation with twice the default number of threads. It also accounts for differences in record order caused by column collation differences in PostgreSQL endpoints. Also, it provides a validation reporting delay to account for additional time to process any validation failures.

```
"ValidationSettings": {  
    "EnableValidation": true,  
    "ThreadCount": 10,  
    "HandleCollationDiff": true,  
    "RecordFailureDelayLimitInMinutes": 30  
}
```

**Note**

For an Oracle endpoint, AWS DMS uses DBMS\_CRYPTO to validate BLOBS. If your Oracle endpoint uses BLOBS, grant the execute permission for DBMS\_CRYPTO to the user account that is used to access the Oracle endpoint. 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 to handle change processing DDL 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.

Following is an example of how task settings that handle change processing DDL appear in a task setting JSON file:

```
"ChangeProcessingDdlHandlingPolicy": {  
    "HandleSourceTableDropped": true,  
    "HandleSourceTableTruncated": true,  
    "HandleSourceTableAltered": true  
},
```

**Note**

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

## Character substitution task settings

You can specify that your replication task perform character substitutions on the target database for all source database columns with the AWS DMS STRING or WSTRING data type. You can configure character substitution for any task with endpoints from the following source and target databases:

- Source databases:
  - Oracle
  - Microsoft SQL Server
  - MySQL

- PostgreSQL
- SAP Adaptive Server Enterprise (ASE)
- IBM Db2 LUW
- Target databases:
  - Oracle
  - Microsoft SQL Server
  - MySQL
  - PostgreSQL
  - SAP Adaptive Server Enterprise (ASE)
  - Amazon Redshift

You can specify character substitutions using the `CharacterSetSettings` parameter in your task settings. These character substitutions occur for characters specified using the Unicode code point value in hexadecimal notation. You can implement the substitutions in two phases, in the following order if both are specified:

1. **Individual character replacement** – AWS DMS can replace the values of selected characters on the source with specified replacement values of corresponding characters on the target. Use the `CharacterReplacements` array in `CharacterSetSettings` to select all source characters having the Unicode code points you specify. Use this array also to specify the replacement code points for the corresponding characters on the target.

To select all characters on the source that have a given code point, set an instance of `SourceCharacterCodePoint` in the `CharacterReplacements` array to that code point. Then specify the replacement code point for all equivalent target characters by setting the corresponding instance of `TargetCharacterCodePoint` in this array. To delete target characters instead of replacing them, set the appropriate instances of `TargetCharacterCodePoint` to zero (0). You can replace or delete as many different values of target characters as you want by specifying additional pairs of `SourceCharacterCodePoint` and `TargetCharacterCodePoint` settings in the `CharacterReplacements` array. If you specify the same value for multiple instances of `SourceCharacterCodePoint`, the value of the last corresponding setting of `TargetCharacterCodePoint` applies on the target.

For example, suppose that you specify the following values for `CharacterReplacements`.

```
"CharacterSetSettings": {  
    "CharacterReplacements": [ {  
        "SourceCharacterCodePoint": 62,  
        "TargetCharacterCodePoint": 61  
    }, {  
        "SourceCharacterCodePoint": 42,  
        "TargetCharacterCodePoint": 41  
    }  
}
```

In this example, AWS DMS replaces all characters with the source code point hex value 62 on the target by characters with the code point value 61. Also, AWS DMS replaces all characters with the source code point 42 on the target by characters with the code point value 41. In other words, AWS DMS replaces all instances of the letter 'b' on the target by the letter 'a'. Similarly, AWS DMS replaces all instances of the letter 'B' on the target by the letter 'A'.

2. **Character set validation and replacement** – After any individual character replacements complete, AWS DMS can make sure that all target characters have valid Unicode code points in the single character set that you specify. You use `CharacterSetSupport` in `CharacterSetSettings` to configure this target character verification and modification. To specify the verification character

set, set `CharacterSet` in `CharacterSetSupport` to the character set's string value. (The possible values for `CharacterSet` follow.) You can have AWS DMS modify the invalid target characters in one of the following ways:

- Specify a single replacement Unicode code point for all invalid target characters, regardless of their current code point. To configure this replacement code point, set `ReplaceWithCharacterCodePoint` in `CharacterSetSupport` to the specified value.
- Configure the deletion of all invalid target characters by setting `ReplaceWithCharacterCodePoint` to zero (0).

For example, suppose that you specify the following values for `CharacterSetSupport`.

```
"CharacterSetSettings": {
    "CharacterSetSupport": {
        "CharacterSet": "UTF16_PlatformEndian",
        "ReplaceWithCharacterCodePoint": 0
    }
}
```

In this example, AWS DMS deletes any characters found on the target that are invalid in the "UTF16\_PlatformEndian" character set. So, any characters specified with the hex value 2FB6 are deleted. This value is invalid because this is a 4-byte Unicode code point and UTF16 character sets accept only characters with 2-byte code points.

#### Note

The replication task completes all of the specified character substitutions before starting any global or table-level transformations that you specify through table mapping. For more information on table mapping, see [Using table mapping to specify task settings \(p. 309\)](#).

The values that AWS DMS supports for `CharacterSet` appear in the table following.

UTF-8	ibm-860_P100-1995	ibm-280_P100-1995
UTF-16	ibm-861_P100-1995	ibm-284_P100-1995
UTF-16BE	ibm-862_P100-1995	ibm-285_P100-1995
UTF-16LE	ibm-863_P100-1995	ibm-290_P100-1995
UTF-32	ibm-864_X110-1999	ibm-297_P100-1995
UTF-32BE	ibm-865_P100-1995	ibm-420_X120-1999
UTF-32LE	ibm-866_P100-1995	ibm-424_P100-1995
UTF16_PlatformEndian	ibm-867_P100-1998	ibm-500_P100-1995
UTF16_OppositeEndian	ibm-868_P100-1995	ibm-803_P100-1999
UTF32_PlatformEndian	ibm-869_P100-1995	ibm-838_P100-1995
UTF32_OppositeEndian	ibm-878_P100-1996	ibm-870_P100-1995
UTF-16BE,version=1	ibm-901_P100-1999	ibm-871_P100-1995
UTF-16LE,version=1	ibm-902_P100-1999	ibm-875_P100-1995
UTF-16,version=1	ibm-922_P100-1999	ibm-918_P100-1995
UTF-16,version=2	ibm-1168_P100-2002	ibm-930_P120-1999

UTF-7	ibm-4909_P100-1999	ibm-933_P110-1995
IMAP-mailbox-name	ibm-5346_P100-1998	ibm-935_P110-1999
SCSU	ibm-5347_P100-1998	ibm-937_P110-1999
BOCU-1	ibm-5348_P100-1997	ibm-939_P120-1999
CESU-8	ibm-5349_P100-1998	ibm-1025_P100-1995
ISO-8859-1	ibm-5350_P100-1998	ibm-1026_P100-1995
US-ASCII	ibm-9447_P100-2002	ibm-1047_P100-1995
gb18030	ibm-9448_X100-2005	ibm-1097_P100-1995
ibm-912_P100-1995	ibm-9449_P100-2002	ibm-1112_P100-1995
ibm-913_P100-2000	ibm-5354_P100-1998	ibm-1122_P100-1999
ibm-914_P100-1995	ibm-1250_P100-1995	ibm-1123_P100-1995
ibm-915_P100-1995	ibm-1251_P100-1995	ibm-1130_P100-1997
ibm-1089_P100-1995	ibm-1252_P100-2000	ibm-1132_P100-1998
ibm-9005_X110-2007	ibm-1253_P100-1995	ibm-1137_P100-1999
ibm-813_P100-1995	ibm-1254_P100-1995	ibm-4517_P100-2005
ibm-5012_P100-1999	ibm-1255_P100-1995	ibm-1140_P100-1997
ibm-916_P100-1995	ibm-5351_P100-1998	ibm-1141_P100-1997
ibm-920_P100-1995	ibm-1256_P110-1997	ibm-1142_P100-1997
iso-8859_10-1998	ibm-5352_P100-1998	ibm-1143_P100-1997
iso-8859_11-2001	ibm-1257_P100-1995	ibm-1144_P100-1997
ibm-921_P100-1995	ibm-5353_P100-1998	ibm-1145_P100-1997
iso-8859_14-1998	ibm-1258_P100-1997	ibm-1146_P100-1997
ibm-923_P100-1998	macos-0_2-10.2	ibm-1147_P100-1997
ibm-942_P12A-1999	macos-6_2-10.4	ibm-1148_P100-1997
ibm-943_P15A-2003	macos-7_3-10.2	ibm-1149_P100-1997
ibm-943_P130-1999	macos-29-10.2	ibm-1153_P100-1999
ibm-33722_P12A_P12A-2009_U2macos-35-10.2		ibm-1154_P100-1999
ibm-33722_P120-1999	ibm-1051_P100-1995	ibm-1155_P100-1999
ibm-954_P101-2007	ibm-1276_P100-1995	ibm-1156_P100-1999
euc-jp-2007	ibm-1006_P100-1995	ibm-1157_P100-1999
ibm-1373_P100-2002	ibm-1098_P100-1995	ibm-1158_P100-1999
windows-950-2000	ibm-1124_P100-1996	ibm-1160_P100-1999

ibm-950_P110-1999	ibm-1125_P100-1997	ibm-1164_P100-1999
ibm-1375_P100-2008	ibm-1129_P100-1997	ibm-1364_P110-2007
ibm-5471_P100-2006	ibm-1131_P100-1997	ibm-1371_P100-1999
ibm-1386_P100-2001	ibm-1133_P100-1997	ibm-1388_P103-2001
windows-936-2000	ISO_2022,locale=ja,version=ibm-1390_P110-2003	
ibm-1383_P110-1999	ISO_2022,locale=ja,version=ibm-1399_P110-2003	
ibm-5478_P100-1995	ISO_2022,locale=ja,version=ibm-5123_P100-1999	
euc-tw-2014	ISO_2022,locale=ja,version=ibm-8482_P100-1999	
ibm-964_P110-1999	ISO_2022,locale=ja,version=ibm-16684_P110-2003	
ibm-949_P110-1999	ISO_2022,locale=ko,version=ibm-4899_P100-1998	
ibm-949_P11A-1999	ISO_2022,locale=ko,version=ibm-4971_P100-1999	
ibm-970_P110_P110-2006_U2	ISO_2022,locale=zh,version=ibm-9067_X100-2005	
ibm-971_P100-1995	ISO_2022,locale=zh,version=ibm-12712_P100-1998	
ibm-1363_P11B-1998	ISO_2022,locale=zh,version=ibm-16804_X110-1999	
ibm-1363_P110-1997	HZ	ibm-37_P100-1995,swaplfnl
windows-949-2000	x11-compound-text	ibm-1047_P100-1995,swaplfnl
windows-874-2000	ISCII,version=0	ibm-1140_P100-1997,swaplfnl
ibm-874_P100-1995	ISCII,version=1	ibm-1141_P100-1997,swaplfnl
ibm-1162_P100-1999	ISCII,version=2	ibm-1142_P100-1997,swaplfnl
ibm-437_P100-1995	ISCII,version=3	ibm-1143_P100-1997,swaplfnl
ibm-720_P100-1997	ISCII,version=4	ibm-1144_P100-1997,swaplfnl
ibm-737_P100-1997	ISCII,version=5	ibm-1145_P100-1997,swaplfnl
ibm-775_P100-1996	ISCII,version=6	ibm-1146_P100-1997,swaplfnl
ibm-850_P100-1995	ISCII,version=7	ibm-1147_P100-1997,swaplfnl
ibm-851_P100-1995	ISCII,version=8	ibm-1148_P100-1997,swaplfnl
ibm-852_P100-1995	LMBCS-1	ibm-1149_P100-1997,swaplfnl
ibm-855_P100-1995	ibm-37_P100-1995	ibm-1153_P100-1999,swaplfnl
ibm-856_P100-1995	ibm-273_P100-1995	ibm-12712_P100-1998,swaplfnl
ibm-857_P100-1995	ibm-277_P100-1995	ibm-16804_X110-1999,swaplfnl
ibm-858_P100-1997	ibm-278_P100-1995	ebcdic-xml-us

## Before image task settings

When writing CDC updates to a data-streaming target like Kinesis or Kafka, you can view a source database row's original values before change by an update. To make this possible, AWS DMS populates a *before image* of update events based on data supplied by the source database engine.

`BeforeImageSettings` – Adds a new JSON attribute to every update operation with values collected from the source database system.

### Note

Apply `BeforeImageSettings` to full load plus CDC tasks (which migrate existing data and replicate ongoing changes), or to CDC only tasks (which replicate data changes only). Don't apply `BeforeImageSettings` to tasks that are full load only.

- `EnableBeforeImage` – Enables before imaging when set to `true`. The default is `false`.
- `FieldName` – Assigns a name to the new JSON attribute. When `EnableBeforeImage` is `true`, `FieldName` is required and can't be empty.
- `ColumnFilter` – Specifies a column to add by using before imaging. To add only columns that are part of the table's primary keys, use the default value, `pk-only`. To add only columns that are not of LOB type, use `non-lob`. To add any column that has a before image value, use `all`.

For example:

```
"BeforeImageSettings": {  
    "EnableBeforeImage": true,  
    "FieldName": "before-image",  
    "ColumnFilter": "pk-only"  
}
```

For information on before image settings for Kinesis, including additional table mapping settings, see [Using a before image to view original values of CDC rows for a Kinesis data stream as a target \(p. 224\)](#).

For information on before image settings for Kafka, including additional table mapping settings, see [Using a before image to view original values of CDC rows for Apache Kafka as a target \(p. 235\)](#).

## 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 isn't 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 isn't 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 isn't 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 DataErrorEscalationPolicy. 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 isn't 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 isn't 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 isn't 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 isn't 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 isn't 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 isn't replicated.
  - STOP\_TASK – The task stops and manual intervention is required.
- ApplyErrorEscalationCount – This option 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 table data is handled according to the policy set in the ApplyErrorEscalationPolicy parameter. The default is 0.
- ApplyErrorFailOnTruncationDdl – Set this option to true to cause the task to fail when a truncation is performed on any of the tracked tables during CDC. 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 option 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 it to true to cause a task to fail when a transaction is open for more time than the specified timeout and can 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 in most cases AWS DMS starts CDC, ignoring that transaction. If this option is set to true, the task fails.

- FullLoadIgnoreConflicts – Set this option to true to have AWS DMS ignore "zero rows affected" and "duplicates" errors when applying cached events. If set to false, AWS DMS reports all errors instead of ignoring them. The default is true.

## 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"  
    }  
}
```

```
        "ApplyErrorUpdatePolicy": "LOG_ERROR",
        "ApplyErrorEscalationPolicy": "LOG_ERROR",
        "ApplyErrorEscalationCount": 0,
        "FullLoadIgnoreConflicts": true
    }
}
```

## Setting LOB support for source databases in an 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 are 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.

For information on the task settings to specify these options, see [Target metadata task settings \(p. 282\)](#)

## 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 (full-load) 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.

**Note**

You can migrate views using full-load tasks only. If your task is either a CDC-only task or a full-load task that starts CDC after it completes, the migration includes only tables from the source. Using a full-load-only task, you can migrate views or a combination of tables and views. For more information, see [Specifying table selection and transformations rules using JSON \(p. 314\)](#).

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 set up 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 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
  - PostgreSQL
  - Oracle
  - MySQL

## Determining a CDC native start point

A *CDC native start point* is a point in the database engine's log that defines a time where you can begin CDC. 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 supported source engines:

### 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 \(p. 104\)](#). AWS DMS creates the publication automatically when you use CDC without using a CDC native start point.

### PostgreSQL

You can use a CDC recovery checkpoint for your PostgreSQL source database. This checkpoint value is generated at various points as an ongoing replication task runs for your source database (the

parent task). For more information on checkpoints in general, see [Using a checkpoint as a CDC start point \(p. 304\)](#).

To identify the checkpoint to use as your native start point, use your database pg\_replication\_slots view or your parent task's overview details from the AWS Management Console.

### To find the overview details for your parent task on the console

1. Sign in to the AWS Management Console and open the AWS DMS console at <https://console.aws.amazon.com/dms/v2/>.

If you are signed in as an AWS Identity and Access Management (IAM) user, make sure that you have the appropriate permissions to access AWS DMS. For more information on the permissions required, see [IAM permissions needed to use AWS DMS \(p. 416\)](#).

2. On the navigation pane, choose **Database migration tasks**.
3. Choose your parent task from the list on the **Database migration tasks** page. Doing this opens your parent task page, showing the overview details.
4. Find the checkpoint value under **Change data capture (CDC)**, **Change data capture (CDC) start position**, and **Change data capture (CDC) recovery checkpoint**.

The value appears similar to the following.

```
checkpoint:V1#1#000004AF/B00000D0#0#0###0#0
```

Here, the 4AF/B00000D0 component is what you need to specify this native CDC start point. Set the DMS API CdcStartPosition parameter to this value when you create the CDC task to begin replication at this start point for your PostgreSQL source. For information on using the AWS CLI to create this CDC task, see [Using CDC with an RDS for PostgreSQL DB instance \(p. 119\)](#).

### 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.

To get the current SCN in an Oracle database, you can run the command following:

```
SELECT CURRENT_SCN FROM V$DATABASE
```

However, note that if you use the current SCN to start a CDC task, you will miss and fail to migrate the results of any open transactions (transactions that were started earlier but not committed yet). You can identify the SCN and timestamp to start a CDC task at a point that includes all open transactions. For more information, see [Transactions](#) in the Oracle online documentation.

### 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 needed. The task automatically stops based on the commit or server time that you set. Or, 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.

## Performing bidirectional replication

You can use AWS DMS tasks to perform bidirectional replication between two systems. In *bidirectional replication*, you replicate data from the same table (or set of tables) between two systems in both directions.

For example, you can copy an `EMPLOYEE` table from database A to database B and replicate changes to the table from database A to database B. You can also replicate changes to the `EMPLOYEE` table from database B back to A. Thus, you're performing bidirectional replication.

### Note

AWS DMS bidirectional replication isn't intended as a full multi-master solution including a primary node, conflict resolution, and so on.

Use bidirectional replication for situations where data on different nodes is operationally segregated. In other words, suppose that you have a data element changed by an application operating on node A,

and that node A performs bidirectional replication with node B. That data element on node A is never changed by any application operating on node B.

AWS DMS versions 3.3.1 and higher support bidirectional replication on these database engines:

- Oracle
- SQL Server
- MySQL
- PostgreSQL
- Amazon Aurora with MySQL compatibility
- Aurora with PostgreSQL compatibility

## Creating bidirectional replication tasks

To enable AWS DMS bidirectional replication, configure source and target endpoints for both databases (A and B). For example, configure a source endpoint for database A, a source endpoint for database B, a target endpoint for database A, and a target endpoint for database B.

Then create two tasks: one task for source A to move data to target B, and another task for source B to move data to target A. Also, make sure that each task is configured with loopback prevention. Doing this prevents identical changes from being applied to the targets of both tasks, thus corrupting the data for at least one of them. For more information, see [Preventing loopback \(p. 305\)](#).

For the easiest approach, start with identical datasets on both database A and database B. Then create two CDC only tasks, one task to replicate data from A to B, and another task to replicate data from B to A.

To use AWS DMS to instantiate a new dataset (database) on node B from node A, do the following:

1. Use a full load and CDC task to move data from database A to B. Make sure that no applications are modifying data on database B during this time.
2. When the full load is complete and before applications are allowed to modify data on database B, note the time or CDC start position of database B. For instructions, see [Performing replication starting from a CDC start point \(p. 302\)](#).
3. Create a CDC only task that moves data from database B back to A using this start time or CDC start position.

### Note

Only one task in a bidirectional pair can be full load and CDC.

## Preventing loopback

To show preventing loopback, suppose that in a task T1 AWS DMS reads change logs from source database A and applies the changes to target database B.

Next, a second task, T2, reads change logs from source database B and applies them back to target database A. Before T2 does this, DMS must make sure that the same changes made to target database B from source database A aren't made to source database A. In other words, DMS must make sure that these changes aren't echoed (looped) back to target database A. Otherwise, the data in database A can be corrupted.

To prevent loopback of changes, add the following task settings to each bidirectional replication task. Doing this makes sure that loopback data corruption doesn't occur in either direction.

{

```
. . .

"LoopbackPreventionSettings": {
    "EnableLoopbackPrevention": Boolean,
    "SourceSchema": String,
    "TargetSchema": String
},
. . .
}
```

The `LoopbackPreventionSettings` task settings determine if a transaction is new or an echo from the opposite replication task. When AWS DMS applies a transaction to a target database, it updates a DMS table (`awsdms_loopback_prevention`) with an indication of the change. Before applying each transaction to a target, DMS ignores any transaction that includes a reference to this `awsdms_loopback_prevention` table. Therefore, it doesn't apply the change.

Include these task settings with each replication task in a bidirectional pair. These settings enable loopback prevention. They also specify the schema for each source and target database in the task that includes the `awsdms_loopback_prevention` table for each endpoint.

To enable each task to identify such an echo and discard it, set `EnableLoopbackPrevention` to `true`. To specify a schema at the source that includes `awsdms_loopback_prevention`, set `SourceSchema` to the name for that schema in the source database. To specify a schema at the target that includes the same table, set `TargetSchema` to the name for that schema in the target database.

In the example following, the `SourceSchema` and `TargetSchema` settings for a replication task T1 and its opposite replication task T2 are specified with opposite settings.

Settings for task T1 are as follows.

```
{
. . .

"LoopbackPreventionSettings": {
    "EnableLoopbackPrevention": true,
    "SourceSchema": "LOOP-DATA",
    "TargetSchema": "loop-data"
},
. . .
}
```

Settings for opposite task T2 are as follows.

```
{
. . .

"LoopbackPreventionSettings": {
    "EnableLoopbackPrevention": true,
    "SourceSchema": "loop-data",
    "TargetSchema": "LOOP-DATA"
},
. . .
}
```

#### Note

When using the AWS CLI, use only the `create-replication-task` or `modify-replication-task` commands to configure `LoopbackPreventionSettings` in your bidirectional replications tasks.

## Limitations of bidirectional replication

Bidirectional replication for AWS DMS has the following limitations:

- Loopback prevention tracks only data manipulation language (DML) statements. AWS DMS doesn't support preventing data definition language (DDL) loopback. To do this, configure one of the tasks in a bidirectional pair to filter out DDL statements.
- Tasks that use loopback prevention don't support committing changes in batches. To configure a task with loopback prevention, make sure to set `BatchApplyEnabled` to `false`.
- DMS bidirectional replication doesn't include conflict detection or resolution. To detect data inconsistencies, use data validation on both tasks.

## Modifying a task

You can modify a task if you need to change the task settings, table mapping, or other settings. You can also enable and run premigration assessments before running the modified task. You can 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 can't modify the source or target endpoint of a task.
- You can't change the migration type of a task.
- Tasks 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 such as `ALTER TABLE ADD COLUMN` or `DROP COLUMN` that are 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 open the AWS DMS console at <https://console.aws.amazon.com/dms/v2/>.

If you are signed in as an AWS Identity and Access Management (IAM) user, make sure that you have the appropriate permissions to access AWS DMS. For more information on the permissions required, see [IAM permissions needed to use AWS DMS \(p. 416\)](#).

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 Management Console. At the top, there are several tabs: 'Create task' (highlighted in blue), 'Assess', 'Modify', 'Start/Resume', and 'Stop'. Below these is a search bar labeled 'Filter: Q Filter'. A table follows, with columns for 'ID', 'Status', and 'Source'. One row is visible: 'move-data' (Status: Running, Source: from-mysql-sou...). The main content area is titled 'move-data'. It contains tabs for 'Overview', 'Task monitoring', 'Table statistics' (which is highlighted with a red border), 'Logs', and 'A...'. Below this is another search bar. A large red oval highlights the 'Reload table data' button. A table below shows table statistics: 'employees' (Schema) has tables 'departments' (Load State: Table completed, Inserts: 0, Deletes: 0), 'dept\_emp' (Load State: Table completed, Inserts: 0, Deletes: 0), and 'dept\_manager' (Load State: Table completed, Inserts: 0, Deletes: 0).

Schema	Table	Load State	Inserts	Deletes
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.

When working with table mapping, you can use filters to specify data that you want replicated from table columns. In addition, you can use transformations to modify selected schemas, tables, or views before they are written to the target database.

### Topics

- [Specifying table selection and transformations rules from the console \(p. 309\)](#)
- [Specifying table selection and transformations rules using JSON \(p. 314\)](#)
- [Selection rules and actions \(p. 314\)](#)
- [Transformation rules and actions \(p. 318\)](#)
- [Using transformation rule expressions to define column content \(p. 331\)](#)
- [Table-settings rules and operations \(p. 333\)](#)
- [Using source filters \(p. 349\)](#)

### Note

When working with table mapping for a MongoDB source endpoint, you can use filters to specify data that you want replicated, and specify a database name in place of the `schema_name`. Or, you can use the default "%".

## Specifying table selection and transformations rules 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.

### Note

AWS DMS doesn't support more than one transformation rule per schema level or per table level. However, AWS DMS does support more than one transformation rule per column level.

The following procedure shows how to set up selection rules, based on a table called `Customers` in a schema called `EntertainmentAgencySample`.

### To specify a table selection, filter criteria, and transformations using the console

1. Sign in to the AWS Management Console and open the AWS DMS console at <https://console.aws.amazon.com/dms/v2/>.

If you are signed in as an AWS Identity and Access Management (IAM) user, make sure that you have the appropriate permissions to access AWS DMS. For more information on the permissions required, see [IAM permissions needed to use AWS DMS \(p. 416\)](#).

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.

The screenshot shows the 'Create task' wizard. At the top, there are four input fields: 'Task name\*' (sg-selecttransfrm), 'Replication instance\*' (repl-inst-or2pg - vpc-fd...), 'Source endpoint\*' (sg-mysql2pg-source2), and 'Target endpoint\*' (postgres-target). Below these is a 'Migration type\*' dropdown set to 'Migrate existing data'. Underneath the form, there is a checked checkbox for 'Start task on create'. A horizontal line separates this from the 'Task Settings' section, which contains a 'Table mappings' dropdown. Another horizontal line separates this from the 'Table mappings' section, which has two tabs: 'Guided' (selected) and 'JSON'.

Task name\* sg-selecttransfrm

Replication instance\* repl-inst-or2pg - vpc-fd...

Source endpoint\* sg-mysql2pg-source2

Target endpoint\* postgres-target

Migration type\* Migrate existing data

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 EntertainmentAgencyExample

Table name is like %

Use % as a wildcard.

Action **Include** ?

Filter ?

Add column filter

Add selection rule

The screenshot shows the 'Table mappings' configuration screen. The 'Guided' tab is highlighted with a red circle. The 'Selection rules' section contains a note about requiring at least one selection rule with an 'Include' action. It includes fields for filtering by schema name ('EntertainmentAgencyExample') and table name ('%'), and an 'Action' dropdown set to 'Include'. Below this is a 'Filter' section with a 'Add column filter' link. At the bottom right is a prominent blue 'Add selection rule' button.

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** ⓘ

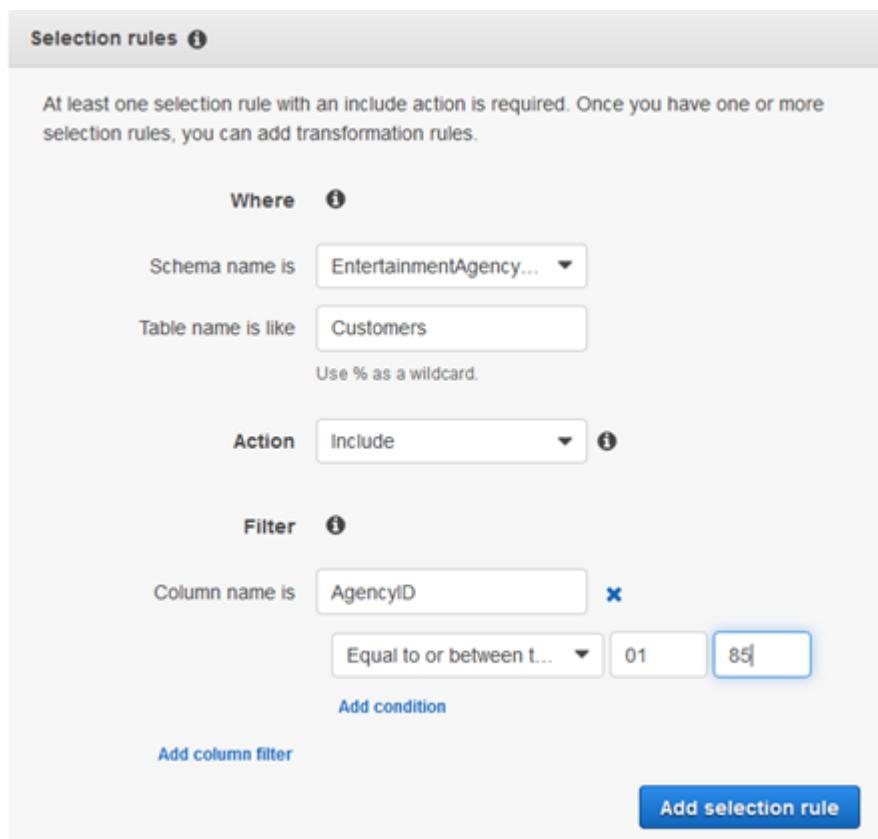
Schema name is EntertainmentAgency... ▾  
Table name is like Customers  
Use % as a wildcard.

Action Include ⓘ

**Filter** ⓘ

Column name is AgencyID ✖  
Equal to or between t... ▾ 01 85|  
Add condition  
Add column filter

Add selection rule



7. When you have created the selections you want, choose **Add selection rule**.
8. 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

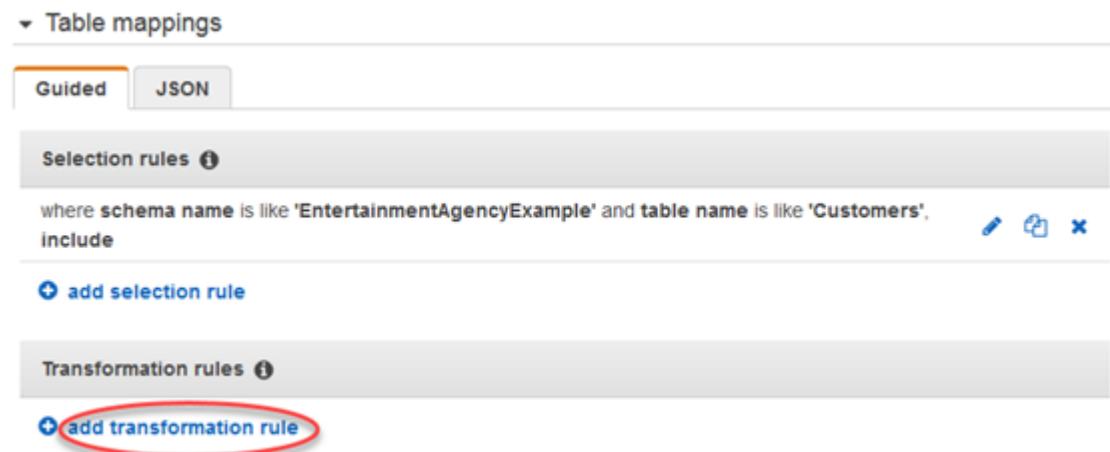
**Selection rules** ⓘ

where schema name is like 'EntertainmentAgencyExample' and table name is like 'Customers'.  
include

+ add selection rule

**Transformation rules** ⓘ

+ add transformation rule



9. 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.

The screenshot shows the 'Selection rules' and 'Transformation rules' configuration interface. The 'Selection rules' section at the top includes a note: 'where schema name is like 'EntertainmentAgencyExample' and table name is like 'Customers', include'. Below it is a link to 'add selection rule'. The 'Transformation rules' section follows, with a 'Target' dropdown set to 'Column'. Under 'Where', there are three filters: 'Schema name is EntertainmentAgencyExample', 'Table name is like Customer', and 'Column name is like AgencyStatus'. A note below says 'Use % as a wildcard.' Under 'Action', a dropdown is set to 'Remove column'. At the bottom right are 'cancel' and 'Add transformation rule' buttons.

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**.

The screenshot shows the 'Create task' dialog. At the top, there are 'Guided' and 'JSON' tabs, with 'Guided' selected. The 'Selection rules' section contains the same note as the previous screenshot: 'where schema name is like 'EntertainmentAgencyExample' and table name is like 'Customer', include'. Below it is a link to 'add selection rule', which is circled in red. The 'Transformation rules' section contains the transformation rule: 'For column where schema name is like 'EntertainmentAgencyExample' and table name is like 'Customer' and column name is like 'AgencyStatus', remove column'. Below it is a link to 'add transformation rule', also circled in red. At the bottom right are 'Cancel' and 'Create task' buttons.

**Note**

AWS DMS doesn't support more than one transformation rule per schema level or per table level. However, AWS DMS does support more than one transformation rule per column level.

## Specifying table selection and transformations rules using JSON

To specify the table mappings that you want to apply during migration, you can create a JSON file. If you create a migration task using the console, you can browse for this JSON file or enter the JSON directly into the table mapping box. If you use the CLI or API to perform migrations, you can specify this file using the `TableMappings` parameter of the `CreateReplicationTask` or `ModifyReplicationTask` API operation.

You can specify what tables, views, and schemas you want to work with. You can also perform table, view, and schema transformations and specify settings for how AWS DMS loads individual tables and views. You create table-mapping rules for these options using the following rule types:

- **selection rules** – Identify the types and names of source tables, views, and schemas to load. For more information, see [Selection rules and actions \(p. 314\)](#).
- **transformation rules** – Specify certain changes or additions to particular source tables and schemas on the source before they are loaded on the target. For more information, see [Transformation rules and actions \(p. 318\)](#).

Also, to define content of new and existing columns, you can use an expression within a transformation rule. For more information, see [Using transformation rule expressions to define column content \(p. 331\)](#).

- **table-settings rules** – Specify how DMS tasks load the data for individual tables. For more information, see [Table-settings rules and operations \(p. 333\)](#).

**Note**

For Amazon S3 targets, you can also tag S3 objects mapped to selected tables and schemas using the post-processing rule type and the add-tag rule action. For more information, see [Amazon S3 object tagging \(p. 189\)](#).

For the targets following, you can specify how and where selected schemas and tables are migrated to the target using the object-mapping rule type:

- Amazon DynamoDB – For more information, see [Using object mapping to migrate data to DynamoDB \(p. 210\)](#).
- Amazon Kinesis – For more information, see [Using object mapping to migrate data to a Kinesis data stream \(p. 227\)](#).
- Apache Kafka – For more information, see [Using object mapping to migrate data to a Kafka topic \(p. 238\)](#).

## Selection rules and actions

Using table mapping, you can specify what tables, views, and schemas you want to work with by using selection rules and actions. For table-mapping rules that use the selection rule type, you can apply the following values.

Parameter	Possible values	Description
rule-type	selection	A selection rule. Define at least one selection rule when specifying a table mapping.
rule-id	A numeric value.	A unique numeric value to identify the rule.
rule-name	An alphanumeric value.	A unique name to identify the rule.
rule-action	include, exclude, explicit	A value that includes or excludes the object or objects selected by the rule. If explicit is specified, you can select and include only one object that corresponds to an explicitly specified table and schema.
object-locator	<p>An object with the following parameters:</p> <ul style="list-style-type: none"> <li>• schema-name – The name of the schema.</li> <li>• table-name – The name of the table.</li> <li>• (Optional) table-type – table   view   all, to indicate if table-name refers only to tables, views, or both tables and views. The default is table.</li> </ul> <p>AWS DMS loads views only in a full-load task. If you have only full-load and change data capture (CDC) tasks, configure at least one full-load-only task to load your views.</p>	<p>The name of each schema and table or view to which the rule applies. You can also specify if a rule includes only tables, only views, or both tables and views. If the rule-action is either include or exclude, you can use the "%" percent sign as a wildcard for all or part of the value for the schema-name and table-name parameter. Thus, you can match these items:</p> <ul style="list-style-type: none"> <li>• A single table or view in a single schema</li> <li>• A single table or view in some or all schemas</li> <li>• Some or all tables and views in a single schema</li> <li>• Some or all tables and views in some or all schemas</li> </ul> <p>If the rule-action is explicit, you can only specify the exact name of a single table or view and its schema (with no wildcards). The supported sources for views include:</p> <ul style="list-style-type: none"> <li>• Oracle</li> <li>• Microsoft SQL Server</li> <li>• MySQL</li> <li>• PostgreSQL</li> <li>• IBM Db2 LUW</li> <li>• SAP Adaptive Server Enterprise (ASE)</li> </ul>

Parameter	Possible values	Description
		<b>Note</b> AWS DMS never loads a source view to a target view. A source view is loaded to an equivalent table on the target with the same name as the view on the source.
<code>load-order</code>	A positive integer. The maximum value is 2,147,483,647.	The priority for loading tables and views. Tables and views with higher values are loaded first.
<code>filters</code>	An array of objects.	One or more objects for filtering the source. You specify object parameters to filter on a single column in the source. You specify multiple objects to filter on multiple columns. For more information, see <a href="#">Using source filters (p. 349)</a> .

### 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"
    }
  ]
}
```

```
{
    "rule-type": "selection",
    "rule-id": "2",
    "rule-name": "2",
    "object-locator": {
        "schema-name": "Test",
        "table-name": "DMS%"
    },
    "rule-action": "exclude"
}
]
```

### Example Migrate a specified single table in single schema

The following example migrates the `Customer` table from the `NewCust` schema in your source to your target endpoint.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "NewCust",
                "table-name": "Customer"
            },
            "rule-action": "explicit"
        }
    ]
}
```

#### Note

You can explicitly select on multiple tables and schemas by specifying multiple selection rules.

### 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"
}
]
```

### Example Migrate some views in a schema

The following example migrates some views from a schema named `Test` in your source to equivalent tables in your target.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "2",
            "rule-name": "2",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "view_DMS%",
                "table-type": "view"
            },
            "rule-action": "include"
        }
    ]
}
```

### Example Migrate all tables and views in a schema

The following example migrates all tables and views from a schema named `report` in your source to equivalent tables in your target.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "3",
            "rule-name": "3",
            "object-locator": {
                "schema-name": "report",
                "table-name": "%",
                "table-type": "all"
            },
            "rule-action": "include"
        }
    ]
}
```

## Transformation rules and actions

You use the transformation actions to specify any transformations you want to apply to the selected schema, table, or view. Transformation rules are optional.

### Note

AWS DMS doesn't support more than one transformation rule per schema level or per table level. However, AWS DMS does support more than one transformation rule per column level.

For table-mapping rules that use the transformation rule type, you can apply the following values.

Parameter	Possible values	Description
rule-type	transformation	A value that applies the rule to each object specified by the selection rule. Use transformation unless otherwise noted.
rule-id	A numeric value.	A unique numeric value to identify the rule.
rule-name	An alphanumeric value.	A unique name to identify the rule.
object-locator	An object with the following parameters: <ul style="list-style-type: none"> <li>• schema-name – The name of the schema.</li> <li>• table-name – The name of the table or view.</li> <li>• table-tablespace-name – The name of an existing table tablespace.</li> <li>• index-tablespace-name – The name of an existing index tablespace.</li> <li>• column-name – The name of an existing column.</li> <li>• data-type – The name of an existing column data type.</li> </ul>	<p>The name of each schema, table or view, table tablespace, index tablespace, and column to which the rule applies. You can use the "%" percent sign as a wildcard for all or part of the value of each object-locator parameter, except data-type. Thus, you can match these items:</p> <ul style="list-style-type: none"> <li>• A single table or view in a single schema</li> <li>• A single table or view in some or all schemas</li> <li>• Some or all tables and views in a single schema</li> <li>• Some or all tables and views in some or all schemas</li> <li>• One or more columns in the specified table or tables, view or views, and schema or schemas.</li> <li>• The columns with a given data-type when multiple columns are specified. For the possible values of data-type, see data-type described following in this table.</li> </ul> <p>Also, the table-tablespace-name or index-tablespace-name parameter is only available to match an Oracle source endpoint. You can specify either table-tablespace-name or index-tablespace-name in a single rule, but not both. Thus, you can match <i>either</i> of the following items:</p> <ul style="list-style-type: none"> <li>• One, some, or all table tablespaces</li> <li>• One, some, or all index tablespaces</li> </ul>
rule-action	add-column, remove-column  rename  convert-lowercase, convert-uppercase	<p>The transformation you want to apply to the object. All transformation rule actions are case-sensitive.</p> <p>The add-column value of the rule-action parameter adds a column to a</p>

Parameter	Possible values	Description
	add-prefix, remove-prefix, replace-prefix  add-suffix, remove-suffix, replace-suffix  define-primary-key  change-data-type  add-before-image-columns	table. When used with the expression and data-type parameters, add-column specifies the value of new column data.  The change-data-type value for rule-action is only available for column rule targets.
rule-target	schema, table, column, table-tablespace, index-tablespace	The type of object that you're transforming.  The table-tablespace and index-tablespace values are only available for an Oracle target endpoint.  Make sure to specify a value for the parameter that you specify as part of the object-locator: table-tablespace-name or index-tablespace-name name.
value	An alphanumeric value that follows the naming rules for the target type.	The new value for actions that require input, such as rename.
old-value	An alphanumeric value that follows the naming rules for the target type.	The old value for actions that require replacement, such as replace-prefix.

Parameter	Possible values	Description
data-type	<p>type – The data type to use if the rule-action is add-column or the replacement data type if the rule-action is change-data-type.</p> <p>Or, the name of the replacement data type when rule-action is change-data-type, the value of column-name is "%", and an additional data-type parameter to identify the existing data type is included in the object-locator.</p> <p>AWS DMS supports column data type transformations for the following DMS data types: "bytes", "date", "time", "datetime", "int1", "int2", "int4", "int8", "numeric", "real4", "real8", "string", "uint1", "uint2", "uint4", "uint8", "wstrin", "blob", "nclob", "clob", "boolean", "set", "list", "map", "tuple"</p> <p>precision – If the added column or replacement data type has a precision, an integer value to specify the precision.</p> <p>length – The length of new column data (when used with add-column)</p>	<p>The following is an example of a data-type parameter to specify the existing data type to be replaced. This existing data-type parameter is included in the object-locator when the value of column-name is "%", shown following.</p> <pre> . . .  "object-locator": {     "schema-name": "dbo",     "table-name": "dms",     "column-name": "%",     "data-type": "int2" }, "data-type": {     "type": "int8" }</pre> <p>Here, any column with the int2 data type is replaced with the int8 data type.</p> <p>The length value for data-type is only available for use with column rule targets of add-column rule actions.</p>
expression	An alphanumeric value that follows SQLite syntax.	<p>When used with the rule-action set to rename-schema, the expression parameter specifies a new schema. When used with the rule-action set to rename-table, expression specifies a new table. When used with the rule-action set to rename-column, expression specifies a new column name value.</p> <p>When used with the rule-action set to add-column, expression specifies data that makes up a new column.</p> <p>For more information on using expressions for transformation rules, see <a href="#">Using transformation rule expressions to define column content (p. 331)</a>.</p>

Parameter	Possible values	Description
primary-key-def	An object with the following parameters: <ul style="list-style-type: none"> <li>name – The name of a new primary key or unique index for the table or view.</li> <li>(Optional) origin – The type of unique key to define: <code>primary-key</code> (the default) or <code>unique-index</code>.</li> <li>columns – An array of strings listing the names of columns in the order they appear in the primary key or unique index.</li> </ul>	This parameter can define the name, type, and content of a unique key on the transformed table or view. It does so when the <code>rule-action</code> is set to <code>define-primary-key</code> and the <code>rule-target</code> is set to <code>table</code> . By default, the unique key is defined as a primary key.
before-image-def	An object with the following parameters: <ul style="list-style-type: none"> <li>column-prefix – A value prepended to a column name. The default value is <code>BI_</code>.</li> <li>column-suffix – A value appended to the column name. The default is empty.</li> <li>column-filter – Requires one of the following values: <code>pk-only</code> (default), <code>non-lob</code> (optional) and <code>all</code> (optional).</li> </ul>	<p>This parameter defines a naming convention to identify the before-image columns and specifies a filter to identify which source columns can have before-image columns created for them on the target. You can specify this parameter when the <code>rule-action</code> is set to <code>add-before-image-columns</code> and the <code>rule-target</code> is set to <code>column</code>.</p> <p>Don't set both <code>column-prefix</code> and <code>column-suffix</code> to empty strings.</p> <p>For <code>column-filter</code>, select:</p> <ul style="list-style-type: none"> <li><code>pk-only</code> – To add only columns that are part of table primary keys.</li> <li><code>non-lob</code> – To add only columns that are not of LOB type.</li> <li><code>all</code> – To add any column that has a before-image value.</li> </ul> <p>For more information on before-image support for AWS DMS target endpoints, see:</p> <ul style="list-style-type: none"> <li><a href="#">Using a before image to view original values of CDC rows for a Kinesis data stream as a target (p. 224)</a></li> <li><a href="#">Using a before image to view original values of CDC rows for Apache Kafka as a target (p. 235)</a></li> </ul>

### Example Rename a schema

The following example renames a schema from `Test` in your source to `Test1` in your target.

```
{
```

```

"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.

```

{
    "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.

```
{
    "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 Rename an Oracle table tablespace

The following example renames the table tablespace named SetSpace for a table named Actor in your Oracle source to SceneTblSpace in your Oracle target endpoint.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "Play",
                "table-name": "%"
            },
            "rule-action": "include"
        },
        {
            "rule-type": "transformation",
            "rule-id": "2",
            "rule-name": "2",
            "rule-action": "rename",
            "rule-target": "table-tablespace",
            "object-locator": {
                "schema-name": "Play",
                "table-name": "Actor",
                "table-tablespace-name": "SetSpace"
            },
            "value": "SceneTblSpace"
        }
    ]
}
```

### Example Rename an Oracle index tablespace

The following example renames the index tablespace named SetISpace for a table named Actor in your Oracle source to SceneIdxSpace in your Oracle target endpoint.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "Play",
                "table-name": "%"
            },
            "rule-action": "include"
        },
        {
            "rule-type": "transformation",
            "rule-id": "2",
            "rule-name": "2",
            "rule-action": "rename",
            "rule-target": "table-tablespace",
            "object-locator": {
                "schema-name": "Play",
                "table-name": "Actor",
                "table-tablespace-name: "SetISpace"
            },
            "value": "SceneIdxSpace"
        }
    ]
}
```

### Example Add a column

The following example adds a datetime column to the table Actor in schema test.

```
{
    "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-column",
            "rule-target": "table",
            "object-locator": {
                "schema-name": "test",
                "table-name": "actor"
            },
            "value": "last_updated",
            "data-type": {
                "type": "datetime",
                "precision": 6
            }
        }
    ]
}
```

### Example Remove a column

The following example transforms the table named `Actor` in your source to remove all columns starting with the characters `col` from it in your target.

```
{  
  "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.

```
{  
  "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.

```
{
```

```

"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.

```

{
  "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.

```
{
  "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.

```
{
    "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 Define a primary key

The following example defines a primary key named `ITEM-primary-key` on three columns of the `ITEM` table migrated 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-id": "1",
    "rule-name": "1",
    "object-locator": {
        "schema-name": "inventory",
        "table-name": "%"
    },
    "rule-action": "include"
}, {
    "rule-type": "transformation",
    "rule-id": "2",
    "rule-name": "2",
    "rule-action": "define-primary-key",
    "rule-target": "table",
    "object-locator": {
        "schema-name": "inventory",
        "table-name": "ITEM"
    },
    "primary-key-def": {
        "name": ITEM-primary-key,
        "columns": [
            "ITEM-NAME",
            "BOM-MODEL-NUM",
            "BOM-PART-NUM"
        ]
    }
}
]
}

```

### Example Define a unique index

The following example defines a unique index named `ITEM-unique-idx` on three columns of the `ITEM` table migrated to your target endpoint.

```

{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "inventory",
                "table-name": "%"
            },
            "rule-action": "include"
        }, {
            "rule-type": "transformation",
            "rule-id": "2",
            "rule-name": "2",
            "rule-action": "define-primary-key",
            "rule-target": "table",
            "object-locator": {
                "schema-name": "inventory",
                "table-name": "ITEM"
            },
            "primary-key-def": {
                "name": ITEM-unique-idx,
                "origin": unique-index,
                "columns": [
                    "ITEM-NAME",
                    "BOM-MODEL-NUM",
                    "BOM-PART-NUM"
                ]
            }
        }
    ]
}

```

## Example Change data type of target column

The following example changes the data type of a target column named `SALE_AMOUNT` from an existing data type to `int8`.

```
{
  "rule-type": "transformation",
  "rule-id": "1",
  "rule-name": "RuleName 1",
  "rule-action": "change-data-type",
  "rule-target": "column",
  "object-locator": {
    "schema-name": "dbo",
    "table-name": "dms",
    "column-name": "SALE_AMOUNT"
  },
  "data-type": {
    "type": "int8"
  }
}
```

## Example Add a before image column

For a source column named `emp_no`, the transformation rule in the example following adds a new column named `BI_emp_no` in the target.

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "%",
        "table-name": "%"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "transformation",
      "rule-id": "2",
      "rule-name": "2",
      "rule-target": "column",
      "object-locator": {
        "schema-name": "%",
        "table-name": "employees"
      },
      "rule-action": "add-before-image-columns",
      "before-image-def": {
        "column-prefix": "BI_",
        "column-suffix": "",
        "column-filter": "pk-only"
      }
    }
  ]
}
```

Here, the statement following populates a `BI_emp_no` column in the corresponding row with 1.

```
UPDATE employees SET emp_no = 3 WHERE emp_no = 1;
```

When writing CDC updates to supported AWS DMS targets, the `BI_emp_no` column makes it possible to tell which rows have updated values in the `emp_no` column.

# Using transformation rule expressions to define column content

To define content for new and existing columns, you can use an expression within a transformation rule. For example, using expressions you can add a column or replicate source table headers to a target. You can also use expressions to flag records on target tables as inserted, updated, or deleted at the source.

## Topics

- [Adding a column using an expression \(p. 331\)](#)
- [Flagging target records using an expression \(p. 331\)](#)
- [Replicating source table headers using expressions \(p. 332\)](#)

## Adding a column using an expression

To add columns to tables using an expression in a transformation rule, use an `add-column` rule action and a `column` rule target.

The following example adds a new column to the `ITEM` table. It sets the new column name to `FULL_NAME`, with a data type of `string`, 50 characters long. The expression concatenates the values of two existing columns, `FIRST_NAME` and `LAST_NAME`, to evaluate to `FULL_NAME`.

```
{  
    "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-column",  
            "rule-target": "column",  
            "object-locator": {  
                "schema-name": "Test",  
                "table-name": "ITEM"  
            },  
            "value": "FULL_NAME",  
            "expression": "$FIRST_NAME || '_' || $LAST_NAME",  
            "data-type": {  
                "type": "string",  
                "length": 50  
            }  
        }  
    ]  
}
```

## Flagging target records using an expression

To flag records in target tables as inserted, updated, or deleted in the source table, use an expression in a transformation rule. The expression uses an `operation_indicator` function to flag records. Records

deleted from the source aren't deleted from the target. Instead, the target record is flagged with a user-provided value to indicate that it was deleted from the source.

**Note**

The `operation_indicator` function works only on tables that have a primary key.

For example, the following transformation rule first adds a new `Operation` column to a target table. It then updates the column with the value `D` whenever a record is deleted from a source table.

```
{
    "rule-type": "transformation",
    "rule-id": "2",
    "rule-name": "2",
    "rule-target": "column",
    "object-locator": {
        "schema-name": "%",
        "table-name": "%"
    },
    "rule-action": "add-column",
    "value": "Operation",
    "expression": "operation_indicator('D', 'U', 'I')",
    "data-type": {
        "type": "string",
        "length": 50
    }
}
```

## Replicating source table headers using expressions

By default, headers for source tables aren't replicated to the target. To indicate which headers to replicate, use a transformation rule with an expression that includes the table column header.

You can use the following column headers in expressions.

Header	Value in ongoing replication	Value in full load	Data type
AR_H_STREAM_POSITION	The stream position value from the source. This value might be the system change number (SCN) or the log sequence number (LSN), depending on the source endpoint.	An empty string.	STRING
AR_H_TIMESTAMP	A timestamp indicating the time of the change.	A timestamp indicating the current time.	DATETIME
AR_H_COMMIT_TIMESTAMP	A timestamp indicating the time of the commit.	A timestamp indicating the current time.	DATETIME
AR_H_OPERATION	INSERT, UPDATE, or DELETE	INSERT	STRING
AR_H_USER	The user name, ID, or any other information that the source provides about the user that made the change.	The transformation that you want to apply to the object. Transformation rule	STRING

Header	Value in ongoing replication	Value in full load	Data type
	This header is supported on the SQL Server and Oracle (version 11.2.0.3 and higher) source endpoints only.	actions are case-sensitive.	

The following example adds a new column to the target by using the stream position value from the source. For SQL Server, the stream position value is the LSN for the source endpoint. For Oracle, the stream position value is the SCN for the source endpoint.

```
{
    "rule-type": "transformation",
    "rule-id": "2",
    "rule-name": "2",
    "rule-target": "column",
    "object-locator": {
        "schema-name": "%",
        "table-name": "%"
    },
    "rule-action": "add-column",
    "value": "transact_id",
    "expression": "$AR_H_STREAM_POSITION",
    "data-type": {
        "type": "string",
        "length": 50
    }
}
```

## Table-settings rules and operations

You use table settings to specify any settings that you want to apply to the selected table or view for a specified operation. Table-settings rules are optional.

### Topics

- [Using parallel load for selected tables and views \(p. 338\)](#)
- [Specifying LOB settings for a selected table or view \(p. 341\)](#)
- [Table-settings examples \(p. 343\)](#)

For table-mapping rules that use the table-settings rule type, you can apply the following parameters.

Parameter	Possible values	Description
rule-type	table-settings	A value that applies the rule to a table or view specified by the selection rule.
rule-id	A numeric value.	A unique numeric value to identify the rule.
rule-name	An alphanumeric value.	A unique name to identify the rule.

Parameter	Possible values	Description
object-locator	<p>An object with the following parameters:</p> <ul style="list-style-type: none"> <li>• schema-name – The name of the schema.</li> <li>• table-name – The name of the table or view.</li> </ul>	The name of a specific schema and table or view (no wildcards).
parallel-load	<p>An object with the following parameters:</p> <ul style="list-style-type: none"> <li>• type – Specifies whether parallel loading is enabled. If it is, this parameter also specifies the mechanism to identify the table or view partitions, subpartitions, or other segments to load in parallel. Partitions are segments that are already defined and identified by name in the source table or view. For Oracle endpoints only, subpartitions are an additional level of segments that are already defined and identified by name in the source table or view. You can identify other segments in the table-settings rule by specifying boundaries in the range of values for one or more table or view columns.</li> <li>• partitions – When type is partitions-list, this value specifies all the partitions to load in parallel.</li> <li>• subpartitions – For Oracle endpoints only, when type is partitions-list this value specifies all the subpartitions to load in parallel.</li> <li>• columns – When type is ranges, this value specifies the names of columns used to identify range-based segments to load in parallel.</li> <li>• boundaries – When type is ranges, this value specifies the values of the columns used to identify range-based segments to load in parallel.</li> </ul>	<p>A value that specifies a parallel load (multithreaded) operation on the table or view identified by the object-locator option. In this case, you can load in parallel in any of these ways:</p> <ul style="list-style-type: none"> <li>• By segments specified by all available partitions or subpartitions.</li> <li>• By selected partitions and subpartitions.</li> <li>• By range-based segments that you specify.</li> </ul> <p>For more information on parallel load, see <a href="#">Using parallel load for selected tables and views (p. 338)</a>.</p>

Parameter	Possible values	Description
type	<p>One of the following for parallel-load:</p> <ul style="list-style-type: none"> <li>• <b>partitions-auto</b> – All partitions of the table or view are loaded in parallel. Every partition is allocated to its own thread.</li> <li>• <b>subpartitions-auto</b> – (Oracle endpoints only) All subpartitions of the table or view are loaded in parallel. Every subpartition is allocated to its own thread.</li> <li>• <b>partitions-list</b> – All specified partitions of the table or view are loaded in parallel. For Oracle endpoints only, all specified subpartitions of the table or view are loaded in parallel. Each partition and subpartition that you specify is allocated to its own thread. You specify the partitions and subpartitions to load in parallel by partition names (<code>partitions</code>) and subpartition names (<code>subpartitions</code>).</li> <li>• <b>ranges</b> – All specified segments of the table or view are loaded in parallel. Each table or view segment that you identify is allocated to its own thread. You specify these segments by column names (<code>columns</code>) and column values (<code>boundaries</code>).</li> <li>• <b>none</b> – The table or view is loaded in a single-threaded task (the default), regardless of its partitions or subpartitions. For more information, see <a href="#">Creating a task (p. 273)</a>.</li> </ul> <p><b>Note</b> PostgreSQL as a source supports only this type of parallel load.</p>	The mechanism to identify the table or view partitions, subpartitions, or segments to load in parallel.

Parameter	Possible values	Description
<code>partitions</code>	When <code>type</code> is <code>partitions-list</code> , this is an array of strings that specify the names of partitions to load in parallel.	The names of partitions to load in parallel.
<code>subpartitions</code>	(Oracle endpoints only) When <code>type</code> is <code>partitions-list</code> , this is an array of strings that specifies the names of subpartitions to load in parallel.	The names of subpartitions to load in parallel.
<code>columns</code>	When <code>type</code> is <code>ranges</code> , an array of strings set to the names of columns that identify range-based table or view segments to load in parallel.	The names of columns that identify range-based table or view segments to load in parallel.
<code>boundaries</code>	When <code>type</code> is <code>ranges</code> , an array of column-value arrays. Each column-value array contains column values in the quantity and order specified by <code>columns</code> . A column-value array specifies the upper boundary of a table or view segment. Each additional column-value array adds the upper boundary for one additional table or view segment. All such range-based table or view segments load in parallel.	Column values that identify range-based table or view partitions to load in parallel.
<code>lob-settings</code>	An object with the following parameters: <ul style="list-style-type: none"> <li>• <code>mode</code> – Specifies the migration handling mode for LOBs.</li> <li>• <code>bulk-max-size</code> – Specifies the maximum size of LOBs, depending on the <code>mode</code> setting.</li> </ul>	A value that specifies LOB handling for the table or view identified by the <code>object-locator</code> option. The specified LOB handling overrides any task LOB settings for this table or view only. For more information on using the LOB settings parameters, see <a href="#">Specifying LOB settings for a selected table or view (p. 341)</a> .

Parameter	Possible values	Description
mode	<p>Specifies the migration handling for LOBs in the specified table or view using the following values:</p> <ul style="list-style-type: none"> <li>• <b>limited</b> – (Default) This value sets migration to limited LOB mode, with all LOBs migrated inline together with all other column data types in the table or view. Use this value when replicating mostly small LOBs (100 MB or less). Also, specify a <code>bulk-max-size</code> value (zero is invalid). All migrated LOBs greater than <code>bulk-max-size</code> are truncated to the size that you set.</li> <li>• <b>unlimited</b> – This value sets migration to full LOB mode. Use this value when all or most of the LOBs that you want to replicate are larger than 1 GB. If you specify a <code>bulk-max-size</code> value of zero, all LOBs are migrated in <i>standard</i> full LOB mode. In this form of <b>unlimited</b> mode, all LOBs are migrated separately from other column data types using a lookup from the source table or view. If you specify a <code>bulk-max-size</code> value greater than zero, all LOBs are migrated in <i>combination</i> full LOB mode. In this form of <b>unlimited</b> mode, LOBs greater than <code>bulk-max-size</code> are migrated using a source table or view lookup, similar to standard full LOB mode. Otherwise, LOBs up to and including this size are migrated inline, similar to limited LOB mode. No LOB is ever truncated in <b>unlimited</b> mode, regardless of the form you use.</li> <li>• <b>none</b> – All table or view LOBs are migrated according to the task LOB settings.</li> </ul> <p>For more information on the task LOB settings,</p>	The mechanism used to migrate LOBs.

Parameter	Possible values	Description
	<p>see <a href="#">Target metadata task settings (p. 282)</a>.</p> <p>For more information on how to migrate LOBs and how to specify these task LOB settings, see <a href="#">Setting LOB support for source databases in an AWS DMS task (p. 300)</a>.</p>	
<code>bulk-max-size</code>	The effect of this value depends on the mode.	The maximum size of LOBs in kilobyte increments. Specify this option only if you need to replicate small LOBs or if the target endpoint doesn't support unlimited LOB size.

## Using parallel load for selected tables and views

To speed up migration and make it more efficient, you can use parallel load for selected relational tables and views. In other words, you can migrate a single segmented table or view using several threads in parallel. To do this, AWS DMS splits a full-load task into threads, with each table segment allocated to its own thread.

Using this parallel-load process, you can first have multiple threads unload multiple tables and views in parallel from the source endpoint. You can then have multiple threads migrate and load the same tables and views in parallel to the target endpoint. For some database engines, you can segment the tables and views by existing partitions or subpartitions. Otherwise, you can segment any table or view by ranges of column values that you specify.

Parallel load is supported for the following source endpoints:

- Oracle
- Microsoft SQL Server
- MySQL
- PostgreSQL
- IBM Db2
- SAP Adaptive Server Enterprise (ASE)

Parallel load for use with table-setting rules are supported for the following target endpoints:

- Oracle
- Microsoft SQL Server
- MySQL
- PostgreSQL
- SAP Adaptive Server Enterprise (ASE)

To specify the maximum number of tables and views to load in parallel, use the `MaxFullLoadSubTasks` task setting. To specify the maximum number of threads per table or view for a parallel-load task, use the `ParallelLoadThreads` task setting. To specify the buffer size for a parallel load task, use the `ParallelLoadBufferSize` task setting. The availability and settings of `ParallelLoadThreads` and `ParallelLoadBufferSize` depend on the target endpoint.

For more information on the `ParallelLoadThreads` and `ParallelLoadBufferSize` settings, see [Target metadata task settings \(p. 282\)](#). For more information on the `MaxFullLoadSubTasks` setting, see [Full-load task settings \(p. 283\)](#). For information specific to target endpoints, see the related topics.

To use parallel load, create a table-mapping rule of type `table-settings` with the `parallel-load` option. Within the `table-settings` rule, you can specify the segmentation criteria for a single table or view that you want to load in parallel. To do so, set the `type` parameter of the `parallel-load` option to one of several options. How to do this depends on how you want to segment the table or view for parallel load:

- By partitions – Load all existing table or view partitions using the `partitions-auto` type. Or load only selected partitions using the `partitions-list` type with a specified `partitions` array.
- (Oracle endpoints only) By subpartitions – Load all existing table or view subpartitions using the `subpartitions-auto` type. Or load only selected subpartitions using the `partitions-list` type with a specified `subpartitions` array.
- By segments that you define – Load table or view segments that you define by using column-value boundaries. To do so, use the `ranges` type with specified `columns` and `boundaries` arrays.

**Note**

PostgreSQL as a source supports only this type of parallel load.

To identify additional tables or views to load in parallel, specify additional `table-settings` objects with `parallel-load` options.

In the following procedures, you can find out how to code JSON for each parallel-load type, from the simplest to the most complex.

### To specify all table or view partitions, or all table or view subpartitions

- Specify `parallel-load` with either the `partitions-auto` type or the `subpartitions-auto` type (but not both).

Every table or view partition or subpartition is then automatically allocated to its own thread.

**Note**

Parallel load includes partitions or subpartitions only if they are already defined for the table or view.

### To specify selected table or view partitions, subpartitions, or both

1. Specify `parallel-load` with the `partitions-list` type.
2. (Optional) Include partitions by specifying an array of partition names as the value of `partitions`.  
Each specified partition is then allocated to its own thread.
3. (Optional, for Oracle endpoints only) Include subpartitions by specifying an array of subpartition names as the value of `subpartitions`.

Each specified subpartition is then allocated to its own thread.

**Note**

Parallel load includes partitions or subpartitions only if they are already defined for the table or view.

You can specify table or view segments as ranges of column values. When you do so, be aware of these column characteristics:

- Specifying indexed columns significantly improves performance.

- You can specify up to 10 columns.
- You can't use columns to define segment boundaries with the following AWS DMS data types: DOUBLE, FLOAT, BLOB, CLOB, and NCLOB
- Records with null values aren't replicated.

### To specify table or view segments as ranges of column values

1. Specify parallel-load with the `ranges` type.
2. Define a boundary between table or view segments by specifying an array of column names as the value of `columns`. Do this for every column for which you want to define a boundary between table or view segments.

The order of columns is significant. The first column is the most significant and the last column is least significant in defining each boundary, as described following.

3. Define the data ranges for all the table or view segments by specifying a boundary array as the value of `boundaries`. A *boundary array* is an array of column-value arrays. To do so, take the following steps:
  - a. Specify each element of a column-value array as a value that corresponds to each column. A *column-value array* represents the upper boundary of each table or view segment that you want to define. Specify each column in the same order that you specified that column in the `columns` array.

**Note**

Enter values for DATE columns in the format supported by the source.

- b. Specify each column-value array as the upper boundary, in order, of each segment from the bottom to the next-to-top segment of the table or view. If any rows exist above the top boundary that you specify, these rows complete the top segment of the table or view. Thus, the number of range-based segments is potentially one more than the number of segment boundaries in the boundary array. Each such range-based segment is allocated to its own thread.

**Note**

All of the non-null data is replicated, even if you don't define data ranges for all of the columns in the table or view.

For example, suppose you define three column-value arrays for columns COL1, COL2, and COL3 as follows.

COL1	COL2	COL3
10	30	105
20	20	120
100	12	99

You have defined three segment boundaries for a possible total of four segments.

To identify the ranges of rows to replicate for each segment, the replication instance applies a search to these three columns for each of the four segments. The search is like the following:

### Segment 1

Replicate all rows where the following is true: The first two-column values are less than or equal to their corresponding **Segment 1** upper boundary values. Also, the values of the third column are less than its **Segment 1** upper boundary value.

### Segment 2

Replicate all rows (except **Segment 1** rows) where the following is true: The first two-column values are less than or equal to their corresponding **Segment 2** upper boundary values. Also, the values of the third column are less than its **Segment 2** upper boundary value.

### Segment 3

Replicate all rows (except **Segment 2** rows) where the following is true: The first two-column values are less than or equal to their corresponding **Segment 3** upper boundary values. Also, the values of the third column are less than its **Segment 3** upper boundary value.

### Segment 4

Replicate all remaining rows (except the **Segment 1, 2, and 3** rows).

In this case, the replication instance creates a WHERE clause to load each segment as follows:

### Segment 1

```
((COL1 < 10) OR ((COL1 = 10) AND (COL2 < 30)) OR ((COL1 = 10) AND (COL2 = 30) AND (COL3 < 105)))
```

### Segment 2

```
NOT ((COL1 < 10) OR ((COL1 = 10) AND (COL2 < 30)) OR ((COL1 = 10) AND (COL2 = 30) AND (COL3 < 105))) AND ((COL1 < 20) OR ((COL1 = 20) AND (COL2 < 20)) OR ((COL1 = 20) AND (COL2 = 20) AND (COL3 < 120)))
```

### Segment 3

```
NOT ((COL1 < 20) OR ((COL1 = 20) AND (COL2 < 20)) OR ((COL1 = 20) AND (COL2 = 20) AND (COL3 < 120))) AND ((COL1 < 100) OR ((COL1 = 100) AND (COL2 < 12)) OR ((COL1 = 100) AND (COL2 = 12) AND (COL3 < 99)))
```

### Segment 4

```
NOT ((COL1 < 100) OR ((COL1 = 100) AND (COL2 < 12)) OR ((COL1 = 100) AND (COL2 = 12) AND (COL3 < 99)))
```

## Specifying LOB settings for a selected table or view

You can set task LOB settings for one or more tables by creating a table-mapping rule of type table-settings with the `lob-settings` option for one or more table-settings objects.

Specifying LOB settings for selected tables or views is supported for the following source endpoints:

- Oracle
- Microsoft SQL Server
- MySQL
- PostgreSQL
- IBM Db2, depending on the `mode` and `bulk-max-size` settings, described following
- SAP Adaptive Server Enterprise (ASE), depending on the `mode` and `bulk-max-size` settings, as described following

Specifying LOB settings for selected tables or views is supported for the following target endpoints:

- Oracle
- Microsoft SQL Server
- MySQL
- PostgreSQL
- SAP ASE, depending on the `mode` and `bulk-max-size` settings, as described following

**Note**

You can use LOB data types only with tables and views that include a primary key.

To use LOB settings for a selected table or view, you create a table-mapping rule of type `table-settings` with the `lob-settings` option. Doing this specifies LOB handling for the table or view identified by the `object-locator` option. Within the `table-settings` rule, you can specify a `lob-settings` object with the following parameters:

- `mode` – Specifies the mechanism for handling LOB migration for the selected table or view as follows:
  - `limited` – The default limited LOB mode is the fastest and most efficient mode. Use this mode only if all of your LOBs are small (within 100 MB in size) or the target endpoint doesn't support an unlimited LOB size. Also if you use `limited`, all LOBs need to be within the size that you set for `bulk-max-size`.

In this mode for a full load task, the replication instance migrates all LOBs inline together with other column data types as part of main table or view storage. However, the instance truncates any migrated LOB larger than your `bulk-max-size` value to the specified size. For a change data capture (CDC) load task, the instance migrates all LOBs using a source table lookup, as in standard full LOB mode (see the following). It does so regardless of LOB size.

**Note**

You can migrate views for full-load tasks only.

- `unlimited` – The migration mechanism for full LOB mode depends on the value you set for `bulk-max-size` as follows:
  - **Standard full LOB mode** – When you set `bulk-max-size` to zero, the replication instance migrates all LOBs using standard full LOB mode. This mode requires a lookup in the source table or view to migrate every LOB, regardless of size. This approach typically results in a much slower migration than for limited LOB mode. Use this mode only if all or most of your LOBs are large (1 GB or larger).
  - **Combination full LOB mode** – When you set `bulk-max-size` to a nonzero value, this full LOB mode uses a combination of limited LOB mode and standard full LOB mode. That is for a full load task, if a LOB size is within your `bulk-max-size` value, the instance migrates the LOB inline as in limited LOB mode. If the LOB size is greater than this value, the instance migrates the LOB using a source table or view lookup as in standard full LOB mode. For a change data capture (CDC) load task, the instance migrates all LOBs using a source table lookup, as in standard full LOB mode (see the following). It does so regardless of LOB size.

**Note**

You can migrate views for full-load tasks only.

This mode results in a migration speed that is a compromise between the faster, limited LOB mode and the slower, standard full LOB mode. Use this mode only when you have a mix of small and large LOBs, and most of the LOBs are small.

This combination full LOB mode is available only for the following endpoints:

- IBM Db2 as source
- SAP ASE as source or target

Regardless of the mechanism you specify for unlimited mode, the instance migrates all LOBs fully, without truncation.

- none – The replication instance migrates LOBs in the selected table or view using your task LOB settings. Use this option to help compare migration results with and without LOB settings for the selected table or view.

If the specified table or view has LOBs included in the replication, you can set the `BatchApplyEnabled` task setting to `true` only when using limited LOB mode.

In some cases, you might set `BatchApplyEnabled` to `true` and `BatchApplyPreserveTransaction` to `false`. In these cases, the instance sets `BatchApplyPreserveTransaction` to `true` if the table or view has LOBs and the source and target endpoints are Oracle.

- `bulk-max-size` – Set this value to a zero or non-zero value in kilobytes, depending on the mode as described for the previous items. In limited mode, you must set a nonzero value for this parameter.

The instance converts LOBs to binary format. Therefore, to specify the largest LOB you need to replicate, multiply its size by three. For example, if your largest LOB is 2 MB, set `bulk-max-size` to 6,000 (6 MB).

## Table-settings examples

Following, you can find some examples that demonstrate the use of table settings.

### Example Load a table segmented by partitions

The following example loads a `SALES` table in your source more efficiently by loading it in parallel based on all its partitions.

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "%",
        "table-name": "%"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "table-settings",
      "rule-id": "2",
      "rule-name": "2",
      "object-locator": {
        "schema-name": "HR",
        "table-name": "SALES"
      },
      "parallel-load": {
        "type": "partitions-auto"
      }
    }
  ]
}
```

### Example Load a table segmented by subpartitions

The following example loads a `SALES` table in your Oracle source more efficiently by loading it in parallel based on all its subpartitions.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "%",
                "table-name": "%"
            },
            "rule-action": "include"
        },
        {
            "rule-type": "table-settings",
            "rule-id": "2",
            "rule-name": "2",
            "object-locator": {
                "schema-name": "HR",
                "table-name": "SALES"
            },
            "parallel-load": {
                "type": "subpartitions-auto"
            }
        }
    ]
}
```

### Example Load a table segmented by a list of partitions

The following example loads a `SALES` table in your source by loading it in parallel by a particular list of partitions. Here, the specified partitions are named after values starting with portions of the alphabet, for example `ABCD`, `EFGH`, and so on.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "%",
                "table-name": "%"
            },
            "rule-action": "include"
        },
        {
            "rule-type": "table-settings",
            "rule-id": "2",
            "rule-name": "2",
            "object-locator": {
                "schema-name": "HR",
                "table-name": "SALES"
            },
            "parallel-load": {
                "type": "partitions-list",
                "partitions": [
                    "ABCD",
                    "EFGH",
                    "IJKL",
                    "MNOP",
                    "QRST",
                    "UVWXYZ"
                ]
            }
        }
}
```

```
        ]
    }
```

### Example Load an Oracle table segmented by a selected list of partitions and subpartitions

The following example loads a `SALES` table in your Oracle source by loading it in parallel by a selected list of partitions and subpartitions. Here, the specified partitions are named after values starting with portions of the alphabet, for example `ABCD`, `EFGH`, and so on. The specified subpartitions are named after values starting with numerals, for example `01234` and `56789`.

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "%",
        "table-name": "%"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "table-settings",
      "rule-id": "2",
      "rule-name": "2",
      "object-locator": {
        "schema-name": "HR",
        "table-name": "SALES"
      },
      "parallel-load": {
        "type": "partitions-list",
        "partitions": [
          "ABCD",
          "EFGH",
          "IJKL",
          "MNOP",
          "QRST",
          "UVWXYZ"
        ],
        "subpartitions": [
          "01234",
          "56789"
        ]
      }
    }
  ]
}
```

### Example Load a table segmented by ranges of column values

The following example loads a `SALES` table in your source by loading it in parallel by segments specified by ranges of the `SALES_NO` and `REGION` column values.

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

```

        "rule-action": "include"
    },
    {
        "rule-type": "table-settings",
        "rule-id": "2",
        "rule-name": "2",
        "object-locator": {
            "schema-name": "HR",
            "table-name": "SALES"
        },
        "parallel-load": {
            "type": "ranges",
            "columns": [
                "SALES_NO",
                "REGION"
            ],
            "boundaries": [
                [
                    [
                        "1000",
                        "NORTH"
                    ],
                    [
                        "3000",
                        "WEST"
                    ]
                ]
            ]
        }
    }
}

```

Here, two columns are specified for the segment ranges with the names, `SALES_NO` and `REGION`. Two boundaries are specified with two sets of column values (`[ "1000", "NORTH" ]` and `[ "3000", "WEST" ]`).

These two boundaries thus identify the following three table segments to load in parallel:

#### Segment 1

Rows with `SALES_NO` less than or equal to 1,000 and `REGION` less than "NORTH". In other words, sales numbers up to 1,000 in the EAST region.

#### Segment 2

Rows other than **Segment 1** with `SALES_NO` less than or equal to 3,000 and `REGION` less than "WEST". In other words, sales numbers over 1,000 up to 3,000 in the NORTH and SOUTH regions.

#### Segment 3

All remaining rows other than **Segment 1** and **Segment 2**. In other words, sales numbers over 3,000 in the "WEST" region.

### Example Load two tables: One segmented by ranges and another segmented by partitions

The following example loads a `SALES` table in parallel by segment boundaries that you identify. It also loads an `ORDERS` table in parallel by all of its partitions, as with previous examples.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "%",

```

```

        "table-name": "%"
    },
    "rule-action": "include"
},
{
    "rule-type": "table-settings",
    "rule-id": "2",
    "rule-name": "2",
    "object-locator": {
        "schema-name": "HR",
        "table-name": "SALES"
    },
    "parallel-load": {
        "type": "ranges",
        "columns": [
            "SALES_NO",
            "REGION"
        ],
        "boundaries": [
            [
                [
                    "1000",
                    "NORTH"
                ],
                [
                    "3000",
                    "WEST"
                ]
            ]
        ]
    }
},
{
    "rule-type": "table-settings",
    "rule-id": "3",
    "rule-name": "3",
    "object-locator": {
        "schema-name": "HR",
        "table-name": "ORDERS"
    },
    "parallel-load": {
        "type": "partitions-auto"
    }
}
]
}

```

### **Example Load a table with LOBs using limited LOB mode**

The following example loads an `ITEMS` table including LOBs in your source using limited LOB mode (the default) with a maximum nontruncated size of 100 MB. Any LOBs that are larger than this size are truncated to 100 MB. All LOBs are loaded inline with all other column data types.

```

{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "%",
                "table-name": "%"
            },
            "rule-action": "include"
        },
        {
            "rule-type": "table-settings",

```

```

        "rule-id": "2",
        "rule-name": "2",
        "object-locator": {
            "schema-name": "INV",
            "table-name": "ITEMS"
        },
        "lob-settings": {
            "bulk-max-size": "100000"
        }
    }
]
}

```

### **Example Load a table with LOBs using standard full LOB mode**

The following example loads an `ITEMS` table in your source, including all its LOBs without truncation, using standard full LOB mode. All LOBs, regardless of size, are loaded separately from other data types using a lookup for each LOB in the source table.

```

{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "%",
                "table-name": "%"
            },
            "rule-action": "include"
        },
        {
            "rule-type": "table-settings",
            "rule-id": "2",
            "rule-name": "2",
            "object-locator": {
                "schema-name": "INV",
                "table-name": "ITEMS"
            },
            "lob-settings": {
                "mode": "unlimited"
                "bulk-max-size": "0"
            }
        }
    ]
}

```

### **Example Load a table with LOBs using combination full LOB mode**

The following example loads an `ITEMS` table in your source, including all its LOBs without truncation, using combination full LOB mode. All LOBs within 100 MB in size are loaded inline along with other data types, as in limited LOB mode. All LOBs over 100 MB in size are loaded separately from other data types. This separate load uses a lookup for each such LOB in the source table, as in standard full LOB mode.

```

{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "%",
                "table-name": "%"
            },
        }
    ]
}

```

```

        "rule-action": "include"
    },
    {
        "rule-type": "table-settings",
        "rule-id": "2",
        "rule-name": "2",
        "object-locator": {
            "schema-name": "INV",
            "table-name": "ITEMS"
        },
        "lob-settings": {
            "mode": "unlimited"
            "bulk-max-size": "100000"
        }
    }
]
}

```

### Example Load a table with LOBs using the task LOB settings

The following example loads an `ITEMS` table in your source, including all LOBs, using its task LOB settings. The `bulk-max-size` setting of 100 MB is ignored and left only for a quick reset to `limited` or `unlimited` mode.

```

{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "%",
                "table-name": "%"
            },
            "rule-action": "include"
        },
        {
            "rule-type": "table-settings",
            "rule-id": "2",
            "rule-name": "2",
            "object-locator": {
                "schema-name": "INV",
                "table-name": "ITEMS"
            },
            "lob-settings": {
                "mode": "none"
                "bulk-max-size": "100000"
            }
        }
    ]
}

```

## 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, view names, and schema names are case-sensitive.

The following limitations apply to using source filters:

- Filters don't calculate columns of right-to-left languages.
- Don't apply filters to LOB columns.
- Apply filters only to *immutable* columns, which aren't updated after creation. If source filters are applied to *mutable* columns, which can be updated after creation, adverse behavior can result.

For example, a filter to exclude or include specific rows in a column always excludes or includes the specified rows even if the rows are later changed. Suppose that you exclude or include rows 1–10 in column A, and they later change to become rows 11–20. In this case, they continue to be excluded or included even when the data is no longer the same.

Similarly, suppose that a row outside of the filter's scope is later updated (or updated and deleted), and should then be excluded or included as defined by the filter. In this case, it's replicated at the target.

## Creating source filter rules in JSON

You can create source filters using the JSON `filters` parameter of a selection rule. The `filters` parameter specifies an array of one or more JSON objects. Each object has parameters that specify the source filter type, column name, and filter conditions. These filter conditions include one or more filter operators and filter values.

The following table shows the parameters for specifying source filtering in a `filters` object.

Parameter	Value
<code>filter-type</code>	<code>source</code>
<code>column-name</code>	The name of the source column that you want the filter applied to. The name is case-sensitive.
<code>filter-conditions</code>	An array of one or more objects containing a <code>filter-operator</code> parameter and an appropriate <code>value</code> parameter, depending on the <code>filter-operator</code> value.
<code>filter-operator</code>	This parameter can have one of the following values: <ul style="list-style-type: none"> <li>• <code>lt</code> – less than or equal to</li> <li>• <code>gt</code> – greater than or equal to</li> <li>• <code>eq</code> – equal to</li> <li>• <code>between</code> – equal to or between two values</li> </ul>
<code>value</code> or	The value of the <code>filter-operator</code> parameter. If the <code>filter-operator</code> has a value other than <code>between</code> , use <code>value</code> . If the <code>filter-operator</code> has a value of <code>between</code> , use <code>value</code> for the first value and <code>value2</code> for the second value.

Parameter	Value
start-value	operator is set to between, provide two values, one for start-value and one for end-value.
end-value	

The following examples show some common ways to use source filters.

### Example Single filter

The following filter replicates all employees where empid  $\geq 100$  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 ( $empid \leq 10$ ) OR ( $empid$  is between 50 and 75) OR ( $empid \geq 100$ ) 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": "ste",
              "value": "10"
            },
            {
              "filter-operator": "between",
              "start-value": "50",
              "end-value": "75"
            }
          ]
        }
      ]
    }
  }
}
```

```

        },
        {
            "filter-operator": "gte",
            "value": "100"
        }
    ]
}
}

```

### 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": "ste",
                            "value": "100"
                        }
                    ],
                    {
                        "filter-type": "source",
                        "column-name": "dept",
                        "filter-conditions": [
                            {
                                "filter-operator": "eq",
                                "value": "tech"
                            }
                        ]
                    }
                ]
            }
        }
    ]
}

```

## 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](#) in the SQLite documentation.

The following example shows how to filter on a date. It replicates all employees where `empstartdate >= January 1, 2002` to the target database.

### Example Single date filter

```

{
    "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": "ge",
                            "value": "2002-01-01"
                        }
                    ]
                }
            ]
        }
    ]
}

```

```
"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"  
     ]  
    }]  
]
```

## Enabling and working with premigration assessments for a task

A premigration assessment evaluates specified components of a database migration task to help identify any problems that might prevent a migration task from running as expected. This assessment gives you a chance to identify issues before you run a new or modified task. You can then fix problems before they occur while running the migration task itself. This can avoid delays in completing a given database migration needed to repair data and your database environment.

AWS DMS provides access two different types of premigration assessments. The first type of premigration assessment, a premigration assessment run, is a functional superset of the second type, a data type assessment. They are described in the following topics:

**Note**

If you do a premigration assessment run that includes the data type assessment, you don't need to do a data type assessment separately.

1. [Specifying, starting, and viewing premigration assessment runs \(p. 354\)](#) – A premigration assessment run specifies one or more individual assessments to run based on a new or existing migration task configuration. Each individual assessment evaluates a specific element of a supported relational source or target database depending on considerations such as the migration type, supported objects, index configuration, and other task settings, such as table mappings that identify the schemas and tables to migrate. For example, an individual assessment might evaluate what source data types or primary key formats can and cannot be migrated, possibly based on the AWS DMS engine version. You can start and view the results of the latest assessment run and view the results of all prior assessment runs for a task either using the AWS DMS Management Console or using the AWS CLI and SDKs to access the AWS DMS API. You can also view the results of prior assessment runs for a task in an Amazon S3 bucket that you have selected for AWS DMS to store these results.

**Note**

The number and types of available individual assessments can increase over time. For more information on periodic updates, see [Specifying individual assessments \(p. 354\)](#).

2. [Starting and viewing data type assessments \(p. 359\)](#) – A data type assessment returns the results of a single type of premigration assessment in a single JSON structure: the data types that might not be migrated correctly in a supported relational source database instance. This report returns the results for all problem data types found in the columns of every schema and table in the source database that is mapped for migration. You can create and view the results of the latest data type assessment using the AWS CLI and SDKs to access the AWS DMS API. You can also view the results of the latest data type assessment using the AWS DMS Management Console. You can view the results of prior data type assessments in an Amazon S3 bucket for your account where AWS DMS stores these reports.

# Specifying, starting, and viewing premigration assessment runs

A premigration assessment run specifies one or more individual assessments to run based on a new or existing migration task configuration. Each individual assessment evaluates a specific element of the source or target database depending on considerations such as the migration type, supported objects, index configuration, and other task settings, such as table mappings to identify the schemas and tables to migrate. For example, an individual assessment might evaluate what source data types or primary key formats can and cannot be migrated, possibly based on the AWS DMS engine version.

## Specifying individual assessments

The table following provides a brief overview of the individual assessments that are applicable to a given migration task based on its configuration. You can choose from among the individual assessments to include in a new assessment run that are applicable to your task configuration.

Assessment name (console and API)	Description	Source must be relational	Target must be relational	Target must be Amazon ES	Target must be DynamoDB	Migration type must perform CDC	Applicable AWS DMS engine versions
Console – Unsupported data types  API – unsupported-data-types-in-source	Checks for data types unsupported by AWS DMS in the source endpoint. Not all data types can be migrated between engines.	–	–	–	–	–	All supported versions
Console – Large objects (LOBs) are used but target LOB columns are not nullable  API – full-lob-not-nullable-at-target	Checks for the nullability of a LOB column in the target when full LOB mode or inline LOB mode is used. AWS DMS requires a LOB column to be null when using these LOB modes.	X	X	–	–	–	All supported versions
Console – Source table with LOBs but without primary keys or unique constraints  API – table-with-lob-but-without-primary-key-or-unique-constraint	Checks for the presence of source tables with LOBs but without a primary key or a unique key. Currently, a table must have a primary key or a unique key for AWS DMS to migrate LOBs.	X	–	–	–	–	All supported versions

Assessment name (console and API)	Description	Source must be relational	Target must be relational	Target must be Amazon ES	Target must be DynamoDB	Migration type must perform CDC	Applicable AWS DMS engine versions
Console – Source table without primary key for CDC or full load and CDC tasks only  API – table-with-no-primary-key-or-unique-constraint	Checks for the presence of a primary key or a unique key in source tables for a full-load and change data capture (CDC) migration or a CDC-only migration. Lack of a primary key or a unique key can cause performance issues during the CDC migration.	X	-	-	-	X	All supported versions
Console – Target table without primary keys for CDC tasks only  API – target-table-has-unique-key-or-primary-key-for-cdc	Checks for the presence of a primary key or a unique key in already-created target tables for a CDC-only migration. Lack of a primary key or a unique key can cause full table scans in the target when AWS DMS applies updates and deletes resulting in performance issues during the CDC migration.	-	X	-	-	X	All supported versions
Console – Unsupported source primary key types - composite primary keys  API – unsupported-source-pk-type-for-dynamodb-target	Checks for the presence of composite primary keys in source tables when migrating to Amazon DynamoDB. The primary key of the source table must consist of a single column.	X	-	-	X	-	All supported versions

Assessment name (console and API)	Description	Source must be relational	Target must be relational	Target must be Amazon ES	Target must be DynamoDB	Migration type must perform CDC	Applicable AWS DMS engine versions
Console – Unsupported source primary key types - composite primary keys  API – unsupported-source-pk-type-for-elasticsearch-target	Checks for the presence of composite primary keys in source tables when migrating to Amazon Elasticsearch Service (Amazon ES). The primary key of the source table must consist of a single column.	X	–	X	–	–	Versions 3.3.2 and earlier

#### Note

AWS DMS versions 3.3.3 and later support migrating a source database to an Amazon ES target where the source primary key consists of multiple columns.

AWS DMS supports premigration assessment runs for the following relational databases:

- Oracle
- SQL Server
- PostgreSQL
- MySQL
- MariaDB
- Amazon Aurora

## Starting and viewing premigration assessment runs

You can start a premigration assessment run for a new or existing migration task using the AWS DMS Management Console, the AWS CLI, and the AWS DMS API.

### To start a premigration assessment run for a new or existing task

1. From the **Database migration tasks** page in the AWS DMS Management Console, do one of the following:

- Choose **Create task**. The **Create database migration task** page opens:
  1. Enter the task settings required to create your task, including table mapping.
  2. In the **Premigration assessment** section, select **Enable premigration assessment run**. The section expands with options to specify an assessment run for the new task.

#### Note

When creating a new task, enabling a premigration assessment run disables the option to start the task automatically on task creation. You can start the task manually after the assessment run completes.

- Choose the **Identifier** for an existing task on the **Database migration tasks** page. The task page for the chosen existing task opens:

1. Choose **Actions** and select **Create premigration assessment**. A **Create premigration assessment** page opens with options to specify an assessment run for the existing task.

These options include:

2. Enter a unique name for your assessment run.
3. Select the available individual assessments that you want to include in this assessment run. You can only select the available individual assessments based on your current task settings. By default, all available individual assessments are enabled and selected. All other individual assessments are disabled for this assessment run.
4. Search for and choose an Amazon S3 bucket and folder in your account to store your assessment result report.
5. Select or enter an IAM role with full account access to your chosen Amazon S3 bucket and folder.
6. Optionally choose a setting to encrypt the assessment result report in your Amazon S3 bucket.
7. Choose **Create task** for a new task or choose **Create** for an existing task.

The **Database migration tasks** page opens listing your new or modified task with a **Status of Creating...** and a banner message indicating that your premigration assessment run will start once the task is created.

AWS DMS provides access to the latest and all prior premigration assessment runs using the AWS DMS Management Console, the AWS CLI, or the AWS DMS API.

#### To view results for the latest assessment run

1. From the AWS DMS Management Console, choose the **Identifier** for your existing task on the **Database migration tasks** page. The task page for the existing task opens.
2. Choose the **Premigration assessments** tab on the existing task page. This opens a **Latest assessment results** section on that page showing results of the latest assessment run for this task.

These assessment run results start with the name of the latest assessment run and an overview of its status followed by a listing of the specified individual assessments and their status. You can then explore details of the status of each individual assessment by choosing its name in the list, with results available down to the table column level.

Both the status overview for an assessment run and each individual assessment shows a **Status** value. This value indicates the overall status of the assessment run and a similar status for each individual assessment. Following is a list of the **Status** values for the assessment run:

- "cancelling" – The assessment run was cancelled.
- "deleting" – The assessment run was deleted.
- "failed" – At least one individual assessment completed with a **failed** status.
- "error-provisioning" – An internal error occurred while resources were provisioned (during provisioning status).
- "error-executing" – An internal error occurred while individual assessments ran (during **running** status).
- "invalid state" – The assessment run is in an unknown state.
- "passed" – All individual assessments have completed, and none has a **failed** status.
- "provisioning" – Resources required to run individual assessments are being provisioned.
- "running" – Individual assessments are being run.
- "starting" – The assessment run is starting, but resources are not yet being provisioned for individual assessments.

Following is a list of the **Status** values for each individual assessment of the assessment run:

- "cancelled" – The individual assessment was cancelled as part of cancelling the assessment run.
- "error" – The individual assessment did not complete successfully.
- "failed" – The individual assessment completed successfully with a failed validation result: view the details of the result for more information.
- "invalid state" – The individual assessment is in an unknown state.
- "passed" – The individual assessment completed with a successful validation result.
- "pending" – The individual assessment is waiting to run.
- "running" – The individual assessment is running.
- "warning" – The individual assessment completed successfully with a warning validation result: view the details of the result for more information.

You can also view the JSON files for the assessment run results on Amazon S3.

#### To view the JSON files for the assessment run on Amazon S3

1. From the AWS DMS Management Console, choose the Amazon S3 bucket link shown in the status overview. This displays a list of bucket folders and other Amazon S3 objects stored in the bucket. If your results are stored in a bucket folder, open the folder.
2. You can find your assessment run results in several JSON files. A `summary.json` file contains the overall results of the assessment run. The remaining files are each named for an individual assessment that was specified for the assessment run, such as `unsupported-data-types-in-source.json`. These files each contain the results for the corresponding individual assessment from the chosen assessment run.

#### To view the results for all previous assessment runs

1. Choose **Previous assessment results** under the **Latest assessment results** section. This shows a list of the previous assessment runs listed by name in reverse chronological order.
2. Choose the name of the previous assessment run whose results you want to view. The status overview and individual assessment results for your chosen assessment run display in place of the **Latest assessment results** section.
3. You can then view the results for the chosen assessment run in the same way as for the latest assessment results shown initially.

To start and view the results of premigration assessment runs for an existing migration task, you can run the following CLI and AWS DMS API commands and operations:

- CLI: `describe-applicable-individual-assessments`, API: `DescribeApplicableIndividualAssessments` – Provides a list of individual assessments that you can specify for a new premigration assessment run, given one or more task configuration parameters.
- CLI: `start-replication-task-assessment-run`, API: `StartReplicationTaskAssessmentRun` – Starts a new premigration assessment run for one or more individual assessments of an existing migration task.
- CLI: `describe-replication-task-assessment-runs`, API: `DescribeReplicationTaskAssessmentRuns` – Returns a paginated list of premigration assessment runs based on filter settings.
- CLI: `describe-replication-task-individual-assessments`, API: `DescribeReplicationTaskIndividualAssessments` – Returns a paginated list of individual assessments based on filter settings.

- CLI: [cancel-replication-task-assessment-run](#), API: [CancelReplicationTaskAssessmentRun](#) – Cancels, but does not delete, a single premigration assessment run.
- CLI: [delete-replication-task-assessment-run](#), API: [DeleteReplicationTaskAssessmentRun](#) – Deletes the record of a single premigration assessment run.

## Starting and viewing data type assessments

A data type assessment identifies data types in a source database that might not get migrated correctly. During this assessment, AWS DMS reads the source database schemas for a migration task and creates a list of the column data types. It then compares this list to a predefined list of data types supported by AWS DMS. AWS DMS creates a report that you can look at to see if your migration task has any unsupported data types.

AWS DMS supports creating data type assessment reports for the following relational databases:

- Oracle
- SQL Server
- PostgreSQL
- MySQL
- MariaDB
- Amazon Aurora

You can start and view a data type assessment report using the CLI and SDKs to access the AWS DMS API:

- The CLI uses the [start-replication-task-assessment](#) command to start a data type assessment and uses the [describe-replication-task-assessment-results](#) command to view the latest data type assessment report in JSON format.
- The AWS DMS API uses the [StartReplicationTaskAssessment](#) operation to start a data type assessment and uses the [DescribeReplicationTaskAssessmentResults](#) operation to view the latest data type assessment report in JSON format.

You can also view the latest data type assessment report in the AWS DMS Management Console.

The data type assessment report is a single JSON file that includes a summary that lists the unsupported data types and the column count for each one. It includes a list of data structures for each unsupported data type including the schemas, tables, and columns that have the 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 appear on the report as not supported can't be migrated. Data types that appear on the report as partially supported might be converted to another data type, but not migrate as you expect.

The example following shows a sample data type assessment report that you might view.

```
{  
    "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"
                    ]
                }
            ]
        }
    ]
}
]

```

```
        "column10"
    ],
},
{
    "table-name": "table2",
    "columns": [
        "column11",
        "column12"
    ]
}
]
}
]
```

To view the latest data type assessment report from the AWS DMS Management Console, use the **Assessment results** tab on the task page that opens when you select the **Identifier** for a given task on the **Database migration tasks** page.

AWS DMS also stores the latest and all previous data type assessments in an Amazon S3 bucket created by AWS DMS in your account. The Amazon S3 bucket name has the following format, where *customerId* is your customer ID and *customerDNS* is an internal identifier.

dms-*customerId*-*customerDNS*

#### Note

By default, you can create up to 100 Amazon S3 buckets in each of your AWS accounts. Since AWS DMS creates a bucket in your account, make sure it does not exceed your bucket limit. Otherwise, the data type assessment will fail.

All data type assessment reports for a given migration task are stored in a bucket folder named with the task identifier. Each report's file name is the date of the data type assessment in the format yyyy-mm-dd-hh-mm. You can view and compare previous data type assessment reports from the Amazon S3 Management Console.

AWS DMS also creates an AWS Identity and Access Management (IAM) role to allow access to the S3 bucket created for these reports. The role name is dms-access-for-tasks. The role uses the AmazonDMSRedshiftS3Role policy.

## Specifying supplemental data for task settings

When you create or modify a replication task for some AWS DMS endpoints, the task might require additional information to perform the migration. You can specify this additional information using an option in the DMS console. Or you can specify it using the `TaskData` parameter for the DMS API operation `CreateReplicationTask` or `ModifyReplicationTask`.

If your target endpoint is Amazon Neptune, you need to specify mapping data, supplemental to table mapping. This supplemental mapping data specifies how to convert source relational data into the target graph data that a Neptune database can consume. In this case, you can use one of two possible formats. For more information, see [Specifying graph-mapping rules using Gremlin and R2RML for Amazon Neptune as a target \(p. 263\)](#).

# Monitoring AWS DMS tasks

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS DMS and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. AWS provides several tools for monitoring your AWS DMS tasks and resources, and responding to potential incidents:

## AWS DMS events and notifications

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. 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. 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. For more information, see [Working with events and notifications in AWS Database Migration Service \(p. 375\)](#)

## Task status

You can monitor the progress of your task by checking the task status and by monitoring the task's control table. Task status indicates the condition of a AWS DMS task and its associated resources. It includes such indications as if the task is being created, starting, running, or stopped. It also includes the current state of the tables that the task is migrating, such as if a full load of a table has begun or is in progress and details such as the number of inserts, deletes, and updates have occurred for the table. For more information about monitoring task and task resource condition, see [Task status \(p. 363\)](#) and [Table state during tasks \(p. 364\)](#). For more information about control tables, see [Control table task settings \(p. 285\)](#).

## Amazon CloudWatch alarms and logs

Using Amazon CloudWatch alarms, you watch one or more task metrics over a time period that you specify. If a metric exceeds a given threshold, a notification is sent to an Amazon SNS topic. CloudWatch alarms do not invoke actions because they are in a particular state. Rather the state must have changed and been maintained for a specified number of periods. AWS DMS also uses 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. For more information about using CloudWatch with AWS DMS, see [Monitoring replication tasks using Amazon CloudWatch \(p. 365\)](#). For more information about monitoring AWS DMS metrics, see [AWS Database Migration Service metrics \(p. 367\)](#). For more information about using AWS DMS task logs, see [Viewing and managing AWS DMS task logs \(p. 370\)](#).

## AWS CloudTrail logs

AWS DMS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, IAM 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 API operations. 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. For more information, see [Logging AWS DMS API calls with AWS CloudTrail \(p. 371\)](#).

## Database logs

You can view, download, and watch database logs for your task endpoints using the AWS Management Console, AWS CLI, or the API for your AWS database service. For more information, see the documentation for your database service at [AWS documentation](#).

For more information, see the following topics.

### Topics

- [Task status \(p. 363\)](#)
- [Table state during tasks \(p. 364\)](#)
- [Monitoring replication tasks using Amazon CloudWatch \(p. 365\)](#)
- [AWS Database Migration Service metrics \(p. 367\)](#)
- [Viewing and managing AWS DMS task logs \(p. 370\)](#)
- [Logging AWS DMS API calls with AWS CloudTrail \(p. 371\)](#)

# 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
<b>Creating</b>	AWS DMS is creating the task.
<b>Running</b>	The task is performing the migration duties specified.
<b>Stopped</b>	The task is stopped.
<b>Stopping</b>	The task is being stopped. This is usually an indication of user intervention in the task.
<b>Deleting</b>	The task is being deleted, usually from a request for user intervention.
<b>Failed</b>	The task has failed. See the task log files for more information.
<b>Starting</b>	The task is connecting to the replication instance and to the source and target endpoints. Any filters and transformations are being applied.
<b>Ready</b>	The task is ready to run. This status usually follows the "creating" status.
<b>Modifying</b>	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.

Note that the "last updated" column in 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.

## 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', 'Delete', and a refresh icon. Below this is a search bar labeled 'Filter' with a magnifying glass icon and a close button 'X'. A table lists two tasks:

ID	Status	Source	Target	Type
sg-mysql2pg-task1	Load complete	sg-mysql2pg-so	sg-mysql2pg-ta	Full Load
sg-mysql2pg-task2	Load complete	sg-mysql2pg-so	sg-mysql2pg-ta	Full Load

Below the table, the task 'sg-mysql2pg-task1' is selected. The navigation tabs at the bottom are 'Overview', 'Task monitoring', 'Table statistics' (which is highlighted with a red oval), and 'Logs'.

The screenshot shows the 'Table statistics' tab for the selected task. It displays a table of table states:

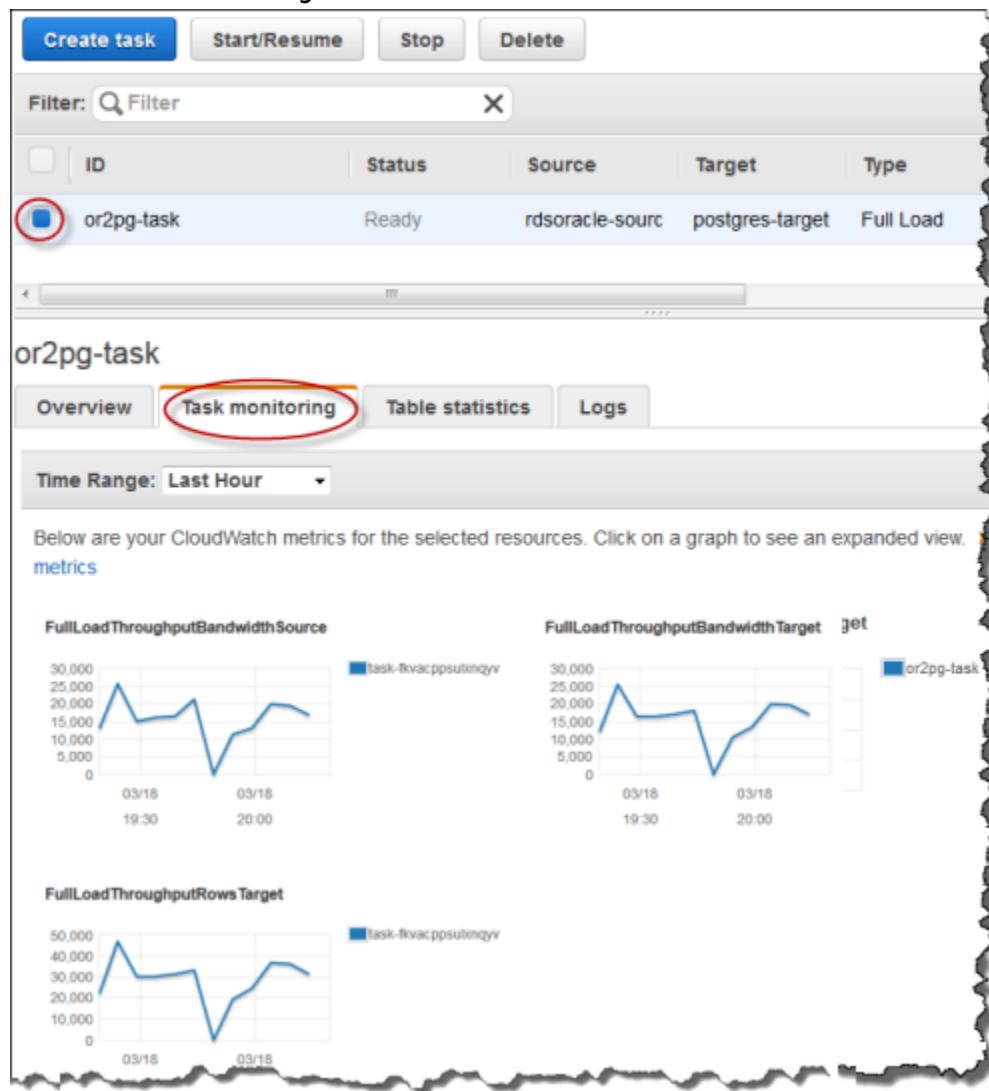
Schema	Table	State	Inserts	Deletes	Updates	Errors
EntertainmentAgencyModify	Agents	Table completed	0	0	0	0
EntertainmentAgencyModify	Customers	Table completed	0	0	0	0
EntertainmentAgencyModify	Engagements	Table completed	0	0	0	0

<b>State</b>	<b>Description</b>
<b>Table does not exist</b>	AWS DMS cannot find the table on the source endpoint.
<b>Before load</b>	The full load process has been enabled, but it hasn't started yet.
<b>Full load</b>	The full load process is in progress.
<b>Table completed</b>	Full load has completed.
<b>Table cancelled</b>	Loading of the table has been cancelled.
<b>Table error</b>	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-fkvacppslxnqyv

Overview Task monitoring **Table statistics** Logs

Schema	Table	State	Inserts	Deletes	Updates	DDLs	Full Load Rows	Total	Last updated
aat	T20	Full load	0	0	0	0	16,080,394	16,080,394	3/16 22:30
aat	T21	Full load	0	0	0	0	16,079,437	16,079,437	3/16 22:30
aat	T22	Full load	0	0	0	0	15,804,000	15,804,000	3/16 22:30

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.

Create replication instance Delete

Filter: Filter X

	Identifier	Status	VPC	Instance class	Availability zone	IP address
<input type="checkbox"/>	mh-test-test-test	available	vpc-fd8c1b99	dms.t2.medium	us-east-1a	52.0.245
<input checked="" type="checkbox"/>	or2pg-replserver	available	vpc-5ba9473e	dms.t2.large	us-east-1a	52.70.19
<input type="checkbox"/>	rdsor2rdspostgres	available	vpc-fd8c1b99	dms.t2.large	us-east-1b	52.2.183

or2pg-replserver

Overview **Monitoring**

Time Range: Last Hour

Below are your CloudWatch metrics for the selected resources. Click on a graph to see an expanded view. > [View all CloudWatch metrics](#)

CPUUtilization (Percent)

Free Storage Space (MB)

FreeableMemory (MB)

WriteIOPS (Count/Second)

# AWS Database 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 \(p. 367\)](#).
- **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 \(p. 368\)](#).
- **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: Percent

### 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 write I/O operations per second.

Units: Count/Second

#### **ReadIOPS**

The average number of disk read 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 data received from a full load from the source in kilobytes (KB) per second.

**FullLoadThroughputBandwidthTarget**

Outgoing data transmitted 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. You can view this metric together with CDCChangesDiskSource.

**CDCChangesMemoryTarget**

Amount of rows accumulating in a memory and waiting to be committed to the target. You can view this metric together with CDCChangesDiskTarget.

**CDCChangesDiskSource**

Amount of rows accumulating on disk and waiting to be committed from the source. You can view this metric together with CDCChangesMemorySource.

**CDCChangesDiskTarget**

Amount of rows accumulating on disk and waiting to be committed to the target. You can view this metric together with CDCChangesMemoryTarget.

**CDCThroughputBandwidthSource**

Incoming data received for the source in KB per second. CDCThroughputBandwidth records incoming data received on sampling points. If no task network traffic is found, the value is zero. Because CDC does not issue long-running transactions, network traffic may not be recorded.

**CDCThroughputBandwidthTarget**

Outgoing data transmitted for the target in KB per second. CDCThroughputBandwidth records outgoing data transmitted on sampling points. If no task 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. CDCLatencySource represents the latency between source and replication instance. High CDCLatencySource means the process of capturing changes from source is delayed. To identify latency in an ongoing replication, you can view this metric together with CDCLatencyTarget. If both CDCLatencySource and CDCLatencyTarget are high, investigate CDCLatencySource first.

### **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. CDCLatencyTarget represents the latency between replication instance and target. When CDCLatencyTarget is high, it indicates the process of applying change events to the target is delayed. To identify latency in an ongoing replication, you can view this metric together with CDCLatencySource. If CDCLatencyTarget is high but CDCLatencySource isn't high, investigate if:

- No primary keys or indexes are in the target
- Resource bottlenecks occur in the target or replication instance
- Network issues reside between replication instance and target

### **CPUUtilization**

The percentage of CPU being used by a task. Units: Percent

### **CPUAllocated**

The percentage of CPU maximally allocated for the task (0 means no limit). Units: Percent

### **MemoryAllocated**

The maximum allocation of memory for the task (0 means no limits). Units: MiB

### **SwapUsage**

The amount of swap used by the task. Units: Bytes

### **MemoryUsage**

The resident set size (RSS) occupied by a task. It indicates the portion of memory occupied by a task held in main memory (RAM). Since parts of the occupied memory are paged out, or parts of the executable are never loaded, MemoryUsage doesn't include memory held in swap space or file system.

### **FreeMemory**

The amount of physical memory available for use by applications, page cache, and for the kernel's own data structures. For more information, see `MemFree` value in `/proc/memInfo` section of the [Linux man-pages](#).

Units: Bytes

### **AvailableMemory**

An estimate of how much memory is available for starting new applications, without swapping. For more information, see `MemAvailable` value in `/proc/memInfo` section of the [Linux man-pages](#).

Units: Bytes

## Viewing and managing AWS DMS task logs

You can use Amazon CloudWatch to log task information during an AWS DMS migration process. You enable logging when you select task settings. For more information, see [Logging task settings \(p. 284\)](#).

To view logs of a task that ran, follow these steps:

1. Open the AWS DMS console, and choose **Database migration tasks** from the navigation pane. The Database migration tasks dialog appears.
2. Select the name of your task. The Overview details dialog appears.
3. Locate the **Migration task logs** section and choose **View CloudWatch Logs**.

In addition, you can use the AWS CLI or AWS DMS API to view information about 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:CDSFSFSFFFSSUFCAY
```

A sample response from the command is as follows.

```
{
  "ReplicationInstanceTaskLogs": [
    {
      "ReplicationTaskArn": "arn:aws:dms:us-east-1:237565436:task:MY34U6Z4MSY52GRTIX3O4AY",
      "ReplicationTaskName": "mysql-to-ddb",
      "ReplicationInstanceTaskLogSize": 3726134
    }
  ],
  "ReplicationInstanceArn": "arn:aws:dms:us-east-1:237565436:rep:CDSFSFSFFFSSUFCAY"
}
```

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.

**Note**

Stop the associated task from running before deleting a log. A log cannot be deleted using the AWS CLI or the AWS DMS Console while the associated task continues to run.

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 API operations. 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 AWS Regions. The trail logs events from all AWS 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 AWS 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": "ASIAFYFI33SINADOJJEZW",
    "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",
        "replicationInstanceState": "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"
}

```

# 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 \(p. 376\)](#), 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 deselecting the **Enabled** option 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 AWS Regions where Amazon SNS is supported. For a list of AWS Regions and countries where Amazon SNS supports SMS messaging, see [Supported Regions and countries](#). 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 \(p. 378\)](#).

# 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	The replication instance class for this replication instance is being changed.
Configuration Change	DMS-EVENT-0014	The replication instance class for this replication instance has changed.
Configuration Change	DMS-EVENT-0018	The storage for the replication instance is being increased.
Configuration Change	DMS-EVENT-0017	The storage for the replication instance has been increased.
Configuration Change	DMS-EVENT-0024	The replication instance is transitioning to a Multi-AZ configuration.
Configuration Change	DMS-EVENT-0025	The replication instance has finished transitioning to a Multi-AZ configuration.
Configuration Change	DMS-EVENT-0030	The replication instance is transitioning to a Single-AZ configuration.
Configuration Change	DMS-EVENT-0029	The replication instance has finished transitioning to a Single-AZ configuration.
Creation	DMS-EVENT-0067	A replication instance is being created.
Creation	DMS-EVENT-0005	A replication instance has been created.
Deletion	DMS-EVENT-0066	The replication instance is being deleted.
Deletion	DMS-EVENT-0003	The replication instance has been deleted.
Maintenance	DMS-EVENT-0047	Management software on the replication instance has been updated.
Maintenance	DMS-EVENT-0026	Offline maintenance of the replication instance is taking place. The replication instance is currently unavailable.
Maintenance	DMS-EVENT-0027	Offline maintenance of the replication instance is complete. The replication instance is now available.
Maintenance	DMS-EVENT-0068	Replication instance is in a state that cannot be upgraded.
LowStorage	DMS-EVENT-0007	Free storage for the replication instance is low.
Failover	DMS-EVENT-0013	Failover started for a Multi-AZ replication instance.

Category	DMS event ID	Description
Failover	DMS-EVENT-0049	Failover has been completed for a Multi-AZ replication instance.
Failover	DMS-EVENT-0015	Multi-AZ failover to standby complete.
Failover	DMS-EVENT-0050	Multi-AZ activation has started.
Failover	DMS-EVENT-0051	Multi-AZ activation completed.
Failover	DMS-EVENT-0034	If you request Failover too frequently, this event occurs instead of regular failover events.
Failure	DMS-EVENT-0031	The replication instance has gone into storage failure.
Failure	DMS-EVENT-0036	The replication instance has failed due to an incompatible network.
Failure	DMS-EVENT-0037	When service is unable to access the KMS key used to encrypt the data volume.

The following table shows the possible categories and events for the replication task source type.

Category	DMS event ID	Description
State Change	DMS-EVENT-0069	The replication task has started.
State Change	DMS-EVENT-0077	The replication task has started CDC.
State Change	DMS-EVENT-0081	Reload of table details has been requested.
State Change	DMS-EVENT-0079	The replication task has stopped.
Failure	DMS-EVENT-0078	A replication task has failed.
Failure	DMS-EVENT-0082	A call to clean task data has failed.
ConfigurationChange	DMS-EVENT-0080	A replication task has been modified.
Deletion	DMS-EVENT-0073	The replication task has been deleted.
Creation	DMS-EVENT-0074	The replication task has been created.

The following example models an AWS DMS event subscription with State Change category.

```

Resources:
  DMSEvent:
    Type: AWS::DMS::EventSubscription
    Properties:
      Enabled: true
      EventCategories: State Change
      SnsTopicArn: arn:aws:sns:us-east-1:123456789:testSNS
      SourceIds: []
      SourceType: replication-task
  
```

# 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. In a notification subscription, you choose how and where to send notifications. You specify the type of source you want to be notified of; currently AWS DMS supports the replication instance and replication task source types. And, depending on the source type you select, you choose the event categories and identify the source you want to receive event notifications for.

## Using 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. Under **Details**, for **Name**, type a name for the event notification subscription.
  - b. Select **Enabled** to enable the subscription. If you want to create the subscription but not have notifications sent yet, don't select **Enabled**.
  - c. Under **Target**, choose either **Existing topics**, **Create new email topic** or **Create new SMS topic** to send notifications. You must have either an existing Amazon SNS topic to send notices to or you must create the topic. If you choose to create a topic, you can enter an email address where notifications will be sent.
  - d. Under **Event source**, for **Source type**, choose a source type. The only options are **replication-instance** and **replication-task**.
  - e. Depending on the source type you selected, choose the event categories and sources you want to receive event notifications for.

## Create event subscription

### Details

#### Name

The name for your event subscription

EventSubscriptionExample

Enabled

### Target

Send notification to

- Existing topics
- Create new email topic
- Create new SMS topic

#### Topic name

ProdEventSubscription

#### With these recipients

Email addresses or phone numbers of SMS enabled devices to send the notifications to

user@domain.com

### Event source

#### Source type

Source Type of resource this subscription will consume events from

replication-instance

API Version API Version 2016-01-01

Event categories 379

All event categories

Select specific event categories

- f. Select **Create event subscription**.

The AWS DMS console indicates that the subscription is being created.

## Using AWS DMS API and CLI

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 created using the AWS DMS API or CLI, 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 type of source by specifying a *source type* value. You define the source generating the event by specifying a *source identifier* value.

The following `create-event-subscription` example shows the syntax to create event notification subscriptions using the AWS CLI.

```
aws dms create-event-subscription \
  --subscription-name string \
  --sns-topic-arn string \
  [--source-type string] \
  [--event-categories string] \
  [--source-ids string] \
  [--enabled | --no-enabled] \
  [--tags string] \
  [--cli-input-json string] \
  [--generate-cli-skeleton string]
```

To subscribe to AWS DMS event notification using the AWS DMS API, call the `CreateEventSubscription` action. The following provides an example request syntax for the `CreateEventSubscription` API.

```
{
  "Enabled": boolean,
  "EventCategories": [ "string" ],
  "SnsTopicArn": "string",
  "SourceIds": [ "string" ],
  "SourceType": "string",
  "SubscriptionName": "string",
  "Tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ]
}
```

# AWS DMS data validation

## Topics

- [Using JSON editor to modify validation rules \(p. 382\)](#)
- [Replication task statistics \(p. 382\)](#)
- [Replication task statistics with Amazon CloudWatch \(p. 384\)](#)
- [Revalidating tables during a task \(p. 384\)](#)
- [Troubleshooting \(p. 385\)](#)
- [Limitations \(p. 386\)](#)

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 wherever AWS DMS supports them as source and target endpoints:

- Oracle
- PostgreSQL
- MySQL
- MariaDB
- Microsoft SQL Server
- Amazon Aurora (MySQL)
- Amazon Aurora (PostgreSQL)
- IBM Db2 LUW

For more information about the supported endpoints, see [Working with AWS DMS endpoints \(p. 63\)](#).

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:

- `EnableValidation` – Enables or disables data validation.
- `FailureMaxCount` – Specifies the maximum number of records that can fail validation before validation is suspended for the task.
- `HandleCollationDiff` – Accounts for column collation differences in PostgreSQL endpoints when identifying source and target records to compare.
- `RecordFailureDelayLimitInMinutes` – Specifies the delay before reporting any validation failure details.

- **TableFailureMaxCount** – Specifies the maximum number of tables that can fail validation before validation is suspended for the task.
- **ThreadCount** – Adjusts the number of execution threads that AWS DMS uses during validation.
- **ValidationOnly** – Previews the validation for the task without performing any migration or replication of data. To use this option, set the task migration type to **Replicate data changes only** in the AWS DMS console, or set the migration type to **cdc** in the AWS DMS API. In addition, set the target table task setting, **TargetTablePrepMode**, to **DO NOTHING**.

The following JSON example turns on validation, increases the number of threads to eight, and suspends validation if any table has a validation failure.

```
ValidationSettings": {  
    "EnableValidation":true,  
    "ThreadCount":8,  
    "TableFailureMaxCount":1  
}
```

For more information about these settings, see [Data validation task settings \(p. 289\)](#).

## Using JSON editor to modify validation rules

To add a validation rule to a task using the JSON editor from the AWS DMS Console Dashboard, do the following:

1. Select **Database migration tasks**.
2. Select your task from the list of migration tasks.
3. From the **Actions** drop down menu, select **Stop**.
4. Once the task has stopped, from the **Actions** drop down menu, select **Modify**.
5. Choose the **Mapping rules** tab.
6. Select **Mapping rules (JSON)** and add your validation rule in your table mappings.

## 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_v1` 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 \(p. 296\)](#)
- **ValidationFailed**—The number of records that didn't pass the data validation phase. For more information, see [Error handling task settings \(p. 296\)](#).

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:VFPFTYKK2RYSI",
  "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.

## Replication task statistics with Amazon CloudWatch

When Amazon CloudWatch is enabled, AWS DMS provides the following replication task statistics:

- **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.

To collect data validation information from CloudWatch enabled statistics, select **Enable CloudWatch logs** when you create or modify a task using the console. Then, to view the data validation information and ensure that your data was migrated accurately from source to target, do the following.

1. Choose the task on the **Database migration tasks** page.
2. Choose the **CloudWatch metrics** tab.
3. Select **Validation** from the drop down menu.

## 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 \(p. 416\)](#).

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 (you can choose up to 10 tables at one time). If the task is no longer running, you can't revalidate the table(s).
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) NOT NULL</code>	AWS DMS task identifier.
<code>TABLE_OWNER</code>	<code>VARCHAR(128) NOT NULL</code>	Schema (owner) of the table.
<code>TABLE_NAME</code>	<code>VARCHAR(128) NOT NULL</code>	Table name.
<code>FAILURE_TIME</code>	<code>DATETIME(3) NOT NULL</code>	Time when the failure occurred.
<code>KEY_TYPE</code>	<code>VARCHAR(128) NOT NULL</code>	Reserved for future use (value is always 'Row')
<code>KEY</code>	<code>TEXT NOT NULL</code>	This is the primary key for row record type.
<code>FAILURE_TYPE</code>	<code>VARCHAR(128) NOT NULL</code>	Severity of validation error. Can be either <code>RECORD_DIFF</code> , <code>MISSING_SOURCE</code> or <code>MISSING_TARGET</code> .
<code>DETAILS</code>	<code>VARCHAR(8000) NOT NULL</code>	JSON formatted string of all source/target column values which do not match for the given key.

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'
TASK_NAME      VFPFKH4FJR3FTYKK2RYSI
```

```
TABLE_OWNER      DB2PERF
TABLE_NAME       PERFTEST
FAILURE_TIME    2020-06-11 21:58:44
KEY_TYPE        Row
KEY              {"key": ["3451491"]}
FAILURE_TYPE    RECORD_DIFF
DETAILS          [{"MYREAL": "+1.10106036e-01"}, {"MYREAL": "+1.10106044e-01"}], ]
```

You can look at the `DETAILS` field to determine which columns don't match. Since 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 LOBs. If your Oracle endpoint uses LOBs, then you must grant the execute permission on dbms\_crypto to the user account 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:

- Certificates
- Endpoints
- Event subscriptions
- Replication instances
- Replication subnet (security) groups
- Replication tasks

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 a 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 \(p. 16\)](#).

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 \(p. 16\)](#).

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.

Tagging element	Description
Key	<p>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: "<math>^([\u00p{L}\u00p{Z}\u00p{N}_.:/+=\u00p{-}]*)\$</math>").</p> <p>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.</p>
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: "<math>^([\u00p{L}\u00p{Z}\u00p{N}_.:/+=\u00p{-}]*)\$</math>").</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>

# Security in AWS Database Migration Service

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security of the cloud and security *in the cloud*:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS compliance programs](#). To learn about the compliance programs that apply to AWS DMS, see [AWS services in scope by compliance program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS DMS. The following topics show you how to configure AWS DMS to meet your security and compliance objectives. You also learn how to use other AWS services that help you monitor and secure your AWS DMS resources.

You can manage access to your AWS DMS resources and your databases (DBs). The method you use to manage access depends on the replication task you need to perform with AWS DMS:

- Use AWS Identity and Access Management (IAM) policies to assign permissions that determine who is allowed to manage AWS DMS resources. AWS DMS requires that you have the appropriate permissions if you sign in as an IAM user. For example, you can use IAM to determine who is allowed to create, describe, modify, and delete DB instances and clusters, tag resources, or modify security groups. For more information about IAM and using it with AWS DMS, see [Identity and access management for AWS Database Migration Service \(p. 394\)](#).
- AWS DMS uses Secure Sockets Layer (SSL) for your endpoint connections with Transport Layer Security (TLS). For more information about using SSL/TLS with AWS DMS, see [Using SSL with AWS Database Migration Service \(p. 435\)](#).
- AWS DMS uses AWS Key Management Service (AWS KMS) encryption keys to encrypt the storage used by your replication instance and its endpoint connection information. AWS DMS also uses AWS KMS encryption keys to secure your target data at rest for Amazon S3 and Amazon Redshift target endpoints. For more information, see [Setting an encryption key and specifying AWS KMS permissions \(p. 431\)](#).
- AWS DMS always creates your replication instance in a virtual private cloud (VPC) based on the Amazon VPC service for the greatest possible network access control. For your DB instances and instance clusters, use the same VPC as your replication instance, or additional VPCs to match this level of access control. Each Amazon VPC that you use 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.

For more information about available network configurations for AWS DMS, see [Setting up a network for a replication instance \(p. 45\)](#). For more information about creating a DB instance or instance cluster

in a VPC, see the security and cluster management documentation for your Amazon databases at [AWS documentation](#). For more information about network configurations that AWS DMS supports, see [Setting up a network for a replication instance \(p. 45\)](#).

- To view database migration logs, you need the appropriate Amazon CloudWatch Logs permissions for the IAM role you are using. For more information about logging for AWS DMS, see [Monitoring replication tasks using Amazon CloudWatch \(p. 365\)](#).

## Topics

- [Data protection in AWS Database Migration Service \(p. 392\)](#)
- [Identity and access management for AWS Database Migration Service \(p. 394\)](#)
- [Compliance validation for AWS Database Migration Service \(p. 413\)](#)
- [Resilience in AWS Database Migration Service \(p. 414\)](#)
- [Infrastructure security in AWS Database Migration Service \(p. 415\)](#)
- [IAM permissions needed to use AWS DMS \(p. 416\)](#)
- [Creating the IAM roles to use with the AWS CLI and AWS DMS API \(p. 420\)](#)
- [Fine-grained access control using resource names and tags \(p. 424\)](#)
- [Setting an encryption key and specifying AWS KMS permissions \(p. 431\)](#)
- [Network security for AWS Database Migration Service \(p. 433\)](#)
- [Using SSL with AWS Database Migration Service \(p. 435\)](#)
- [Changing the database password \(p. 439\)](#)

# Data protection in AWS Database Migration Service

AWS DMS conforms to the AWS [shared responsibility model](#), which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and Amazon Partner Network (APN) partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection, we recommend that you protect AWS account credentials and set up principals with AWS Identity and Access Management (IAM). Doing this means that each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields of a database. This recommendation applies especially when you work with AWS database services used as AWS DMS source endpoints from the console, API, AWS CLI, or AWS SDKs. Any data that you enter into these database services might get picked up for inclusion in diagnostic logs. Also, when you provide a URL to an external server, don't include credentials of any kind in the URL to validate your request to that server.

For more information about data protection, see the [AWS shared responsibility model and GDPR blog post](#) on the AWS Security Blog.

## Data encryption

You can enable encryption for data resources of supported AWS DMS target endpoints. AWS DMS also encrypts connections to AWS DMS and between AWS DMS and all its source and target endpoints. In addition, you can manage the keys that AWS DMS and its supported target endpoints use to enable this encryption.

### Topics

- [Encryption at rest \(p. 392\)](#)
- [Encryption in transit \(p. 393\)](#)
- [Key management \(p. 393\)](#)

## Encryption at rest

AWS DMS supports encryption at rest by allowing you to specify the server-side encryption mode that you want used to push your replicated data to Amazon S3 before it is copied to supported AWS DMS target endpoints. You can specify this encryption mode by setting the `encryptionMode` extra connection attribute for the endpoint. If this `encryptionMode` setting specifies KMS key encryption mode, you can also create custom KMS keys specifically to encrypt the target data for the following AWS DMS target endpoints:

- Amazon Redshift – For more information about setting encryptionMode, see [Extra connection attributes when using Amazon Redshift as a target for AWS DMS \(p. 180\)](#). For more information about creating a custom KMS encryption key, see [Creating and using AWS KMS keys to encrypt Amazon Redshift target data \(p. 176\)](#).
- Amazon S3 – For more information about setting encryptionMode, see [Extra connection attributes when using Amazon S3 as a target for AWS DMS \(p. 200\)](#). For more information about creating a custom KMS encryption key, see [Creating AWS KMS keys to encrypt Amazon S3 target objects \(p. 194\)](#).

## Encryption in transit

AWS DMS supports encryption in transit by ensuring that the data it replicates moves securely from the source endpoint to the target endpoint. This includes encrypting an S3 bucket on the replication instance that your replication task uses for intermediate storage as the data moves through the replication pipeline. To encrypt task connections to source and target endpoints AWS DMS uses Secure Socket Layer (SSL) with Transport Layer Security (TLS). By encrypting connections to both endpoints, AWS DMS ensures that your data is secure as it moves both from the source endpoint to your replication task and from your task to the target endpoint. For more information about using SSL/TLS with AWS DMS, see [Using SSL with AWS Database Migration Service \(p. 435\)](#)

AWS DMS supports both default and custom keys to encrypt both intermediate replication storage and connection information. You manage these keys by using AWS KMS. For more information, see [Setting an encryption key and specifying AWS KMS permissions \(p. 431\)](#).

## Key management

AWS DMS supports default or custom keys to encrypt replication storage, connection information, and the target data storage for certain target endpoints. You manage these keys by using AWS KMS. For more information, see [Setting an encryption key and specifying AWS KMS permissions \(p. 431\)](#).

## Internetwork traffic privacy

Connections are provided with protection between AWS DMS and source and target endpoints in the same AWS Region, whether running on premises or as part of an AWS service in the cloud. (At least one endpoint, source or target, must run as part of an AWS service in the cloud.) This protection applies whether these components share the same virtual private cloud (VPC) or exist in separate VPCs, if the VPCs are all in the same AWS Region. For more information about the supported network configurations for AWS DMS, see [Setting up a network for a replication instance \(p. 45\)](#). For more information about the security considerations when using these network configurations, see [Network security for AWS Database Migration Service \(p. 433\)](#).

# Identity and access management for AWS Database Migration Service

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS DMS resources. IAM is an AWS service that you can use with no additional charge.

## Topics

- [Audience \(p. 394\)](#)
- [Authenticating with identities \(p. 394\)](#)
- [Managing access using policies \(p. 396\)](#)
- [How AWS Database Migration Service works with IAM \(p. 397\)](#)
- [AWS Database Migration Service identity-based policy examples \(p. 402\)](#)
- [Resource-based policy examples for AWS KMS \(p. 407\)](#)
- [Troubleshooting AWS Database Migration Service identity and access \(p. 410\)](#)

## Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in AWS DMS.

**Service user** – If you use the AWS DMS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS DMS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS DMS, see [Troubleshooting AWS Database Migration Service identity and access \(p. 410\)](#).

**Service administrator** – If you're in charge of AWS DMS resources at your company, you probably have full access to AWS DMS. It's your job to determine which AWS DMS features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS DMS, see [How AWS Database Migration Service works with IAM \(p. 397\)](#).

**IAM administrator** – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS DMS. To view example AWS DMS identity-based policies that you can use in IAM, see [AWS Database Migration Service identity-based policy examples \(p. 402\)](#).

## Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see [The IAM Console and Sign-in Page](#) in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the [AWS Management Console](#), use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If

you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see [Signature Version 4 Signing Process](#) in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Using Multi-Factor Authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

## AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the [best practice of using the root user only to create your first IAM user](#). Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

## IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see [Managing Access Keys for IAM Users](#) in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [When to Create an IAM User \(Instead of a Role\)](#) in the *IAM User Guide*.

## IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by [switching roles](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Using IAM Roles](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an *identity provider*. For more information about federated users, see [Federated Users and Roles](#) in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access.

However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [How IAM Roles Differ from Resource-based Policies](#) in the *IAM User Guide*.

- **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see [Creating a Role to Delegate Permissions to an AWS Service](#) in the *IAM User Guide*.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances](#) in the *IAM User Guide*.

To learn whether to use IAM roles, see [When to Create an IAM Role \(Instead of a User\)](#) in the *IAM User Guide*.

## Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON Policies](#) in the *IAM User Guide*.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

## Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM Policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choosing Between Managed Policies and Inline Policies](#) in the *IAM User Guide*.

## Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

## Access control lists (ACLs)

Access control lists (ACLs) are a type of policy that controls which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format. Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access Control List \(ACL\) Overview](#) in the *Amazon Simple Storage Service Developer Guide*.

## Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions Boundaries for IAM Entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see [How SCPs Work](#) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see [Session Policies](#) in the *IAM User Guide*.

## Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy Evaluation Logic](#) in the *IAM User Guide*.

# How AWS Database Migration Service works with IAM

Before you use IAM to manage access to AWS DMS, you should understand what IAM features are available to use with AWS DMS. To get a high-level view of how AWS DMS and other AWS services work with IAM, see [AWS services that work with IAM](#) in the *IAM User Guide*.

### Topics

- [AWS DMS identity-based policies](#) (p. 398)

- [AWS DMS resource-based policies \(p. 400\)](#)
- [Authorization based on AWS DMS tags \(p. 400\)](#)
- [IAM roles for AWS DMS \(p. 400\)](#)

## AWS DMS identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources, and also the conditions under which actions are allowed or denied. AWS DMS supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

### Actions

The Action element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in AWS DMS use the following prefix before the action: dms:. For example, to grant someone permission to create a replication task with the AWS DMS CreateReplicationTask API operation, you include the dms:CreateReplicationTask action in their policy. Policy statements must include either an Action or NotAction element. AWS DMS defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows.

```
"Action": [  
    "dms:action1",  
    "dms:action2"]
```

You can specify multiple actions using wildcards (\*). For example, to specify all actions that begin with the word `Describe`, include the following action.

```
"Action": "dms:Describe*"
```

To see a list of AWS DMS actions, see [Actions Defined by AWS Database Migration Service](#) in the *IAM User Guide*.

### Resources

The Resource element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. You specify a resource using an ARN or using the wildcard (\*) to indicate that the statement applies to all resources.

AWS DMS works with the resources following:

- Certificates
- Endpoints
- Event subscriptions
- Replication instances
- Replication subnet (security) groups
- Replication tasks

The resource or resources that AWS DMS requires depends on the action or actions that you invoke. You need a policy that permits these actions on the associated resource or resources specified by the resource ARNs.

For example, an AWS DMS endpoint resource has the following ARN:

```
arn:${Partition}:dms:${Region}:${Account}:endpoint/${InstanceId}
```

For more information about the format of ARNs, see [Amazon Resource Names \(ARNs\) and AWS service namespaces](#).

For example, to specify the `1A2B3C4D5E6F7G8H9I0J1K2L3M` endpoint instance for the `use-east-2` region in your statement, use the following ARN.

```
"Resource": "arn:aws:dms:us-east-2:987654321098:endpoint/1A2B3C4D5E6F7G8H9I0J1K2L3M"
```

To specify all endpoints that belong to a specific account, use the wildcard (\*).

```
"Resource": "arn:aws:dms:us-east-2:987654321098:endpoint/*"
```

Some AWS DMS actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (\*).

```
"Resource": "*"
```

Some AWS DMS API actions involve multiple resources. For example, `StartReplicationTask` starts and connects a replication task to two database endpoint resources, a source and a target, so an IAM user must have permissions to read the source endpoint and to write to the target endpoint. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [  
    "resource1",  
    "resource2" ]
```

For more information on controlling access to AWS DMS resources using policies, see [??? \(p. 424\)](#). To see a list of AWS DMS resource types and their ARNs, see [Resources Defined by AWS Database Migration Service](#) in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see [Actions Defined by AWS Database Migration Service](#).

## Condition keys

The Condition element (or Condition block) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple Condition elements in a statement, or multiple keys in a single Condition element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM Policy Elements: Variables and Tags](#) in the *IAM User Guide*.

AWS DMS defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

AWS DMS defines a set of standard tags that you can use in its condition keys and also allows you to define your own custom tags. For more information, see [Using tags to control access \(p. 426\)](#).

To see a list of AWS DMS condition keys, see [Condition Keys for AWS Database Migration Service](#) in the *IAM User Guide*. To learn which actions and resources you can use a condition key, see [Actions Defined by AWS Database Migration Service](#) and [Resources Defined by AWS Database Migration Service](#).

## Examples

To view examples of AWS DMS identity-based policies, see [AWS Database Migration Service identity-based policy examples \(p. 402\)](#).

## AWS DMS resource-based policies

Resource-based policies are JSON policy documents that specify what actions a specified principal can perform on a given AWS DMS resource and under what conditions. AWS DMS supports resource-based permissions policies for AWS KMS encryption keys that you create to encrypt data migrated to supported target endpoints. The supported target endpoints include Amazon Redshift and Amazon S3. By using resource-based policies, you can grant the permission for using these encryption keys to other accounts for each target endpoint.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the [principal in a resource-based policy](#). Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, you must also grant the principal entity permission to access the resource. Grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.

The AWS DMS service supports only one type of resource-based policy called a *key policy*, which is attached to an AWS KMS encryption key. This policy defines which principal entities (accounts, users, roles, and federated users) can encrypt migrated data on the supported target endpoint.

To learn how to attach a resource-based policy to an encryption key that you create for the supported target endpoints, see [Creating and using AWS KMS keys to encrypt Amazon Redshift target data \(p. 176\)](#) and [Creating AWS KMS keys to encrypt Amazon S3 target objects \(p. 194\)](#).

## Examples

For examples of AWS DMS resource-based policies, see [Resource-based policy examples for AWS KMS \(p. 407\)](#).

## Authorization based on AWS DMS tags

You can attach tags to AWS DMS resources or pass tags in a request to AWS DMS. To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `dms:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition key. AWS DMS defines a set of standard tags that you can use in its condition keys and also enables you to define your own custom tags. For more information, see [Using tags to control access \(p. 426\)](#).

For an example identity-based policy that limits access to a resource based on tags, see [Accessing AWS DMS resources based on tags \(p. 407\)](#).

## IAM roles for AWS DMS

An [IAM role](#) is an entity within your AWS account that has specific permissions.

## Using temporary credentials with AWS DMS

You can use temporary credentials to sign in with federation, assume an IAM role, or assume a cross-account role. You get temporary security credentials by calling AWS STS API operations such as [AssumeRole](#) or [GetFederationToken](#).

AWS DMS supports using temporary credentials.

## Service-linked roles

[Service-linked roles](#) allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

AWS DMS does not support service-linked roles.

## Service roles

This feature allows a service to assume a [service role](#) on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

AWS DMS supports two types of service roles that you must create to use certain source or target endpoints:

- Roles with permissions to allow AWS DMS access to the following source and target endpoints (or their resources):
  - Amazon DynamoDB as a target – For more information see [Prerequisites for using DynamoDB as a target for AWS Database Migration Service \(p. 209\)](#).
  - Elasticsearch as a target – For more information see [Prerequisites for using Amazon Elasticsearch Service as a target for AWS Database Migration Service \(p. 244\)](#).
  - Amazon Kinesis as a target – For more information see [Prerequisites for using a Kinesis data stream as a target for AWS Database Migration Service \(p. 226\)](#).
  - Amazon Redshift as a target – You need to create the specified role only for creating a custom KMS encryption key to encrypt the target data or for specifying a custom S3 bucket to hold intermediate task storage. For more information, see [Creating and using AWS KMS keys to encrypt Amazon Redshift target data \(p. 176\)](#) or [Amazon S3 bucket settings \(p. 180\)](#).
  - Amazon S3 as a source or as a target – For more information, see [Prerequisites when using Amazon S3 as a source for AWS DMS \(p. 149\)](#) or [Prerequisites for using Amazon S3 as a target \(p. 188\)](#).

For example, to read data from an S3 source endpoint or to push data to an S3 target endpoint, you must create a service role as a prerequisite to accessing S3 for each of these endpoint operations.

- Roles with permissions required to use the AWS CLI and AWS DMS API – Two IAM roles that you need to create are `dms-vpc-role` and `dms-cloudwatch-logs-role`. If you use Amazon Redshift as a target database, you must also create and add the IAM role `dms-access-for-endpoint` to your AWS account. For more information, see [Creating the IAM roles to use with the AWS CLI and AWS DMS API \(p. 420\)](#).

## Choosing an IAM role in AWS DMS

If you use the AWS CLI or the AWS DMS API for your database migration, you must add certain 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. For more information, see [Creating the IAM roles to use with the AWS CLI and AWS DMS API \(p. 420\)](#).

# AWS Database Migration Service identity-based policy examples

By default, IAM users and roles don't have permission to create or modify AWS DMS resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see [Creating policies on the JSON tab](#) in the *IAM User Guide*.

## Topics

- [Policy best practices \(p. 402\)](#)
- [Using the AWS DMS console \(p. 402\)](#)
- [Allow users to view their own permissions \(p. 405\)](#)
- [Accessing one Amazon S3 bucket \(p. 406\)](#)
- [Accessing AWS DMS resources based on tags \(p. 407\)](#)

## Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete AWS DMS resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get Started Using AWS Managed Policies** – To start using AWS DMS quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see [Get Started Using Permissions With AWS Managed Policies](#) in the *IAM User Guide*.
- **Grant Least Privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see [Grant Least Privilege](#) in the *IAM User Guide*.
- **Enable MFA for Sensitive Operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see [Using Multi-Factor Authentication \(MFA\) in AWS](#) in the *IAM User Guide*.
- **Use Policy Conditions for Extra Security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see [IAM JSON Policy Elements: Condition](#) in the *IAM User Guide*.

## Using the AWS DMS console

The policy following gives you access to AWS DMS, including the AWS DMS console, and also specifies permissions for certain actions needed from other Amazon services such as Amazon EC2.

```
{  
    "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 required for using the console is necessary.

The following section is required to allow the user to list their available AWS KMS keys and alias for display in the console. This entry is not required if you know the Amazon Resource Name (ARN) for the KMS key and you are using only the AWS Command Line Interface (AWS CLI).

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

The following 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 aren't 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. 420).

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

The following section is required because AWS DMS needs to create the Amazon 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": "*"  
}
```

The following 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": "*"
}

```

The following section is required when using Amazon Redshift as a target. It allows AWS DMS to validate that the Amazon 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 about adding these roles, see [Creating the IAM roles to use with the AWS CLI and AWS DMS API \(p. 420\)](#).

For more information on the requirements for using this policy to access AWS DMS, see [IAM permissions needed to use AWS DMS \(p. 416\)](#)

## Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam>ListGroupsForUser",
                "iam>ListAttachedUserPolicies",
                "iam>ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
    ]
}

```

```
{
    "Sid": "NavigateInConsole",
    "Effect": "Allow",
    "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam>ListAttachedGroupPolicies",
        "iam>ListGroupPolicies",
        "iam>ListPolicyVersions",
        "iam>ListPolicies",
        "iam>ListUsers"
    ],
    "Resource": "*"
}
]
```

## Accessing one Amazon S3 bucket

AWS DMS uses Amazon S3 buckets as intermediate storage for database migration. Typically, AWS DMS manages default S3 buckets for this purpose. However, in certain cases, especially when you use the AWS CLI or the AWS DMS API, AWS DMS enables you to specify your own S3 bucket instead. For example, you can specify your own S3 bucket for migrating data to an Amazon Redshift target endpoint. In this case, you need to create a role with permissions based on the Amazon-managed `AmazonDMSRedshiftS3Role` policy.

The example following shows a version of the `AmazonDMSRedshiftS3Role` policy. It allows AWS DMS to grant an IAM user in your AWS account access to one of your Amazon S3 buckets. It also allows the user to add, update, and delete objects.

In addition to granting the `s3:PutObject`, `s3:GetObject`, and `s3>DeleteObject` permissions to the user, the policy also grants the `s3>ListAllMyBuckets`, `s3:GetBucketLocation`, and `s3>ListBucket` permissions. These are the additional permissions required by the console. Other permissions allow AWS DMS to manage the bucket life cycle. Also, the `s3:GetObjectAcl` action is required to be able to copy objects.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3>CreateBucket",
                "s3>ListBucket",
                "s3>DeleteBucket",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:PutObject",
                "s3>DeleteObject",
                "s3:GetObjectVersion",
                "s3:GetBucketPolicy",
                "s3:PutBucketPolicy",
                "s3:GetBucketAcl",
                "s3:PutBucketVersioning",
                "s3:GetBucketVersioning",
                "s3:PutLifecycleConfiguration",
                "s3:GetLifecycleConfiguration",
                "s3>DeleteBucketPolicy"
            ],
            "Resource": "arn:aws:s3:::dms-*"
        }
    ]
}
```

```
    ]  
}
```

For more information on creating a role based on this policy, see [Amazon S3 bucket settings \(p. 180\)](#).

## Accessing AWS DMS resources based on tags

You can use conditions in your identity-based policy to control access to AWS DMS resources based on tags. This example shows how you might create a policy that allows access to all AWS DMS endpoints. However, permission is granted only if the endpoint database tag `Owner` has the value of that user's user name.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "dms:*",  
            "Resource": "arn:aws:dms:*:*:endpoint/*",  
            "Condition": {  
                "StringEquals": {"dms:endpoint-tag/Owner": "${aws:username}"}  
            }  
        }  
    ]  
}
```

You can attach this policy to the IAM users in your account. If a user named `richard-roe` attempts to access an AWS DMS endpoint, the endpoint database must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise, this user is denied access. The condition tag key `Owner` matches both `Owner` and `owner` because condition key names are not case-sensitive. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.

## Resource-based policy examples for AWS KMS

AWS DMS allows you to create custom AWS KMS encryption keys to encrypt supported target endpoint data. To learn how to create and attach a key policy to the encryption key you create for supported target data encryption, see [Creating and using AWS KMS keys to encrypt Amazon Redshift target data \(p. 176\)](#) and [Creating AWS KMS keys to encrypt Amazon S3 target objects \(p. 194\)](#).

### Topics

- [A policy for a custom AWS KMS encryption key to encrypt Amazon Redshift target data \(p. 407\)](#)
- [A policy for a custom AWS KMS encryption key to encrypt Amazon S3 target data \(p. 409\)](#)

## A policy for a custom AWS KMS encryption key to encrypt Amazon Redshift target data

The example following shows the JSON for the key policy created for a AWS KMS encryption key that you create to encrypt Amazon Redshift target data.

```
{  
    "Id": "key-consolepolicy-3",  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Enable IAM User Permissions",  
            "Effect": "Allow",  
            "Principal": "*"  
        }  
    ]  
}
```

```

"Principal": {
    "AWS": [
        "arn:aws:iam::987654321098:root"
    ],
    "Action": "kms:*",
    "Resource": "*"
},
{
    "Sid": "Allow access for Key Administrators",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::987654321098:role/Admin"
        ]
    },
    "Action": [
        "kms>Create*",
        "kms>Describe*",
        "kms>Enable*",
        "kms>List*",
        "kms>Put*",
        "kms>Update*",
        "kms>Revoke*",
        "kms>Disable*",
        "kms>Get*",
        "kms>Delete*",
        "kms>TagResource",
        "kms>UntagResource",
        "kms>ScheduleKeyDeletion",
        "kms>CancelKeyDeletion"
    ],
    "Resource": "*"
},
{
    "Sid": "Allow use of the key",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::987654321098:role/DMS-Redshift-endpoint-access-role"
        ]
    },
    "Action": [
        "kms>Encrypt",
        "kms>Decrypt",
        "kms>ReEncrypt*",
        "kms>GenerateDataKey*",
        "kms>DescribeKey"
    ],
    "Resource": "*"
},
{
    "Sid": "Allow attachment of persistent resources",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::987654321098:role/DMS-Redshift-endpoint-access-role"
        ]
    },
    "Action": [
        "kms>CreateGrant",
        "kms>ListGrants",
        "kms>RevokeGrant"
    ],
    "Resource": "*",
    "Condition": {

```

```

        "Bool": {
            "kms:GrantIsForAWSResource": true
        }
    }
}
]
```

Here, you can see where the key policy references the role for accessing Amazon Redshift target endpoint data that you created before creating the key. In the example, that is `DMS-Redshift-endpoint-access-role`. You can also see the different key actions permitted for the different principals (users and roles). For example, any user with `DMS-Redshift-endpoint-access-role` can encrypt, decrypt, and re-encrypt the target data. Such a user can also generate data keys for export to encrypt the data outside of AWS KMS. They can also return detailed information about a customer master key (CMK), such as the key that you just created. In addition, such a user can manage attachments to AWS resources, such as the target endpoint.

## A policy for a custom AWS KMS encryption key to encrypt Amazon S3 target data

The example following shows the JSON for the key policy created for a AWS KMS encryption key that you create to encrypt Amazon S3 target data.

```
{
    "Id": "key-consolepolicy-3",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::987654321098:root"
                ]
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow access for Key Administrators",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::987654321098:role/Admin"
                ]
            },
            "Action": [
                "kms>Create*",
                "kms>Describe*",
                "kms>Enable*",
                "kms>List*",
                "kms>Put*",
                "kms>Update*",
                "kms>Revoke*",
                "kms>Disable*",
                "kms>Get*",
                "kms>Delete*",
                "kms>TagResource",
                "kms>UntagResource",
                "kms>ScheduleKeyDeletion",
                "kms>CancelKeyDeletion"
            ],
            "Condition": {
                "Bool": {
                    "kms:GrantIsForAWSResource": true
                }
            }
        }
    ]
},
```

```
        "Resource": "*"
    },
{
    "Sid": "Allow use of the key",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::987654321098:role/DMS-S3-endpoint-access-role"
        ]
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
    ],
    "Resource": "*"
},
{
    "Sid": "Allow attachment of persistent resources",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::987654321098:role/DMS-S3-endpoint-access-role"
        ]
    },
    "Action": [
        "kms>CreateGrant",
        "kms>ListGrants",
        "kms:RevokeGrant"
    ],
    "Resource": "*",
    "Condition": {
        "Bool": {
            "kms:GrantIsForAWSResource": true
        }
    }
}
]
```

Here, you can see where the key policy references the role for accessing Amazon S3 target endpoint data that you created prior to creating the key. In the example, that is DMS-S3-endpoint-access-role. You can also see the different key actions permitted for the different principals (users and roles). For example, any user with DMS-S3-endpoint-access-role can encrypt, decrypt, and re-encrypt the target data. Such a user can also generate data keys for export to encrypt the data outside of AWS KMS. They can also return detailed information about a customer master key (CMK), such as the key that you just created. In addition, such a user can manage attachment to AWS resources, such as the target endpoint.

## Troubleshooting AWS Database Migration Service identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS DMS and IAM.

### Topics

- [I am not authorized to perform an action in AWS DMS \(p. 411\)](#)
- [I am not authorized to perform iam:PassRole \(p. 411\)](#)
- [I want to view my access keys \(p. 411\)](#)

- I'm an administrator and want to allow others to access AWS DMS (p. 412)
- I want to allow people outside of my AWS account to access my AWS DMS resources (p. 412)

## I am not authorized to perform an action in AWS DMS

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a AWS DMS endpoint but does not have `dms:DescribeEndpoint` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:  
dms:DescribeEndpoint on resource: my-postgresql-target
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `my-postgresql-target` endpoint resource using the `dms:DescribeEndpoint` action.

## I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to AWS DMS.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS DMS. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

## I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJAlrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

### Important

Do not provide your access keys to a third party, even to help [find your canonical user ID](#). By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see [Managing Access Keys](#) in the *IAM User Guide*.

## I'm an administrator and want to allow others to access AWS DMS

To allow others to access AWS DMS, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in AWS DMS.

To get started right away, see [Creating Your First IAM Delegated User and Group](#) in the *IAM User Guide*.

## I want to allow people outside of my AWS account to access my AWS DMS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS DMS supports these features, see [How AWS Database Migration Service works with IAM \(p. 397\)](#).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing Access to an IAM User in Another AWS Account That You Own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing Access to AWS Accounts Owned by Third Parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing Access to Externally Authenticated Users \(Identity Federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [How IAM Roles Differ from Resource-based Policies](#) in the *IAM User Guide*.

# Compliance validation for AWS Database Migration Service

Third-party auditors assess the security and compliance of AWS Database Migration Service as part of multiple AWS compliance programs. These include the programs following:

- SOC
- PCI
- ISO
- FedRAMP
- DoD CC SRG
- HIPAA BAA
- MTCS
- CS
- K-ISMS
- ENS High
- OSPAR
- HITRUST CSF

For a list of AWS services in scope of specific compliance programs, see [AWS services in scope by compliance program](#). For general information, see [AWS compliance programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading reports in AWS artifact](#).

Your compliance responsibility when using AWS DMS is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security and compliance quick start guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA security and compliance whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- [AWS compliance resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [AWS Config](#) – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

# Resilience in AWS Database Migration Service

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS global infrastructure](#).

In addition to the AWS global infrastructure, AWS DMS provides high availability and failover support for a replication instance using a Multi-AZ deployment when you choose the **Multi-AZ** option.

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 information on working with Multi-AZ deployments, see [Working with an AWS DMS replication instance \(p. 39\)](#).

# Infrastructure security in AWS Database Migration Service

As a managed service, AWS Database Migration Service is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of security processes](#) whitepaper.

You use AWS published API calls to access AWS DMS through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an AWS Identity and Access Management (IAM) principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

You can call these API operations from any network location, but AWS DMS does support resource-based access policies, which can include restrictions based on the source IP address. You can also use AWS DMS policies to control access from specific Amazon VPC endpoints or specific virtual private clouds (VPCs). Effectively, this isolates network access to a given AWS DMS resource from only the specific VPC within the AWS network.

For more information about using resource-based access policies with AWS DMS, including examples, see [Fine-grained access control using resource names and tags \(p. 424\)](#).

# IAM permissions needed to use AWS DMS

You 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 policy following gives you access to AWS DMS, and also permissions for certain actions needed from other Amazon services such as AWS KMS, IAM, 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 \(p. 424\)](#).

```
{
    "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": "*"
    }
]
}

```

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

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

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

The following section is required to allow the user to list their available AWS KMS keys and alias for display in the console. This entry is not required if you know the Amazon Resource Name (ARN) for the KMS key and you are using only the AWS Command Line Interface (AWS CLI).

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

The following section is required for certain endpoint types that require an IAM role ARN to be passed in with the endpoint. In addition, if the required AWS DMS roles aren't created ahead of time, the AWS DMS console can create the role. If all roles are configured ahead of time, all that is required is `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 \(p. 420\)](#).

```
{
    "Effect": "Allow",
    "Action": [
        "iam:GetRole",
        "iam:PassRole",

```

```
        "iam:CreateRole",
        "iam:AttachRolePolicy"
    ],
    "Resource": "*"
}
```

The following section is required because AWS DMS needs to create the Amazon 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": "*"
}
```

The following 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": "*"
}
```

The following section is required when using Amazon Redshift as a target. It allows AWS DMS to validate that the Amazon 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 about adding these roles, see [Creating the IAM roles to use with the AWS CLI and AWS DMS API \(p. 420\)](#).

# 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 about the specified IAM 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 IAM 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": "sts:AssumeRole",  
                    "Effect": "Allow",  
                    "Principal": "dms-vpc-role",  
                    "Condition": {}  
                },  
                {  
                    "Action": "sts:AssumeRole",  
                    "Effect": "Allow",  
                    "Principal": "dms-cloudwatch-logs-role",  
                    "Condition": {}  
                }  
            ]  
        }  
    }  
}
```

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

You can use the same commands to get information about `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 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": "s3:GetBucketLocation"  
        }  
    ]  
}
```

```
        "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 resource names and resource tags based on Amazon Resource Names (ARNs) 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 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 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 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:UO3YR4N47DXH3ATT4YMWOIT"
        }
    ]
}

```

For example, the following commands 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:UO3YR4N47DXH3ATT4YMWOIT"

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:UO3YR4N47DXH3ATT4YMWOIT

```

## 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 \(p. 387\)](#).

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. You can add these to AWS DMS resources, including replication instances, endpoints, and tasks. These tags are matched by using 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 that 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 matches a resource with the given tag.

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

You add the following tag to a replication instance that matches 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 can be applied to a given resource. In this case, the tag prefix is "req".

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

```

"Condition":
{
    "streq":
    {
        "dms:rep-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 an 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 AWS 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 (AWS KMS) key that is unique to your AWS account. You can view and manage this key with AWS KMS. You can use the default AWS KMS key in your account (`aws/dms`) or you can create a custom AWS KMS key. If you have an existing AWS KMS key, you can also use that key for encryption.

## Note

Any custom or existing AWS KMS key that you use as an encryption key must be a symmetric key. AWS DMS does not support the use of asymmetric encryption keys. For more information on symmetric and asymmetric encryption keys, see <https://docs.aws.amazon.com/kms/latest/developerguide/symmetric-asymmetric.html> in the *AWS Key Management Service Developer Guide*.

The default AWS KMS key (`aws/dms`) is created when you first launch a replication instance, if you haven't selected a custom AWS KMS master key from the **Advanced** section of the **Create Replication Instance** page. If you use the default AWS 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 AWS KMS key, see [IAM permissions needed to use AWS DMS \(p. 416\)](#).

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

- Add the IAM user account used for the migration as a key administrator or key user for the AWS KMS custom key. Doing this ensures that necessary AWS KMS permissions are granted to the IAM user account. This action is in addition to the IAM permissions that you 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](#) in the *AWS Key Management Service Developer Guide*.
- If you don't want to add the IAM user account as a key administrator or key user for your custom AWS 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 also works with AWS KMS key aliases. For more information about creating your own AWS KMS keys and giving users access to an AWS KMS key, see the [KMS Developer Guide](#).

If you don't specify an AWS KMS key identifier, then AWS DMS uses your default encryption key. AWS 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 AWS KMS keys used for encrypting your AWS DMS resources, use the AWS Key Management Service. AWS KMS combines secure, highly available hardware and software to provide a key management system scaled for the cloud. Using AWS KMS, you can create encryption keys and define the policies that control how these keys can be used.

### You can find AWS KMS in the AWS Management Console following

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. Choose one of the options following to work with AWS KMS keys:
  - To view the keys in your account that AWS creates and manages for you, in the navigation pane, choose **AWS managed keys**.
  - To view the keys in your account that you create and manage, in the navigation pane choose **Customer managed keys**.

AWS KMS supports AWS CloudTrail, so you can audit key usage to verify that keys are being used appropriately. Your AWS KMS keys can be used in combination with AWS DMS and supported AWS services such as Amazon RDS, Amazon S3, Amazon Redshift, and Amazon EBS.

You can also create custom AWS KMS keys specifically to encrypt target data for the following AWS DMS endpoints:

- Amazon Redshift – For more information, see [Creating and using AWS KMS keys to encrypt Amazon Redshift target data \(p. 176\)](#).
- Amazon S3 – For more information, see [Creating AWS KMS keys to encrypt Amazon S3 target objects \(p. 194\)](#).

After you have created your AWS DMS resources with an AWS KMS 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.

# 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 Amazon 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 that you can use for database migration each require specific security considerations:

- [Configuration with all database migration components in one VPC \(p. 45\)](#) – 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 \(p. 46\)](#) – 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 \(p. 46\)](#) – 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 \(p. 47\)](#) – 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 RDS DB instance not in a VPC to a DB instance in a VPC using ClassicLink \(p. 47\)](#)
  - 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. This Elastic network interface 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 instead appears 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.
- **VPC endpoints for non-RDBMS engines** – AWS DMS doesn't support VPC endpoints for non-RDBMS engines.



# 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 \(p. 80\)](#).

## Topics

- [Limitations on using SSL with AWS DMS \(p. 436\)](#)
- [Managing certificates \(p. 436\)](#)
- [Enabling SSL for a MySQL-compatible, PostgreSQL, or SQL Server endpoint \(p. 437\)](#)

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 in the `rds-combined-ca-bundle.pem` file hosted by Amazon RDS. For more information about downloading this file, see [Using SSL/TLS to encrypt a connection to a DB instance](#) in the *Amazon RDS User Guide*.

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 DMS

Following are limitations on using SSL with AWS DMS:

- SSL connections to Amazon Redshift target endpoints aren't supported. AWS DMS uses an Amazon S3 bucket to transfer data to the Amazon Redshift database. This transmission is encrypted by Amazon Redshift by default.
- SQL timeouts can occur when performing change data capture (CDC) tasks with SSL-enabled Oracle endpoints. If you have an issue where CDC counters don't reflect the expected numbers, set the `MinimumTransactionSize` parameter from the `ChangeProcessingTuning` section of the task settings to a lower value. You can start with a value as low as 100. For more information about the `MinimumTransactionSize` parameter, see [Change processing tuning settings \(p. 288\)](#).
- You can import certificates only in the .pem and .sso (Oracle wallet) formats.
- In some cases, your server SSL certificate might be signed by an intermediate certificate authority (CA). If so, make sure that 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 doesn't 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.

Identifier	Signing algorithm	Owner	Key size	Valid from
my-cert-1	SHA1withRSA	[redacted]	2048	Thu Feb 05 01:11:
my-cert-2	SHA1withRSA	[redacted]	2048	Thu Feb 05 01:11:

# 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 about the permissions required for database migration, see [IAM permissions needed to use AWS DMS \(p. 416\)](#).

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 about the permissions required for database migration, see [IAM permissions needed to use AWS DMS \(p. 416\)](#).

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 a value for **SSL mode**.

If you choose **verify-ca** or **verify-full** mode, specify the certificate that you want to use for **CA certificate**, 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-premises or in the cloud. Details should be specified in the form below. It is recommended that you test your endpoint connections here to avoid errors during primary key migration.

Endpoint type\*  Source  Target ?

Endpoint identifier\* e.g. ProdEndpoint ?

Source engine\* sqlserver ▼ ?

Server name\*

Port\*

SSL mode\* verify-ca ▼ ?

CA certificate\* - Select One - ▼ ?  
[Add new CA certificate](#)

User name\*

Password\*

» Advanced

7. Choose **Modify**.
8. When the endpoint has been modified, choose 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 about creating a task, see [Step 4: Create a task \(p. 25\)](#).

# 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. If you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information about the permissions required, see [IAM permissions needed to use AWS DMS \(p. 416\)](#).
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 AWS 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 following table lists the AWS DMS resources and their limits per AWS Region.

Resource	Default limit
Replication instances	60
Total amount of storage for a replication instance	30,000 GB
Event subscriptions	60
Replication subnet groups	60
Subnets per replication subnet group	60
Endpoints	1000
Tasks	600
Endpoints per instance	100
Certificates	100

### Note

- The 30,000 GB limit for storage applies to an AWS DMS replication instance. This storage is used to cache changes if the target cannot keep up with the source, and for storing log information.
- Storage size of source and target endpoints can exceed 30,000 GB.

# Troubleshooting migration tasks in AWS Database Migration Service

Following, you can find topics about troubleshooting issues with AWS Database Migration Service (AWS DMS). These topics can help you to resolve common issues using both AWS DMS and selected endpoint databases.

If you have opened an AWS Support case, your support engineer might identify a potential issue with one of your endpoint database configurations. Your engineer might also ask you to run a support script to return diagnostic information about your database. For details about downloading, running, and uploading the diagnostic information from this type of support script, see [Working with diagnostic support scripts in AWS DMS \(p. 458\)](#).

## Topics

- [Migration tasks run slowly \(p. 441\)](#)
- [Task status bar doesn't move \(p. 442\)](#)
- [Task completes but nothing was migrated \(p. 442\)](#)
- [Foreign keys and secondary indexes are missing \(p. 442\)](#)
- [Issues occur with connecting to Amazon RDS \(p. 443\)](#)
- [Networking issues occur \(p. 443\)](#)
- [CDC is stuck after full load \(p. 444\)](#)
- [Primary key violation errors occur when you restart a task \(p. 444\)](#)
- [Initial load of a schema fails \(p. 444\)](#)
- [Tasks fail with an unknown error \(p. 444\)](#)
- [Task restart loads tables from the beginning \(p. 444\)](#)
- [Number of tables per task causes issues \(p. 444\)](#)
- [Tasks fail when a primary key is created on a LOB column \(p. 445\)](#)
- [Duplicate records occur on a target table without a primary key \(p. 445\)](#)
- [Source endpoints fall in the reserved IP range \(p. 445\)](#)
- [Troubleshooting issues with Oracle \(p. 445\)](#)
- [Troubleshooting issues with MySQL \(p. 448\)](#)
- [Troubleshooting issues with PostgreSQL \(p. 452\)](#)
- [Troubleshooting issues with Microsoft SQL Server \(p. 454\)](#)
- [Troubleshooting issues with Amazon Redshift \(p. 456\)](#)
- [Troubleshooting issues with Amazon Aurora MySQL \(p. 457\)](#)
- [Working with diagnostic support scripts in AWS DMS \(p. 458\)](#)

## Migration tasks run slowly

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. To make sure that your instance has enough resources for the tasks you are running on it, check your replication instance's use of CPU, memory, swap files, and IOPS. For example, multiple tasks with Amazon Redshift as an endpoint are I/O 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 \(p. 42\)](#).

You can increase the speed of an initial migration load by doing the following:

- If your target is an Amazon RDS DB instance, make sure that Multi-AZ isn't 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 after your migration is complete.
- If the feature is available on your target, use provisioned IOPS.
- If your migration data contains LOBs, make sure that the task is optimized for LOB migration. For more information on optimizing for LOBs, see [Target metadata task settings \(p. 282\)](#).

## Task status bar doesn't move

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, AWS DMS can't provide any kind of percentage complete estimate. In this case, use the task state and the indication of rows loaded to confirm that the task is running and making progress.

## Task completes but nothing was migrated

Do the following if nothing was migrated after your task has completed.

- Check if the user that created the endpoint has read access to the table you intend to migrate.
- Check if the object you want to migrate is a table. If it is a view, update table mappings and specify the object-locator as "view" or "all". For more information, see [Specifying table selection and transformations rules from the console \(p. 309\)](#).

## Foreign keys and secondary indexes are missing

AWS DMS creates tables, primary keys, and in some cases unique indexes, but it doesn't create any other objects that aren't 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 AWS Schema Conversion Tool (AWS SCT) if you are migrating to a different database engine than that used by your source database to migrate secondary objects.

## Issues occur with connecting to Amazon RDS

There can be several reasons why you can't connect to an Amazon RDS DB instance that you set as a source or target. Some items to check follow:

- Check that the user name and password combination is correct.
- 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 aren't in the same virtual private cloud (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. This error indicates that there is an error in the connection string. An example is a space after the host IP address. Another is a bad character copied into the connection string.

## Networking issues occur

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. In many cases, you modify this security group or use your own security group. If so, at a minimum, make sure to give egress to the source and target endpoints on their respective database ports.

Other configuration-related issues can include the following:

- **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. Make sure 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 an internet gateway)** – The VPC security group must include routing rules that send traffic that isn't 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 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. This NAT gateway receives a NAT identifier (nat-#####).

In some cases, the VPC includes a default route to that NAT gateway instead of the internet gateway. In such cases, the replication instance instead appears to contact the database endpoint using the public IP address of the internet gateway. Here, 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.

For information about using your own on-premises name server, see [Using your own on-premises name server \(p. 35\)](#).

## CDC is 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, suppose that the **Target table preparation mode** parameter is set to **Do nothing** or **Truncate**. In this case, 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, AWS DMS does a full table scan for each update. This approach can affect performance significantly.

## Primary key violation errors occur when you restart a task

This error can occur when data remains in the target database from a previous migration task. If the **Target table preparation mode** option is set to **Do nothing**, AWS DMS doesn't do any preparation on the target table, including cleaning up data inserted from a previous task.

To restart your task and avoid these errors, remove rows inserted into the target tables from the previous running of the task.

## Initial load of a schema fails

In some cases, the initial load of your schemas might fail with an error of `Operation:getSchemaListDetails:errType=, status=0, errMessage=, errDetails=`.

In such cases, the user account used by AWS DMS to connect to the source endpoint doesn't have the necessary permissions.

## Tasks fail with an unknown error

The cause of unknown types of error can be varied. However, often we find that the issue involves insufficient resources allocated to the AWS DMS replication instance.

To make sure that your replication instance has enough resources to perform the migration, check your instance's use of CPU, memory, swap files, and IOPS. For more information on monitoring, see [AWS Database Migration Service metrics \(p. 367\)](#).

## Task restart loads tables from the beginning

AWS DMS restarts table loading from the beginning when it hasn't finished the initial load of a table. When a task is restarted, AWS DMS reloads tables from the beginning when the initial load didn't complete.

## Number of tables per task causes issues

There is no set limit on the number of tables per replication task. However, we recommend limiting the number of tables in a task to less than 60,000, as a rule of thumb. Resource use can often be a bottleneck when a single task uses more than 60,000 tables.

## Tasks fail when a primary key is created on a LOB column

In FULL LOB or LIMITED LOB mode, AWS DMS doesn't support replication of primary keys that are LOB data types.

DMS initially migrates a row with a LOB column as null, then later updates the LOB column. So, when the primary key is created on a LOB column, the initial insert fails since the primary key can't be null. As a workaround, add another column as primary key and remove the primary key from the LOB column.

## Duplicate records occur on a target table without a primary key

Running a full load and CDC task can create duplicate records on target tables that don't have a primary key or unique index. To avoid duplicating records on target tables during full load and CDC tasks, make sure that target tables have a primary key or unique index.

## Source endpoints fall in the reserved IP range

If an AWS DMS source database uses an IP address within the reserved IP range of 192.168.0.0/24, the source endpoint connection test fails. The steps following provide a possible workaround:

1. Find one Amazon EC2 instance that isn't in the reserved range that can communicate to the source database at 192.168.0.0/24.
2. Install a socat proxy and run it. The following shows an example.

```
yum install socat

socat -d -d -l:local2 tcp4-listen:database_port,bind=0.0.0.0,reuseaddr,fork
      tcp4:source_database_ip_address:database_port
&
```

Use the EC2 instance IP address and the database port given above for the AWS DMS endpoint. Make sure that the endpoint has the security group that allows AWS DMS to talk to it at the database port.

## Troubleshooting issues with Oracle

Following, you can learn about troubleshooting issues specific to using AWS DMS with Oracle databases.

### Topics

- [Pulling data from views \(p. 446\)](#)
- [Migrating LOBs from Oracle 12c \(p. 446\)](#)
- [Switching between Oracle LogMiner and Binary Reader \(p. 446\)](#)
- [Error: Oracle CDC stopped 122301 oracle CDC maximum retry counter exceeded. \(p. 446\)](#)
- [Automatically add supplemental logging to an Oracle source endpoint \(p. 447\)](#)
- [LOB changes aren't being captured \(p. 447\)](#)
- [Error: ORA-12899: Value too large for column column-name \(p. 447\)](#)

- NUMBER data type being misinterpreted (p. 448)
- Records missing during full load (p. 448)

## Pulling data from views

You can pull data once from a view; you can't use it for ongoing replication. To be able to extract data from views, you must add the following code to **Extra connection attributes** in the **Advanced** section of the Oracle source endpoint page. 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 doesn't 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. Choose **Endpoints**.
3. Choose the Oracle source endpoint that you want to use Binary Reader.
4. Choose **Modify**.
5. Choose **Advanced**, and then add the following code for **Extra connection attributes**.

```
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. Choose **Endpoints**.
3. Choose the Oracle source endpoint that you want to add supplemental logging to.
4. Choose **Modify**.
5. Choose **Advanced**, and then add the following code to the **Extra connection attributes** text box:

```
addSupplementalLogging=Y
```

6. Choose **Modify**.

## LOB changes aren't 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 couple of issues.

In one of these issues, there's a mismatch in the character sets used by the source and target databases.

In another of these issues, national language support (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 data types, depending on the precision and scale of NUMBER. These conversions are documented here [Source data types for Oracle \(p. 95\)](#). 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 \(p. 88\)](#).

## Records missing during full load

When performing a full load, AWS DMS looks for open transactions at the database level and waits for the transaction to be committed. For example, based on the task setting `TransactionConsistencyTimeout=600`, AWS DMS waits for 10 minutes even if the open transaction is on a table not included in table mapping. But if the open transaction is on a table included in table mapping, and the transaction is not committed in time, missing records in the target table result.

You can modify the `TransactionConsistencyTimeout` task setting and increase wait time if you know that open transactions will take longer to commit.

Also, note the default value of the `FailOnTransactionConsistencyBreached` task setting is `false`. This means AWS DMS continues to apply other transactions but open transactions are missed. If you want the task to fail when open transactions aren't closed in time, you can set `FailOnTransactionConsistencyBreached` to `true`.

## Troubleshooting issues with MySQL

Following, you can learn about troubleshooting issues specific to using AWS DMS with MySQL databases.

### Topics

- [CDC task failing for Amazon RDS DB instance endpoint because binary logging disabled \(p. 448\)](#)
- [Connections to a target MySQL instance are disconnected during a task \(p. 449\)](#)
- [Adding autocommit to a MySQL-compatible endpoint \(p. 449\)](#)
- [Disable foreign keys on a target MySQL-compatible endpoint \(p. 449\)](#)
- [Characters replaced with question mark \(p. 450\)](#)
- ["Bad event" log entries \(p. 450\)](#)
- [Change data capture with MySQL 5.5 \(p. 450\)](#)
- [Increasing binary log retention for Amazon RDS DB instances \(p. 450\)](#)
- [Log message: Some changes from the source database had no impact when applied to the target database. \(p. 451\)](#)
- [Error: Identifier too long \(p. 451\)](#)
- [Error: Unsupported character set causes field data conversion to fail \(p. 451\)](#)
- [Error: Codepage 1252 to UTF8 \[120112\] a field data conversion failed \(p. 451\)](#)

## 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, you might see the following type of errors in the task log.

```
[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.
```

In this case, you might need to adjust some of your task settings.

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. Choose **Endpoints**.
3. Choose the MySQL-compatible target endpoint that you want to add autocommit to.
4. Choose **Modify**.
5. Choose **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. Choose **Endpoints**.
3. Choose the MySQL, Aurora MySQL, or MariaDB target endpoint that you want to disable foreign keys.
4. Choose **Modify**.
5. Choose **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 engine versions prior to version 3.1.1 didn't support the UTF8MB4 character set.

### "Bad event" log entries

"Bad event" entries in the migration logs usually indicate that an unsupported data definition language (DDL) operation was attempted on the source database endpoint. Unsupported DDL operations cause an event that the replication instance can't skip, so a bad event is logged.

To fix this issue, restart the task from the beginning. Doing this reloads the tables and starts 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 isn't 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. These engines update 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)
```

In some cases, you set AWS DMS to create the tables and primary keys in the target database. In these cases, DMS currently doesn't use the same names for the primary keys that were used in the source database. Instead, DMS creates the primary key name based on the table name. When the table name is long, the autogenerated identifier created can be longer than the allowed limits for MySQL.

To solve this issue, the current approach is to first pre-create the tables and primary keys in the target database. Then 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)
```

In AWS DMS engine versions prior to 3.1.1, this error often occurred because of tables or databases using UTF8MB4 encoding. These engine versions didn't support the UTF8MB4 character set. In addition, check your database's parameters related to connections. The following command can be used to set 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 issues with PostgreSQL

Following, you can learn about troubleshooting issues specific to using AWS DMS with PostgreSQL databases.

### Topics

- [JSON data types being truncated \(p. 452\)](#)
- [Columns of a user-defined data type not being migrated correctly \(p. 453\)](#)
- [Error: No schema has been selected to create in \(p. 453\)](#)
- [Deletes and updates to a table aren't being replicated using CDC \(p. 453\)](#)
- [Truncate statements aren't being propagated \(p. 453\)](#)
- [Preventing PostgreSQL from capturing DDL \(p. 453\)](#)
- [Selecting the schema where database objects for capturing DDL are created \(p. 454\)](#)
- [Oracle tables missing after migrating to PostgreSQL \(p. 454\)](#)
- [Task using view as a source has no rows copied \(p. 454\)](#)

## 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, suppose that limited LOB mode is set to 4,096 KB. In this case, any JSON data larger than 4,096 KB is truncated at the 4,096 KB limit and fails the validation test in PostgreSQL.

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

In some case, you might see the error "SQL\_ERROR SqlState: 3F000 NativeError: 7 Message: ERROR: no schema has been selected to create in".

This error can occur when your JSON table mapping contains a wildcard value for the schema but the source database doesn't support that value.

## Deletes and updates to a table aren't being replicated using CDC

Delete and update operations during change data capture (CDC) are ignored if the source table doesn't have a primary key. 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 (WAL) logs don't include a before image of the database row. In this case, AWS DMS can't update the table. For delete operations to be replicated, create a primary key on the source table.

## Truncate statements aren't being propagated

When using change data capture (CDC), TRUNCATE operations aren't 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.

```
captureDDLS=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

In this case, your tables and data are generally still accessible.

Oracle defaults to uppercase table names, and PostgreSQL defaults to lowercase table names. When you perform a migration from Oracle to PostgreSQL, we suggest that you supply certain transformation rules under your task's table-mapping section. These are transformation rules to convert the case of your table names.

If you migrated your tables without using transformation rules to convert the case of your table names, enclose your table names in quotation marks when referencing them.

## Task using view as a source has no rows copied

To migrate a view, set **table-type** to **all** or **view**. For more information, see [Specifying table selection and transformations rules from the console \(p. 309\)](#).

Sources that support views include the following.

- Oracle
- Microsoft SQL Server
- MySQL
- PostgreSQL
- IBM Db2 LUW
- SAP Adaptive Server Enterprise (ASE)

## Troubleshooting issues with Microsoft SQL Server

Following, you can learn about troubleshooting issues specific to using AWS DMS with Microsoft SQL Server databases.

### Topics

- [Special permissions for AWS DMS user account to use CDC \(p. 455\)](#)
- [Errors capturing changes for SQL server database \(p. 455\)](#)
- [Missing identity columns \(p. 455\)](#)
- [Error: SQL Server doesn't support publications \(p. 455\)](#)

- [Changes don't appear in your target \(p. 455\)](#)
- [Non-uniform table mapped across partitions \(p. 456\)](#)

## 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).

## Errors capturing changes for SQL server database

Errors during change data capture (CDC) can often indicate that one of the prerequisites wasn't met. For example, the most common overlooked prerequisite 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 prerequisites listed for using SQL Server as a source in [Using a Microsoft SQL Server database as a source for AWS DMS \(p. 98\)](#).

## Missing identity columns

AWS DMS doesn't support identity columns when you create a target schema. You must add them after the initial load has completed.

## Error: SQL Server doesn't 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 doesn't support SQL Server Express as a source or target.

## Changes don't appear in your 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 isn't 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. However, as soon as a checkpoint occurs the log is automatically truncated. This truncation means that the log becomes available for reuse and older log entries can be overwritten. When log entries are overwritten, changes can't be captured. This issue is why AWS

DMS doesn't support the SIMPLE data recovery model. For information on other required prerequisites for using SQL Server as a source, see [Using a Microsoft SQL Server database as a source for AWS DMS \(p. 98\)](#).

## Non-uniform table mapped across partitions

During change data capture (CDC), migration of a table with a specialized structure is suspended when AWS DMS can't properly perform CDC on the table. Messages like these are issued:

```
[SOURCE_CAPTURE ]W: Table is not uniformly mapped across partitions. Therefore - it is excluded from CDC (sqlserver_log_metadata.c:1415)
[SOURCE_CAPTURE ]I: Table has been mapped and registered for CDC.
(sqlserver_log_metadata.c:835)
```

When running CDC on SQL Server tables, AWS DMS parses the SQL Server tlogs. On each tlog record, AWS DMS parses hexadecimal values containing data for columns that were inserted, updated, or deleted during a change.

To parse the hexadecimal record, AWS DMS reads the table metadata from the SQL Server system tables. Those system tables identify what the specially structured table columns are and reveal some of their internal properties, such as "xoffset" and "null bit position".

AWS DMS expects that metadata to be the same for all raw partitions of the table. But in some cases, specially structured tables don't have the same metadata on all of their partitions. In these cases, AWS DMS can suspend CDC on that table to avoid parsing changes incorrectly and providing the target with incorrect data. Workarounds include the following:

- If the table has a clustered index, perform an index rebuild.
- If the table doesn't have a clustered index, add a clustered index to the table (you can drop it later if you want).

## Troubleshooting issues with Amazon Redshift

Following, you can learn about troubleshooting issues specific to using AWS DMS with Amazon Redshift databases.

### Topics

- [Loading in to an Amazon Redshift cluster in a different AWS Region \(p. 456\)](#)
- [Error: Relation "awsdms\\_apply\\_exceptions" already exists \(p. 457\)](#)
- [Errors with tables whose name begins with "awsdms\\_changes" \(p. 457\)](#)
- [Seeing tables in clusters with names like dms.awsdms\\_changes000000000XXXX \(p. 457\)](#)
- [Permissions required to work with Amazon Redshift \(p. 457\)](#)

## Loading in to an Amazon Redshift cluster in a different AWS Region

You can't load into an Amazon Redshift cluster in a different AWS Region than your AWS DMS replication instance. DMS requires that your replication instance and your Amazon 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"

Table error messages with names that begin with "awsdms\_changes" can occur when two tasks trying to load data into the same Amazon Redshift cluster run concurrently. Due to the way temporary tables are named, concurrent tasks can conflict when updating the same table.

## Seeing tables in clusters with names like dms.awsdms\_changes000000000XXXX

AWS DMS creates temporary tables when data is being loaded from files stored in Amazon S3. The names of these temporary tables each 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 that you use to access Amazon Redshift must have the following permissions:

- CRUD (Choose, Insert, Update, Delete)
- Bulk load
- Create, alter, drop (if required by the task's definition)

To see the prerequisites required for using Amazon Redshift as a target, see [Using an Amazon Redshift database as a target for AWS Database Migration Service \(p. 173\)](#).

## Troubleshooting issues with Amazon Aurora MySQL

Following, you can learn about troubleshooting issues specific to using AWS DMS with Amazon Aurora MySQL databases.

### Topics

- [Error: CHARACTER SET UTF8 fields terminated by ';' enclosed by "" lines terminated by '\n' \(p. 458\)](#)

## Error: CHARACTER SET UTF8 fields terminated by ;' enclosed by """ lines terminated by '\n'

If you are using Amazon Aurora MySQL as a target, you might see an error like the following in the logs. This type of error 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 quotation marks to be handled like quotation marks 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/7X04FJHCVON7TYTLQ6RX3CQHDU/data_files/4/LOAD00001DF.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)
```

## Working with diagnostic support scripts in AWS DMS

If you encounter an issue when working with AWS DMS, your support engineer might need more information about either your source or target database. We want to make sure that AWS Support gets as much of the required information as possible in the shortest possible time. Therefore, we developed scripts to query this information for several of the major relational database engines.

If a support script is available for your database, you can download it using the link in the corresponding script topic described following. After verifying and reviewing the script (described following), you can run it according to the procedure described in the script topic. When the script run is complete, you can upload its output to your AWS Support case (again, described following).

Before running the script, you can detect any errors that might have been introduced when downloading or storing the support script. To do this, compare the checksum for the script file with a value provided by AWS. AWS uses the SHA256 algorithm for the checksum.

### To verify the support script file using a checksum

1. Open the latest checksum file provided to verify these support scripts at <https://d2pwp9zz55emqw.cloudfront.net/sha256Check.txt>. For example, the file might have content like the following.

```
MYSQL  dfafdf0d511477c699f96c64693ad0b1547d47e74d5c5f2f2025b790b1422e3c8
ORACLE  6c41ebcfcc99518cfa8a10cb2ce8943b153b2cc7049117183d0b5de3d551bc312
POSTGRES  6ccd274863d14f6f3146fbdbbba43f2d8d4c6a4c25380d7b41c71883aa4f9790
SQL_SERVER  971a6f2c46aec8d083d2b3b6549b1e9990af3a15fe4b922e319f4fdd358debe7
```

2. Run the SHA256 validation command for your operating system in the directory that contains the support file. For example, on the macOS operating system you can run the command following on an Oracle support script described later in this topic.

```
shasum -a 256 awsdms_support_collector_oracle.sql
```

3. Compare the results of the command with the value shown in the latest sha256Check.txt file that you opened. The two values should match. If they don't, contact your support engineer about the mismatch and how you can obtain a clean support script file.

If you have a clean support script file, before running the script make sure to read and understand the SQL from both a performance and security perspective. If you aren't comfortable running any of the SQL in this script, you can comment out or remove the problem SQL. You can also consult with your support engineer about any acceptable workarounds.

Upon successful completion and unless otherwise noted, the script returns output in a readable HTML format. The script is designed to exclude from this HTML any data or security details that might compromise your business. It also makes no modifications to your database or its environment. However, if you find any information in the HTML that you are uncomfortable sharing, feel free to remove the problem information before uploading the HTML. When the HTML is acceptable, upload it using the **Attachments** in the **Case details** of your support case.

Each of the following topics describes the scripts available for a supported AWS DMS database and how to run them. Your support engineer will direct you to a specific script documented following.

#### Topics

- [Oracle diagnostic support scripts \(p. 459\)](#)
- [SQL Server diagnostic support scripts \(p. 461\)](#)
- [Diagnostic support scripts for MySQL-compatible databases \(p. 463\)](#)
- [PostgreSQL diagnostic support scripts \(p. 464\)](#)

## Oracle diagnostic support scripts

Following, you can find the diagnostic support scripts available to analyze an on-premises or Amazon RDS for Oracle database in your AWS DMS migration configuration. These scripts work with either a source or target endpoint. The scripts are all written to run in the SQL\*Plus command-line utility. For more information on using this utility, see [A Using SQL Command Line](#) in the Oracle documentation.

Before running the script, ensure that the user account that you use has the necessary permissions to access your Oracle database. The permissions settings shown assume a user created as follows.

```
CREATE USER script_user IDENTIFIED BY password;
```

For an on-premises database, set the minimum permissions as shown following for *script\_user*.

```
GRANT CREATE SESSION TO script_user;
GRANT SELECT on V$DATABASE to script_user;
GRANT SELECT on V$VERSION to script_user;
GRANT SELECT on GV$SGA to script_user;
GRANT SELECT on GV$INSTANCE to script_user;
GRANT SELECT on GV$DATAGUARD_CONFIG to script_user;
GRANT SELECT on GV$LOG to script_user;
GRANT SELECT on DBA_TABLESPACES to script_user;
GRANT SELECT on DBA_DATA_FILES to script_user;
GRANT SELECT on DBA_SEGMENTS to script_user;
GRANT SELECT on DBA_LOBS to script_user;
GRANT SELECT on V$ARCHIVED_LOG to script_user;
GRANT SELECT on DBA_TAB_MODIFICATIONS to script_user;
GRANT SELECT on DBA_TABLES to script_user;
GRANT SELECT on DBA_TAB_PARTITIONS to script_user;
GRANT SELECT on DBA_MVIEWS to script_user;
```

```

GRANT SELECT on DBA_OBJECTS to script_user;
GRANT SELECT on DBA_TAB_COLUMNS to script_user;
GRANT SELECT on DBA_LOG_GROUPS to script_user;
GRANT SELECT on DBA_LOG_GROUP_COLUMNS to script_user;
GRANT SELECT on V$ARCHIVE_DEST to script_user;
GRANT SELECT on DBA_SYS_PRIVS to script_user;
GRANT SELECT on DBA_TAB_PRIVS to script_user;
GRANT SELECT on DBA_TYPES to script_user;
GRANT SELECT on DBA_CONSTRAINTS to script_user;
GRANT SELECT on V$TRANSACTION to script_user;
GRANT SELECT on GV$ASM_DISK_STAT to script_user;
GRANT SELECT on GV$SESSION to script_user;
GRANT SELECT on GV$SQL to script_user;
GRANT SELECT on DBA_ENCRYPTED_COLUMNS to script_user;

GRANT EXECUTE on dbms_utility to script_user;

```

For an Amazon RDS database, set the minimum permissions as shown following.

```

GRANT CREATE SESSION TO script_user;
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$DATABASE', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$VERSION', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('GV_$SGA', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('GV_$INSTANCE', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('GV_$DATAGUARD_CONFIG', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('GV$_LOG', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_TABLESPACES', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_DATA_FILES', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_SEGMENTS', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_LOBS', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$ARCHIVED_LOG', 'script_user', 'SELECT');
exec
rdsadmin.rdsadmin_util.grant_sys_object('DBA_TAB_MODIFICATIONS', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_TABLES', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_TAB_PARTITIONS', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_MVIEWS', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_OBJECTS', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_TAB_COLUMNS', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_LOG_GROUPS', 'script_user', 'SELECT');

exec
rdsadmin.rdsadmin_util.grant_sys_object('DBA_LOG_GROUP_COLUMNS', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$ARCHIVE_DEST', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_SYS_PRIVS', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_TAB_PRIVS', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_TYPES', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_CONSTRAINTS', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('V_$TRANSACTION', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('GV$ASM_DISK_STAT', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('GV$SESSION', 'script_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('GV$SQL', 'script_user', 'SELECT');

exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_UTLILITY', 'script_user', 'EXECUTE');

```

Following, you can find descriptions how to download, review, and run each SQL\*Plus support script available for Oracle. You can also find how to review and upload the output to your AWS Support case.

## Topics

- [awsdms\\_support\\_collector\\_oracle.sql script \(p. 461\)](#)

## awsdms\_support\_collector\_oracle.sql script

Download the [awsdms\\_support\\_collector\\_oracle.sql](#) script.

This script collects information about your Oracle database configuration. Remember to verify the checksum on the script, and if the checksum verifies, review the SQL code in the script to comment out any of the code that you are uncomfortable running. After you are satisfied with the integrity and content of the script, you can run it.

### To run the script and upload the results to your support case

1. Run the script from your database environment using the SQL\*Plus command line following.

```
SQL> @awsdms_support_collector_oracle.sql
```

```
<result>
```

The script displays a brief description and a prompt to either continue or abort the run. Press [Enter] to continue.

```
</result>
```

2. At the prompt following, enter the name of only one of the schemas that you want to migrate.
3. At the prompt following, enter the name of the user (*script\_user*) that you have defined to connect to the database.
4. At the prompt following, enter the number of days of data you want to examine, or accept the default. The script then collects the specified data from your database.

```
<result>
```

After the script completes, it displays the name of the output HTML file, for example `dms_support_oracle-2020-06-22-13-20-39-ORCL.html`. The script saves this file in your working directory.

```
</result>
```

5. Review this HTML file and remove any information that you are uncomfortable sharing. When the HTML is acceptable for you to share, upload the file to your AWS Support case. For more information on uploading this file, see [Working with diagnostic support scripts in AWS DMS \(p. 458\)](#).

## SQL Server diagnostic support scripts

Following, you can find a description of the diagnostic support scripts available to analyze an on-premises or Amazon RDS for SQL Server database in your AWS DMS migration configuration. These scripts work with either a source or target endpoint. For an on-premises database, run these scripts in the `sqlcmd` command-line utility. For more information on using this utility, see [sqlcmd - Use the utility](#) in the Microsoft documentation.

For an Amazon RDS database, you can't connect using the `sqlcmd` command-line utility. Instead, run these scripts using any client tool that connects to Amazon RDS SQL Server.

Before running the script, ensure that the user account that you use has the necessary permissions to access your SQL Server database. For both an on-premises and an Amazon RDS database, you can use the same permissions you use to access your SQL Server database without the `SysAdmin` role.

### To set up the minimum permissions to run for an on-premises SQL Server database

1. Create a new SQL Server account with password authentication using SQL Server Management Studio (SSMS), for example *on-prem-user*.
2. In the **User Mappings** section of SSMS, choose the **MSDB** and **MASTER** databases (which gives public permission), and assign the `DB_OWNER` role to the database where you want to run the script.

3. Open the context (right-click) menu for the new account, and choose **Security** to explicitly grant the `Connect SQL` privilege.
4. Run the grant commands following.

```
GRANT VIEW SERVER STATE TO on-prem-user;
USE MSDB;
GRANT SELECT ON MSDB.DBO.BACKUPSET TO on-prem-user;
GRANT SELECT ON MSDB.DBO.BACKUPMEDIAFAMILY TO on-prem-user;
GRANT SELECT ON MSDB.DBO.BACKUPFILE TO on-prem-user;
```

### To run with the minimum permissions for an Amazon RDS SQL Server database

1. Create a new SQL Server account with password authentication using SQL Server Management Studio (SSMS), for example `rds-user`.
2. In the **User Mappings** section of SSMS, choose the **MSDB** database (which gives public permission), and assign the `DB_OWNER` role to the database where you want to run the script.
3. Open the context (right-click) menu for the new account, and choose **Security** to explicitly grant the `Connect SQL` privilege.
4. Run the grant commands following.

```
GRANT VIEW SERVER STATE TO rds-user;
USE MSDB;
GRANT SELECT ON MSDB.DBO.BACKUPSET TO rds-user;
GRANT SELECT ON MSDB.DBO.BACKUPMEDIAFAMILY TO rds-user;
GRANT SELECT ON MSDB.DBO.BACKUPFILE TO rds-user;
```

The following topics describe how to download, review, and run each support script available for SQL Server. They also describe how to review and upload the script output to your AWS Support case.

#### Topics

- [awsdms\\_support\\_collector\\_sql\\_server.sql script \(p. 462\)](#)

## awsdms\_support\_collector\_sql\_server.sql script

Download the [awsdms\\_support\\_collector\\_sql\\_server.sql](#) script.

#### Note

Run this SQL Server diagnostic support script on SQL Server 2014 and later versions only.

This script collects information about your SQL Server database configuration. Remember to verify the checksum on the script, and if the checksum verifies, review the SQL code in the script to comment out any of the code that you are uncomfortable running. After you are satisfied with the integrity and content of the script, you can run it.

### To run the script for an on-premises SQL Server database

1. Run the script using the `sqlcmd` command line following.

```
sqlcmd -Uon-prem-user -Ppassword -SDMS-SQL17AG-N1 -y 0
-iC:\Users\admin\awsdms_support_collector_sql_server.sql -oC:\Users\admin
\DMSSupport_Report_SQLServer.html -dsqlserverdb01
```

The specified `sqlcmd` command parameters include the following:

- **-U** – Database user name.
  - **-P** – Database user password.
  - **-S** – SQL Server database server name.
  - **-y** – Maximum width of columns output from the sqlcmd utility. A value of 0 specifies columns of unlimited width.
  - **-i** – Path of the support script to run, in this case `awsdms_support_collector_sql_server.sql`.
  - **-o** – Path of the output HTML file, with a file name that you specify, containing the collected database configuration information.
  - **-d** – SQL Server database name.
2. After the script completes, review the output HTML file and remove any information that you are uncomfortable sharing. When the HTML is acceptable for you to share, upload the file to your AWS Support case. For more information on uploading this file, see [Working with diagnostic support scripts in AWS DMS \(p. 458\)](#).

With Amazon RDS for SQL Server, you can't connect using the sqlcmd command line utility, so use the following procedure.

#### To run the script for an RDS SQL Server database

1. Run the script using any client tool that allows you to connect to RDS SQL Server as the **Master** user and save the output as an HTML file.
2. Review the output HTML file and remove any information that you are uncomfortable sharing. When the HTML is acceptable for you to share, upload the file to your AWS Support case. For more information on uploading this file, see [Working with diagnostic support scripts in AWS DMS \(p. 458\)](#).

## Diagnostic support scripts for MySQL-compatible databases

Following, you can find the diagnostic support scripts available to analyze an on-premises or Amazon RDS for MySQL-compatible database in your AWS DMS migration configuration. These scripts work with either a source or target endpoint. The scripts are all written to run on the MySQL SQL command line.

Before running a script, ensure that the user account that you use has the necessary permissions to access your MySQL-compatible database. Use the following procedure to create a user account and provide the minimum permissions needed to run this script.

#### To set up a user account with the minimum permissions to run these scripts

1. Create the user to run the scripts.

```
create user 'username'@'hostname' identified by password;
```

2. Grant the select command on databases to analyze them.

```
grant select on database-name.* to username;  
grant replication client on *.* to username;
```

3. 

```
grant execute on procedure mysql.rds_show_configuration to username;
```

The following topics describe how to download, review, and run each support script available for a MySQL-compatible database. They also describe how to review and upload the script output to your AWS Support case.

#### Topics

- [awsdms\\_support\\_collector\\_MySQL.sql script \(p. 464\)](#)

## awsdms\_support\_collector\_MySQL.sql script

Download the [awsdms\\_support\\_collector\\_MySQL.sql](#) script.

This script collects information about your MySQL-compatible database configuration. Remember to verify the checksum on the script, and if the checksum verifies, review the SQL code in the script to comment out any of the code that you are uncomfortable running. After you are satisfied with the integrity and content of the script, you can run it.

Run the script after connecting to your database environment using the command line.

#### To run this script and upload the results to your support case

1. Connect to your database using the `mysql` command following.

```
mysql -h hostname -P port -u username database-name
```

2. Run the script using `mysql source` command following.

```
mysql> source awsdms_support_collector_MySQL_compatible_DB.sql
```

Review the generated report and remove any information that you are uncomfortable sharing. When the content is acceptable for you to share, upload the file to your AWS Support case. For more information on uploading this file, see [Working with diagnostic support scripts in AWS DMS \(p. 458\)](#).

#### Note

- If you already have a user account with required privileges described in [Diagnostic support scripts for MySQL-compatible databases \(p. 463\)](#), you can use the existing user account as well to run the script.
- Remember to connect to your database before running the script.
- The script generates its output in text format.
- Keeping security best practices in mind, if you create a new user account only to execute this MySQL diagnostic support script, we recommend that you delete this user account after successful execution of the script.

## PostgreSQL diagnostic support scripts

Following, you can find the diagnostic support scripts available to analyze any PostgreSQL RDBMS (on-premises, Amazon RDS, or Aurora PostgreSQL) in your AWS DMS migration configuration. These scripts work with either a source or target endpoint. The scripts are all written to run in the `psql` command-line utility.

Before running these scripts, ensure that the user account that you use has the necessary permissions following to access any PostgreSQL RDBMS:

- PostgreSQL 10.x or later – A user account with execute permission on the `pg_catalog.pg_ls_waldir` function.
- PostgreSQL 9.x or earlier – A user account with default permissions.

We recommend using an existing account with the appropriate permissions to run these scripts.

If you need to create a new user account or grant permissions to an existing account to run these scripts, you can execute the SQL commands following for any PostgreSQL RDBMS based on the PostgreSQL version.

#### To grant account permissions to run these scripts for a PostgreSQL 10.x or later database

- Do one of the following:
  - For a new user account, run the following.

```
CREATE USER script_user WITH PASSWORD 'password';  
GRANT EXECUTE ON FUNCTION pg_catalog.pg_ls_waldir TO script_user;
```

- For an existing user account, run the following.

```
GRANT EXECUTE ON FUNCTION pg_catalog.pg_ls_waldir TO script_user;
```

#### To grant account permissions to run these scripts for a PostgreSQL 9.x or earlier database

- Do one of the following:
  - For a new user account, run the following with default permissions.

```
CREATE USER script_user WITH PASSWORD password;
```

- For an existing user account, use the existing permissions.

#### Note

These scripts do not support certain functionality related to finding WAL size for PostgreSQL 9.x and earlier databases. For more information, work with AWS Support.

The following topics describe how to download, review, and run each support script available for PostgreSQL. They also describe how to review and upload the script output to your AWS Support case.

#### Topics

- [awsdms\\_support\\_collector\\_postgres.sql script \(p. 465\)](#)

## awsdms\_support\_collector\_postgres.sql script

Download the `awsdms_support_collector_postgres.sql` script.

This script collects information about your PostgreSQL database configuration. Remember to verify the checksum on the script. If the checksum verifies, review the SQL code in the script to comment out any of the code that you are uncomfortable running. After you are satisfied with the integrity and content of the script, you can run it.

#### Note

You can run this script with psql client version 10 or later.

You can use the procedures following to run this script either from your database environment or from the command line. In either case, you can then upload your file to AWS Support later.

**To run this script and upload the results to your support case**

1. Do one of the following:

- Run the script from your database environment using the `psql` command line following.

```
dbname=# \i awsdms_support_collector_postgres.sql
```

At the prompt following, enter the name of only one of the schemas that you want to migrate.

At the prompt following, enter the name of the user (*script\_user*) that you have defined to connect to the database.

- Run the script directly from directly from the command line following. This option avoids any prompts prior to script execution.

```
psql -h database-hostname -p port -U script_user -d database-name -f  
awsdms_support_collector_postgres.sql
```

2. Review the output HTML file and remove any information that you are uncomfortable sharing. When the HTML is acceptable for you to share, upload the file to your AWS Support case. For more information on uploading this file, see [Working with diagnostic support scripts in AWS DMS \(p. 458\)](#).

# Migrating large data stores using AWS Database Migration Service and AWS Snowball Edge

Larger data migrations can include many terabytes of information. This process can be cumbersome due to network bandwidth limits or just the sheer amount of data. AWS Database Migration Service (AWS DMS) can use [Snowball Edge](#) and Amazon S3 to migrate large databases more quickly than by other methods.

Snowball Edge is an AWS service that provides an Edge device that you can use to transfer data to the cloud at faster-than-network speeds. An Edge device is an AWS-owned appliance with large amounts of on-board storage. 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. Snowball Edge offers many additional features; for more information, see [What is an Snowball Edge?](#) in the *AWS Snowball Edge Developer Guide*.

Amazon S3 is an AWS 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 for the object and also for any metadata. For more information, see the [S3 documentation](#).

When you're using an Edge device, the data migration process has the following stages:

1. You use the AWS Schema Conversion Tool (AWS SCT) to extract the data locally and move it to an Edge device.
2. You ship the Edge device or devices back to AWS.
3. After AWS receives your shipment, the Edge device automatically loads its data into an Amazon S3 bucket.
4. 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 then applied to the target data store.

In this sections following, you can learn about using an Edge device to migrate relational databases with AWS SCT and AWS DMS. You can also use an Edge device and AWS SCT to migrate on-premises data warehouses to the AWS Cloud. For more information about data warehouse migrations, see [Migrating data from an on-premises data warehouse to Amazon Redshift](#) in the *AWS Schema Conversion Tool User Guide*.

## Topics

- [Overview of migrating large data stores using AWS DMS and AWS Snowball Edge \(p. 468\)](#)
- [Prerequisites for migrating large data stores using AWS DMS and AWS Snowball Edge \(p. 469\)](#)
- [Migration checklist \(p. 469\)](#)
- [Step-by-step procedures for migrating data using AWS DMS with AWS Snowball Edge \(p. 471\)](#)
- [Limitations when working with AWS Snowball Edge and AWS DMS \(p. 487\)](#)

# Overview of migrating large data stores using AWS DMS and AWS Snowball Edge

The process of using AWS DMS and Snowball Edge incorporates both on-premises applications and Amazon-managed services. We use the terms *local* and *remote* to distinguish these components.

Local components include the following:

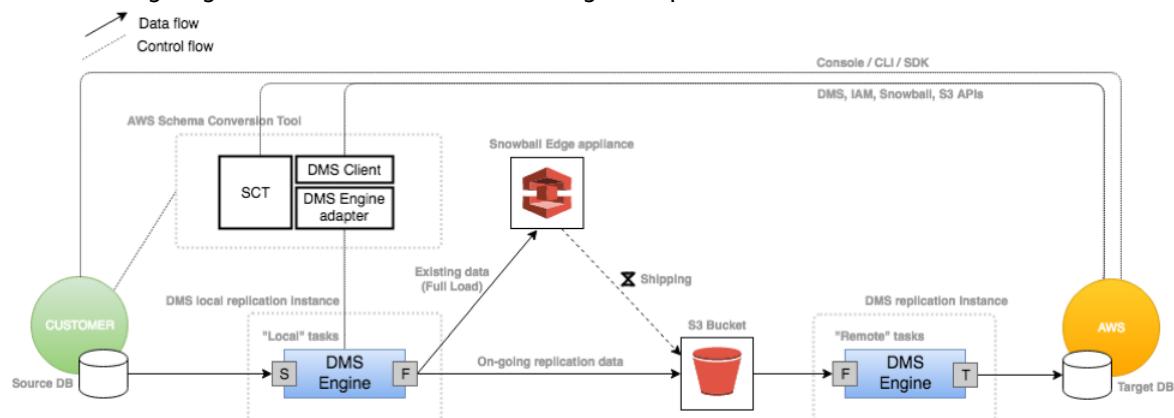
- AWS SCT
- AWS DMS Agent (a local version of AWS DMS that works on-premises)
- Snowball Edge devices

Remote components include the following:

- Amazon S3
- AWS DMS

In the following sections, you can find a step-by-step guide to configuring, installing, and managing an AWS DMS migration using an Edge device or devices.

The following diagram shows an overview of the migration process.



The migration involves a local task, where you move data to an Edge device using the DMS Agent. After an Edge device is loaded, you return it to AWS. If you have multiple Edge devices, you can return them at the same time, or in sequence. When AWS receives an Edge device, a remote task using AWS DMS loads the data to the target data store on AWS.

To migrate from a local data store to an AWS data store, you follow these steps:

1. Use the AWS Snow Family Management Console to create a new job for importing data to S3 with a **Snowball Edge Storage Optimized** device. The job involves requesting the device be sent to your address.
2. Set up AWS SCT on a local machine that can access AWS. Install the Snowball Edge client tool on a different local machine.
3. When the Edge device arrives, power it on, connect to it, then unlock it with the client tool. For step-by-step information, see [Getting started with AWS Snowball edge](#) in the *AWS Snowball Edge Developer Guide*.
4. Install the Open Database Connectivity (ODBC) drivers for your data sources. Put these on the machine with the Edge client tool.

5. Install and configure the AWS DMS Agent host on the machine with the Edge client tool.

The AWS DMS Agent must have connectivity to the source database, AWS SCT, AWS, and the Snowball Edge. The AWS DMS Agent is supported on the following Linux platforms only:

- Red Hat Enterprise Linux versions 6.2 through 6.8, 7.0, and 7.1 (64-bit)
- SUSE Linux version 12 (64-bit)

Although the AWS DMS Agent comes in the AWS SCT installation package, it's better that they aren't located in the same place. We recommend that you install the DMS Agent on a different machine—not the machine you installed AWS SCT on.

6. Create a new project in AWS SCT.

7. Configure AWS SCT to use the Snowball Edge device.

8. Register the AWS DMS Agent with AWS SCT.

9. Create a local and AWS DMS task in SCT.

10. Run and monitor the task in SCT.

## Prerequisites for migrating large data stores using AWS DMS and AWS Snowball Edge

Before you start the migration process, you need the following prerequisites:

- You are familiar with the basic operation of AWS SCT.
- You have or can create the S3 bucket or buckets to use for the migration.
- You have an AWS DMS replication instance in the same AWS Region as the S3 bucket.
- You are comfortable with using the AWS Command Line Interface (AWS CLI).
- You are familiar with the [Snowball Edge developer guide](#).

## Migration checklist

To make things easier during migration, you can use the following checklist to create a list of the items that you need during the migration.

DMS Migration Checklist

This checklist is for my schemas named:

The database engine that my schemas reside on is:

AWS Region for the migration:

Name of migration job that you created in the AWS Snowball Management Console:

S3 bucket (and folder) for this job:

IAM role that has access to the S3 Bucket and the target database on AWS:

Path to the installation directory of AWS SCT (needed for a future step):

Name/IP of Machine #1 (SCT):

Name/IP of Machine #2 (Connectivity):

---

IP address of your Snowball Edge:

Port for the Snowball Edge:

Unlock code for the Snowball Edge device:

Path to the manifest file:

Output of the command snowballEdge get-secret-access-key:

    AWS access key ID:

    AWS secret access Key:

---

Confirm ODBC drivers is installed on Machine #2 (Connectivity):

---

Confirm DMS Agent is installed on Machine #2 (Connectivity):

Confirm DMS Agent is running two processes:

DMS Agent password:

DMS Agent port number:

Confirm that your firewall allows connectivity:

---

Name of SCT project:

---

Confirm that DMS Agent is registered with SCT:

New agent or service profile name that you provided:

---

Confirm local and DMS task exists:

Task name that you provided:

---

Confirm:

DMS Agent connects to the following:

- \_\_ The source database
- \_\_ The staging S3 bucket
- \_\_ The Edge device

DMS task connects to the following:

- \_\_ The staging S3 bucket
- \_\_ The target database on AWS

Confirm the following:

- Stopped Edge client
- Powered off Edge device
- Returned Edge device to AWS

## Step-by-step procedures for migrating data using AWS DMS with AWS Snowball Edge

In the following sections, you can find detailed information on the migration steps.

### Step 1: Create a Snowball Edge job

Follow the steps outlined in the section [Getting started with an Snowball Edge device](#) in the *AWS Snowball Edge Developer Guide*. Open the AWS Snow Family Management Console, and create a new job for **Import into Amazon S3**.

Be sure to request a Snowball Edge device (**Snowball Edge Storage Optimized**), because regular Snowball devices are not supported for AWS DMS. Follow the screen prompts for remaining settings. You have a chance to review your settings before you create the job.

### Step 2: Download and install the AWS Schema Conversion Tool (AWS SCT)

You need two local machines to run this process, in addition to the Edge device.

Download the AWS Schema Conversion Tool app and install it on a local machine that can access AWS. For instructions, which include information on compatible operating systems, see [Installing and updating the AWS Schema Conversion Tool](#).

On a different machine, where you plan to install the AWS DMS Agent, download and install the Snowball Edge client from [AWS Snowball Edge resources](#).

After you finish this step, you should have two machines:

- Machine #1 (SCT), with AWS SCT installed
- Machine #2 (Connectivity), with the Edge client, where you plan to install the AWS DMS Agent and the ODBC drivers for the databases you are migrating

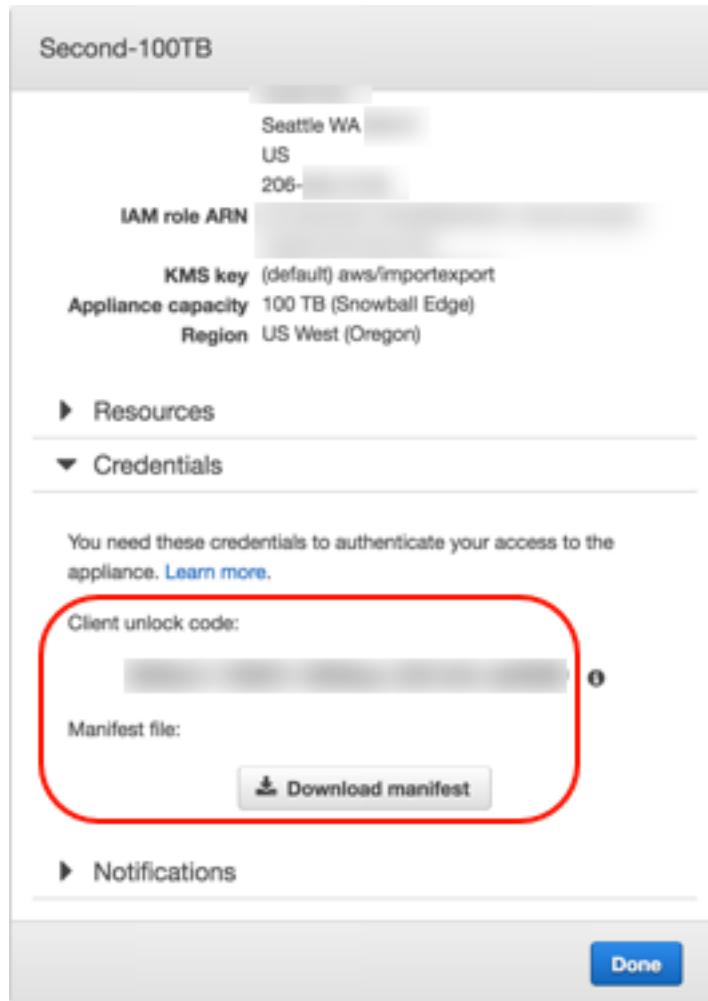
### Step 3: Unlock the AWS Snowball Edge device

When the Edge device arrives, prepare it for use.

Follow the steps outlined in the section [Getting started with an AWS Snowball Edge device](#) in the *AWS Snowball Edge Developer Guide*.

You can also check out the guided tour [How do I unlock an AWS Snowball Edge?](#) from AWS Support, or see the [AWS Snowball Edge getting started marketing page](#) for more resources.

Power the device on, [connect it to your local network](#), record the IP address of the Edge device, and [obtain the unlock code and manifest file](#) from the Snowball Edge console. In the console, choose your job, choose **View job details**, and then **Credentials**. Save both the client unlock code and the manifest file.



On the Edge device screen, get the IP of the Edge device from the **Connection** tab. Then unlock the device by using the `snowballEdge unlock` command with the IP and the credentials information. The following example shows the sample syntax for this command.

```
snowballEdge unlock -i IP_Address -m Local_path_to_manifest_file -  
u 29_character_unlock_code
```

Following is an example command.

```
snowballEdge unlock \  
-i 192.0.2.0 \  
-m /Downloads/JID2EXAMPLE-0c40-49a7-9f53-916aEXAMPLE81-manifest.bin \  
-u 12345-abcd-e12345-ABCDE-12345
```

Finally, [retrieve the Snowball Edge access key and secret key](#) from the device using the Edge client. The following shows example input and output for the command to get the access key.

#### Example input

```
snowballEdge list-access-keys \
--endpoint https://192.0.2.0 \
--manifest-file Path_to_manifest_file \
--unlock-code 12345-abcde-12345-ABCDE-12345
```

#### Example output

```
{ "AccessKeyId" : [ "AKIAIOSFODNN7EXAMPLE" ] }
```

The following shows example input and output for the command to get the secret key.

#### Example input

```
snowballEdge get-secret-access-key \
--access-key-id AKIAIOSFODNN7EXAMPLE \
--endpoint https://192.0.2.0 \
--manifest-file /Downloads/JID2EXAMPLE-0c40-49a7-9f53-916aEXAMPLE81-manifest.bin \
--unlock-code 12345-abcde-12345-ABCDE-12345
```

#### Example output

```
[snowballEdge]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

When the Snowball Edge is ready to use, you can interact with it directly by using the AWS CLI or S3 SDK Adapter for Snowball. This adapter also works with the Edge device.

## Step 4: Configure the AWS DMS agent host with ODBC drivers

Using Machine #2 (Connectivity) from step 2, where the Edge client is already installed, install the necessary ODBC drivers. These drivers are necessary to connect to your source database. The required driver varies by database engine. In the following sections, you can find information for each database engine.

### Topics

- [Oracle \(p. 473\)](#)
- [Microsoft SQL Server \(p. 474\)](#)
- [ASE SAP Sybase \(p. 474\)](#)
- [MySQL \(p. 474\)](#)
- [PostgreSQL \(p. 475\)](#)

## 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 you use the following.

```
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 /opt/amazon/aws-schema-conversion-tool-dms-agent/bin/site_arep_login.sh  
  
export ORACLE_HOME=/usr/lib/oracle/12.2/client64;  
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib
```

## Microsoft SQL Server

Install the Microsoft ODBC Driver.

For ODBC 17 drivers, update the `site_arep_login.sh` script with the following code.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/microsoft/msodbcsql17/lib64/
```

For ODBC drivers earlier than ODBC 17, update the `site_arep_login.sh` script with the following code.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/microsoft/msodbcsql/lib64/
```

## ASE SAP Sybase

The SAP Sybase ASE ODBC 64-bit client should be installed.

If the installation directory is `/opt/sap`, update the `site_arep_login.sh` script with the following.

```
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` file should include the following entries.

```
[Sybase]  
Driver=/opt/sap/DataAccess64/ODBC/lib/libsybdrvodb.so  
Description=Sybase ODBC driver
```

## 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 shown in the following example.

```
[MySQL ODBC 5.2.6 Unicode Driver]  
Driver = /usr/lib64/libmyodbc5w.so  
UsageCount = 1
```

The `/etc/odbcinst.ini` file contains information about ODBC drivers available to users and can be edited by hand. If necessary, you can use the `odbcinst -j` command to look for the `/etc/odbcinst.ini` file, as shown in the following example.

```
$ odbcinst -j  
  
DRIVERS.....: /etc/odbcinst.ini
```

## PostgreSQL

Install `postgresql94-9.4.4-1PGDG.OS Version.x86_64.rpm`. This package 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 shown in the following example.

```
[PostgreSQL]  
Description = PostgreSQL ODBC driver  
Driver = /usr/pgsql-9.4/lib/pgsqlodbc.so  
Setup = /usr/pgsql-9.4/lib/pgsqlodbcw.so  
Debug = 0  
CommLog = 1  
UsageCount = 2
```

## Step 5: Install the AWS DMS Agent

Using Machine #2 (Connectivity) from step 2, where the Edge client and the ODBC drivers are already installed, install and configure the AWS DMS Agent. The AWS DMS Agent is provided as part of the [AWS SCT installation package](#), described in the [AWS Schema Conversion Tool User Guide](#).

After you finish this step, you should have two local machines prepared:

- Machine #1 (SCT) with AWS SCT installed
- Machine #2 (Connectivity) with the Edge client, the ODBC drivers, and the DMS Agent installed

### To install the AWS DMS Agent

1. In the AWS SCT installation directory, locate the RPM file called `aws-schema-conversion-tool-dms-agent-2.4.1-R1.x86_64.rpm`.  
Copy it to Machine #2 (Connectivity), the AWS DMS Agent machine. SCT and the DMS Agent should be installed on separate machines. AWS DMS Agent should be located on the same machine as the Edge client and the ODBC drivers.
2. On Machine #2 (Connectivity), run the following command to install the DMS Agent. To simplify permissions, run this command as the `root` user.

```
sudo rpm -i aws-schema-conversion-tool-dms-agent-2.4.0-R2.x86_64.rpm
```

This command uses the default installation location of `/opt/amazon/aws-schema-conversion-tool-dms-agent`. To install the DMS Agent to a different location, use the following option.

```
sudo rpm --prefix installation_directory -i aws-schema-conversion-tool-dms-agent-2.4.0-R2.x86_64.rpm
```

3. To verify that the AWS DMS Agent is running, use the following command.

```
ps -ef | grep repctl
```

The output of this command should show two processes running.

To configure the AWS DMS Agent, you must provide a password and port number. You use the password later to register the AWS DMS Agent with AWS SCT, so keep it handy. Pick an unused port number for the AWS DMS Agent to listen on for AWS SCT connections. You might have to configure your firewall to allow connectivity.

Now configure the AWS DMS Agent using the `configure.sh` script.

```
sudo /opt/amazon/aws-schema-conversion-tool-dms-agent/bin/configure.sh
```

The following prompt appears. Enter the password. When prompted, enter the password again to confirm it.

```
Configure the AWS Schema Conversion Tool DMS Agent server
Note: you will use these parameters when configuring agent in AWS Schema
      Conversion Tool

Please provide password for the server
Use minimum 8 and up to 20 alphanumeric characters with at least one digit and one
      capital case character

Password:
```

The output is as follows. Provide a port number.

```
chown: missing operand after 'amazon:amazon'
Try 'chown --help' for more information.
/opt/amazon/aws-schema-conversion-tool-dms-agent/bin/repctl:
    /opt/amazon/aws-schema-conversion-tool-dms-agent/lib/libcom_err.so.3: no version
        information available (required by
            /opt/amazon/aws-schema-conversion-tool-dms-agent/lib/libgssapi_krb5.so.2)
    /opt/amazon/aws-schema-conversion-tool-dms-agent/bin/repctl:
        /opt/amazon/aws-schema-conversion-tool-dms-agent/lib/libcom_err.so.3: no version
            information available (required by
                /opt/amazon/aws-schema-conversion-tool-dms-agent/lib/libkrb5.so.3)
[setserverpassword command] Succeeded

Please provide port number the server will listen on (default is 3554)
Note: you will have to configure your firewall rules accordingly
Port:
```

The output is as follows, confirming that the service is started.

```
Starting service...
/opt/amazon/aws-schema-conversion-tool-dms-agent/bin/repctl:
    /opt/amazon/aws-schema-conversion-tool-dms-agent/lib/libcom_err.so.3: no version
        information available (required by
            /opt/amazon/aws-schema-conversion-tool-dms-agent/lib/libgssapi_krb5.so.2)
    /opt/amazon/aws-schema-conversion-tool-dms-agent/bin/repctl:
```

```
/opt/amazon/aws-schema-conversion-tool-dms-agent/lib/libcom_err.so.3: no version
information available (required by
/opt/amazon/aws-schema-conversion-tool-dms-agent/lib/libkrb5.so.3)
AWS Schema Conversion Tool DMS Agent was sent a stop signal
AWS Schema Conversion Tool DMS Agent is no longer running
[service command] Succeeded
/opt/amazon/aws-schema-conversion-tool-dms-agent/bin/repctl:
/opt/amazon/aws-schema-conversion-tool-dms-agent/lib/libcom_err.so.3: no version
information available (required by
/opt/amazon/aws-schema-conversion-tool-dms-agent/lib/libgssapi_krb5.so.2)
/opt/amazon/aws-schema-conversion-tool-dms-agent/bin/repctl:
/opt/amazon/aws-schema-conversion-tool-dms-agent/lib/libcom_err.so.3: no version
information available (required by
/opt/amazon/aws-schema-conversion-tool-dms-agent/lib/libkrb5.so.3)
AWS Schema Conversion Tool DMS Agent was started as PID 1608
```

We recommend that you install the [AWS Command Line Interface](#) (AWS CLI). Using the AWS CLI, you can interrogate the Snowball Edge to see the data files written to the device. You use the AWS credentials retrieved from the Edge to access the Edge device. For example, you might run the following command.

```
aws s3 ls --profile SnowballEdge --endpoint https://192.0.2.0:8080 bucket-name --recursive
```

This command produces the following output.

```
2018-08-20 10:55:31 53074692 streams/load00000001000573E166ACF4C0/00000001.fcd.gz
2018-08-20 11:14:37 53059667 streams/load00000001000573E166ACF4C0/00000002.fcd.gz
2018-08-20 11:31:42 53079181 streams/load00000001000573E166ACF4C0/00000003.fcd.gz
```

To stop the AWS DMS Agent, run the following command in the /opt/amazon/aws-schema-conversion-tool-dms-agent/bin directory.

```
./aws-schema-conversion-tool-dms-agent stop
```

To start the AWS DMS Agent, run the following command in the /opt/amazon/aws-schema-conversion-tool-dms-agent/bin directory.

```
./aws-schema-conversion-tool-dms-agent start
```

## Step 6: Create a new AWS SCT project

Next, you create a new AWS SCT project that specifies the source and target databases. For more information, see [Creating an AWS Schema Conversion Tool project](#) in the *AWS Schema Conversion Tool User Guide*.

### To create a new project in AWS SCT

1. Start AWS SCT, and choose **File** then **New Project**. The **New Project** dialog box appears.
2. Add the following project information.

For this parameter	Do this
<b>Project Name</b>	Enter a name for your project, which is stored locally on your computer.
<b>Location</b>	Enter the location for your local project file.
<b>OLTP</b>	Choose <b>Transactional Database (OLTP)</b> .

For this parameter	Do this
<b>Source DB Engine</b>	Choose your source data store.
<b>Target DB Engine</b>	Choose your target data store.

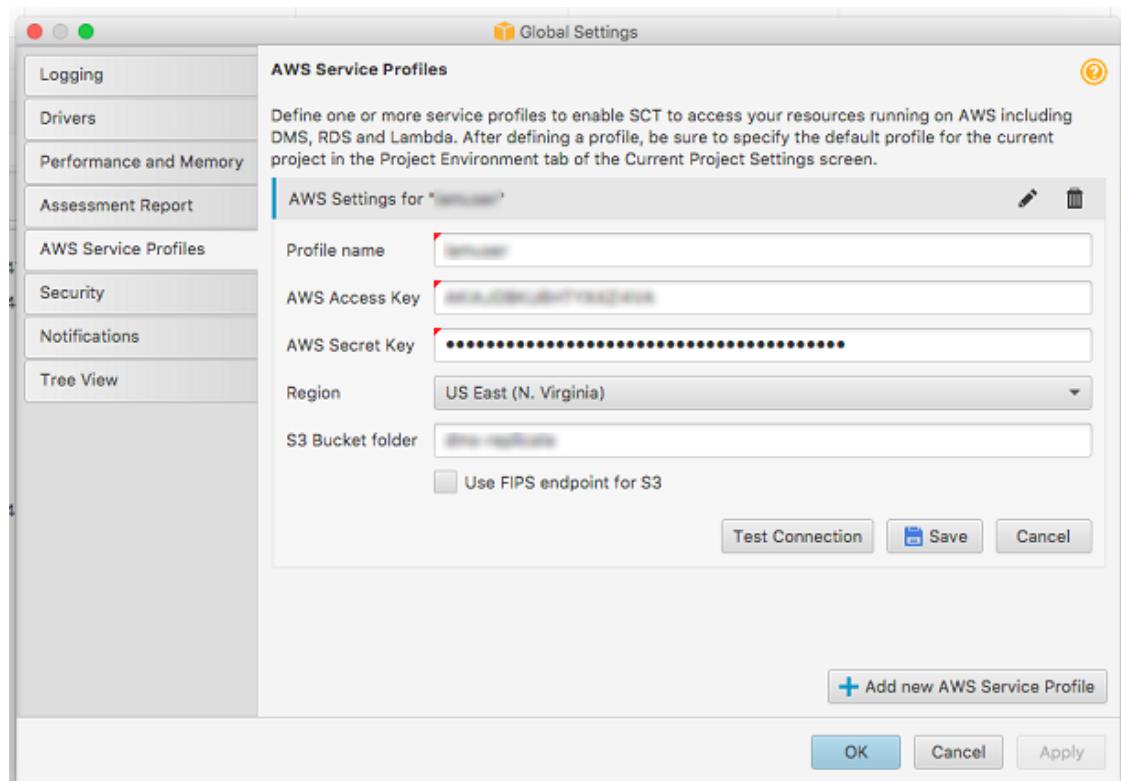
3. Choose **OK** to create your AWS SCT project.
4. Connect to your source and target databases.

## Step 7: Configure AWS SCT to use the AWS Snowball Edge

Your AWS SCT service profile must be updated to use the AWS DMS Agent, which is a local AWS DMS that works on-premises.

### To update the AWS SCT profile to work with the AWS DMS agent

1. Start AWS SCT.
2. Choose **Settings, Global Settings, AWS Service Profiles**.
3. Choose **Add New AWS Service Profile**.



4. Add the following profile information.

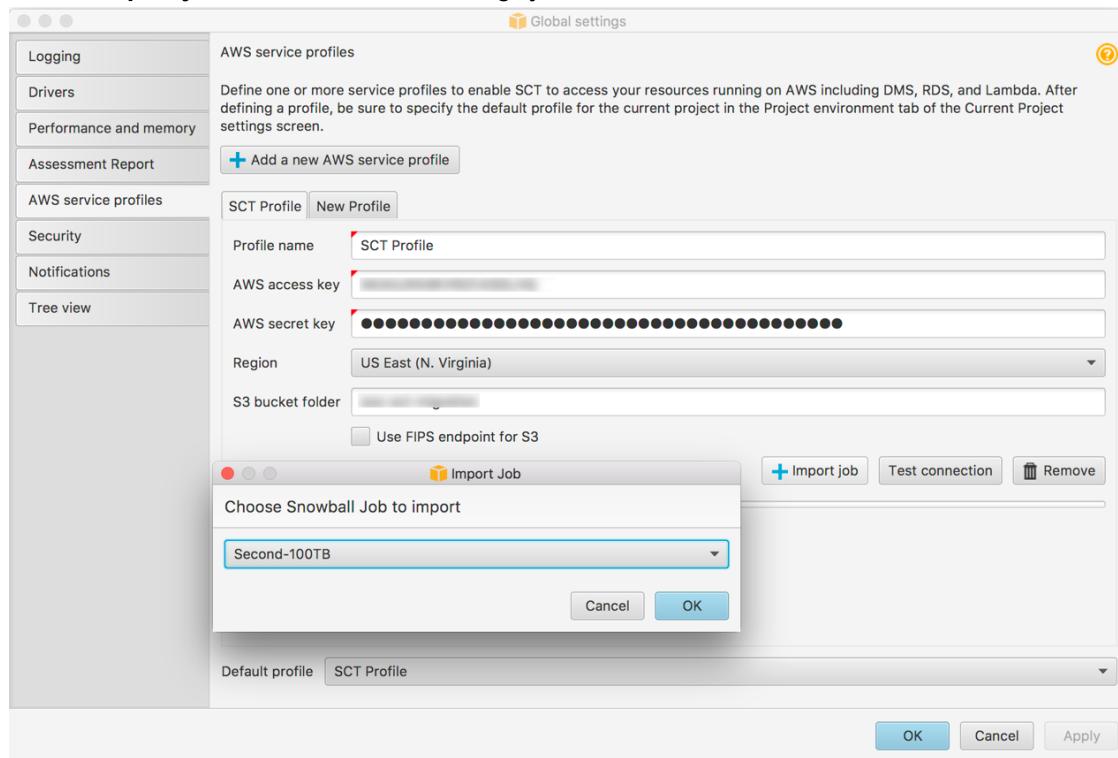
For this parameter	Do this
<b>Profile Name</b>	Enter a name for your project, which is stored locally on your computer.

For this parameter	Do this
<b>AWS Access Key</b>	Enter the AWS access key for the AWS account and AWS Region that you plan to use for the migration. The supplied credentials must have permissions to access the Snowball Edge job in AWS.
<b>AWS Secret Key</b>	Enter the AWS secret key for the AWS account and AWS Region that you plan to use for the migration.
<b>Region</b>	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.
<b>S3 Bucket folder</b>	Enter a name for S3 bucket that you were assigned when you created the Snowball Edge job.

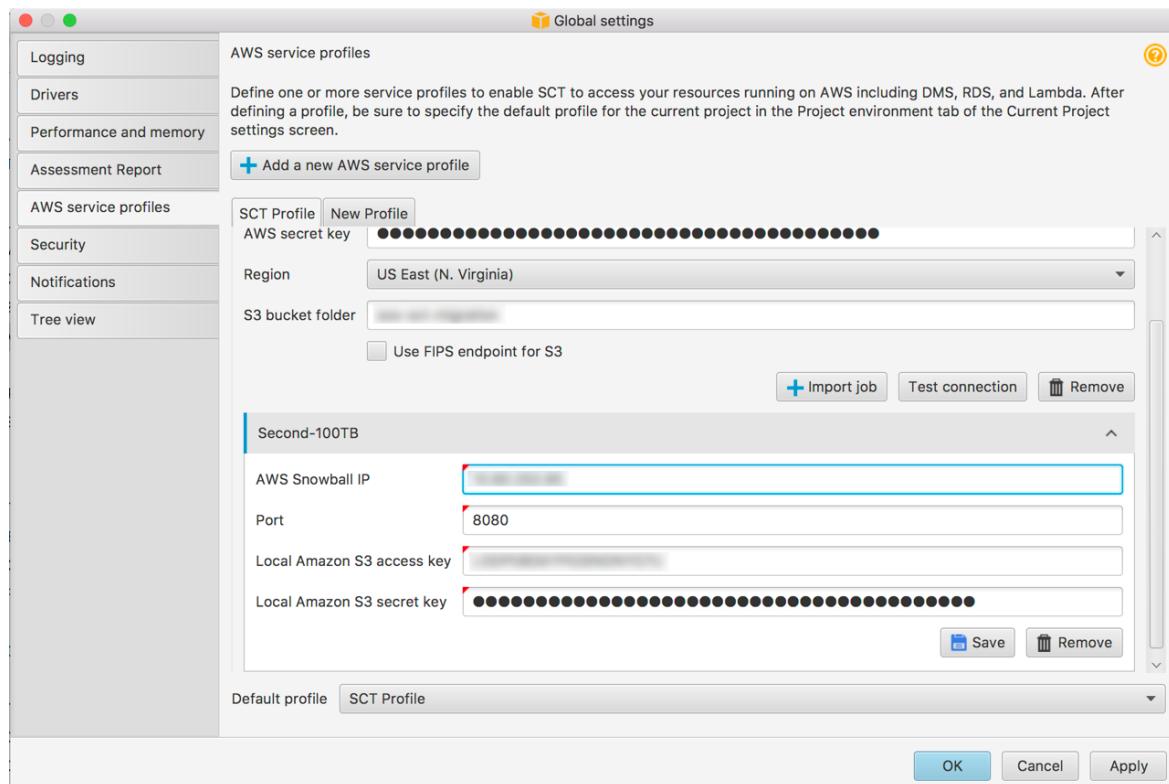
- After you have entered the information, choose **Test Connection** to verify that AWS SCT can connect to the Amazon S3 bucket.

The **OLTP Local & AWS 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 is missing privileges to access the Amazon S3 bucket.

- If the test passes, choose **OK** and then **OK** again to close the window and dialog box.
- Choose **Import job**, choose the Snowball Edge job from the list, and then choose **OK**.



Now configure AWS SCT to use the Snowball Edge. Enter the IP address of the Snowball Edge, the listening port on the device (the default is 8080), and the Snowball Edge access keys and secret keys you retrieved earlier. Choose **OK** to save your changes.

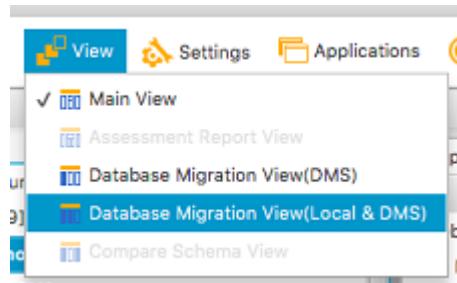


## Step 8: Register the AWS DMS Agent in AWS SCT

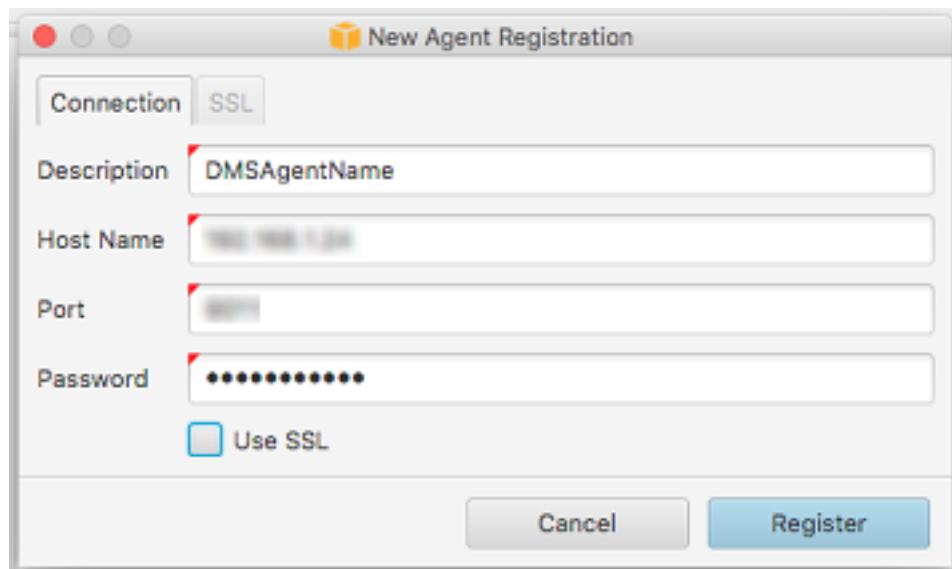
Next, you register the AWS DMS Agent in AWS SCT. SCT then tries to connect to the agent, showing status. When the agent is available, the status turns to active.

### To register the AWS DMS Agent

1. Start AWS SCT, choose **View**, and then choose **Database Migration View (Local & DMS)**.



2. Choose the **Agent** tab, and then choose **Register**. The **New Agent Registration** dialog box appears.



3. Enter your information in the **New Agent Registration** dialog box.

For this parameter	Do this
Description	Enter the name of the agent.
Host Name	Enter the IP address of the machine where you installed the DMS Agent.
Port	Enter the port number that you used when you configured the DMS Agent.
Password	Enter the password that you used when you configured the DMS Agent.

4. Choose **Register** to register the agent with your AWS SCT project.

## Step 9: Create a local and AWS DMS task

Next, you create the task that is the end-to-end migration task. This task includes two subtasks:

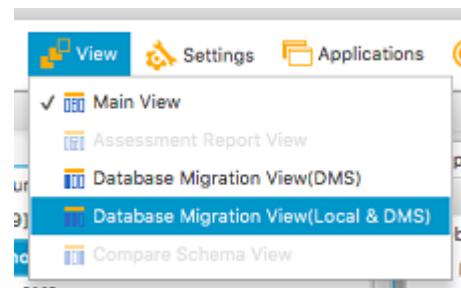
- The local subtask – This task migrates data from the source database to the Snowball Edge appliance.
- The AWS DMS subtask – This task moves the data from the appliance into an Amazon S3 bucket and migrates it to the target database.

### Note

We recommend that you test your migration before you use the Snowball Edge device. You can do this by setting up a task to send data, such as a single table, to an Amazon S3 bucket instead of to the Snowball Edge device.

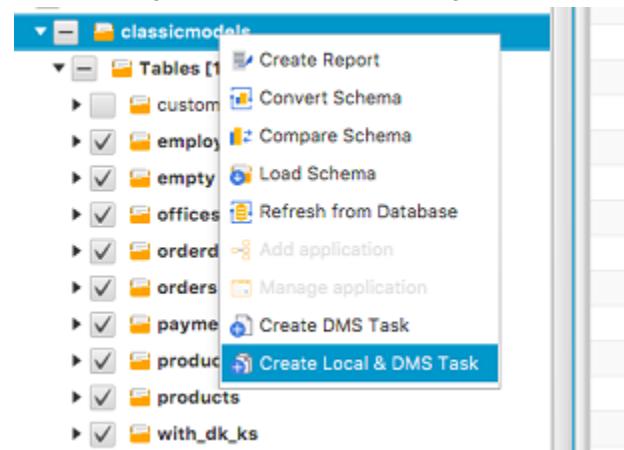
### To create the end-to-end migration task

1. Start AWS SCT, choose **View**, and then choose **Database Migration View (Local & DMS)**.

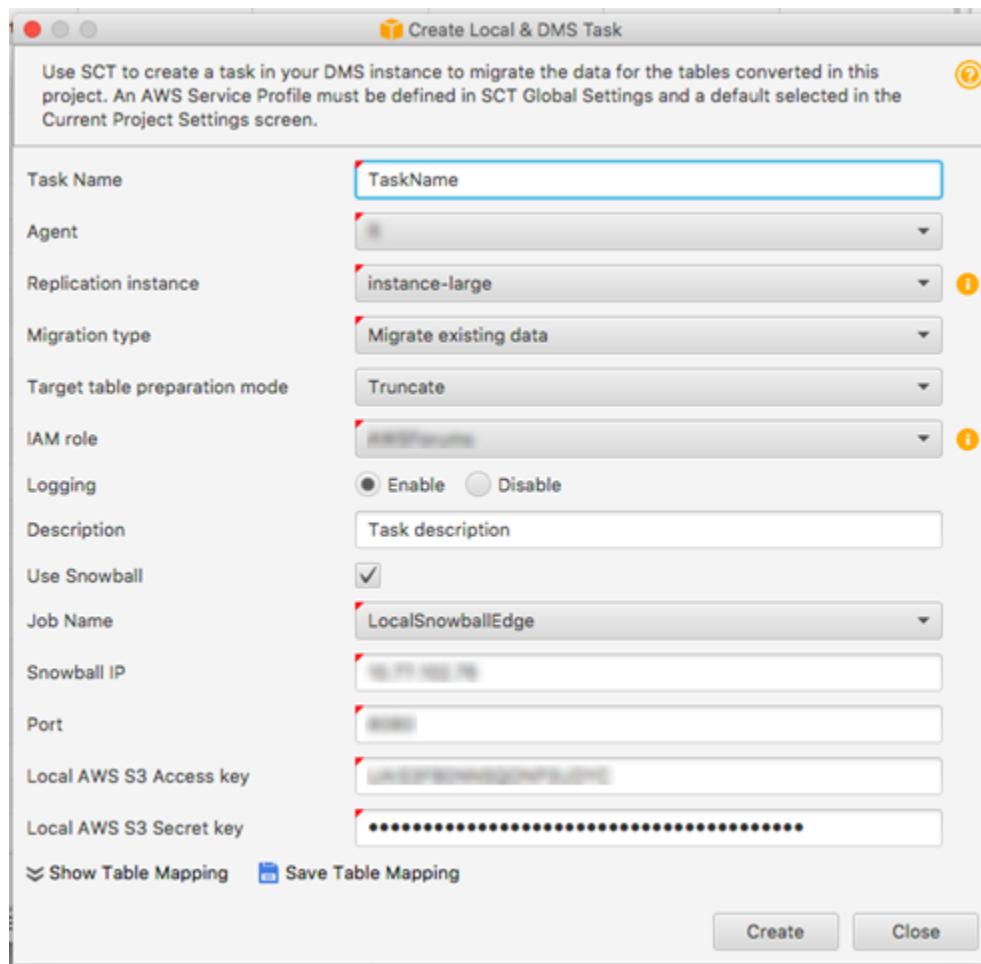


2. In the left panel that displays the schema from your source database, choose a schema to migrate. Open the context (right-click) menu for the schema, and then choose **Create Local & DMS Task**.

You can't migrate individual tables using AWS DMS and Snowball Edge.



The following screen appears.



3. Add your task information.

For this parameter	Do this
<b>Task Name</b>	Enter a name for the task.
<b>Agent</b>	Choose <b>DMS Agent</b> .
<b>Replication Instance</b>	Choose the AWS DMS replication instance that you want to use.
<b>Migration Type</b>	<p>Choose the type of migration you want:</p> <p>Choose <b>Migrate existing data</b> to migrate the contents of the chosen schema. This process is called a <i>full load</i> in AWS DMS.</p> <p>Choose <b>Migrate existing data and replicate ongoing changes</b> to migrate the contents of the chosen schema and capture all ongoing changes to the database. This process is called <i>full load and CDC</i> in AWS DMS.</p>
<b>Target table preparation mode</b>	Choose the preparation mode you want to use:

For this parameter	Do this
	<p><b>Truncate</b> – Tables are truncated without affecting table metadata.</p> <p><b>Drop tables on target</b> – The tables are dropped and new tables are created in their place.</p> <p><b>Do nothing</b> – Data and metadata of the target tables are not changed.</p>
<b>IAM role</b>	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 <a href="#">Prerequisites when using S3 as a source for AWS DMS</a> .
<b>Compression format</b>	<p>Choose whether to have uploaded files compressed or not:</p> <p><b>GZIP</b> – Files are compressed before loading. This is the default.</p> <p><b>No Compression</b> – Extracts are faster but take more space.</p>
<b>Logging</b>	Choose <b>Enable</b> to have Amazon CloudWatch create logs for the migration. You incur charges for this service. For more information about CloudWatch, see <a href="#">How amazon CloudWatch works</a> .
<b>Description</b>	Enter a description of the task.
<b>S3 Bucket</b>	Enter the name of an S3 bucket configured for this Snowball Edge job in the Snowball Edge console
<b>Use Snowball Edge</b>	Choose this check box to use Snowball Edge. If this box is not checked, then data is uploaded directly to the S3 bucket.
<b>Job Name</b>	Choose the Snowball Edge job name you created.
<b>Snowball Edge IP</b>	Enter the IP address of the Snowball Edge appliance.
<b>Port</b>	Enter the port value for the Snowball Edge appliance.
<b>Local Amazon S3 Access key</b>	Enter the local Snowball Edge access key you retrieved from the device.
<b>Local Amazon S3 Secret key</b>	Enter the local Snowball Edge secret key you retrieved from the device.

4. Choose **Create** to create the task.

## Step 10: Run and monitor the task in SCT

You can start the migration task when connections to all endpoints are successful, including the following:

- AWS DMS Agent connections to these:
  - The source database
  - The staging Amazon S3 bucket
  - The Edge device
- AWS DMS Task connections to these:
  - The staging Amazon S3 bucket
  - The target database on AWS

If all connections are functioning properly, your SCT console resembles the following screenshot, and you are ready to begin.

The screenshot shows the AWS SCT interface. The top navigation bar has tabs for 'Agents', 'Tasks', and 'Snowball'. The 'Tasks' tab is selected. Below the tabs is a table with the following columns: Task Name, Status, Complete %, Elapsed Time, Tables Loaded, Tables Loading, and Tables... . A single row is present with the task name 'abhinavmigration', status 'Running' (indicated by a plus sign icon), complete percentage at 0%, elapsed time at 0, and tables loaded, loading, and total at 0. Below the table is a toolbar with buttons for Start (blue triangle), Stop (grey square), Delete (trash can), Test (blue document), Refresh (refresh arrow), and Show log (log icon). The main area below the toolbar is titled 'Task details'. It contains sections for 'Version:' (Round Trip Latency (ms): 0), 'Source Endpoint:' (Name: [empty], Status: Successful, Connection message: To sqlserver: 'Connection successful' / To S3/Snowball: 'Connection successful'), and 'Target Endpoint:' (Name: [empty], Status: Successful, Connection message: To S3: 'Connection successful' / To aurora-postgresql: 'Connection successful').

Use the following procedure to start the migration.

### To start the migration task

1. Choose the migration task, then choose **Start**.

Task Name	Status	Complete %	Elapsed Time	Tables Loaded	Tables Loading	Tables...
abhinavmigration		0%		0	0	0
abhinavmigration-Local-generated		0%		0	0	0
abhinavmigration-DMS-generated		0%		0	0	0
snowball-migration		0%		0	0	0
snowball-migration-Local-generated		0%		0	0	0
snowball-migration-DMS-generated		0%		0	0	0

▶ Start ■ Stop >Delete Test Refresh Show log

Task details

ID: f26b735c2dbc462283d5a89f2d5a7961  
Task name: abhinavmigration

- To monitor the AWS DMS Agent, choose **Show Log**. The log details include the agent server (**Agent Log**) and local running task (**Task Log**) logs. The endpoint connectivity is done by the server. Because the local task isn't running, it has 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...

- Verify that the status of the migration task is 50 percent. You can use the Snowball Edge console or AWS SCT to check the status of the device.

Task Name	Status	Complete %	Elapsed Time	Tables
DemoTask		50%		0
DemoTask-Local		100%	0h:00m:11s	10
DemoTask-DMS		0%	0h:01m:00s	0

After the source tables have been loaded onto the Snowball Edge appliance, AWS SCT updates the status of the task to show it is 50 percent complete. This is because the other half of the task involves AWS DMS taking the data from Amazon S3 to the target data store.

- Follow the steps outlined in the Snowball Edge documentation, starting with the section named [Stop the Snowball client, and power off the Snowball Edge](#). These steps include the following:
  - Stopping the Snowball Edge client

- Powering off the Edge device
  - Returning the Edge device to AWS
5. Finishing the migration after the device returns to AWS involves waiting for the remote task to complete.

When the Snowball Edge appliance arrives at AWS, the remote (DMS) task starts to run. If the migration type you chose was **Migrate existing data**, the status for the AWS DMS task show 100 percent complete after the data has been transferred from Amazon S3 to the target data store.

If you set the task mode to include ongoing replication, then after the full load is complete the task continues to run while AWS DMS applies ongoing changes.

## Limitations when working with AWS Snowball Edge and AWS DMS

There are some limitations you should be aware of when working with AWS Snowball Edge:

- Every AWS SCT task creates two endpoint connections on AWS DMS. If you create multiple tasks, you can reach a resource limit for the number of endpoints that you can create.
- A schema is the minimum task scope when using Snowball Edge. You can't migrate individual tables or subsets of tables using Snowball Edge.
- The AWS DMS Agent doesn't support HTTP/HTTPS or SOCKS proxy configurations. Connections to the source and target can fail if the AWS DMS agent host uses proxies.
- The LOB mode limits LOB file size to 32 K. LOBs larger than 32 K aren't migrated.
- In some cases, an error can occur when loading from the local database to the Edge device or when loading data from Amazon S3 to the target database. In some of these cases, the error is recoverable and the task can restart. If AWS DMS can't recover from the error, the migration stops. If this happens, contact AWS Support.

# 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 homogeneous 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 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 \(p. 488\)](#)

## Data types for AWS Database Migration Service

AWS Database Migration Service uses built-in data types to migrate data from a source database engine type to a target 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.
INT1	A one-byte, signed integer.

AWS DMS data types	Description
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.

**Note**

AWS DMS can't migrate any LOB data type to an Apache Kafka endpoint.

# AWS DMS release notes

Following, you can find release notes for current and previous versions of AWS Database Migration Service (AWS DMS).

AWS DMS uses a semantic versioning scheme to identify a service release. A version consist of a three-component number in the format of X.Y.Z where X represents a **major** version, Y represents a **minor** version, and Z represents a **patch** (Major.Minor.Patch).

**Note**

You can upgrade any version of AWS Database Migration Service to any later version. To stay current with the latest features, enhancements, and performance improvements, upgrade to AWS DMS version 3.4.1 Beta.

## AWS Database Migration Service 3.4.1 Beta release notes

The table following shows the new features and enhancements introduced in AWS DMS version 3.4.1 Beta.

New feature or enhancement	Description
New MongoDB version	MongoDB version 4.0 is now supported as a source.
TLS 1.2 support for SQL Server	AWS DMS now supports TLS 1.2 for SQL Server endpoints.

The issues resolved in AWS DMS 3.4.1 Beta include the following:

- Improved Oracle 19c TDE support.
- Improved support of utf8mb4 character set and identity data type using Redshift as a target.
- Improved replication task failure handling when using MySQL as a source and binary log is not present.
- Improved data validation support on various data types and character sets.
- Improved null value handling with a new endpoint setting `IncludeNullAndEmpty` when using Kinesis and Kafka as a target.
- Improved error logging and handling when using Kafka as a target.
- Improved DST time offset when using SQL Server as a source.
- Fixed an issue where replication tasks try to create existing tables for Oracle as a target.
- Fixed an issue where replication tasks get stuck after the database connection is killed when using Oracle as a source.
- Fixed an issue where replication tasks failed to detect and reconnect to the new primary when using SQL Server as a source with AlwaysON setting.

- Fixed an issue where replication tasks do not add a "D" for "OP" column under certain conditions for S3 as a target.

## AWS Database Migration Service 3.4.0 Beta release notes

The table following shows the new features and enhancements introduced in AWS DMS version 3.4.0.

New feature or enhancement	Description
New MySQL version	AWS DMS now supports MySQL version 8.0 as a source, except when the transaction payload is compressed.
TLS 1.2 support for MySQL	AWS DMS now supports TLS 1.2 for MySQL endpoints.
New MariaDB version	AWS DMS now supports MariaDB version 10.3.13 as a source.
Non-SysAdmin access to self-managed Microsoft SQL Server sources	AWS DMS now supports access by non-SysAdmin users to on-premise and EC2-hosted SQL Server source endpoints. <b>Note</b> This feature is currently in Beta. If you want to try it out, contact AWS support for more information.
CDC tasks and Oracle source tables created using CREATE TABLE AS	AWS DMS now supports both full-load and CDC and CDC-only tasks running against Oracle source tables created using the CREATE TABLE AS statement.

The issues resolved in AWS DMS 3.4.0 include the following:

- Improved premigration task assessments. For more information, see [Enabling and working with premigration assessments for a task \(p. 353\)](#).
- Improved data validation for float, real, and double data types.
- Improved Amazon Redshift as a target by better handling this error: "The specified key does not exist."
- Supports multithreaded CDC load task settings, including `ParallelApplyThreads`, `ParallelApplyBufferSize`, and `ParallelApplyQueuesPerThread`, for Amazon Elasticsearch Service (Amazon ES) as a target.
- Improved Amazon ES as a target by supporting its use of composite primary keys.
- Fixed an issue where test connection fails when using PostgreSQL as a source and the password has special characters in it.
- Fixed an issue with using SQL Server as a source when some VARCHAR columns are truncated.
- Fixed an issue where AWS DMS does not close open transactions when using Amazon RDS SQL Server as a source. You can also prevent this from happening in these cases:
  - If the task is configured for full load and CDC or CDC only, set the a capture polling interval to more than 300 seconds. If this interval is set to 300 seconds or shorter, the task fails with an error message.
  - If you do not configure an appropriate polling interval level for a Capture Job, AWS DMS completes the migration with missing records on the target. Amazon RDS SQL Server transaction logs are regularly copied and removed from the instance. AWS DMS can refer to backup transaction logs to find older change records if the transaction log backup file is not already removed from the instance storage.

- Fixed an issue for Oracle Standby as source where CDC tasks would stop unexpectedly when using Binary Reader.
- Fixed an issue for IBM DB2 for LUW where the task failed with the message "The Numeric literal 0 is not valid because its value is out of range."
- Fixed an issue for a PostgreSQL to PostgreSQL migration when a new column was added on the PostgreSQL source and the column was created with a different data type than the data type for which the column was originally created on the source.
- Fixed an issue with a MySQL source the migration task stopped unexpectedly when it was unable to fetch binlogs.
- Fixed an issue related to an Oracle target when `BatchApply` was being used.
- Fixed an issue for MySQL and MariaDb when migrating the `TIME` data type.
- Fixed an issue for an IBM DB2 LUW source where migrating tables with LOBs fail when the tables don't have a primary key or unique key.

## AWS Database Migration Service 3.3.4 release notes

The issues resolved in AWS DMS 3.3.4 include the following:

- Fixed an issue where transactions are dropped or duplicated when using PostgreSQL as a source.
- Improved the support of using dollar sign (\$) in schema names.
- Fixed an issue where replication instances do not close open transactions when using RDS SQL Server as a source.
- Fixed an issue where test connection fails when using PostgreSQL as a source and the password has special characters in it.
- Improved Amazon Redshift as a target by better handling this error: "The specified key does not exist."
- Improved data validation support on various data types and character sets.
- Fixed an issue where replication tasks try to create existing tables for Oracle as a target.
- Fixed an issue where replication tasks do not add a "D" for "OP" column under certain conditions for Amazon S3 as a target.

## AWS Database Migration Service 3.3.3 release notes

The table following shows the new features and enhancements introduced in AWS DMS version 3.3.3.

New feature or enhancement	Description
New PostgreSQL version	PostgreSQL version 12 is now supported as a source and target.
Support for composite primary key with Amazon Elasticsearch Service as target	As of AWS DMS 3.3.3, use of a composite primary key is supported by Amazon ES targets.

New feature or enhancement	Description
Support for Oracle extended data types	Oracle extended data types for both Oracle source and targets are now supported.
Increased number of AWS DMS resources per account	The limit on the number of AWS DMS resources you can create has increased. For more information, see <a href="#">Limits for AWS Database Migration Service (p. 440)</a> .

The issues resolved in AWS DMS 3.3.3 include the following:

- Fixed an issue where a task crashes using a specific update statement with Parallel Apply in Amazon Kinesis.
- Fixed an issue where a task crashes on the ALTER TABLE statement with Amazon S3 as a target.
- Fixed an issue where values on polygon columns are truncated when using Microsoft SQL Server as a source.
- Fixed an issue on Unicode converter of JA16SJISTILDE and JA16EUCTILDE when using Oracle as a source.
- Fixed an issue where MEDIUMTEXT and LONGTEXT columns failed to migrate from MySQL to S3 comma-separated value (CSV) format.
- Fixed an issue where boolean columns were transformed to incorrect types with Apache Parquet output.
- Fixed an issue with extended varchar columns in Oracle.
- Fixed an issue where data validation tasks failed due to certain timestamp combinations.
- Fixed an issue with Sybase data definition language (DDL) replication.
- Fixed an issue involving an Oracle Real Application Clusters (RAC) source crashing with Oracle Binary Reader.
- Fixed an issue with validation for Oracle targets with schema names' case.
- Fixed an issue with validation of IBM Db2 versions 9.7 and 10.
- Fixed an issue for a task not stopping two times with `StopTaskCachedChangesApplied` and `StopTaskCachedChangesNotApplied` enabled.

## AWS Database Migration Service (AWS DMS) 3.3.2 release notes

The table following shows the features and bug fixes for version 3.3.2 of AWS Database Migration Service (AWS DMS).

New feature or enhancement	Description
Neptune as a target	Adds support to migrate data from an SQL relational database source to Amazon Neptune as a target. In addition to any table mapping to select and transform the SQL source data, this AWS DMS version supports additional mechanisms to map the selected relational data to a graph-mapping format used to load the target Neptune graph database.

New feature or enhancement	Description
Data transformations	Adds support in table mapping for using expressions with selected rule actions of the transformation rule type. For more information, see <a href="#">Using transformation rule expressions to define column content (p. 331)</a> . This also includes support for adding the before image of updated or deleted columns in the JSON written to either Kinesis Data Streams or Apache Kafka as a target. This includes a choice of all columns from the source, the primary key column, or columns without LOBs as a separate field. For more information, see <a href="#">Before image task settings (p. 296)</a> .
New Oracle version	Oracle version 19c is now supported as a source and target. Note that this support currently doesn't include the use of Oracle transparent data encryption (TDE) with Oracle version 19c as a source.
New SQL Server version	SQL Server 2019 is now supported as a source and target.
Support for LOBs in Kafka as a target	Replicating LOB data to a Kafka target is now supported.
Support for partial validation of LOBs	Adds support for partial validation of LOBs for MySQL, Oracle, and PostgreSQL endpoints.

The issues resolved are as follows:

- Multiple data validation related fixes:
  - Disabled validation for Oracle long types.
  - Improved retry mechanisms based on driver error codes.
  - Fixed an issue where validation hangs indefinitely while fetching data from source and target.
  - Fixed an issue with PostgreSQL UUID type validation.
  - Fixed multiple stuck validation issues.
- Fixed an issue where data loss occurs with S3 as a target when the replication task process crashes on the replication instance.
- Fixed an issue where columns of timestamp data types do not migrate when using attribute mappings in Kinesis and Kafka.
- Fixed an issue where replication tasks with Oracle or PostgreSQL as a source enter a "failed" state when you try to stop them.
- Fixed an issue where batch apply does not work as expected for replications to Redshift when using composite primary keys.
- Fixed an out of memory issue on the replication instance when using MariaDB as a source.
- Fixed an issue where AWS DMS inserts random values in an S3 target instead of null values as expected.

## AWS Database Migration Service (AWS DMS) 3.3.1 release notes

The table following shows the features and bug fixes for version 3.3.1 of AWS Database Migration Service (AWS DMS).

New feature or enhancement	Description
Support for bidirectional replication between two separate databases	Adds support to replicate changes between two different databases using two simultaneous tasks. One task replicates changes from a database A as source to a database B as target. The other task replicates changes from the database B as source to the database A as target. You can use bidirectional replication between a limited combination of endpoint types. Bidirectional replication also includes loopback prevention to prevent an identical change from being replicated in both tasks, which can corrupt the data. For more information, see <a href="#">Performing bidirectional replication (p. 304)</a>
Apache Kafka as a target	Adds support for both a self-managed Apache Kafka cluster and an Amazon Managed Streaming for Apache Kafka (Amazon MSK) cluster as a target.
Support for LOBs in Kinesis Data Streams as a target	Adds support for migrating LOBs from any of our supported sources to Kinesis Data Streams as a target.
Performance improvements for Kinesis Data Streams as a target	Adds improvements to PutRecords API call performance when writing ongoing changes to Kinesis Data Streams as a target.
Spatial data type migration support	Adds functionality to support migration of spatial data from Oracle to Oracle and PostgreSQL, and from PostgreSQL to PostgreSQL.
Improved support for replication from SQL Server when change tracking is enabled	Adds support for replicating ongoing changes from SQL Server as a source when change tracking is enabled. The task will not fail when change tracking is enabled as it previously did.
Support for the JSON data type in MySQL as a source	Adds functionality to support migration of the JSON data type from MySQL to any supported AWS DMS target.
Support for binary reader-based change capture in RDS for Oracle 12.2 as a source	Adds support for binary reader-based change capture in RDS for Oracle as a source in AWS DMS versions 12.2 and later.
Oracle driver upgrade for both sources and targets	Upgrades the underlying Oracle driver in the AWS DMS replication instance to take advantage of: <ul style="list-style-type: none"> <li>Better error handling</li> <li>Full support for data types, such as timestamp with time zone and timestamp with local time zone</li> <li>SSL V3</li> </ul>
Data validation for IBM Db2 LUW as a source	Adds support for data validation for Db2 LUW as a source.
Data validation improvements	Supports multiple data validation-related improvements.
S3 as a target (data lake) improvements	Adds functionality to capture and replicate inserts and updates only for Amazon S3 as a target.
DynamoDB as a target improvements	Adds support for migrating the CLOB data type to Amazon DynamoDB as a target.

The issues resolved are as follows:

- Resolved an issue where ongoing replication to PostgreSQL 11+ as a source fails with AWS DMS 3.3.0 Beta.
- Fixed the issues following with respect to replicating ongoing changes in Parquet/CSV format to Amazon S3 as a target:
  - Fixed a CSV/Parquet file missing data issue during task stop and resume.
  - Fixed a Parquet format bug when replicating null data.
  - Fixed handling of INT types when replicating data in Parquet format.
  - Fixed an issue where an ongoing replication only task with S3 as a target fails from a race condition in the replication engine.
  - Fixed a segmentation fault issue when writing ongoing changes in Parquet format to S3 as a target.
- Fixed an issue in Kinesis Data Streams as a target when `partition-key-type` in object mapping is not set to `primary-key`.
- Fixed an issue in data validation where a bulk fetch during ongoing replication has high latency. This causes issues with validation metrics.
- Fixed a memory leak issue in the AWS DMS replication instance in version 3.3 that causes replication tasks to stop.
- Fixed an issue with a parallel partition-based unload from Oracle as a source, where metadata queries used to construct reads from each partition are slow and sometimes hang.
- Fixed an issue where the `datetime` data type in Oracle is not migrated as expected to the targets RDS for PostgreSQL and Aurora PostgreSQL.
- Fixed a missing data issue when replicating ongoing changes from Oracle as a source to Amazon Redshift.
- Fixed multiple logging related issues for targets such as Amazon S3 and Amazon Redshift.
- Fixed an issue where DMS switches to using savepoints for PostgreSQL as a target, is stuck in a loop and never switches back to not use savepoints causing memory pressure.
- Fixed an issue where auto-partitions based unload does not work as expected with Db2 LUW as a source.
- Fixed an issue where AWS DMS does not cleanly handle the disconnect from the oplog for MongoDB source and continues to work without reporting disconnect issues.
- Fixed an issue where zero dates are not handled as expected when migrated to PostgreSQL and Redshift as a target.
- Fixed a timestamp formatting issue in Kinesis Data Streams, DynamoDB, and Elasticsearch as a target. If the precision for seconds is a single digit, it does not contain extra 0 padding.
- Fixed an issue with all relational targets where an alter table statement erroneously includes a NULL parameter. This causes an unexpected change to the target column's nullability.

## AWS Database Migration Service (AWS DMS) 3.3.0 Beta release notes

The table following shows the features and bug fixes for version 3.3.0 Beta of AWS Database Migration Service (AWS DMS).

### Note

Version 3.3.0 is now retired. You can directly upgrade your 3.3.0 version to version 3.3.2.

New feature or enhancement	Description
New Oracle versions	Oracle 18c is now supported as a source and target endpoint.
New PostgreSQL versions	PostgreSQL 10.x and 11.x are now supported as a source and target endpoint.
Enhancements to Oracle as a source endpoint	<p>Added support for the following:</p> <ul style="list-style-type: none"> <li>• Nested tables.</li> <li>• The <code>RESETLOGS</code> command.</li> <li>• Controlling the number of parallel read threads and read-ahead buffers for Oracle Automatic Storage Management (ASM) when collecting changes for a Change Data Capture (CDC) load. For more information on these enhancements for using Oracle ASM, see <a href="#">Configuration for change data capture (CDC) on an Oracle source database (p. 68)</a>.</li> </ul>
Enhancements to Microsoft SQL Server as a source endpoint	Added enhanced AlwaysOn support. You can now migrate changes from an AlwaysOn replica.
Support for large updates to Amazon DynamoDB	Added support to break large updates into multiple chunks less than 4 KB each and apply them to DynamoDB to work around an update limitation. A warning message notes when this happens.

The issues resolved are as follows:

- Fixed an issue where DMS cannot connect to an Azure SQL Server source.
- Fixed an issue where JSON LOB columns in PostgreSQL were not being migrated.
- Fixed an issue where 0000-00-00 dates were being migrated as 0101-01-01 in a MySQL to MySQL migration.
- Fixed an issue where DMS couldn't create a table with a numeric array column during ongoing replication to PostgreSQL.
- Fixed an issue where updates were not being applied in the target Amazon Redshift instance when the table did not have a PK.
- Fixed an issue where DMS couldn't select tables from schemas where names begin with "pg%" in PostgreSQL sources.
- Fixed an issue where a task failed when migrating a Boolean value from Aurora PostgreSQL to Amazon Redshift.
- Fixed an issue where special characters with EUC\_JP encoding were not being migrated from PostgreSQL to PostgreSQL.
- Fixed an issue where a DMS task fails because it can't create the `aws_dms_apply_exceptions` control table.
- Fixed issues with SQL Server and Oracle as a source where ongoing replication from a point in time fails with a "Not a valid time value supplied" error.
- Fixed an issue where ongoing replication breaks when a table has a primary key and a unique constraint, and there are ongoing changes to the unique constraint column.
- Fixed an issue where MySQL `TIMESTAMP(0)` column migration causes the task to stop replicating changes.

# AWS Database Migration Service (AWS DMS) 3.1.4 release notes

The table following shows the features and bug fixes for version 3.1.4 of AWS Database Migration Service (AWS DMS).

New feature or enhancement	Description
Adding a source commit timestamp column to Amazon S3 target files	You can now use a target endpoint setting to insert the source commit timestamp in an additional column of .csv or .parquet files when migrating to S3. You can also specify the name of the additional column. For more information, see <a href="#">timestampColumnName</a> .
Including the operation column for full load in Amazon S3 target files	To keep S3 target files consistent for both full load and ongoing replication operations, you can now specify a target endpoint setting that adds an operation column during full load (.csv format only). For more information, see <a href="#">includeOpForFullLoad</a> .
Making the formats of Amazon S3 source and Amazon S3 target files consistent	Added improvements to make S3 source file format consistent with S3 target file format. For more information on how operation column values are coded in S3 files, see the notes in <a href="#">Using Amazon S3 as a source for AWS DMS (p. 145)</a> . For more information on the options for coding these values, see <a href="#">Indicating source DB operations in migrated S3 data (p. 206)</a> .
Specifying the precision for <code>TIMESTAMP</code> columns in Amazon S3 target files with .parquet file format	For .parquet formatted S3 target files only, you can specify that all <code>TIMESTAMP</code> columns have either microsecond or millisecond precision. Specifying millisecond precision allows the data to be processed by Amazon Athena and AWS Glue. For more information, see <a href="#">parquetTimestampInMillisecond</a> .
Multithreaded loads into Amazon Kinesis Data Streams	This version of DMS enables faster loads into Kinesis Data Streams by using multiple threads.
Support for large updates to Amazon DynamoDB	Added support to break large updates into multiple chunks less than 4 KB each and apply them to DynamoDB to work around an update limitation. A warning message notes when this happens.
Various performance improvements to data validation	Added various performance improvements to data validation, including the ability to handle nullable unique keys.
Logging improvements	Added various logging improvements.

The issues resolved are as follows:

- Fixed an issue where DMS was failing to capture changes from an Oracle source with ASM configured and DMS is performing parallel change reads.
- Fixed an issue where DMS was failing to capture changes for an nvarchar column from Microsoft SQL Server sources when reading changes from transaction log backups.
- Fixed an issue where a DMS task was crashing because the replication process could not find details about a column from a table with supplemental logging enabled from table metadata in Oracle.

- Fixed an issue where DMS was passing an invalid parameter when trying to resume collecting changes from where it left off and looks for older changes that have been purged.
- Changed stream buffer messages in DMS task logs to log once instead of multiple times. Logging these messages multiple times caused underlying storage to become full and created other performance issues with the task.
- Fixed an error when DMS was returning the task state to be failed instead of stopped after a stop task was attempted by the user.
- Fixed a segmentation fault issue with IBM Db2 for Linux, UNIX, and Windows (Db2 LUW) as a source when extra connection attributes are used with Db2 as a source.
- Fixed several issues with fail on truncation and fail on LOB truncation with PostgreSQL as a source.
- Fixed an issue where the task didn't fail if an S3 source bucket didn't exist. The task now fails with a relevant error message.
- Fixed a memory leak when DMS migrates S3 data in Parquet file format.
- Included various fixes to data validation.
- Fixed an issue where a drop collection command in MongoDB caused a task failure if the collection being dropped wasn't in the task's scope.
- Fixed an issue where tasks with an S3 target endpoint crash because of NULL values in the data.
- Fixed an issue where bulk loads to S3 were slow because of new memory pool creates. Full loads to S3 are now 20 percent faster in DMS 3.1.4.
- Fixed an issue where LOB columns were migrating incorrectly from Oracle to Oracle.

## AWS Database Migration Service (AWS DMS) 3.1.3 release notes

The following tables show the features and bug fixes for version 3.1.3 of AWS Database Migration Service (AWS DMS).

**Note**

Version 3.1.3 is now retired. You can directly upgrade your 3.1.3 version to version 3.3.2.

New feature or enhancement	Description
Support for Amazon DocumentDB (with MongoDB compatibility) as a target	You can now use Amazon DocumentDB as a target to deploy a fast, reliable, and fully managed database service that makes it easy for you to set up, operate, and scale MongoDB-compatible databases.
Apache Parquet storage format support for Amazon S3 targets	You can now select and configure .parquet files as the storage format for S3 target objects.
AWS Key Management Service (AWS KMS) key encryption for Amazon Redshift and Amazon S3 target data	You can now use KMS keys for server-side encryption of data for Amazon Redshift and S3 targets.
Enhanced selection and data transformation with table mapping	You can now explicitly select a table and its schema. You can also transform table and index tablespaces for an Oracle target and update the primary key for a table on any target.

New feature or enhancement	Description
Support for tagging Amazon S3 target objects	You can now associate tags with each object written to your S3 target bucket.
TIMESTAMP data types migrated from Microsoft SQL Server to selected targets	TIMESTAMP data types migrated from Microsoft SQL Server to Amazon Elasticsearch Service or Amazon Kinesis targets are now converted from binary to a hex-encoded string (for example, 0x654738). Previously, these TIMESTAMP values were lost when mapped as binary data.
Transitions to DEFAULT or INFO logging level	AWS DMS now logs when log levels are changed back to DEFAULT or INFO level. Previously, these log level transitions weren't logged.
Data validation and filter settings	Data validation now honors the row filter settings when validating data between the source and target.
String handling option for Amazon S3 source objects	A new <code>rfc4180</code> extra connection attribute is available to control how source strings with double quotation marks ("") are handled.
Character substitution supported for several relational databases as sources or targets and Amazon Redshift as a target	You can now specify task settings that substitute characters in the target database based on the Unicode code points of selected source database characters and their replacements on the target. Also, you can now specify that a single character substitution take place on the target for all characters that are invalid in a specified character set. This ensures that your target database stores only valid characters in the single chosen character set.

The issues resolved are as follows.

Date reported	Description
December 17, 2018	Fixed an issue where enabling Secure Sockets Layer (SSL) caused a task failure with an Amazon S3 source and PostgreSQL target.
December 11, 2018	Fixed an issue where percentage complete for MySQL was reported incorrectly.
November 11, 2018	Fixed an issue with an Amazon S3 source where a double quotation mark ("") in a string was causing a task failure.
November 6, 2018	Fixed an issue where a custom .csv delimiter value in a header column for an Amazon S3 target was not applied properly.
August 16, 2018	Fixed an issue that caused a duplicate constraint error when checkpoint recovery was enabled.

## 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 Amazon Kinesis Data Streams as a target	You can now use Kinesis Data Streams as a target to process large amounts of data in real time.
Support for Amazon Elasticsearch Service as a target	You can now use Amazon ES as a target to deploy and operate a complex query and analytics engine for all your data using open-source Elasticsearch. Among other features, this includes support for both a multithreaded full load and change data capture (CDC) to keep the query engine current.
Support for an updated version of IBM Db2 for Windows (Db2 LUW) as a source	You can now use Db2 LUW Version 11.1 as a source, with all fix packs supported.
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 <a href="#">Using a Microsoft SQL Server database as a source for AWS DMS (p. 98)</a> .
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 <a href="#">Table-settings rules and operations (p. 333)</a> .
Control large object (LOB) settings per table	You can now control LOB settings per table. For more information, see <a href="#">Target metadata task settings (p. 282)</a> . 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 <a href="#">Using table mapping to specify task settings (p. 309)</a> .
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 create 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) 2.4.5 release notes

The following tables show the features and bug fixes for version 2.4.5 of AWS Database Migration Service (AWS DMS).

New feature or enhancement	Description
KMS key encryption for Amazon Redshift and Amazon S3 target data	You can now use KMS keys for server-side encryption of data for Amazon Redshift and S3 targets.
Transitions to DEFAULT or INFO logging level	AWS DMS now logs when log levels are changed back to DEFAULT or INFO level. Previously, these log level transitions were not logged.
Data validation and filter settings	Data validation now honors the row filter settings when validating data between the source and target.
String handling option for Amazon S3 source objects	A new <code>rfc4180</code> extra connection attribute is available to control how source strings with double-quotes ("") are handled.

The issues resolved are as follows.

Date reported	Description
January 19, 2019	Fixed an issue for Oracle endpoints where <code>securityDbEncryptionName</code> was not set properly when specifying the <code>password</code> and <code>securityDbEncryptionName</code> without an <code>asm_password</code> .
December 11, 2018	Fixed an issue where Percentage Complete for MySQL was reported incorrectly.
November 11, 2018	Fixed an issue with an Amazon S3 source where a double-quote ("") in a string was causing a task failure.
November 6, 2018	Fixed an issue where a custom .csv delimiter value in a header column for an Amazon S3 target was not applied properly.
August 16, 2018	Fixed an issue which was caused a duplicate constraint error when checkpoint recovery was enabled.

## 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.
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 VARCHAR 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 the table cached in the replication instance. By default, this attribute is set to false.</p>
Performance improvements for data validation	<p>Improvements include the following:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• Improvements to handle collation differences automatically based on task settings.</li> </ul>

New feature or enhancement	Description
	<ul style="list-style-type: none"> <li>Improvements to identify false positives during validation, which also reduces false positives during cached changes phase.</li> <li>General logging improvements to data validation.</li> <li>Improvements to queries used by data validation when tables have composite primary keys.</li> </ul>

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 <a href="#">additional Amazon Redshift copy parameters</a> using extra connection attributes. For more information, see <a href="#">Extra connection attributes when using Amazon Redshift as a target for AWS DMS (p. 180)</a>.</p> <ul style="list-style-type: none"> <li>TRUNCATECOLUMNS</li> <li>REMOVEQUOTES</li> <li>TRIMBLANKS</li> </ul>
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 <a href="#">ApplyErrorFailOnTruncationDdl</a> setting in the section <a href="#">Error handling task settings (p. 296)</a> .
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.

New feature or enhancement	Description
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 <a href="#">Replication task statistics (p. 382)</a> .

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	Fixed various data validation issues: <ul style="list-style-type: none"> <li>• Fixed wide string handling issues in Oracle source and target validation.</li> <li>• Handle validation when a column is removed for a table in table mappings.</li> <li>• Improved validation performance for sources with a high rate of change.</li> </ul>

## 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 <a href="#">additional copy parameters</a> by using these extra connection attributes:
	<ul style="list-style-type: none"> <li>• ACCEPTANYDATE</li> <li>• DATEFORMAT</li> <li>• TIMEFORMAT</li> <li>• EMPTYASNULL</li> </ul>

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 binary log (binlog) from a MySQL source.
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.

Date reported	Description
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 ROWDEPENDENCIES 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 <a href="#">here</a> .
Support for cross-account Amazon S3 access	Adds functionality to support canned ACLs (predefined grants) to support cross-account access with S3 endpoints. Find more details about canned ACLs in <a href="#">Canned ACL in the Amazon Simple Storage Service Developer Guide</a> .  Usage: Set an extra connect attribute in S3 endpoint, that is CannedAclForObjects=value. Possible values are as follows: <ul style="list-style-type: none"><li>• NONE</li><li>• PRIVATE</li><li>• PUBLIC_READ</li><li>• PUBLIC_READ_WRITE</li><li>• AUTHENTICATED_READ</li><li>• AWS_EXEC_READ</li><li>• BUCKET_OWNER_READ</li><li>• BUCKET_OWNER_FULL_CONTROL</li></ul>

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.

Date reported	Description
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	<a href="#">Using Amazon S3 as a source for AWS DMS (p. 145)</a>
SQL Azure as a source	<a href="#">Using Microsoft Azure SQL database as a source for AWS DMS (p. 111)</a>
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.
Oracle source: CDC support for Oracle Active Data Guard standby as a source	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.  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.

New feature or enhancement	Description
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"> <li>• <code>HeartbeatEnable</code> is set to true (default is false).</li> <li>• <code>HeartbeatSchema</code> is the schema for heartbeat artifacts (default is public).</li> <li>• <code>HeartbeatFrequency</code> is the heartbeat frequency in minutes (default is 5 and minimum value is 1)</li> </ul>
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 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.

Date reported	Description
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. For more details on AWS DMS version releases, see [AWS DMS release notes \(p. 490\)](#),

update-history-change	update-history-description	update-history-date
<a href="#">Support for Amazon Neptune as a target</a>	Added support for Amazon Neptune as a target for data migration.	June 1, 2020
<a href="#">Support for Apache Kafka as a target</a>	Added support for Apache Kafka as a target for data migration.	March 20, 2020
<a href="#">Updated security content</a>	Updated and standardized security content as a response to customer requests.	December 20, 2019
<a href="#">Migrating with AWS Snowball Edge</a>	Added support for using AWS Snowball Edge to migrate large databases.	January 24, 2019
<a href="#">Support for Amazon DocumentDB (with MongoDB compatibility) as a target</a>	Added support for Amazon DocumentDB (with MongoDB compatibility) as a target for data migration.	January 9, 2019
<a href="#">Support for Amazon Elasticsearch Service and Amazon Kinesis Data Streams as targets</a>	Added support for Amazon ES and Kinesis Data Streams as targets for data migration.	November 15, 2018
<a href="#">CDC native start support</a>	Added support for native start points when using change data capture (CDC).	June 28, 2018
<a href="#">R4 support</a>	Added support for R4 replication instance classes.	May 10, 2018
<a href="#">Db2 LUW support</a>	Added support for IBM Db2 LUW as a source for data migration.	April 26, 2018
<a href="#">Task log support</a>	Added support for seeing task log usage and purging task logs.	February 8, 2018
<a href="#">SQL Server as target support</a>	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 before January 2018.

<b>Change</b>	<b>Description</b>	<b>Date changed</b>
New feature	Added support for using AWS DMS with AWS Snowball to migrate large databases. For more information, see <a href="#">Migrating large data stores using AWS Database Migration Service and AWS Snowball Edge (p. 467)</a> .	November 17, 2017
New feature	Added support for task assessment report and data validation. For more information about the task assessment report, see <a href="#">Enabling and working with premigration assessments for a task (p. 353)</a> . For more information about data validation, see <a href="#">Data validation task settings (p. 289)</a> .	November 17, 2017
New feature	Added support for AWS CloudFormation templates. For more information, see <a href="#">AWS DMS support for AWS CloudFormation (p. 13)</a> .	July 11, 2017
New feature	Added support for using Amazon Dynamo as a target. For more information, see <a href="#">Using an Amazon DynamoDB database as a target for AWS Database Migration Service (p. 207)</a> .	April 10, 2017
New feature	Added support for using MongoDB as a source. For more information, see <a href="#">Using MongoDB as a source for AWS DMS (p. 141)</a> .	April 10, 2017
New feature	Added support for using Amazon S3 as a target. For more information, see <a href="#">Using Amazon S3 as a target for AWS Database Migration Service (p. 187)</a> .	March 27, 2017
New feature	Adds support for reloading database tables during a migration task. For more information, see <a href="#">Reloading tables during a task (p. 307)</a> .	March 7, 2017
New feature	Added support for events and event subscriptions. For more information, see <a href="#">Working with events and notifications in AWS Database Migration Service (p. 375)</a> .	January 26, 2017
New feature	Added support for SSL endpoints for Oracle. For more information, see <a href="#">SSL support for an Oracle endpoint (p. 80)</a> .	December 5, 2016
New feature	Added support for using change data capture (CDC) with an Amazon RDS PostgreSQL DB instance. For more information, see <a href="#">Setting up an Amazon RDS PostgreSQL DB instance as a source (p. 119)</a> .	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 AWS Regions, see <a href="#">What is AWS Database Migration Service? (p. 1)</a> .	August 3, 2016
New feature	Added support for ongoing replication. For more information, see <a href="#">Ongoing replication (p. 35)</a> .	July 13, 2016
New feature	Added support for secured connections using SSL. For more information, see <a href="#">Using SSL with AWS Database Migration Service (p. 435)</a> .	July 13, 2016

Change	Description	Date changed
New feature	Added support for SAP Adaptive Server Enterprise (ASE) as a source or target endpoint. For more information, see <a href="#">Using an SAP ASE database as a source for AWS DMS (p. 136)</a> and <a href="#">Using a SAP ASE database as a target for AWS Database Migration Service (p. 185)</a> .	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 <a href="#">Using source filters (p. 349)</a> .	May 2, 2016
New feature	Added support for Amazon Redshift as a target endpoint. For more information, see <a href="#">Using an Amazon Redshift database as a target for AWS Database Migration Service (p. 173)</a> .	May 2, 2016
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*.