User Guide

AWS Database Migration Service

AWS Database Migration Service: User Guide

Table of Contents

What is AWS Database Migration Service?

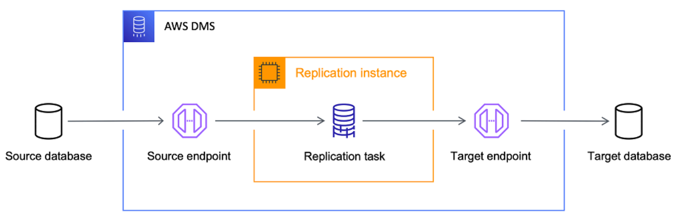
AWS Database Migration Service (AWS DMS) is a cloud service that makes it possible to migrate relational databases, data warehouses, NoSQL databases, and other types of data stores. You can use AWS DMS to migrate your data into the AWS Cloud or between combinations of cloud and on-premises setups.

With AWS DMS, you can discover your source data stores, convert your source schemas, and migrate your data.

- To discover your source data infrastructure, you can use DMS Fleet Advisor. This service collects data from your on-premises database and analytic servers, and builds an inventory of servers, databases, and schemas that you can migrate to the AWS Cloud.
- To migrate to a different database engine, you can use DMS Schema Conversion. This service automatically assesses and converts your source schemas to a new target engine. Alternatively, you can download the AWS Schema Conversion Tool (AWS SCT) to your local PC to convert your source schemas.
- After you convert your source schemas and apply the converted code to your target database, you can use AWS DMS to migrate your data. You can perform one-time migrations or replicate ongoing changes to keep sources and targets in sync. 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.

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

The following diagram illustrates the AWS DMS replication process.



References

- **AWS Regions that support AWS DMS** – For information about what AWS Regions support AWS DMS, see Working with an AWS DMS replication instance.
- **Cost of database migration** – For information on the cost of database migration, see the AWS Database Migration Service pricing page.
- **AWS DMS features and benefits** – For information about AWS DMS features and benefits, see AWS Database Migration Service Features.
- **Available database options** – To learn more about the variety of database options available on Amazon Web Services, see Choosing the right database for your organization.

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.
- 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 Fleet Advisor automatically inventories your data infrastructure. It creates reports that help you identify migration candidates and plan your migration.
- AWS DMS Schema Conversion automatically assesses the complexity of your migration for your source data provider. It also converts database schemas and code objects to a format compatible with the target database and then applies the converted code.

- 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 Relational Database Service (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 source endpoints. For more information, see Sources for data migration.
- AWS DMS provides a broad coverage of available target engines. For more information, see Targets for data migration.
- 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.

AWS DMS Terminology and concepts

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

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:

- Discover databases in your network environment that are good candidates for migration.
- Automatically convert your source database schemas and most of the database code objects to a format compatible with the target database.

- 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:

- Migration of existing data (Full load)
- The application of cached changes
- Ongoing replication (Change Data Capture)

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. Once the table is loaded and the cached changes applied, AWS DMS begins to collect changes as transactions for the ongoing replication phase. If a transaction has tables not yet fully loaded, the changes are stored locally on the replication instance. After AWS DMS applies all cached changes to all tables, 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 a data migration. You can use AWS DMS to take a minimalist approach and create only those objects required to efficiently migrate the data. Using this approach, AWS DMS creates tables, primary keys, and in some cases unique indexes, but it won't create any other objects that are not required to efficiently migrate the data from the source.

Alternatively, you can use DMS Schema Conversion within AWS DMS to automatically convert your source database schemas and most of the database code objects to a format compatible with the target database. This conversion includes tables, views, stored procedures, functions, data types, synonyms, and so on. Any objects that DMS Schema Conversion can't convert automatically are clearly marked. To complete the migration, you can convert these objects manually.

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 five components: discovery of databases to migrate, automatic schema conversion, 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.

Database discovery

DMS Fleet Advisor collects data from multiple database environments to provide insight into your data infrastructure. DMS Fleet Advisor collects data from your on-premises database and analytic servers from one or more central locations without the need to install it on every computer. Currently, DMS Fleet Advisor supports Microsoft SQL Server, MySQL, Oracle, and PostgreSQL database servers.

Based on data discovered from your network, DMS Fleet Advisor builds an inventory that you can review to determine which database servers and objects to monitor. As details about these servers, databases, and schemas are collected, you can analyze the feasibility of your intended database migrations.

Schema and code migration

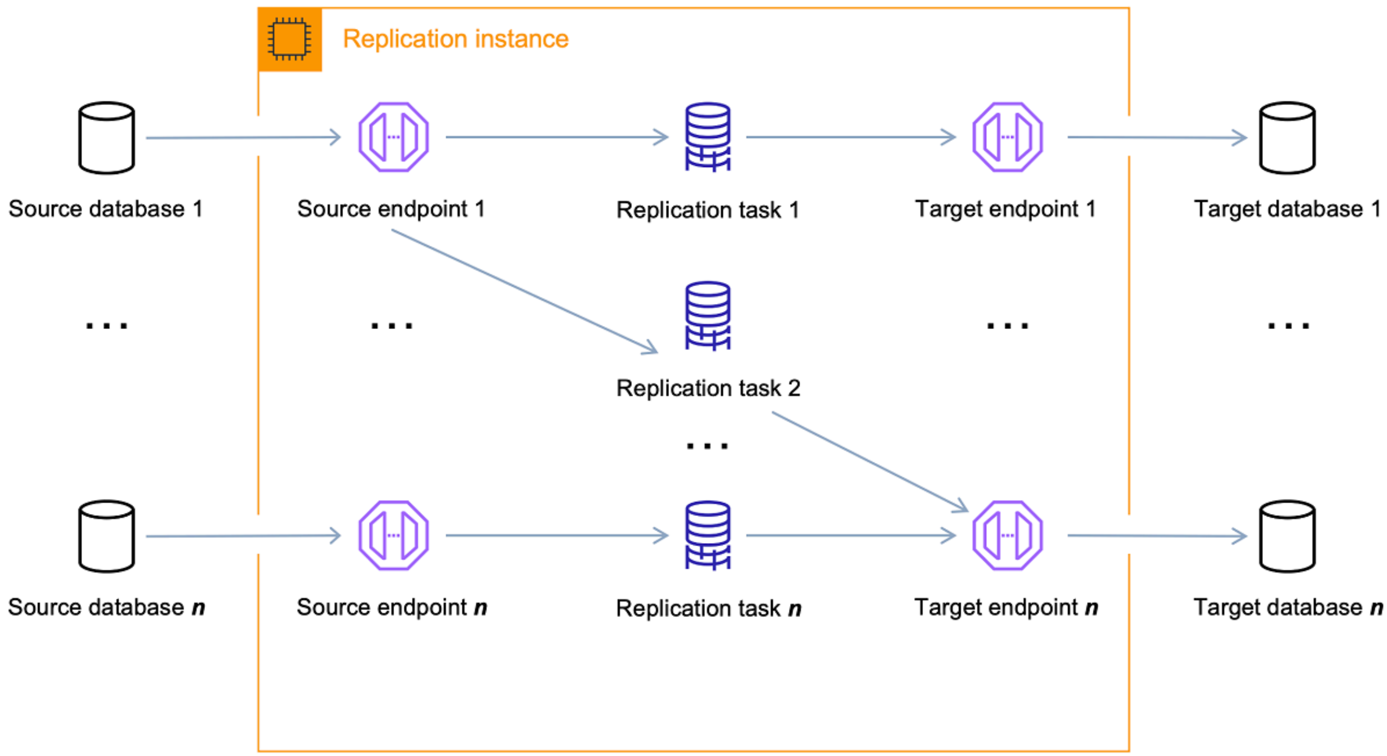
DMS Schema Conversion in AWS DMS makes database migrations between different types of databases more predictable. You can use DMS Schema Conversion to evaluate the complexity of your migration for your source data provider, and then use it to convert database schemas and code objects. You can then apply the converted code to your target database.

At a high level, DMS Schema Conversion operates with the following three components: instance profiles, data providers, and migration projects. An *instance profile* specifies network and security settings. A *data provider* stores database connection credentials. A *migration project* contains data providers, an instance profile, and migration rules. AWS DMS uses data providers and an instance profile to design a process that converts database schemas and code objects.

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](#).

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. As log files are also written to the disk, increasing the level of severity for logging will also lead to a higher storage consumption.

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.

Rather than creating and managing a replication instance, you can let AWS DMS provision your replication automatically using AWS DMS Serverless. For more information, see Working with AWS DMS Serverless.

Endpoint

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 AWS 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 endpoint settings. *Endpoint settings* can control various behavior such as logging detail, file size, and other parameters. Each data store engine type has different endpoint settings available. You can find the specific endpoint settings 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 and Targets for AWS DMS.

For more detailed information about AWS DMS endpoints, see Working with AWS DMS endpoints.

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:

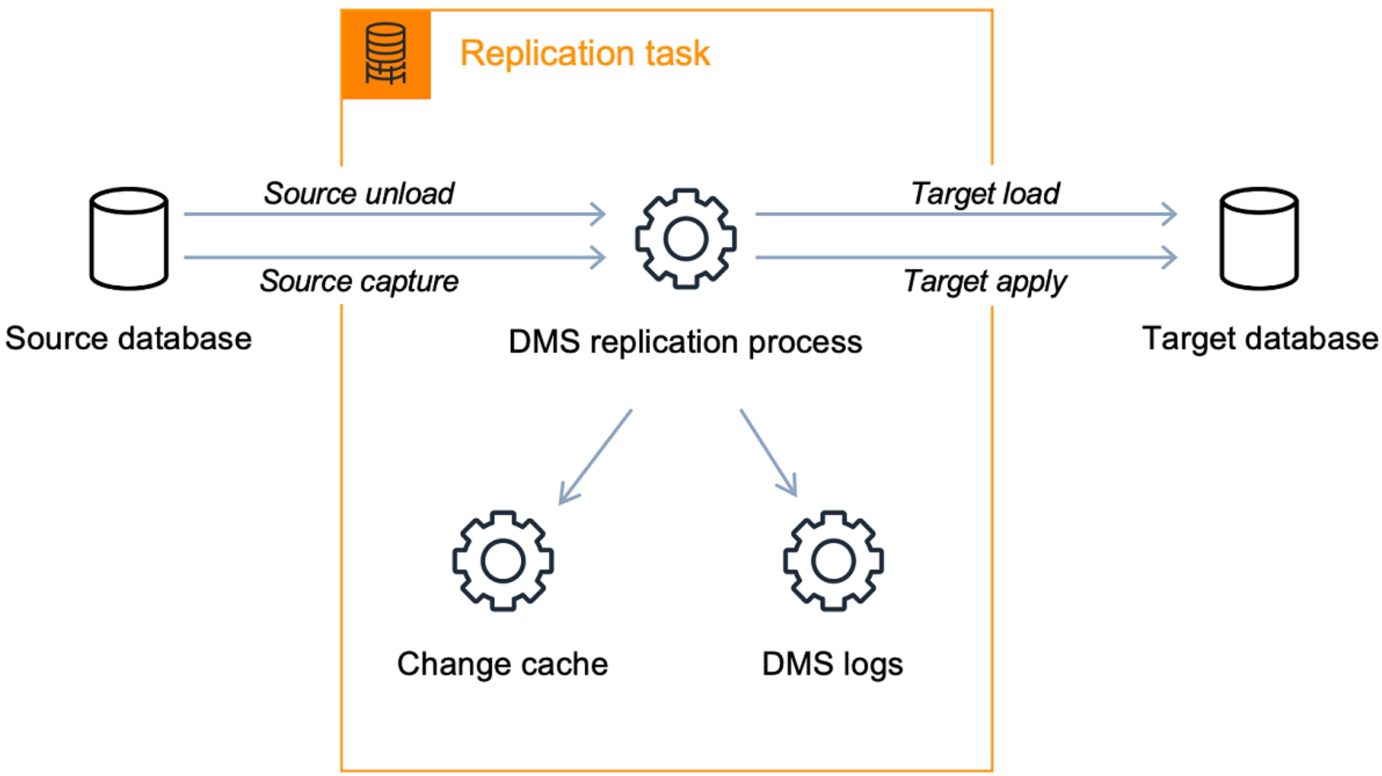
- Replication instance – the instance to host and run the task
- Source endpoint
- Target endpoint
- Migration type options, as listed following. For a full explanation of the migration type options, see Creating a task.
  - Full load (Migrate existing data) – If you can afford an outage long enough 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.
  - Do nothing – AWS DMS assumes that the target tables are precreated 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.
  - 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.
- Data transformations, as listed following. For more information on data transformations, see Specifying table selection and transformations rules using JSON.
  - 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](#).

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 preceding. 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.

Sources for AWS DMS

You can use different source data stores in different AWS DMS features. The following sections contain the lists of supported source data stores for each AWS DMS feature.

Topics

Source endpoints for data migration

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

- Microsoft SQL Server versions 2008 (supported in DMS v3.5.4), 2008R2(supported in DMS v3.5.4), 2012, 2014, 2016, 2017, 2019, and 2022.
  - The Enterprise, Standard, Workgroup, Developer, and Web editions support full-load replication.
  - The Enterprise, Standard (version 2016 and higher), and Developer editions support CDC (ongoing) replication in addition to full-load.
  - The Express edition isn't supported.
- MySQL versions 5.5, 5.6, 5.7, 8.0, and 8.4

> **Note**
>
> **MySQL and DMS Compatibility**
>
> | MySQL Version | DMS Version | Compressed transaction payload supporting |
> | --- | --- | --- |
> | 8 | 3.4.0 and above | No |
> | 8.0 (Google Cloud) | 3.4.6 and above | No |
> | 8.4 | 3.5.4 and above | No |

- MariaDB (supported as a MySQL-compatible data source) versions 10.0 (only versions 10.0.24 and higher), 10.2, 10.3, 10.4, 10.5, 10.6, and 11.4.3 to 11.4.5.

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

- PostgreSQL version 9.4 and higher (for versions 9.x), 10.x, 11.x, 12.x, 13.x 14.x, 15.x, 16.x, 17.x, and 18.x.

> **Note**
>
> - AWS DMS only supports PostgreSQL version 9.x, 10.x, 11.x, 12.x, 13.x, 14.x, 15.x, and 16.x in versions 3.5.3 and higher.

  - AWS DMS only supports PostgreSQL version 17.x and 18.x in versions 3.5.4 (see limitations) and higher.

- MongoDB versions 3.x, 4.0, 4.2, 4.4, 5.0, 6.0, 7.0 and 8.0.
- SAP Adaptive Server Enterprise (ASE) versions 12.5, 15, 15.5, 15.7, 16, and higher
- IBM Db2 for Linux, UNIX, and Windows (Db2 LUW) versions:
  - Version 9.7, all fix packs
  - Version 10.1, all fix packs
  - Version 10.5, all fix packs except for Fix Pack 5
  - Version 11.1, all fix packs
  - Version 11.5, Mods (0-8) with only Fix Pack Zero
- IBM Db2 for z/OS version 12

Third-party managed database services:

- Microsoft Azure SQL Database
- Microsoft Azure PostgreSQL Flexible Server versions 11.2, 12.15, 13.11, 14.8, and 15.3.
- Microsoft Azure MySQL Flexible Server versions 5.7 and 8.
- Google Cloud for MySQL versions 5.6, 5.7, and 8.0.
- Google Cloud for PostgreSQL versions 9.6, 10, 11, 12, 13, 14, and 15.
- OCI MySQL Heatwave version 8.0.34.

Amazon RDS instance databases, and Amazon Simple Storage Service (Amazon S3)

- Oracle versions 11g (versions 11.2.0.4 and higher) and up to 12.2, 18c, and 19c for the Enterprise, Standard, Standard One, and Standard Two editions.
- Microsoft SQL Server versions 2016, 2017, 2019, and 2022 for the Enterprise, Standard, Workgroup, and Developer editions. For more information, see Amazon RDS for Microsoft SQL Server in the *Amazon RDS user guide*.

> **Note**
>
> AWS DMS does not support SQL Server Express. The Web edition is only supported for full-load only replication.

- MySQL versions 5.5, 5.6, 5.7, 8.0, and 8.4.

> **Note**
>
> **MySQL and DMS Compatibility**
>
> | MySQL Version | DMS Version | Compressed transaction payload supporting |
> |---|---|---|
> | 8 | 3.4.0 and above | No |
> | 8.0 (Google Cloud) | 3.4.6 and above | No |
> | 8.4 | 3.5.4 and above | No |

- MariaDB (supported as a MySQL-compatible data source) versions 10.0.24 to 10.0.28, 10.2, 10.3, 10.4, 10.5, 10.6, and 11.4.3 to 11.4.5.

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

- PostgreSQL version 10.x, 11.x, 12.x, 13.x, 14.x, 15.x, 16.x, 17.x, and 18.x.

> **Note**
>
> - AWS DMS only supports PostgreSQL version 10.x, 11.x, 12.x, 13.x, 14.x, 15.x, and 16.x in versions 3.5.3 and higher.
> - AWS DMS only supports PostgreSQL version 17.x and 18.x in versions 3.5.4 (see limitations) and higher.

- 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
- Amazon DocumentDB (with MongoDB compatibility) versions 3.6, 4.0, and 5.0.
- Amazon RDS for IBM Db2 LUW.

For information about working with a specific source, see Working with AWS DMS endpoints.

For information about supported target endpoints, see Target endpoints for data migration.

Source databases for DMS Fleet Advisor

DMS Fleet Advisor supports the following source databases.

- Microsoft SQL Server version 2012 and up to 2019
- MySQL version 5.6 and up to 8
- Oracle version 11g Release 2 and up to 12c, 19c, and 21c
- PostgreSQL version 9.6 and up to 13

For information about working with a specific source, see Creating database users for AWS DMS Fleet Advisor.

For the list of databases that DMS Fleet Advisor uses to generate target recommendations, see Targets for DMS Fleet Advisor.

Source data providers for DMS Schema Conversion

DMS Schema Conversion supports the following data providers as sources for your migration projects.

- Microsoft SQL Server version 2008 R2, 2012, 2014, 2016, 2017, 2019, and 2022
- Oracle version 10.2 and higher, 11g and up to 12.2, 18c, and 19c, and Oracle Data Warehouse
- PostgreSQL version 9.2 and higher
- MySQL version 5.5, 5.6, 5.7, and 8.0.
- IBM Db2 for z/OS version 12
- SAP ASE (Sybase ASE) version 16

Note

DMS Schema Conversion supports all the Amazon RDS version sources listed in this topic.

Your source data provider can be a self-managed engine running on-premises or on an Amazon Elastic Compute Cloud (Amazon EC2) instance.

For information about working with a specific source, see Creating source data providers in DMS Schema Conversion.

For information about supported target databases, see Target data providers for DMS Schema Conversion.

The AWS Schema Conversion Tool (AWS SCT) supports more source and target databases than DMS Schema Conversion. For information about databases that AWS SCT supports, see What is the AWS Schema Conversion Tool.

Source data providers for DMS homogeneous data migrations

You can use the following data providers as sources for homogeneous data migrations to on-premises and EC2 instance databases.

- MySQL version 5.7 and 8.0
- MariaDB version 10.2x
- PostgreSQL version 10.4 to 16.x.
- MongoDB version 4.x, 5.x, 6.0, 7.0, 8.0
- Amazon DocumentDB version 3.6, 4.0, 5.0

You can use the following data providers as sources for homogeneous data migrations to Amazon Relational Database Service instance databases.

- Aurora MySQL version 5.7 and 8.0
- RDS for MySQL version 5.7 and 8.0
- RDS for MariaDB version 10.2.x
- Aurora PostgreSQL version 11.x to 17.x

- RDS for PostgreSQL version 11.x to 17.x

Your source data provider can be a self-managed engine running on-premises or on an Amazon EC2 instance. Also, you can use an Amazon RDS DB instance as a source data provider.

For information about working with a specific source, see Creating source data providers for homogeneous data migrations in AWS DMS.

For information about supported target databases, see Target data providers for DMS homogeneous data migrations.

Targets for AWS DMS

You can use different target data stores in different AWS DMS features. The following sections contain the lists of supported target data stores for each AWS DMS feature.

Topics

Target endpoints for data migration

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
- Microsoft SQL Server versions 2005, 2008, 2008R2, 2012, 2014, 2016, 2017, 2019, and 2022 for the Enterprise, Standard, Workgroup, and Developer editions

> **Note**
>
> AWS DMS doesn't support SQL Server Web and Express editions.

- MySQL versions 5.5, 5.6, 5.7, 8.0, and 8.4
- MariaDB (supported as a MySQL-compatible data target) versions 10.0.24 to 10.0.28, 10.2, 10.3, 10.4, 10.5, 10.6, and 11.4.3 to 11.4.5.

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

- PostgreSQL version 9.4 and higher (for versions 9.x), 10.x, 11.x, 12.x, 13.x, 14.x, 15.x, 16.x, 17.x, and 18.x.

> **Note**
>
> - AWS DMS only supports PostgreSQL version 9.x, 10.x, 11.x, 12.x, 13.x, 14.x, 15.x, and 16.x in versions 3.5.3 and higher.
> - AWS DMS only supports PostgreSQL version 17.x and 18.x in versions 3.5.4 and higher.

- SAP Adaptive Server Enterprise (ASE) versions 15, 15.5, 15.7, 16, and higher
- Redis OSS versions 6.x

Amazon RDS instance databases, Amazon Redshift, Amazon Redshift Serverless, Amazon DynamoDB, Amazon S3, Amazon OpenSearch Service, Amazon ElastiCache (Redis OSS), Amazon Kinesis Data Streams, Amazon DocumentDB, Amazon Neptune, and Apache Kafka

- Oracle versions 11g (versions 11.2.0.3.v1 and higher), 12c, 18c, and 19c for the Enterprise, Standard, Standard One, and Standard Two editions
- Microsoft SQL Server versions 2012, 2014, 2016, 2017, 2019, and 2022 for the Enterprise, Standard, Workgroup, and Developer editions

> **Note**
>
> AWS DMS doesn't support SQL Server Web and Express editions.

- MySQL versions 5.5, 5.6, 5.7, 8.0, and 8.4.
- MariaDB (supported as a MySQL-compatible data target) versions 10.0.24 to 10.0.28, 10.2, 10.3, 10.4, 10.5, 10.6, and 11.4.3 to 11.4.5.

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

- PostgreSQL version 10.x, 11.x, 12.x, 13.x, 14.x, 15.x, 16.x, 17.x, and 18.x.

> **Note**
>
> - AWS DMS only supports PostgreSQL version 10.x, 11.x, 12.x, 13.x, 14.x, 15.x, and 16.x in versions 3.5.3 and higher.
> - AWS DMS only supports PostgreSQL version 17.x and 18.x in versions 3.5.4 and higher.

- IBM Db2 LUW versions 11.1 and 11.5
- Amazon Aurora MySQL-Compatible Edition
- Amazon Aurora PostgreSQL-Compatible Edition
- Amazon Aurora PostgreSQL Limitless
- Amazon Aurora Serverless v2
- Amazon Redshift
- Amazon Redshift Serverless
- Amazon S3
- Amazon DynamoDB
- Amazon OpenSearch Service
- Amazon ElastiCache (Redis OSS)
- 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
- Babelfish (version 3.2.0 and higher) for Aurora PostgreSQL (versions 15.3/14.8 and higher)

For information about working with a specific target, see Working with AWS DMS endpoints.

For information about supported source endpoints, see Source endpoints for data migration.

Target databases for DMS Fleet Advisor

DMS Fleet Advisor generates target recommendations using the latest version of the following target databases.

- Amazon Aurora MySQL
- Amazon Aurora PostgreSQL
- Amazon RDS for MySQL
- Amazon RDS for Oracle
- Amazon RDS for PostgreSQL
- Amazon RDS for SQL Server

For information about target recommendations in DMS Fleet Advisor, see Using the AWS DMS Fleet Advisor Target Recommendations feature.

For information about supported source databases, see Source databases for DMS Fleet Advisor.

Target data providers for DMS Schema Conversion

DMS Schema Conversion supports the following data providers as targets for your migration projects.

- Amazon Aurora MySQL 8.0.32
- Amazon Aurora PostgreSQL 14.x, 15.x, 16.x, 17.x
- Amazon RDS for MySQL 8.0.23
- Amazon RDS for PostgreSQL 14.x, 15.x, 16.x, 17.x, 18.x
- Amazon Redshift
- Amazon RDS for Db2 version 11.5.

For information about working with a specific target, see Creating and setting target data providers in DMS Schema Conversion.

For information about supported source databases, see Source data providers for DMS Schema Conversion.

Target data providers for DMS homogeneous data migrations

You can use the following data providers as targets for homogeneous data migrations.

- Amazon Aurora MySQL version 5.7 and 8.0
- Amazon Aurora PostgreSQL version 10.4 to 16.x
- Amazon Aurora Serverless v2
- Amazon RDS for MySQL version 5.7 and 8.0
- Amazon RDS for MariaDB version 10.2x
- Amazon RDS for PostgreSQL version 10.4 to 16.x
- Amazon DocumentDB version 4.0, 5.0 and DocumentDB Elastic cluster

For information about working with a specific target, see Creating and setting a target database to work with AWS DMS schema conversion.

For information about supported source databases, see Source data providers for DMS homogeneous data migrations.

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
Europe (Milan) Region	eu-south-1
EU (Frankfurt) Region	eu-central-1
Europe (Ireland) Region	eu-west-1
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

- *account number* is your account number with dashes omitted. To find your account number, sign 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 to use when constructing an ARN for a particular AWS DMS resource.

AWS DMS resource type	ARN format
Replication instance	`arn:aws:dms:` *region*: *account*`:rep:` *resourcename*
Endpoint	`arn:aws:dms:` *region*:*account*`:endpoint:` *resourcename*
Replication task	`arn:aws:dms:` *region*:*account*`:task:`*resourcename*
Subnet group	`arn:aws:dms:` *region*:*account*`:subgrp:`*resourcename*

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

The following table shows examples of ARNs for AWS DMS resources. Here, we assume 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	`arn:aws:dms:us-east-1:123456789012:rep:QLXQZ64MH7CXF4QCQMGRVYVXAI`
Endpoint	`arn:aws:dms:us-east-1:123456789012:endpoint:D3HMZ2IGUCGFF3NTAXUXGF6S5A`
Replication task	`arn:aws:dms:us-east-1:123456789012:task:2PVREMWNPGYJCVU2IBPTOYTIV4`

Resource type	Sample ARN
Subnet group	`arn:aws:dms:us-east-1:123456789012:subgrp:test-tag-grp`

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 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 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

Getting started with AWS Database Migration Service

In the following tutorial, you can find out how to perform a database migration with AWS Database Migration Service (AWS DMS).

To perform a database migration, take the following steps:

1. Set up your AWS account by following the steps in Set up for AWS Database Migration Service.
2. Create your sample databases and an Amazon EC2 client to populate your source database and test replication. Also, create a virtual private cloud (VPC) based on the Amazon Virtual Private Cloud (Amazon VPC) service to contain your tutorial resources. To create these resources, follow the steps in Complete prerequisites to set up AWS Database Migration Service.
3. Populate your source database using a sample database creation script.
4. Use DMS Schema Conversion or the AWS Schema Conversion Tool (AWS SCT) to convert the schema from the source database to the target database. To use DMS Schema Conversion, follow the steps in Getting started with DMS Schema Conversion. To convert the schema with AWS SCT, follow the steps in Migrate schema.
5. Create a replication instance to perform all the processes for the migration. To do this and the following tasks, take the steps in Replication.
6. Specify source and target database endpoints. For information about creating endpoints, see Creating source and target endpoints.
7. Create a task to define what tables and replication processes you want to use, and start replication. For information about creating database migration tasks, see Creating a task.
8. Verify that replication is working by running queries on the target database.

Learn more about working with AWS Database Migration Service

Later in this guide, you can learn how to use AWS DMS to migrate your data to and from the most widely used commercial and open-source databases.

We also recommend that you check the following resources as you prepare and perform a database migration project:

- AWS DMS Step-by-Step Migration Guide – This guide provides step-by-step walkthroughs that go through the process of migrating data to AWS.
- AWS DMS API Reference – This reference describes all the API operations for AWS Database Migration Service in detail.
- AWS CLI for AWS DMS – This reference provides information about using the AWS Command Line Interface (AWS CLI) with AWS DMS.

Set up for AWS Database Migration Service

Sign up for an AWS account

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 or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see [Add groups] in the *AWS IAM Identity Center User Guide*.

Complete prerequisites to set up AWS Database Migration Service

In this section, you can learn the prerequisite tasks for AWS DMS, such as setting up your source and target databases. As part of these tasks, you also set up a virtual private cloud (VPC) based on the Amazon VPC service to contain your resources. In addition, you set up an Amazon EC2 instance that you use to populate your source database and verify replication on your target database.

> **Note**
>
> Populating the source database takes up to 45 minutes.

For this tutorial, you create a MariaDB database as your source, and a PostgreSQL database as your target. This scenario uses commonly used, low-cost database engines to demonstrate replication. Using different database engines demonstrates AWS DMS features for migrating data between heterogeneous platforms.

The resources in this tutorial use the US West (Oregon) Region. If you want to use a different AWS Region, specify your chosen Region instead wherever US West (Oregon) appears.

> **Note**
>
> For the sake of simplicity, the databases that you create for this tutorial don't use encryption or other advanced security features. You must use security features to keep your production databases secure. For more information, see Security in Amazon RDS.

For prerequisite steps, see the following topics.

Topics

- [Create your source Amazon RDS database]
- [Create your target Amazon RDS database]
- [Create an Amazon EC2 client]
- [Populate your source database]

Create a VPC

In this section, you create a VPC to contain your AWS resources. Using a VPC is a best practice when using AWS resources, so that your databases, Amazon EC2 instances, security groups, and so on, are logically organized and secure.

Using a VPC for your tutorial resources also ensures that you delete all of the resources you use when you are done with the tutorial. You must delete all of the resources that a VPC contains before you can delete the VPC.

To create a VPC for use with AWS DMS

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. On the navigation pane, choose **VPC Dashboard**, and then choose **Create VPC**.
3. On the **Create VPC** page, enter the following options:

   - **Resources to create**: **VPC and more**
   - **Name tag auto generation**: Choose **Auto-generate**, and enter **`DMSVPC`**.
   - **IPv4 block**: **`10.0.1.0/24`**
   - **IPv6 CIDR block**: **No IPv6 CIDR block**
   - **Tenancy**: **Default**
   - **Number of availability zones**: 2
   - **Number of public subnets**: 2
   - **Number of private subnets**: 2
   - **NAT gateways ($)**: **None**
   - **VPC endpoints**: **None**

   Choose **Create VPC**.

4. On the navigation pane, choose **Your VPCs**. Note the VPC ID for **DMSVPC**.
5. On the navigation pane, choose **Security Groups**.
6. Choose the group named **default** that has a **VPC ID** that matches the ID that you noted for **DMSVPC**.
7. Choose the **Inbound rules** tab, and choose **Edit inbound rules**.
8. Choose **Add rule**. Add a rule of type **MySQL/Aurora** and choose **Anywhere-IPv4** for **Source**.
9. Choose **Add rule** again. Add a rule of type **PostgreSQL** and choose **Anywhere-IPv4** for **Source**.
10. Choose **Save rules**.

Create Amazon RDS parameter groups

To specify settings for your source and target databases for AWS DMS, use Amazon RDS parameter groups. To allow initial and ongoing replication between your databases, make sure to configure the following:

- Your source database's binary log, so that AWS DMS can determine what incremental updates it needs to replicate.
- Your target database's replication role, so that AWS DMS ignores foreign key constraints during the initial data transfer. With this setting, AWS DMS can migrate data out of order.

To create parameter groups for use with AWS DMS

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. On the navigation pane, choose **Parameter groups**.
3. On the **Parameter groups** page, choose **Create parameter group**.
4. On the **Create parameter group** page, enter the following settings:

   - **Parameter group family**: **mariadb10.6**
   - **Group name**: **`dms-mariadb-parameters`**
   - **Description**: **`Group for specifying binary log settings for replication`**

   Choose **Create**.
5. On the **Parameter groups** page, choose **dms-mariadb-parameters**, and on the **dms-mariadb-parameters** page, choose **Edit**.

6. Set the following parameters to the following values:

   - **binlog_checksum**: **NONE**
   - **binlog_format**: **ROW**

   Choose **Save changes**.
7. On the **Parameter groups** page, choose **Create parameter group** again.
8. On the **Create parameter group** page, enter the following settings:

   - **Parameter group family**: **postgres16**
   - **Group name**: `dms-postgresql-parameters`
   - **Description**: `Group for specifying role setting for replication`

   Choose **Create**.
9. On the **Parameter groups** page, choose **dms-postgresql-parameters**.
10. On the **dms-postgresql-parameters** page, choose **Edit**, and set **session_replication_role parameter** to **replica**. Note that the **session_replication_role** parameter is not on the first page of parameters. Use the pagination controls or the search field to find the parameter.
11. Choose **Save changes**.

Create your source Amazon RDS database

Use the following procedure to create your source Amazon RDS database.

To create your source Amazon RDS for MariaDB database

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. On the **Dashboard** page, choose **Create Database** in the **Database** section. Don't choose **Create Database** in the **Try the new Amazon RDS Multi-AZ deployment option for MySQL and PostgreSQL** section at the top of the page.
3. On the **Create database** page, set the following options:

   - **Choose a database creation method**: Choose **Standard Create**.
   - **Engine options**: For **Engine type**, choose **MariaDB**. For **Version**, leave **MariaDB 10.6.14** selected.

- **Templates**: Choose **Dev/Test**.
- **Settings**:
  - **DB instance identifier**: Enter **`dms-mariadb`**.
  - In the **Credentials settings** section, enter the following:
    - **Master username**: Leave as **`admin`**.
    - Leave **Manage master credentials in AWS Secrets Manager** unchecked.
    - **Auto generate a password**: Leave unselected.
    - **Master password**: Enter **`changeit`**.
    - **Confirm password**: Enter **`changeit`** again.
- **Instance configuration**:
  - **DB instance class**: Leave **Standard classes** chosen.
  - For **DB instance class**, choose **db.m5.large**.
- **Storage**:
  - Clear the **Enable storage autoscaling** box.
  - Leave the rest of the settings as they are.
- **Availability and Durability**: Leave **Do not create a standby instance** selected.
- **Connectivity**:
  - **Compute resource** Leave **Don't connect to an EC2 compute resource**
  - **Network type**: Leave **IPv4** selected.
  - **Virtual private cloud**: **DMSVPC-vpc**
  - **Public access**: **Yes**. You must enable public access to use the AWS Schema Conversion Tool.
  - **Availability zone**: **us-west-2a**
  - Leave the rest of the settings as they are.
- **Database authentication**: Leave **Password authentication** selected.
- Under **Monitoring**, clear the **Turn on Performance Insights** box. Expand the **Additional configuration** section, and clear the **Enable Enhanced monitoring** box.
- Expand **Additional configuration**:
  - Under **Database options**, enter **`dms_sample`** for **Initial database name**.
  - Under **DB parameter group**, choose **dms-mariadb-parameters**.
  - For **Option group**, leave **default:mariadb-10-6** selected.

  - Under **Backup**, do the following:
    - Leave **Enable automatic backups** selected. Your source database must have automatic backups enabled to support ongoing replication.
    - For **Backup retention period**, choose **1 day**.
    - For **Backup window**, leave **No preference** selected.
    - Clear the **Copy tags to snapshots** box.
    - Leave the **Enable replication in another AWS region** unchecked.
  - Under **Encryption**, clear the **Enable encryption** box.
  - Leave the **Log exports** section as it is.
  - Under **Maintenance**, clear the **Enable auto minor version upgrade** box, and leave the **Maintenance window** setting as **No preference**.
  - Leave **Enable deletion protection** unchecked.

4. Choose **Create database**.

Create your target Amazon RDS database

Repeat the previous procedure to create your target Amazon RDS database, with the following changes.

To create your target RDS for PostgreSQL database

1. Repeat steps 1 and 2 from the previous procedure.
2. On the **Create database** page, set the same options, except for these:

   a. For **Engine options**, choose **PostgreSQL**.

   b. For **Version**, choose an available **PostgreSQL 16** version

   c. For **DB instance identifier**, enter **`dms-postgresql`**.

   d. For **Master username**, leave **`postgres`** selected.

   e. For **DB parameter group**, choose **dms-postgresql-parameters**.

   f. Clear **Enable automatic backups**.
3. Choose **Create database**.

Create an Amazon EC2 client

In this section, you create an Amazon EC2 client. You use this client to populate your source database with data to replicate. You also use this client to verify replication by running queries on the target database.

Using an Amazon EC2 client to access your databases provides the following advantages over accessing your databases over the internet:

- You can restrict access to your databases to clients that are in the same VPC.
- We have confirmed that the tools you use in this tutorial work, and are easy to install, on Amazon Linux 2023, which we recommend for this tutorial.
- Data operations between components in a VPC generally perform better than those over the internet.

To create and configure an Amazon EC2 client to populate your source database

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the **Dashboard**, choose **Launch instance**.
3. On the **Launch an Instance** page, enter the following values:

   a. In the **Name and tags** section, enter `DMSClient` for **Name**.

   b. In the **Application and OS Images (Amazon Machine Image)** section, leave the settings as they are.

   c. In the **Instance Type** section, choose **t2.xlarge**.

   d. In the **Key pair (login)** section, choose **Create a new key pair**.

   e. On the **Create key pair** page, enter the following:

      - **Key pair name**: `DMSKeyPair`
      - **Key pair type**: Leave as **RSA**.
      - **Private key file format**: Choose **pem** for OpenSSH on MacOS or Linux, or **ppk** for PuTTY on Windows.

      Save the key file when prompted.

> **Note**
>
> You can also use an existing Amazon EC2 key pair rather than creating a new one.

f. In the **Network Settings** section, choose **Edit**. Choose the following settings:

- **VPC - *required***: Choose the VPC with the ID that you recorded for the **DMSVPC-vpc** VPC.
- **Subnet**: Choose the first public subnet.
- **Auto-assign public IP**: Choose **Enable**.

Leave the rest of the settings as they are, and choose **Launch instance**.

Populate your source database

In this section, you find endpoints for your source and target databases for later use and use the following tools to populate the source database:

- Git, to download the script that populates your source database.
- MariaDB client, to run this script.

Get endpoints

Find and note the endpoints of your RDS for MariaDB and RDS for PostgreSQL DB instances for later use.

To find your DB instance endpoints

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. On the navigation pane, choose **Databases**.
3. Choose the **dms-mariadb** database, and note the **Endpoint** value for the database.
4. Repeat the previous steps for the **dms-postgresql** database.

Populate your source database

Next, connect to your client instance, install the necessary software, download AWS sample database scripts from Git, and run the scripts to populate your source database.

To populate your source database

1. Connect to the client instance using the host name and public key that you saved in previous steps.

   For more information on connecting to an Amazon EC2 instance, see Accessing Instances in the *Amazon EC2 User Guide*.

   > **Note**
   >
   > If you are using PuTTY, enable TCP keepalives on the **Connection** settings page so that your connection doesn't time out from inactivity.

2. Install Git, MariaDB, and PostgreSQL. Confirm installation as needed.

   ```
   $ sudo yum install git
   $ sudo dnf install mariadb105
   $ sudo dnf install postgresql15
   ```

3. Run the following command to download the database creation scripts from GitHub.

   ```
   git clone https://github.com/aws-samples/aws-database-migration-samples.git
   ```

4. Change to the `aws-database-migration-samples/mysql/sampledb/v1/` directory.

5. Run the following command. Provide the endpoint for your source RDS instance that you noted previously, for example `dms-mariadb.cdv5fbeyiy4e.us-east-1.rds.amazonaws.com`.

   ```
   mysql -h dms-mariadb.abcdefghij01.us-east-1.rds.amazonaws.com -P 3306 -u admin -p
    dms_sample < ~/aws-database-migration-samples/mysql/sampledb/v1/install-rds.sql
   ```

6. Let the database creation script run. The script takes up to 45 minutes to create the schema and populate the data. You can safely ignore errors and warnings that the script displays.

Migrating your source schema to your target database using AWS SCT

In this section, you use the AWS Schema Conversion Tool to migrate your source schema to your target database. Alternatively, you can use DMS Schema Conversion to convert your source database schemas. For more information, see Getting started with DMS Schema Conversion.

To migrate your source schema to your target database with AWS SCT

1. Install the AWS Schema Conversion Tool. For more information, see Installing, verifying, and updating the AWS SCT in the *AWS Schema Conversion Tool User Guide*.

   When you download JDBC drivers for MySQL and PostgreSQL, note where you save the drivers, in case the tool prompts you for their locations.
2. Open the AWS Schema Conversion Tool. Choose **File**, then choose **New project**.
3. In the **New project** window, set the following values:

   - Set **Project name** to **`DMSProject`**.
   - Keep **Location** as it is to store your AWS SCT project in the default folder.

   Choose **OK**.
4. Choose **Add source** to add a source MySQL database to your project, then choose **MySQL**, and choose **Next**.
5. In the **Add source** page, set the following values:

   - **Connection name**: **`source`**
   - **Server name**: Enter the endpoint for the MySQL database that you noted previously.
   - **Server port**: **`3306`**
   - **User name**: **`admin`**
   - **Password**: **`changeit`**
6. Choose **Add target** to add a target Amazon RDS for PostgreSQL database to your project, then choose **Amazon RDS for PostgreSQL**. Choose **Next**.
7. In the **Add target** page, set the following values:

   - **Connection name**: **`target`**

- **Server name**: Enter the endpoint for the PostgreSQL database that you noted previously.
- **Server port**: **5432**
- **Database**: Enter the name of your PostgreSQL database.
- **User name**: **`postgres`**
- **Password**: **`changeit`**

8. In the left pane, choose **dms_sample** under **Schemas**. In the right pane, choose your target Amazon RDS for PostgreSQL database. Choose **Create mapping**. You can add multiple mapping rules to a single AWS SCT project. For more information about mapping rules, see Creating mapping rules.
9. Choose **Main view**.
10. In the left pane, choose **dms_sample** under **Schemas**. Open the context (right-click) menu and choose **Convert schema**. Confirm the action.

    After the tool converts the schema, the **dms_sample** schema appears in the right pane.
11. In the right pane, under **Schemas**, open the context (right-click) menu for **dms_sample** and choose **Apply to database**. Confirm the action.

Verify that the schema migration completed. Perform the following steps.

To check your schema migration

1. Connect to your Amazon EC2 client.
2. Start the PSQL client with the following command. Specify your PostgreSQL database endpoint, and provide the database password when prompted.

```
psql \
   --host=dms-postgresql.abcdefg12345.us-west-2.rds.amazonaws.com \
   --port=5432 \
   --username=postgres \
   --password \
   --dbname=dms_sample
```

3. Query one of the (empty) tables to verify that AWS SCT applied the schema correctly,

```
dms_sample=> SELECT * from dms_sample.player;
 id | sport_team_id | last_name | first_name | full_name
----+---------------+-----------+------------+-----------
```

```
(0 rows)
```

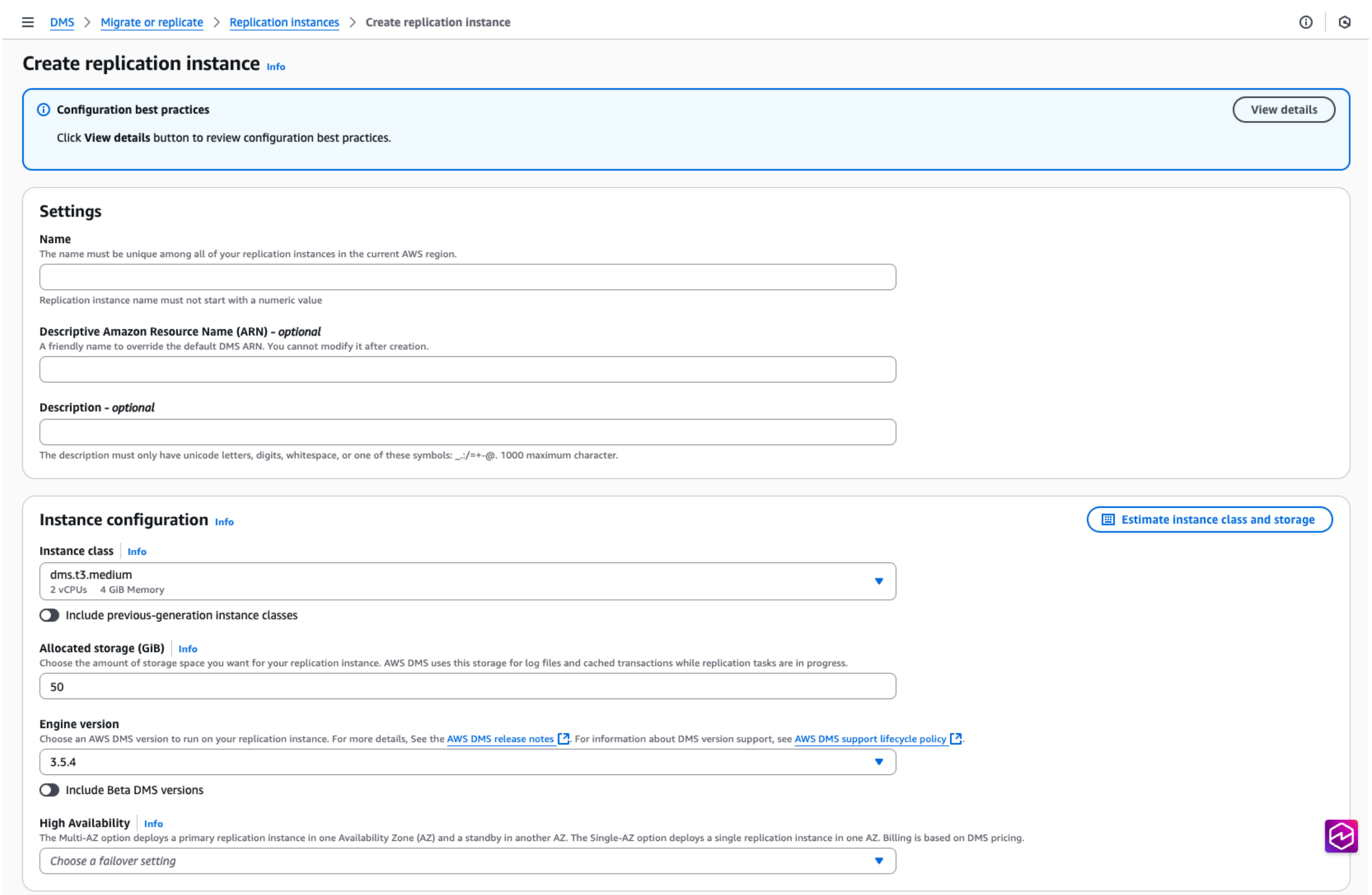
Set up replication for AWS Database Migration Service

In this topic, you set up replication between the source and target databases.

Step 1: Create a replication instance using the AWS DMS console

To start work with AWS DMS, create a replication instance.

A *replication instance* performs the actual data migration between source and target endpoints. Your instance needs enough storage and processing power to perform the tasks that migrate data from your source database to your target database. How large this replication instance should be depends on the amount of data to migrate and the tasks your instance needs to do. For more information about replication instances, see Working with an AWS DMS replication instance.



To create a replication instance 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/.
2. On the navigation pane, choose **Replication instances**, and then choose **Create replication instance**.
3. On the **Create replication instance** page, specify your replication instance configuration:

   a. For **Name**, enter **`DMS-instance`**.

   b. For **Description**, enter a short description for your replication instance (optional).

   c. For **Instance class**, leave **dms.t3.medium** chosen.

      The instance needs enough storage, networking, and processing power for your migration. For more information about how to choose an instance class, see Choosing the right AWS DMS replication instance for your migration.

   d. For **Engine version**, accept the default.

   e. For **Multi AZ**, choose **Dev or test workload (Single-AZ)**.

   f. For **Allocated storage (GiB)**, accept the default of 50 GiB.

      In AWS DMS, storage is mostly used by log files and cached transactions. For cache transactions, storage is used only when the cached transactions need to be written to disk. As a result, AWS DMS doesn't use a significant amount of storage.

   g. For **Network type** choose **IPv4**.

   h. For **VPC**, choose **DMSVPC**.

   i. For **Replication subnet group**, leave the replication subnet group currently chosen.

   j. Clear **Publicly accessible**.
4. Choose the **Advanced security and network configuration** tab to set values for network and encryption settings if you need them:

   a. For **Availability zone**, choose **us-west-2a**.

   b. For **VPC security group(s)**, choose the **Default** security group if it isn't already chosen.

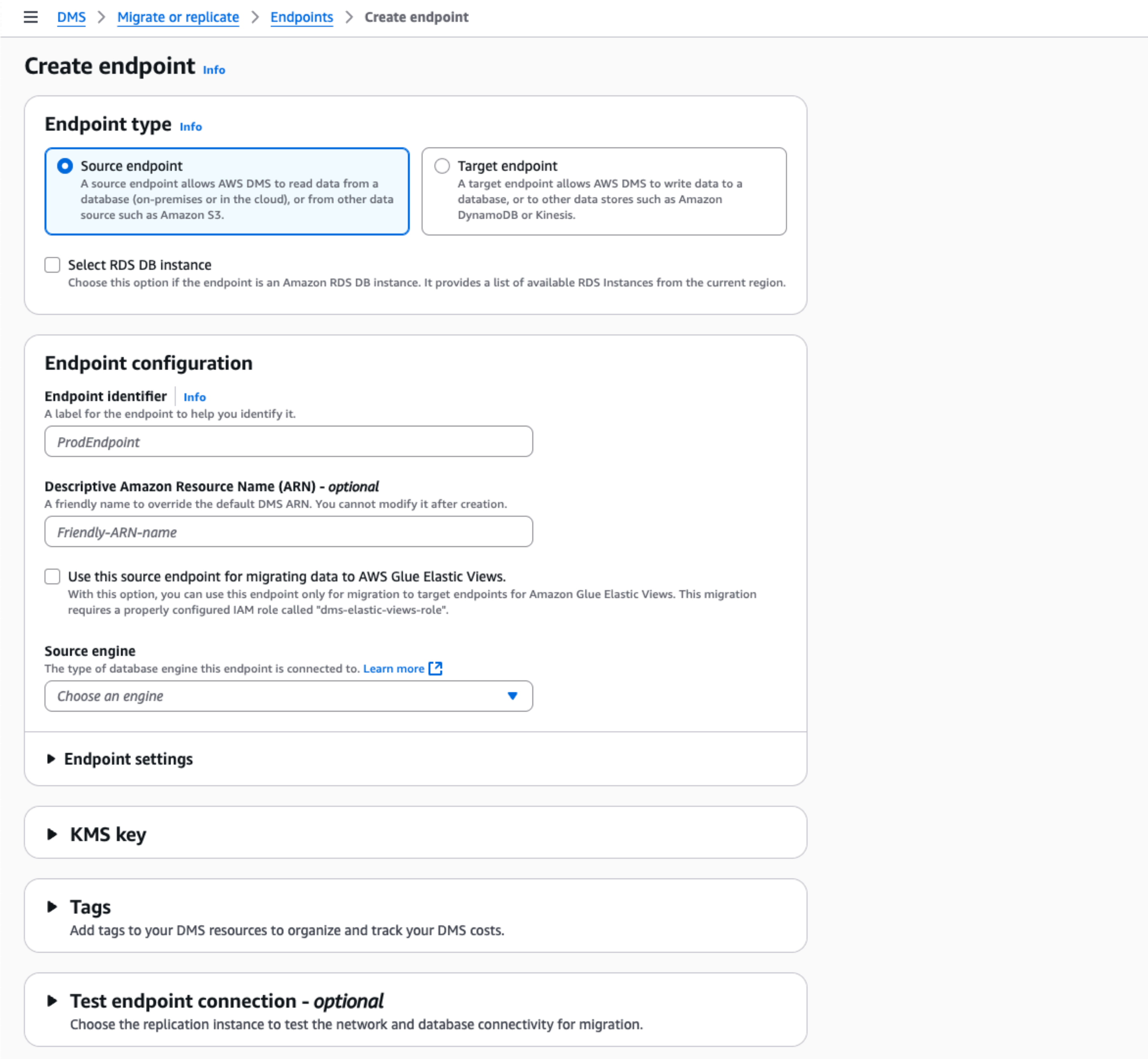
   c. For **AWS KMS key**, leave **(Default) aws/dms** chosen.
5. Leave the settings on the **Maintenance** tab as they are. The default is a 30-minute window selected at random from an 8-hour block of time for each AWS Region, occurring on a random day of the week.

6. Choose **Create**.

AWS DMS creates a replication instance to perform your migration.

Step 2: Specify source and target endpoints

While your replication instance is being created, you can specify the source and target data store endpoints for the Amazon RDS databases you created previously. You create each endpoint separately.



To specify a source endpoint and database endpoint using the AWS DMS console

1. On the console, choose **Endpoints** from the navigation pane and then choose **Create Endpoint**.
2. On the **Create endpoint** page, choose the **Source** endpoint type. Select the **Select RDS DB instance** box, and choose the **dms-mariadb** instance.
3. In the **Endpoint configuration** section, enter **`dms-mysql-source`** for **Endpoint identifier**.

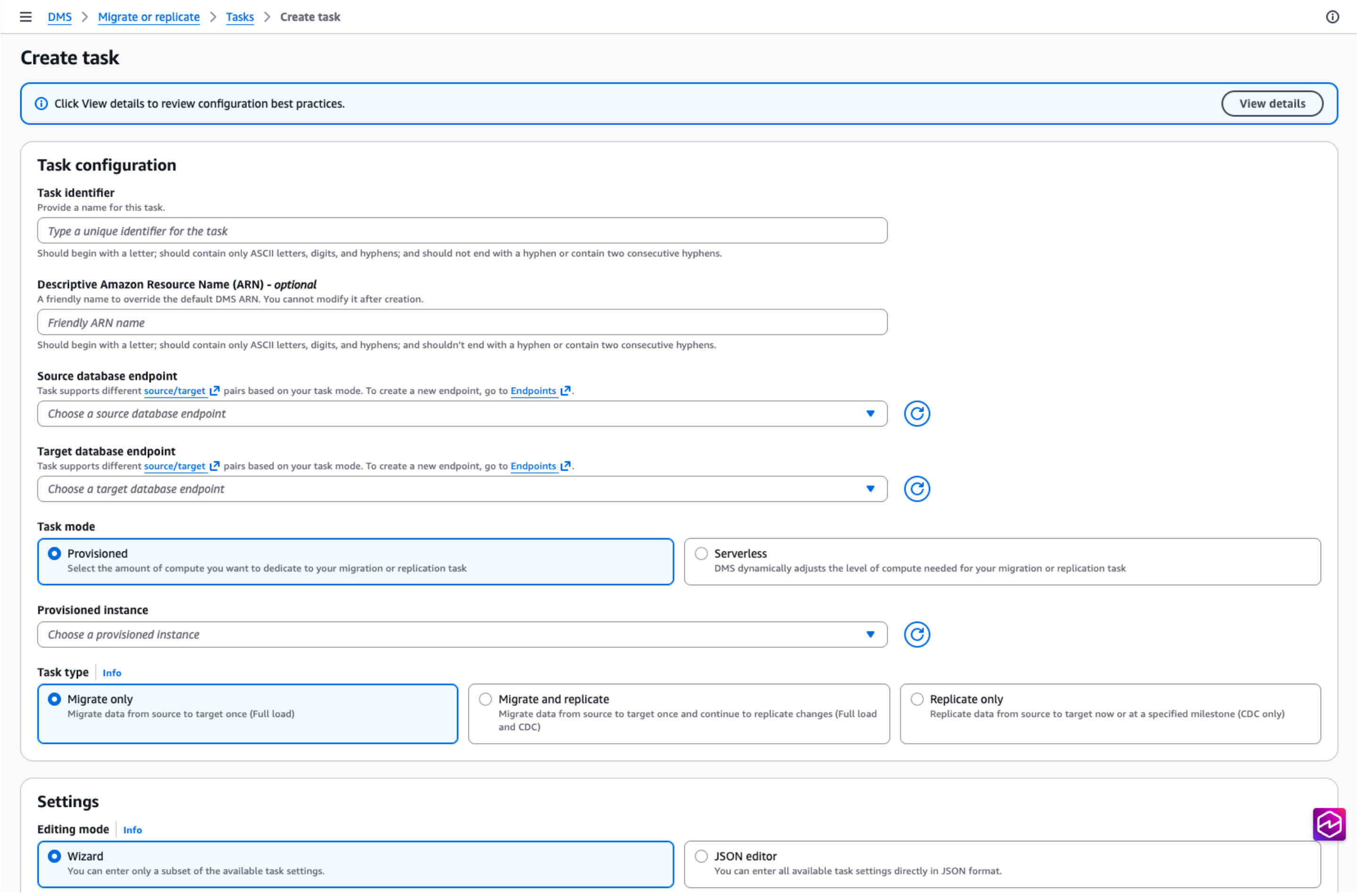
4. For **Source engine**, leave **MySQL** chosen.
5. For **Access to endpoint database**, choose **Provide access information manually**. Verify that the **Port**, **Secure Socket Layer (SSL) mode**, **User name**, and **Password** are correct.
6. Choose the **Test endpoint connection (optional)** tab. For **VPC**, choose **DMSVPC**.
7. For **Replication instance**, leave **dms-instance** chosen.
8. Choose **Run test**.

   After you choose **Run test**, AWS DMS creates the endpoint with the details that you provided and connects to it. If the connection fails, edit the endpoint definition and test the connection again. You can also delete the endpoint manually.
9. After you have a successful test, choose **Create endpoint**.
10. Specify a target database endpoint using the AWS DMS console. To do this, repeat the steps preceding, with the following settings:
    - **Endpoint type**: **Target endpoint**
    - **RDS Instance**: **dms-postgresql**
    - **Endpoint identifier**: **`dms-postgresql-target`**
    - **Target engine**: Leave **`PostgreSQL`** chosen.

When you're finished providing all information for your endpoints, AWS DMS creates your source and target endpoints for use during database migration.

Step 3: Create a task and migrate data

In this step, you create a task to migrate data between the databases you created.



To create a migration task and start your database migration

1. In the console navigation pane, choose **Database migration tasks**, and then choose **Create task**. The **Create database migration task** page opens.

2. In the **Task configuration** section, specify the following task options:

   - **Task identifier**: Enter **`dms-task`**.
   - **Replication instance**: Choose your replication instance (**dms-instance-vpc-*<vpc id>***).
   - **Source database endpoint**: Choose **dms-mysql-source**.
   - **Target database endpoint**: Choose **dms-postgresql-target**.
   - **Migration type**: Choose **Migrate existing data and replicate on-going changes**.

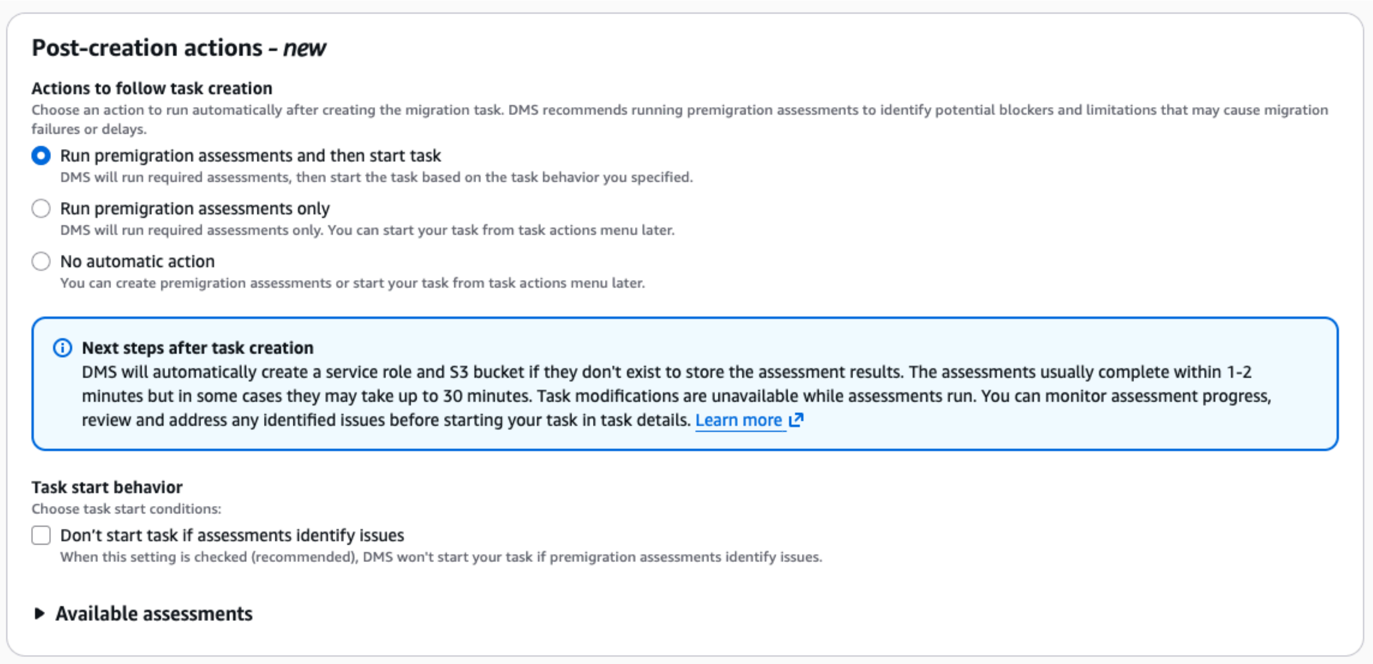
3. Choose the **Task settings** tab. Set the following settings:

   - **Target table preparation mode**: **Do nothing**
   - **Stop task after full load completes**: **Don't stop**

4. Choose the **Table mappings** tab, and expand **Selection rules**. Choose **Add new selection rule**. Set the following settings:

   - **Schema**: **Enter a schema**
   - **Schema name**: **`dms_sample`**

5. Choose the **Migration task startup configuration** tab. Select **Start with premigration assessments (recommended)** to run assessments before starting the task. Alternatively, you can choose **Start automatically on create** to begin migration immediately, or **Run assessments only** to evaluate without starting the task.

6. Choose **Create task**.

AWS DMS then creates the migration task and starts it. The initial database replication takes about 10 minutes. Make sure to do the next step in the tutorial before AWS DMS finishes migrating the data.

Step 4: Test replication

In this section, you insert data into the source database during and after initial replication, and query the target database for the inserted data.

To test replication

1. Make sure that your database migration task shows a status of **Running** but your initial database replication, started in the previous step, isn't complete.
2. Connect to your Amazon EC2 client, and start the MySQL client with the following command. Provide your MySQL database endpoint.

```
mysql -h dms-mysql.abcdefg12345.us-west-2.rds.amazonaws.com -P 3306 -u admin -
pchangeit dms_sample
```

3. Run the following command to insert a record into the source database.

```
MySQL [dms_sample]> insert person (full_name, last_name, first_name) VALUES ('Test
 User1', 'User1', 'Test');
Query OK, 1 row affected (0.00 sec)
```

4. Exit the MySQL client.

```
MySQL [dms_sample]> exit
Bye
```

5. Before replication completes, query the target database for the new record.

   From the Amazon EC2 instance, connect to the target database using the following command, providing your target database endpoint.

```
psql \
   --host=dms-postgresql.abcdefg12345.us-west-2.rds.amazonaws.com \
   --port=5432 \
   --username=postgres \
   --password \
   --dbname=dms_sample
```

   Provide the password (**`changeit`**) when prompted.

6. Before replication completes, query the target database for the new record.

```
dms_sample=> select * from dms_sample.person where first_name = 'Test';
 id | full_name | last_name | first_name
----+-----------+-----------+------------
(0 rows)
```

7. While your migration task is running, you can monitor the progress of your database migration as it happens:

   - In the DMS console navigation pane, choose **Database migration tasks**.
   - Choose **dms-task**.
   - Choose **Table statistics**.

   For more information about monitoring, see Monitoring AWS DMS tasks.

8. After replication completes, query the target database again for the new record. AWS DMS migrates the new record after initial replication completes.

```
dms_sample=> select * from dms_sample.person where first_name = 'Test';
   id    | full_name  | last_name | first_name
---------+------------+-----------+------------
 7077784 | Test User1 | User1     | Test
(1 row)
```

9. Exit the psql client.

```
dms_sample=> quit
```

10. Repeat step 1 to connect to the source database again.

11. Insert another record into the `person` table.

```
MySQL [dms_sample]> insert person (full_name, last_name, first_name) VALUES ('Test
 User2', 'User2', 'Test');
Query OK, 1 row affected (0.00 sec)
```

12. Repeat steps 3 and 4 to disconnect from the source database and connect to the target database.

13. Query the target database for the replicated data again.

```
dms_sample=> select * from dms_sample.person where first_name = 'Test';
   id    | full_name  | last_name | first_name
---------+------------+-----------+------------
 7077784 | Test User1 | User1     | Test
 7077785 | Test User2 | User2     | Test
(2 rows)
```

Step 5: Clean up AWS DMS resources

After you complete the getting started tutorial, you can delete the resources you created. You can use the AWS console to remove them. Make sure to delete the migration tasks before deleting the replication instance and endpoints.

To delete a migration task using the console

1. On the AWS DMS console navigation pane, choose **Database migration tasks**.
2. Choose **dms-task**.
3. Choose **Actions**, **Delete**.

To delete a replication instance using the console

1. On the AWS DMS console navigation pane, choose **Replication instances**.
2. Choose **DMS-instance**.
3. Choose **Actions**, **Delete**.

AWS DMS deletes the replication instance and removes it from the **Replication instances** page.

To remove endpoints using the console

1. On the AWS DMS console navigation pane, choose **Endpoints**.
2. Choose **dms-mysql-source**.
3. Choose **Actions**, **Delete**.

After you delete your AWS DMS resources, make sure also to delete the following resources. For help with deleting resources in other services, see each service's documentation.

- Your RDS databases.
- Your RDS database parameter groups.
- Your RDS subnet groups.
- Any Amazon CloudWatch logs that were created along with your databases and replication instance.

- Security groups that were created for your Amazon VPC and Amazon EC2 client. Make sure to remove the inbound rule from **Default** for the **launch-wizard-1** security groups, which is necessary for you to be able delete them.
- Your Amazon EC2 client.
- Your Amazon VPC.
- Your Amazon EC2 key pair for your Amazon EC2 client.

AWS DMS Fleet Advisor end of support

Important

End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

AWS DMS Fleet Advisor end of support details

End of support plan

After careful consideration, we decided to end support for AWS DMS Fleet Advisor, effective May 20, 2026. AWS DMS Fleet Advisor will no longer accept new customers beginning May 20, 2025. As an existing customer with an account signed up for AWS DMS before May 20, 2025 you can continue to use AWS DMS Fleet Advisor and its features. After May 20, 2026, you will no longer be able to use AWS DMS Fleet Advisor.

To help you plan and migrate away from Fleet Advisor, we will discontinue the offering gradually over 12 months. There are two important dates to note: May 20, 2025 and May 20, 2026.

We will discontinue the service on May 20, 2026. You will not be able to download or use AWS DMS Fleet Advisor capabilities. Support will no longer be available for AWS DMS Fleet Advisor from that time. We recommend that you migrate your database assessment projects to AWS Migration Evaluator before May 20, 2026.

FAQ

What is the cutoff point for “current customers”?

We maintain an allow list of account IDs who have used AWS DMS Fleet Advisor in last 90 days, including detectors, within the service. If an existing customer is having difficulties using service, create a support case.

What does AWS mean by “access” to the service?

Existing AWS DMS Fleet Advisor customers can do all the things they could previously. The only change is that non-existing customers will not be able to use AWS DMS Fleet Advisor or any of its capabilities, including Data Collectors tool, Discovering OS and database servers, Collecting data, and generating Target recommendations.

Can existing AWS DMS Fleet Advisor customers download Data Collectors tool, Discovering OS and database servers, Collecting data, and generating Target recommendations?

Yes. if you have any difficulties, create a support case.

Why are you no longer offering AWS DMS Fleet Advisor?

We have identified that customers prefer Migration Evaluator for their database migration assessment projects as an alternative to AWS DMS Fleet Advisor. Migration Evaluator provides both business and technical stakeholders visibility into the projected cost of running their on-premises workloads in the AWS Cloud. Business stakeholders can download a one-page summary highlighting the estimated savings to re-host at AWS based on usage patterns with costs broken down by infrastructure and software licenses. For more a technical audience, detailed per-server and per-SQL-server data is also available. This export combines on-premises discovery data (server hardware provisioning, SQL configurations, and resource utilization) with recommendations.

What are my options now?

We recommend using AWS Migration Evaluator for all new and ongoing database migration assessment projects. If you have already downloaded Fleet Advisor's Data Collectors tool, you may continue to use them. However, support and updates for these tools will cease after May 20, 2025. For existing Fleet Advisor users, we recommend completing any in-progress assessments with Fleet Advisor, then transitioning to Migration Evaluator to benefit from critical patches, security updates, performance improvements, and new features.

Discovering and evaluating databases for migration with AWS DMS Fleet Advisor

Important

End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

You can use DMS Fleet Advisor to collect metadata and performance metrics from multiple database environments. These collected metrics provide insight into your data infrastructure. DMS Fleet Advisor collects metadata and metrics from your on-premises database and analytic servers from one or more central locations without the need to install it on every computer. Currently, DMS Fleet Advisor supports discovery and metrics collection for Microsoft SQL Server, MySQL, Oracle, and PostgreSQL database servers.

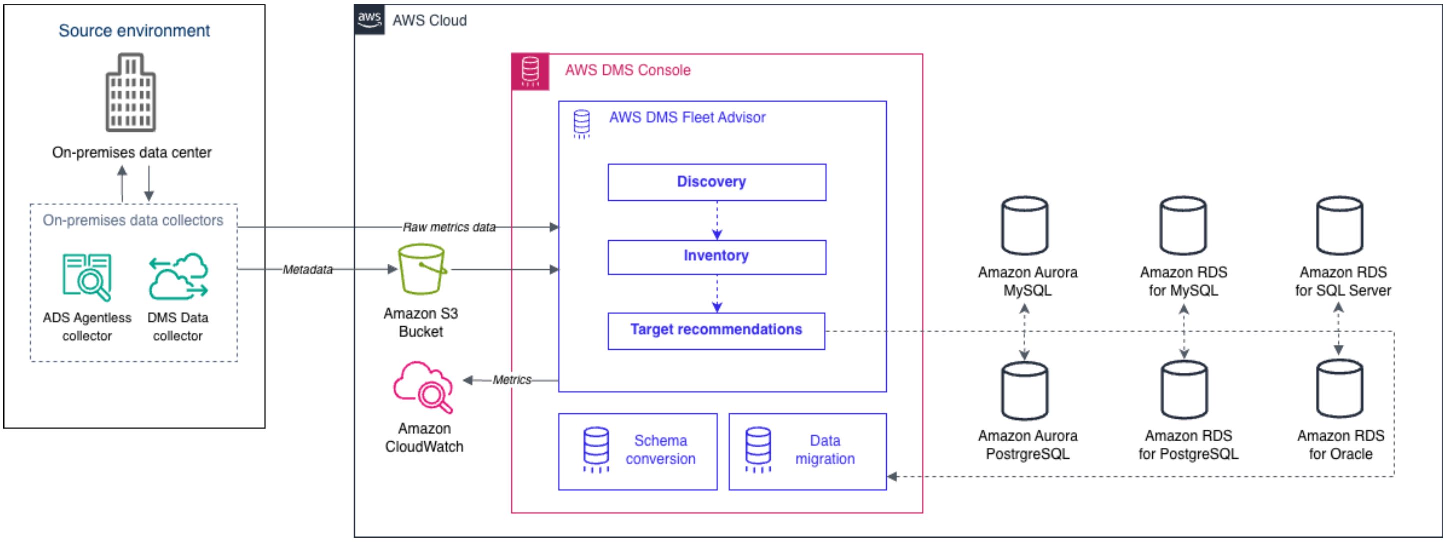
Based on data discovered from the network, you can build an inventory to define the list of database servers for further data collection. After AWS DMS collects information about your servers, databases, and schemas, you can analyze the feasibility of your intended database migrations.

For databases in your inventory that you plan to migrate to the AWS Cloud, DMS Fleet Advisor generates right-sized target recommendations. To generate target recommendations, DMS Fleet Advisor considers the metrics from your data collector and preferred settings. After DMS Fleet Advisor generates recommendations, you can view detailed information for each target database configuration. Your organization's database engineers and administrators can use DMS Fleet Advisor Target Recommendations to plan the migration of their on-premises databases to AWS. You can explore different available migration options and export these recommendations into the AWS Pricing Calculator to further optimize the cost.

For the list of supported source databases, see Sources for DMS Fleet Advisor.

For the list of databases that DMS Fleet Advisor uses to generate target recommendations, see Targets for DMS Fleet Advisor. DMS Fleet Advisor generates like to like recommendations, for example, from source Oracle to target Oracle database. DMS Fleet Advisor also generates heterogenous recommendations, such as migration from source Oracle or Microsoft SQL Server to target RDS for PostgreSQL or Aurora PostgreSQL database.

The following diagram illustrates the AWS DMS Fleet Advisor Target Recommendations process.



Use the following topics to better understand how to use AWS DMS Fleet Advisor.

Supported AWS Regions

You can use DMS Fleet Advisor in the following AWS Regions.

Region Name	Region
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Canada (Central)	ca-central-1
Canada West (Calgary)	ca-west-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Tokyo)	ap-northeast-1

Region Name	Region
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Jakarta)	ap-southeast-3
Asia Pacific (Melbourne)	ap-southeast-4
Asia Pacific (Hyderabad)	ap-south-2
Europe (Frankfurt)	eu-central-1
Europe (Zurich)	eu-central-2
Europe (Stockholm)	eu-north-1
Europe (Spain)	eu-south-2
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Milan)	eu-south-1
Israel (Tel Aviv)	il-central-1
South America (São Paulo)	sa-east-1
Middle East (Bahrain)	me-south-1
Middle East (UAE)	me-central-1

Region Name	Region
Africa (Cape Town)	af-south-1

Understand how to use DMS Fleet Advisor

Important

End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

You can use DMS Fleet Advisor to discover your source on-premises databases for migration to the AWS Cloud. Then, you can determine the right migration target in the AWS Cloud for each of your on-premises databases. Use the following workflow to create an inventory of your source databases and generate target recommendations.

1. Create an Amazon S3 bucket, IAM policies, roles, and users. For more information, see Creating required resources.
2. Create database users with the minimum permissions required for the DMS data collector. For more information, see Creating database users.
3. Create and download a data collector. For more information, see Creating a data collector.
4. Install the data collector in your local environment. Next, configure your data collector to make sure that it can send the collected data to DMS Fleet Advisor. For more information, see Installing a data collector.
5. Discover the OS and database servers in your data environment. For more information, see Discovering OS and database servers.
6. Collect database metadata and resource utilization metrics. For more information, see Collecting data.
7. Analyze your source databases and schemas. DMS Fleet Advisor runs the large-scale assessment of your databases to identify similar schemas. For more information, see Using inventories for analysis in AWS DMS Fleet Advisor.

8. Generate, view, and save a local copy of the target recommendations for your source databases. For more information, see Target recommendations.

After you determine the migration target for each source database, you can use DMS Schema Conversion to convert your database schemas to a new platform. Then, you can use AWS DMS to migrate data. For more information, see Converting database schemas using DMS Schema Conversion and What is AWS Database Migration Service?

This video introduces the DMS Schema Conversion user interface and helps you get familiar with the core components of this service.

Creating required AWS resources for AWS DMS Fleet Advisor

Important

End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

DMS Fleet Advisor needs a set of AWS resources in your account to forward and import inventory information, and to update the status of the DMS data collector.

Before you collect data and create inventories of databases and schemas for the first time, complete the following prerequisites.

To configure your Amazon S3 bucket and IAM resources, do one of the following:

- Configure Amazon S3 and IAM resources using CloudFormation (recommended).
- Configure Amazon S3 and IAM resources in the AWS Management Console

Configure Amazon S3 and IAM resources using CloudFormation

A *CloudFormation stack* is a collection of AWS resources that you can manage as a single unit. To simplify creating required resources for DMS Fleet Advisor, you can use the CloudFormation template files to create CloudFormation stacks. For more information, see Creating a stack on the CloudFormation console in *CloudFormation User Guide*.

> **Note**
>
> This section only applies to using the standalone DMS Fleet Advisor collector. For information about using a single on-premises collector for gathering information about both databases and servers, see [Application Discovery Service Agentless Collector](#) in the *[AWS Application Discovery Service User Guide](#)*.

Amazon S3 and IAM resources created by CloudFormation

When you use the CloudFormation templates, they create stacks that include the following resources in your AWS account:

- An Amazon S3 bucket named `dms-fleetadvisor-data-`*`accountId`*`-`*`region`*
- An IAM user named `FleetAdvisorCollectorUser-`*`region`*
- An IAM service role named `FleetAdvisorS3Role-`*`region`*
- An access policy named `FleetAdvisorS3Role-`*`region`*`-Policy`
- An access policy named `FleetAdvisorCollectorUser-`*`region`*`-Policy`
- An IAM Service Linked Role (SLR) named `AWSServiceRoleForDMSFleetAdvisor`

Follow the steps listed below to configure your resources with CloudFormation.

- [Step 1: Download the CloudFormation template files](#)
- [Step 2: Configure Amazon S3 and IAM using CloudFormation](#)

Step 1: Download the CloudFormation template files

A *CloudFormation template* is a declaration of the AWS resources that make up a stack. The template is stored as a JSON file.

To download the CloudFormation template files

1. Open the context (right-click) menu for one of the following links and choose **Save Link As**:

    - If you plan to use DMS Fleet Advisor, choose [dms-fleetadvisor-iam-slr-s3.zip](#). If you have already created the SLR for DMS Fleet Advisor, choose [dms-fleetadvisor-iam-s3.zip](#)

- If you plan to use the AWS Application Discovery Service (ADS) Agentless Collector and have not created the SLR for it, then choose [dms-fleetadvisor-ads-iam-slr-s3.zip](). If you have created the SLR for DMS Fleet Advisor with ADS before, choose [dms-fleetadvisor-ads-iam-s3.zip]().

2. Save the file to your computer.

Step 2: Configure Amazon S3 and IAM using CloudFormation

When you use the CloudFormation template for IAM, it creates the Amazon S3 and IAM resources listed previously.

To configure Amazon S3 and IAM using CloudFormation

1. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. Start the Create Stack wizard by choosing **Create Stack** and **With new resources** in the dropdown list.
3. On the **Create stack** page, do the following:

    a. For **Prepare template**, choose **Template is ready**.

    b. For **Template source**, choose **Upload a template file**.

    c. For **Choose file**, navigate to, then choose **dms-fleetadvisor-iam-slr-S3.json**, **dms-fleetadvisor-iam-S3.json.**, **dms-fleetadvisor-ads-iam-slr-s3.zip**, or **dms-fleetadvisor-ads-iam-s3.zip**.

    d. Choose **Next**.
4. On the **Specify stack details** page, do the following:

    a. For **Stack name**, enter **`dms-fleetadvisor-iam-slr-s3`**,**`dms-fleetadvisor-iam-s3`**, **`dms-fleetadvisor-ads-iam-slr-s3`**, or **`dms-fleetadvisor-ads-iam-s3`**.

    b. Choose **Next**.
5. On the **Configure stack options page**, choose **Next**.
6. On the **Review dms-fleetadvisor-iam-slr-s3**,**Review dms-fleetadvisor-iam-s3**, **Review dms-fleetadvisor-ads-iam-slr-s3**, or **Review dms-fleetadvisor-ads-iam-s3** page, do the following:

    a. Select the **I acknowledge that CloudFormation might create IAM resources with custom names** check box.

    b. Choose **Submit**.

CloudFormation creates the S3 bucket and IAM roles and user that DMS Fleet Advisor requires. In the left panel, when **dms-fleetadvisor-iam-slr-s3**, **dms-fleetadvisor-iam-s3**, **dms-fleetadvisor-ads-iam-slr-s3**, or **dms-fleetadvisor-ads-iam-s3** shows **CREATE_COMPLETE**, proceed to the next step.

7. In the left panel, choose **dms-fleetadvisor-iam-slr-s3**, **dms-fleetadvisor-iam-s3**, **dms-fleetadvisor-ads-iam-slr-s3**, or **dms-fleetadvisor-ads-iam-s3**. In the right panel, do the following:

   a. Choose **Stack info**. Your stack has an ID in the format **arn:aws:cloudformation:*`region`*:*`account-no`*:stack/dms-fleetadvisor-iam-slr-s3/*`identifier`***, **arn:aws:cloudformation:*`region`*:*`account-no`*:stack/dms-fleetadvisor-iam-s3/*`identifier`***, **arn:aws:cloudformation:*`region`*:*`account-no`*:stack/dms-fleetadvisor-ads-iam-slr-s3/*`identifier`***, or **arn:aws:cloudformation:*`region`*:*`account-no`*:stack/dms-fleetadvisor-ads-iam-s3/*`identifier`***.

   b. Choose **Resources**. You should see the following:

      - An Amazon S3 bucket named `dms-fleetadvisor-data-`*`accountId`*`-`*`region`*
      - A service role named `FleetAdvisorS3Role-`*`region`*
      - An IAM user named `FleetAdvisorCollectorUser-`*`region`*
      - An IAM SLR named `AWSServiceRoleForDMSFleetAdvisor` (if you downloaded `dms-fleet-advisor-iam-slr-s3.zip` or `dms-fleet-advisor-ads-iam-slr-s3.zip`).
      - An access policy named `FleetAdvisorS3Role-`*`region`*`-Policy`
      - An access policy named `FleetAdvisorCollectorUser-`*`region`*`-Policy`

Configure Amazon S3 and IAM resources in the AWS Management Console

Create an Amazon S3 bucket

Create an Amazon S3 bucket where inventory metadata can be stored. We recommend that you preconfigure this S3 bucket before using DMS Fleet Advisor. AWS DMS stores your DMS Fleet Advisor inventory metadata in this S3 bucket.

For more information about creating an S3 bucket, see Create your first S3 bucket in the *Amazon S3 User Guide*.

> **Note**
>
> DMS Fleet Advisor only supports SSE-S3 encrypted buckets.

To create an Amazon S3 bucket to store local data environment information

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose **Create bucket**.
3. On the **Create bucket** page, enter a globally unique name that includes your sign-in name for the bucket, such as **fa-bucket-*`yoursignin`***.
4. Choose the AWS Region where you use the DMS Fleet Advisor.
5. Keep the remaining settings and choose **Create bucket**.

Create IAM resources

In this section, you create IAM resources for your data collector, IAM user, and DMS Fleet Advisor.

Topics

- Create IAM resources for your data collector
- Create the DMS Fleet Advisor service-linked role

Create IAM resources for your data collector

To make sure that your data collector works correctly and uploads the collected metadata to your Amazon S3 bucket, create the following policies. Then, create an IAM user with the following minimum permissions. For more information about DMS data collector, see Discovering databases for migration using data collectors in AWS DMS.

To create an IAM policy for DMS Fleet Advisor and your data collector to access Amazon S3

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. In the **Create policy** page, choose the **JSON** tab.
5. Paste the following JSON into the editor, replacing the example code. Replace *fa_bucket* with the name of the Amazon S3 bucket that you created in the previous section.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "s3:GetObject*",
            "s3:GetBucket*",
            "s3:List*",
            "s3:DeleteObject*",
            "s3:PutObject*"
        ],
        "Resource": [
            "arn:aws:s3:::fa_bucket",
            "arn:aws:s3:::fa_bucket/*"
        ]
    }
    ]
    }
```

6. Choose **Next: Tags** and **Next: Review.**
7. Enter **`FleetAdvisorS3Policy`** for **Name***, and then choose **Create policy**.

To create an IAM policy for DMS data collector to access DMS Fleet Advisor

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. In the **Create policy** page, choose the **JSON** tab.

5. Paste the following JSON code into the editor, replacing the example code.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "dms:DescribeFleetAdvisorCollectors",
            "dms:ModifyFleetAdvisorCollectorStatuses",
            "dms:UploadFileMetadataList"
        ],
        "Resource": "*"
    }
    ]
    }
```

6. Choose **Next: Tags** and **Next: Review.**
7. Enter **`DMSCollectorPolicy`** for **Name***, then choose **Create policy**.

To create an IAM user with minimum permissions to use DMS data collector

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. Choose **Add users**.
4. On the **Add user** page, enter **`FleetAdvisorCollectorUser`** for **User name***. Choose **Access key- Programmatic Access** for **Select AWS Access Type**. Choose **Next: Permissions**.
5. In the **Set permissions** section, choose **Attach existing policies directly**.
6. Use the search control to find and choose the **DMSCollectorPolicy** and **FleetAdvisorS3Policy** policies that you created before. Choose **Next: Tags**.
7. On the **Tags** page, choose **Next: Review**.
8. On the **Review** page, choose **Create user**. On the next page, choose **Download .csv** to save the new user credentials. Use these credentials with DMS Fleet Advisor for minimum required access permissions.

To create an IAM role for DMS Fleet Advisor and your data collector to access Amazon S3

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. On the **Select trusted entity** page, for **Trusted entity type**, choose **AWS Service**. For **Use cases for other AWS services**, choose **DMS**.
5. Select the **DMS** check box and choose **Next**.
6. On the **Add permissions** page, choose **FleetAdvisorS3Policy**. Choose **Next**.
7. On the **Name, review, and create** page, enter **`FleetAdvisorS3Role`** for **Role name**, then choose **Create role**.
8. On the **Roles** page, enter **`FleetAdvisorS3Role`** for **Role name**. Choose **FleetAdvisorS3Role**.
9. On the **FleetAdvisorS3Role** page, choose the **Trust relationships** tab. Choose **Edit trust policy**.
10. On the **Edit trust policy** page, paste the following JSON into the editor, replacing the existing text.

    JSON

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

The preceding policy grants the `sts:AssumeRole` permission to the services that AWS DMS uses to import collected data from the Amazon S3 bucket.

11. Choose **Update policy**.

Create the DMS Fleet Advisor service-linked role

DMS Fleet Advisor uses a service-linked role to manage Amazon CloudWatch metrics in your AWS account. DMS Fleet Advisor uses this service-linked role to publish the collected database performance metrics to CloudWatch on your behalf.

To create the service-linked role for DMS Fleet Advisor

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**. Then, choose **Create role**.
3. For **Trusted entity type**, choose **AWS service**.
4. For **Use cases for other AWS services**, choose **DMS – Fleet Advisor**.
5. Select the **DMS – Fleet Advisor** check box and choose **Next**.
6. On the **Add permissions** page, choose **Next**.
7. On the **Name, review, and create** page, choose **Create role**.

Alternatively, you can create this service-linked role from the AWS API or AWS CLI. For more information, see [Creating a service-linked role for AWS DMS Fleet Advisor](#).

After you create the service-linked role for DMS Fleet Advisor, you can see performance metrics for your source databases in target recommendations. Also, you can see these metrics and in your CloudWatch account. For more information, see [Target recommendations](#).

To create an IAM policy that is required for the DMS Fleet Advisor service-linked role

The minimum required permissions to create the service-linked role are specified in the `DMSFleetAdvisorCreateServiceLinkedRolePolicy` policy. Create this IAM policy for your account if you are unable to create the service-linked role.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. In the **Create policy** page, choose the **JSON** tab.
5. Paste the following JSON code into the editor, replacing the example code.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": "iam:CreateServiceLinkedRole",
        "Resource": "arn:aws:iam::*:role/aws-service-role/dms-fleet-
advisor.amazonaws.com/AWSServiceRoleForDMSFleetAdvisor*",
        "Condition": {"StringLike": {"iam:AWSServiceName": "dms-fleet-
advisor.amazonaws.com"}}
    },
    {
        "Effect": "Allow",
        "Action": [
            "iam:AttachRolePolicy",
            "iam:PutRolePolicy"
        ],
        "Resource": "arn:aws:iam::*:role/aws-service-role/dms-fleet-
advisor.amazonaws.com/AWSServiceRoleForDMSFleetAdvisor*"
    }
    ]
    }
```

6. Choose **Next: Tags** and **Next: Review.**
7. Enter **`DMSFleetAdvisorCreateServiceLinkedRolePolicy`** for **Name***, then choose **Create policy**.

Now, you can use this policy to create the service-linked role for DMS Fleet Advisor.

Creating database users for AWS DMS Fleet Advisor

Important

End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

This section describes how to create users for your source databases with the minimum permissions required for the DMS data collector.

This section contains the following topics:

Using a database user with AWS DMS Fleet Advisor

You can use a database user other than `root` with the DMS data collector. Specify the user name and password after adding the database to the inventory, but before you run your data collector. For more information about adding databases to the inventory, see Managing monitored objects in AWS DMS.

After you finish using the DMS data collector, you can delete the database users that you created. For more information, see Deleting database users.

Important

In the following examples, replace *`{your_user_name}`* with the name of the database user that you created for your database. Then, replace *`{your_password}`* with a secure password.

Creating a database user with MySQL

To create a database user in a MySQL source database, use the following script. Make sure that you keep one version of the GRANT statement that depends on the version of your MySQL database.

```
CREATE USER {your_user_name} identified BY '{your_password}';

GRANT PROCESS ON *.* TO {your_user_name};
GRANT REFERENCES ON *.* TO {your_user_name};
GRANT TRIGGER ON *.* TO {your_user_name};
GRANT EXECUTE ON *.* TO {your_user_name};

# For MySQL versions lower than 8.0, use the following statement.
GRANT SELECT, CREATE TEMPORARY TABLES ON `temp`.* TO {your_user_name};

# For MySQL versions 8.0 and higher, use the following statement.
GRANT SELECT, CREATE TEMPORARY TABLES ON `mysql`.* TO {your_user_name};

GRANT SELECT ON performance_schema.* TO {your_user_name};

SELECT
IF(round(Value1 + Value2 / 100 + Value3 / 10000, 4) > 5.0129, 'GRANT EVENT ON *.*
 TO {your_user_name};', 'SELECT ''Events are not applicable'';') sql_statement
INTO @stringStatement
FROM (
   SELECT
     substring_index(ver, '.', 1)                            value1,
     substring_index(substring_index(ver, '.', 2), '.', - 1) value2,
     substring_index(ver, '.', - 1)                          value3
   FROM  (
     SELECT
       IF((@@version regexp '[^0-9\.]+') != 0, @@innodb_version, @@version) AS ver
     FROM dual
   ) vercase
 ) v;

PREPARE sqlStatement FROM @stringStatement;
SET @stringStatement := NULL;
EXECUTE sqlStatement;
DEALLOCATE PREPARE sqlStatement;
```

Creating a database user with Oracle

To create a database user in an Oracle source database, use the following script.

To run this SQL script, connect to your Oracle database using SYSDBA privileges. After you run this SQL script, connect to your database using the credentials of the user that you created with this script. Also, use the credentials of this user to run the DMS data collector.

The following script adds the C## prefix to the name of the user for Oracle multitenant container databases (CDB).

```
CREATE USER {your_user_name} IDENTIFIED BY "{your_password}";
GRANT CREATE SESSION TO {your_user_name};
GRANT SELECT ANY DICTIONARY TO {your_user_name};
GRANT SELECT ON DBA_WM_SYS_PRIVS TO {your_user_name};
BEGIN
DBMS_NETWORK_ACL_ADMIN.CREATE_ACL(
    acl => UPPER('{your_user_name}') || '_Connect_Access.xml',
    description => 'Connect Network',
    principal => UPPER('{your_user_name}'),
    is_grant => TRUE,
    privilege => 'resolve',
    start_date => NULL,
    end_date => NULL);

DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL(
    acl => UPPER('{your_user_name}') || '_Connect_Access.xml',
    host => '*',
    lower_port => NULL,
    upper_port => NULL);
END;
```

Creating a database user with PostgreSQL

To create a database user in a PostgreSQL source database, use the following script.

```
CREATE USER "{your_user_name}" WITH LOGIN PASSWORD '{your_password}';
GRANT pg_read_all_settings TO "{your_user_name}";

-- For PostgreSQL versions 10 and higher, add the following statement.
GRANT EXECUTE ON FUNCTION pg_ls_waldir() TO "{your_user_name}";
```

Creating a database user with Microsoft SQL Server

To create a database user in a Microsoft SQL Server source database, use the following script.

```
USE master
GO

IF NOT EXISTS (SELECT * FROM sys.sql_logins WHERE name = N'{your_user_name}')

CREATE LOGIN [{your_user_name}] WITH PASSWORD=N'{your_password}',
 DEFAULT_DATABASE=[master], DEFAULT_LANGUAGE=[us_english], CHECK_EXPIRATION=OFF,
 CHECK_POLICY=OFF

GO

GRANT VIEW SERVER STATE TO [{your_user_name}]

GRANT VIEW ANY DEFINITION TO [{your_user_name}]

GRANT VIEW ANY DATABASE TO [{your_user_name}]

IF LEFT(CONVERT(SYSNAME,SERVERPROPERTY('ProductVersion')), CHARINDEX('.',
 CONVERT(SYSNAME,SERVERPROPERTY('ProductVersion')), 0)-1) >= 12
EXECUTE('GRANT CONNECT ANY DATABASE TO [{your_user_name}]')

DECLARE @dbname VARCHAR(100)
DECLARE @statement NVARCHAR(max)

DECLARE db_cursor CURSOR
LOCAL FAST_FORWARD
FOR
SELECT
name
FROM MASTER.sys.databases
WHERE state = 0
AND is_read_only = 0
    OPEN db_cursor
FETCH NEXT FROM db_cursor INTO @dbname
WHILE @@FETCH_STATUS = 0
BEGIN

SELECT @statement = 'USE '+ quotename(@dbname) +';'+ '
```

```
IF NOT EXISTS (SELECT * FROM sys.syslogins WHERE name = ''{your_user_name}'') OR NOT
 EXISTS (SELECT * FROM sys.sysusers WHERE name = ''{your_user_name}'')
  CREATE USER [{your_user_name}] FOR LOGIN [{your_user_name}];

EXECUTE sp_addrolemember N''db_datareader'', [{your_user_name}]'

BEGIN TRY
EXECUTE sp_executesql @statement
END TRY
BEGIN CATCH
DECLARE @err NVARCHAR(255)

SET @err = error_message()

PRINT @dbname
PRINT @err
END CATCH

FETCH NEXT FROM db_cursor INTO @dbname
END
CLOSE db_cursor
DEALLOCATE db_cursor

USE msdb
GO

GRANT EXECUTE ON dbo.agent_datetime TO [{your_user_name}]
```

Deleting database users

After you complete all data collection tasks, you can delete the database users that you created for the DMS data collector. You can use the following scripts to delete the users with minimum permissions from your databases.

To delete the user from your MySQL database, run the following script.

```
DROP USER IF EXISTS "{your_user_name}";
```

To delete the user from your Oracle database, run the following script.

```
DECLARE
-- Input parameters, please set correct value
```

```
cnst$user_name CONSTANT VARCHAR2(255) DEFAULT '{your_user_name}';

-- System variables, please, don't change
var$is_exists INTEGER DEFAULT 0;
BEGIN
SELECT COUNT(hal.acl) INTO var$is_exists
FROM dba_host_acls hal
WHERE hal.acl LIKE '%' || UPPER(cnst$user_name) || '_Connect_Access.xml';
IF var$is_exists > 0 THEN
DBMS_NETWORK_ACL_ADMIN.DROP_ACL(
  acl => UPPER(cnst$user_name) || '_Connect_Access.xml');
END IF;
SELECT COUNT(usr.username) INTO var$is_exists
FROM all_users usr
WHERE usr.username = UPPER(cnst$user_name);
IF var$is_exists > 0 THEN
EXECUTE IMMEDIATE 'DROP USER ' || cnst$user_name || ' CASCADE';
END IF;
END;
```

To delete the user from your PostgreSQL database, run the following script.

```
DROP USER IF EXISTS "{your_user_name}";
```

To delete the user from your SQL Server database, run the following script.

```
USE msdb
GO

REVOKE EXECUTE ON dbo.agent_datetime TO [{your_user_name}]

USE master
GO

DECLARE @dbname VARCHAR(100)
DECLARE @statement NVARCHAR(max)

DECLARE db_cursor CURSOR
LOCAL FAST_FORWARD
FOR
SELECT
name
FROM MASTER.sys.databases
```

```
WHERE state = 0
AND is_read_only = 0
OPEN db_cursor
FETCH NEXT FROM db_cursor INTO @dbname
WHILE @@FETCH_STATUS = 0
BEGIN

SELECT @statement = 'USE '+ quotename(@dbname) +';'+ '
EXECUTE sp_droprolemember N''db_datareader'', [{your_user_name}]

IF EXISTS (SELECT * FROM sys.syslogins WHERE name = ''{your_user_name}'')
OR EXISTS (SELECT * FROM sys.sysusers WHERE name = ''{your_user_name}'')
DROP USER [{your_user_name}];'

BEGIN TRY
EXECUTE sp_executesql @statement
END TRY
BEGIN CATCH
DECLARE @err NVARCHAR(255)

SET @err = error_message()

PRINT @dbname
PRINT @err
END CATCH

FETCH NEXT FROM db_cursor INTO @dbname
END
CLOSE db_cursor
DEALLOCATE db_cursor

GO

IF EXISTS (SELECT * FROM sys.sql_logins WHERE name = N'{your_user_name}')
DROP LOGIN [{your_user_name}]  -- Use for SQL login

GO
```

Discovering databases for migration using data collectors in AWS DMS

Important

End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

To discover your source data infrastructure, you can use either AWS Application Discovery Service Agentless Collector or AWS DMS data collectors. The ADS Agentless Collector is an on-premises application that collects information about your on-premises environment through agentless methods, including server profile information (for example, OS, number of CPUs, amount of RAM), database metadata, and utilization metrics. You install the Agentless Collector as a virtual machine (VM) in your VMware vCenter Server environment using an Open Virtualization Archive (OVA) file. An AWS DMS *data collector* is a Windows application that you install in your local environment. This application connects to your data environment and collects metadata and performance metrics from your on-premises database and analytic servers. Once database metadata and performance metrics have been collected through either the ADS Agentless Collector or a DMS data collector, DMS Fleet Advisor builds an inventory of servers, databases, and schemas that you can migrate to the AWS Cloud.

The DMS data collector is a Windows application which uses .NET libraries, connectors, and data providers to connect to your source databases for database discovery and data collection.

The DMS data collector runs on Windows. However, your DMS data collector can collect data from all supported database vendors regardless of the OS server where they run.

The DMS data collector uses a protected RTPS protocol with TLS encryption to establish a secure connection with DMS Fleet Advisor. Therefore, your data is encrypted during transit by default.

AWS DMS has the maximum number of data collectors that you can create for your AWS account. See the following section for information about AWS DMS service quotas Quotas for AWS Database Migration Service.

Topics

Permissions for a DMS data collector

The database users that you create for the DMS data collector should have read permissions. However, in some cases, the database user requires the `EXECUTE` permission. For more information, see Creating database users for AWS DMS Fleet Advisor.

The DMS data collector requires additional permissions to run the discovery scripts.

- For OS discovery, the DMS data collector needs credentials for the domain server to run requests using the LDAP protocol.
- For database discovery in Linux, the DMS data collector needs credentials with `sudo SSH` grants. Also, you should configure your Linux servers to allow running remote SSH scripts.
- For database discovery in Windows, the DMS data collector needs credentials with grants to run Windows Management Instrumentation (WMI) and WMI Query Language (WQL) queries and read the registry. Also, you should configure your Windows servers to allow running remote WMI, WQL, and PowerShell scripts.

Creating a data collector for AWS DMS Fleet Advisor

Important

End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

Learn how to create and download a DMS data collector.

Before you create a data collector, use the IAM console to create a service-linked role for DMS Fleet Advisor. This role allows principals to publish metric data points to Amazon CloudWatch. DMS Fleet Advisor uses this role to display charts with database metrics. For more information, see Creating a service-linked role for AWS DMS Fleet Advisor.

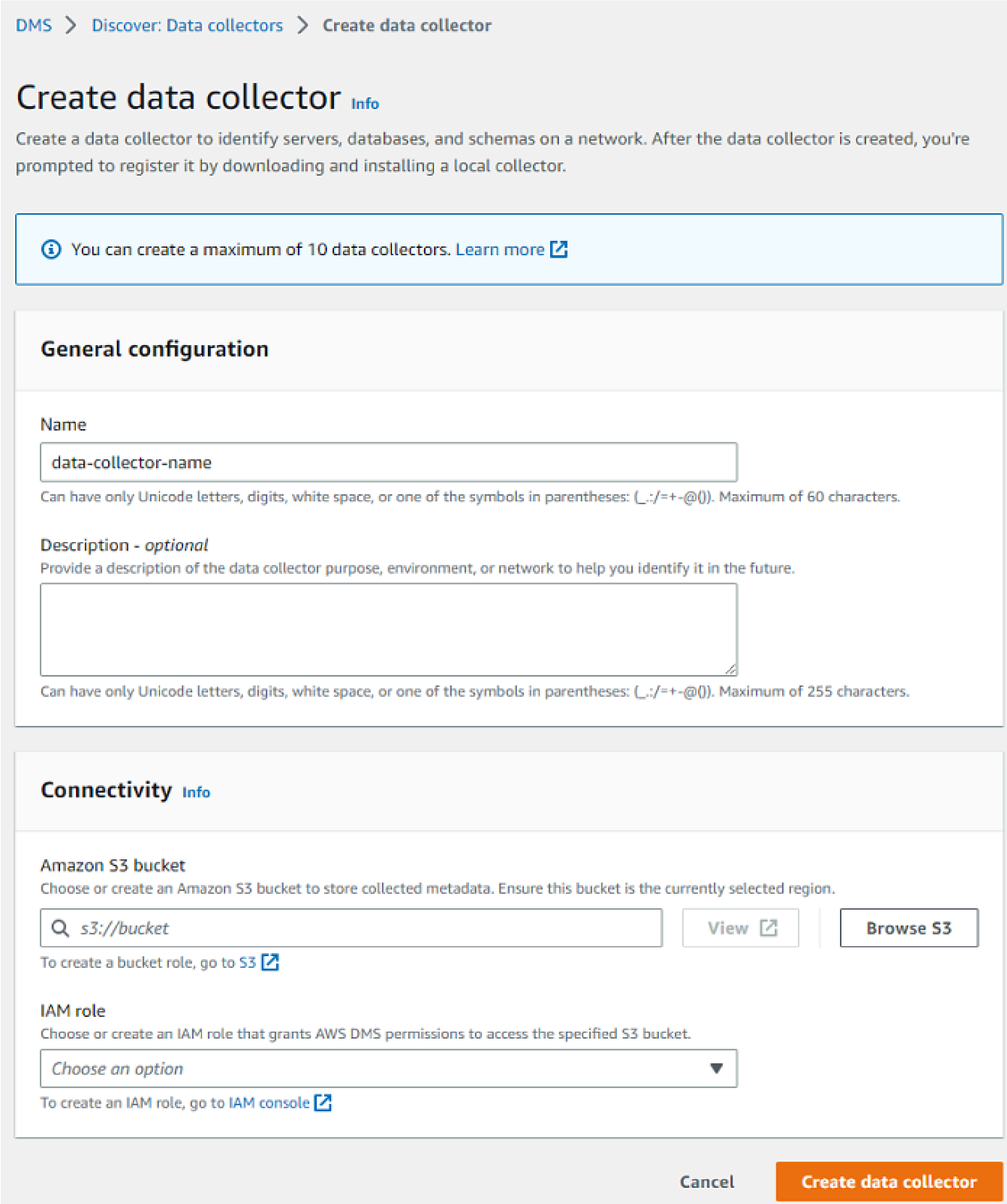
To create and download a data collector

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.

   Choose the Region where you use the DMS Fleet Advisor.

2. In the navigation pane, choose **Data collectors** under **Discover**. The **Data collectors** page opens.

3. Choose **Create data collector**. The **Create data collector** page opens.



4. For **Name** in the **General configuration** section, enter a name of your data collector.

5. In the **Connectivity** section, choose **Browse S3**. Choose the Amazon S3 bucket that you preconfigured from the list that appears.

   AWS DMS stores your DMS Fleet Advisor inventory metadata in this S3 bucket. Make sure that your Amazon S3 bucket is in the same AWS Region where your AWS DMS Fleet Advisor is currently running.

   > **Note**
   >
   > DMS Fleet Advisor only supports SSE-S3 encrypted buckets.

6. In the list of IAM roles, choose the IAM role that you preconfigured from the list that appears. This role grants AWS DMS permissions to access the specified Amazon S3 bucket.
7. Choose **Create data collector**. The **Data collectors** page opens and the created data collector appears in the list.

   When creating your first data collector, AWS DMS configures an environment in your Amazon S3 bucket that formats data and stores attributes for use with DMS Fleet Advisor.
8. Choose **Download local collector** on the information banner to download your newly created data collector. A message informs you that the download is in progress. After the download has finished, you can access the `AWS_DMS_Collector_Installer_`*`version_number`*`.msi` file.

You can now install the DMS data collector on your client. For more information, see Installing and configuring a data collector in AWS DMS.

Installing and configuring a data collector in AWS DMS

> **Important**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

Learn how to install your DMS data collector, how to specify data forwarding credentials, and how to add an LDAP server to your project.

The following table describes hardware and software requirements for installing an DMS data collector.

Minimum	Recommended
Processor: 2 cores with CPU benchmark score greater than 8,000	**Processor**: 4 cores with CPU benchmark score greater than 11,000
RAM: 8 GB	**RAM**: 16 GB
Hard drive size: 256 GB	**Hard drive size**: 512 GB
Operating system: Microsoft Windows Server 2012 or higher	**Operating system**: Windows Server 2016 or higher

To install a data collector on a client on your network

1. Run the **.MSI** installer. The **AWS DMS Fleet Advisor Collector Setup Wizard** page appears.
2. Choose **Next**. The **End-user license agreement** appears.
3. Read and accept the **End-user license agreement**.
4. Choose **Next**. The **Destination folder** page appears.
5. Choose **Next** to install the data collector in the default directory.

   Or choose **Change** to enter another install directory. Then choose **Next**.
6. On the **Desktop shortcut** page, select the box to install an icon on your desktop.
7. Choose **Install**. The data collector is installed in the directory that you chose.
8. On the **Completed DMS Collector Setup Wizard** page, choose **Launch AWS DMS Collector**, and then choose **Finish**.

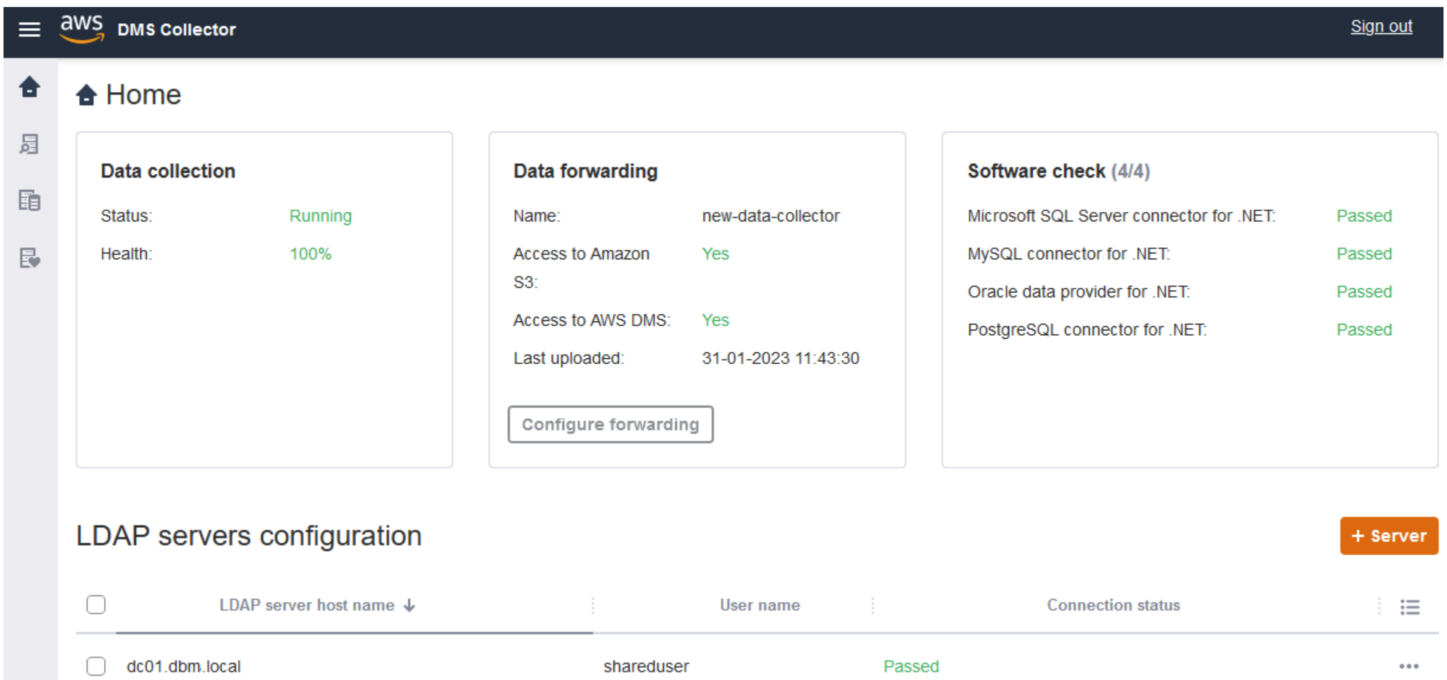
Your DMS data collector uses .NET libraries, connectors, and data providers to connect to your source databases. The DMS data collector installer automatically installs this required software for all supported databases on your server.

After you install the data collector, you can run it from a browser by entering **`http://localhost:11000/`** as the address. Optionally, from the Microsoft Windows **Start** menu, choose

AWS DMS Collector in the list of programs. When you first run DMS data collector, you are asked to configure credentials. Create the user name and password for signing in to the data collector.

On the DMS data collector home page, you can find information for preparing and running metadata collection, including the following status conditions:

- Status and health of your data collection.
- Accessibility to your Amazon S3 bucket and to AWS DMS so the data collector can forward data to AWS DMS.
- Connectivity to your installed database drivers.
- Credentials of an LDAP server to perform initial discovery.



DMS data collector uses an LDAP directory to gather information about the machines and database servers in your network. *Lightweight Directory Access Protocol (LDAP)* is an open standard application protocol. It's used for accessing and maintaining distributed directory information services over an IP network. You can add an existing LDAP server to your project for data collector that you can use for discovering information about the infrastructure of your systems. To do so, choose the **+Server** option, then specify a fully qualified domain name (FQDN) and the credentials for your domain controller. After adding the server, validate the connection check. To get started with the discovery process, see Discovering OS and database servers to monitor in AWS DMS.

Configuring credentials for data forwarding

After you install the data collector, make sure that this application can send the collected data to AWS DMS Fleet Advisor.

To configure credentials for data forwarding in AWS DMS Fleet Advisor

1. On the DMS data collector home page, in the **Data forwarding** section, choose **Configure forwarding**. The **Configure credentials for data forwarding** dialog box opens.
2. Choose the AWS Region where you intend to use DMS Fleet Advisor.
3. Enter your **AWS Access Key ID** and your **AWS Secret Access Key** obtained earlier when you created IAM resources. For more information, see Create IAM resources.
4. Choose **Browse data collectors**.

   If you haven't created a data collector in the specified Region yet, create a data collector before proceeding. For more information, see Creating a data collector.
5. In the **Choose data collector** window, select a data collector in the list and select **Choose**.
6. In the **Configure credentials for data forwarding** dialog box, choose **Save**.

On the **DMS Collector** home page, in the **Data forwarding** card, verify that the statuses of **Access to Amazon S3** and **Access to AWS DMS** are set to **Yes**.

If you see that the status of **Access to Amazon S3** or **Access to AWS DMS** is set to **No**, make sure that you created IAM resources for accessing Amazon S3 and DMS Fleet Advisor. After you create these IAM resources with all required permissions, configure data forwarding again. For more information, see Create IAM resources.

Discovering OS and database servers to monitor in AWS DMS

> **Important**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

You can use the DMS data collector to find and list all available servers in your network. Discovering all the available database servers in your network is recommended, but isn't required. Optionally, you can manually add or upload the list of servers for further data collection. For more information about manually adding a list of servers, see Managing monitored objects.

We recommend that you discover all operating system (OS) servers before discovering databases on those servers. To discover OS servers, you need permission to run remote PowerShell, Secure Shell (SSH), and Windows Management Instrumentation (WMI) scripts and commands, as well as access to the Windows registry. To discover database servers in your network and collect metadata from them, you need read-only administrator permissions for a remote database connection. Make sure that you added an LDAP server before you proceed with discovery. For more information, see Configuring credentials for data forwarding.

To get started with the DMS data collector, complete the following tasks:

- Discover all OS servers in your network.
- Add specific OS servers as objects to monitor.
- Verify connections for monitored OS servers.
- Discover Microsoft SQL Server, MySQL, Oracle, and PostgreSQL databases running on OS servers.
- Add database servers for data collection.
- Verify connections to the monitored databases.

To discover OS servers in your network that you can monitor

1. In the DMS data collector navigation pane, choose **Discovery**. To display the navigation pane, choose the menu icon in the upper-left corner of the DMS data collector home page.

   The **Discovery** page opens.
2. Make sure that the **OS servers** tab is selected, then choose **Run discovery**. The **Discovery parameters** dialog box appears.
3. Enter the LDAP servers that you want to use to scan your network.
4. Choose **Run discovery**. The page displays a list of all OS servers discovered within your network, regardless of whether they're running a database.

   We recommend that you run discovery for all OS servers before running discovery for databases on those servers. Your credentials make discovery possible first for the host servers, then for the databases that reside on them. You want to discover OS servers first before running discovery for databases on those servers. Be aware that the credentials that you use for an LDAP server to find OS servers in your network might differ from the credentials required to discover databases on a particular OS server. Therefore, we recommend that you add OS servers to monitored objects, verify the credentials and correct them if necessary, and then check connectivity before proceeding.

In the list of discovered OS servers in your network, you can now select the servers that you want to add to monitored objects.

To select OS servers as objects to monitor

1. On the **Discovery** page, choose the **OS servers** tab.
2. In the list of discovered OS servers shown, select the check box next to each server that you want to monitor.
3. Choose **Add to monitored objects**.

You can view the list of OS servers to monitor and verify connections on the **Monitor objects** page.

To verify connections of selected OS servers to monitor

1. In the DMS data collector navigation pane, choose **Monitored objects**.
2. On the **Monitored objects** page, choose the **OS servers** tab. A list of discovered OS servers to be monitored appears.
3. Select the check box at the top of the column to choose all the listed OS servers.
4. Choose **Actions**, then **Verify connection**. For each server object, view the results in the **Connections status** column.
5. Select servers with a connection status other than **Success**. Next, choose **Actions**, then choose **Edit**. The **Edit server** dialog box opens.
6. Verify that the information is correct or edit if needed. When finished, choose **Save**. The **Override credentials** dialog box opens.
7. Choose **Overwrite**. DMS data collector verifies and updates the status for each connection as **Success**.

You can now discover databases that reside on servers that you selected to monitor.

Discover databases running on servers

1. In the DMS data collector navigation pane, choose **Discovery**.
2. Choose the **Database servers** tab, and choose **Run discovery**. The **Discovery parameters** dialog box opens.
3. In the **Discovery parameters** dialog box, for **Discovery by**, choose **Monitored objects**. For **Servers**, choose the OS servers on which you want to run database discovery.
4. Choose **Run discovery**. The page displays a list of all databases that reside on the OS servers that you choose to monitor.

View information such as database address, server name, and database engine to help you select databases to monitor.

To select databases to monitor

1. On the **Discovery** page, choose the **Database servers** tab.
2. In the list of discovered databases shown, select the check box next to all the databases you want to monitor.
3. Choose **Add to monitored objects**.

You can now verify the connections to the databases you choose to monitor.

To verify connections to monitored databases

1. In the DMS data collector navigation pane, choose **Monitored objects**.
2. On the **Monitored objects** page, choose the **Database servers** tab. A list of discovered database servers you choose to monitor appears.
3. Select the check box at the top of the column to choose all the listed database servers.
4. Choose **Actions**, then choose **Verify connection**. For each database, view the results in the **Connections status** column.
5. Select connections that have undefined (blank) status or the status of **Failure**. Next, choose **Actions**, then choose **Edit**. The **Edit monitored objects** dialog box opens.
6. Enter your **Login** and **Password** credentials, then choose **Save**. The **Change credentials** dialog box opens.

7. Choose **Overwrite**. DMS data collector verifies and updates the status for each connection as **Success**.

After discovering OS servers and databases to monitor, you can also perform actions to manage monitored objects.

Managing monitored objects in AWS DMS

Important

End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

You can select objects to monitor when you run the server discovery process as described in Discovering OS and database servers. Also, you can manually manage objects, such as operating system (OS) servers and database servers. You can perform the following actions to manage monitored objects:

- Add new objects to monitor
- Remove existing objects
- Edit existing objects
- Export and import a list of objects to monitor
- Check connections to objects
- Start data collection

For example, you can manually add an object to monitor.

To add an object to monitor manually

1. On the **Monitored Objects** page, choose **+Server**. The **Add monitored object** dialog box opens.
2. Add information about the server, then choose **Save**.

You can also use a `.csv` file to import a large list of objects to monitor. Use the following `.csv` file format to import a list of objects to DMS data collector.

```
Hostname - Hostname or IP address of Monitored Object
Port - TCP port of Monitored Object
Engine: (one of the following)
        • Microsoft SQL Server
        • Microsoft Windows
        • Oracle Database
        • Linux
        • MySQL Server
        • PostgreSQL
Connection type: (one of the following)
        • Login/Password Authentication
        • Windows Authentication
        • Key-Based Authentication
Domain name:(Windows authentication)
        • Use domain name for the account
User name
Password
```

To import a .csv file with a list of objects to monitor

1. Choose **Import**. The **Import monitored objects** page opens.
2. Browse to the `.csv` file that you want to import, then choose **Next**.

You can view all of the objects and select those that you want to begin collecting metadata from.

Associating an OS server with a manually added database

DMS Fleet Advisor can't collect performance metrics directly from MySQL and PostgreSQL databases. To collect metrics required for target recommendations, DMS Fleet Advisor uses OS metrics where your databases run.

When you manually add MySQL and PostgreSQL databases to the list of monitored objects, DMS data collector can't identify the OS servers where these databases run. Because of this issue, you should associate your MySQL and PostgreSQL databases with OS servers.

You don't need to manually associate OS servers with databases that DMS Fleet Advisor has automatically discovered.

To associate an OS server with your database

1. In the DMS data collector navigation pane, choose **Monitored objects**.
2. On the **Monitored objects** page, choose the **Database servers** tab. A list of database servers appears.
3. Select the check box next to the MySQL or PostgreSQL database server that you added manually.
4. Choose **Actions**, then choose **Edit**. The **Edit database** dialog box opens.
5. If your DMS data collector has already discovered the OS server where this database runs, then choose **Auto detect**. DMS data collector runs a SQL script to automatically identify the OS server where your database runs. Then, DMS data collector associates this OS server with your database. Skip the next step and save the database configuration that you edited.

   If the DMS data collector can't automatically identify the OS server for your database, make sure that you use the correct credentials and provide database access permissions. Optionally, you can add the OS server manually.
6. To add your OS server manually, choose **+Add OS server**. The **Add host OS server** dialog box opens.

   Add information about your OS server, then choose **Save**.
7. In the **Edit database** dialog box, choose **Verify connection** to make sure that your DMS data collector can connect to the OS server.
8. After you verify the connection, choose **Save**.

If you change the associated OS server for your source database, then DMS Fleet Advisor uses the updated metrics to generate recommendations. However, the Amazon CloudWatch charts display the old data for your database server. For more information about CloudWatch charts, see Recommendation details.

Using SSL with AWS DMS Fleet Advisor

> **Important**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS

DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

To protect your data, AWS DMS Fleet Advisor can use SSL to access your databases.

Supported databases

AWS DMS Fleet Advisor supports using SSL to access following databases:

- Microsoft SQL Server
- MySQL
- PostgreSQL

Setting up SSL

To use SSL to access your database, configure your database server to support SSL. For more information, see the following documentation for your database:

- SQL Server: Enable encrypted connections to the Database Engine
- MySQL: Configuring MySQL to Use Encrypted Connections
- PostgreSQL: Secure TCP/IP Connections with SSL

To use SSL to connect to your database, select **Trust server certificate** and **Use SSL** when adding a server manually. For a MySQL database, you can use a custom certificate. To use a custom certificate, select the **Verify CA** check box. For information about adding a server, see Managing monitored objects.

Checking the Server Certificate Authority (CA) Certificate for SQL Server

If you want to validate your Server Certificate Authority (CA) Certificate for SQL Server, then clear **Trust server certificate** when you add the server. If your server uses a well-known CA, and the CA is installed by default on your OS, then verification should work normally. If DMS Fleet Advisor can't connect to your database server, install the CA certificate that your database server uses. For more information, see Configure client.

Collecting data for AWS DMS Fleet Advisor

> **Important**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

To start collecting data, select the objects on the **Monitored objects** page, and choose **Run data collection**. DMS data collector can collect from up to 100 databases at one time. Also, DMS data collector can use up to eight parallel threads to connect to databases in your environment. From these eight threads, DMS data collector can use up to five parallel threads to connect to a single database instance.

> **Important**
>
> Before starting to collect data, view the **Software check** section on the DMS data collector home page. Verify that all database engines that you want to monitor have the **Passed** status. If some database engines have the **Failed** status, and you have database servers with corresponding engines in your monitored objects list, fix the issue before proceeding. You can find tips next to the **Failed** status listed in the **Software check** section.

DMS data collector can work in two modes: single run or ongoing monitoring. After you start data collection, the **Run data collection** dialog box opens. Next, choose one of the two following options.

Metadata and database capacity

DMS data collector collects information from the database or OS servers. It includes schemas, versions, editions, CPU, memory, and disk capacity. DMS data collector also collects and provides metrics such as IOPS, I/O throughput and active datbase server connections. You can compute target recommendations in DMS Fleet Advisor based on this information. If the source database is over- or underprovisioned, then the target recommendations also will be over- or underprovisioned.

This is the default option.

Metadata, database capacity, and resource utilization

In addition to metadata and database capacity information, DMS data collector collects actual utilization metrics of CPU, memory, and disk capacity for the databases or OS servers. DMS data collector also collects and provides metrics such as IOPS, I/O throughput and active datbase server connections. Target recommendations provided will be more accurate because they are based on the actual database workloads.

If you choose this option, then you set the period of data collection. You can collect data during the **Next 7 days** or set the **Custom range** of 1–60 days.

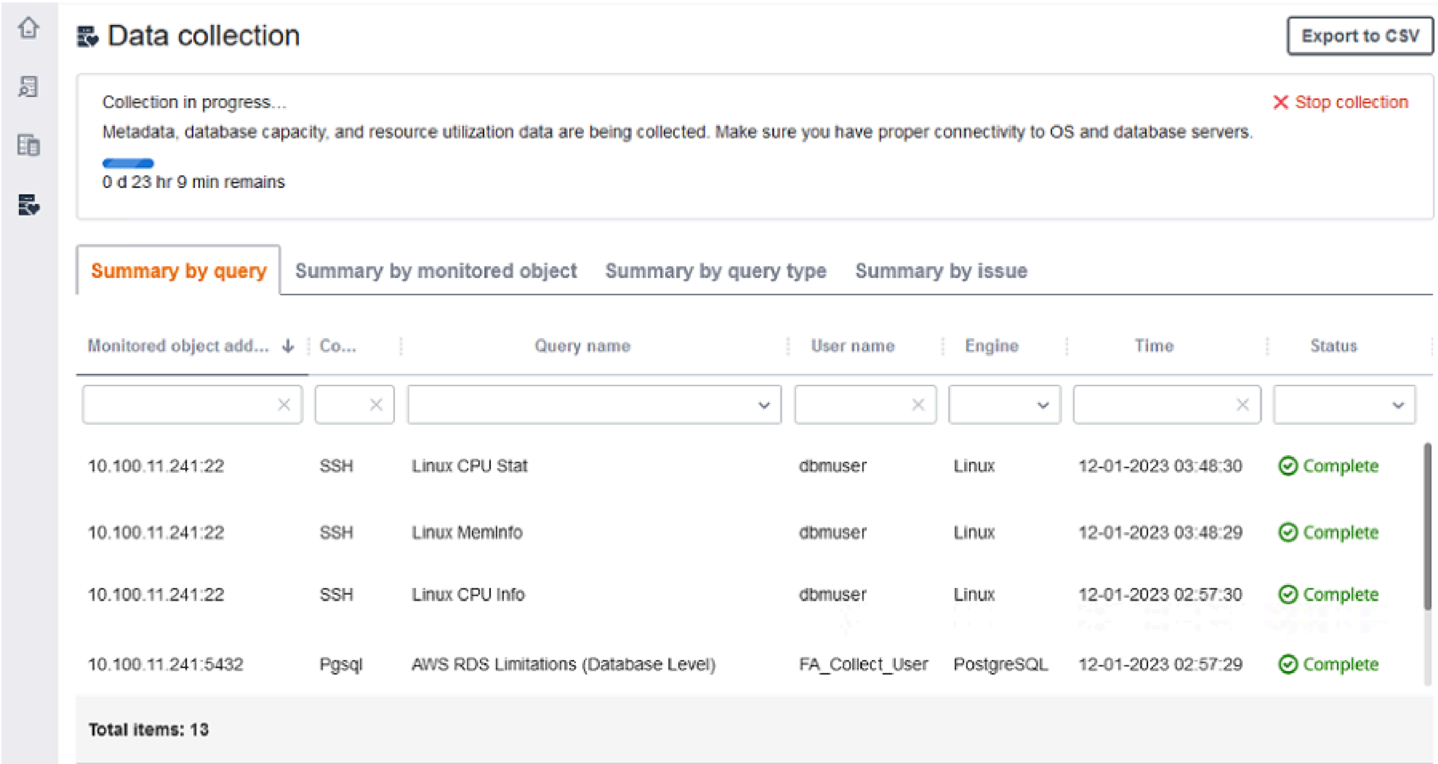
After data collection begins, you're redirected to the **Data collection** page, where you can see how the collection queries run and monitor the live progress. Here, you can view overall collection health or on the DMS data collector home page. If overall data collection health is less than 100 percent, you might need to fix issues related to the collection.

If you run the DMS data collector in the **Metadata and database capacity** mode, then you can see the number of completed queries on the **Data collection** page.

If you run the DMS data collector in the **Metadata, database capacity, and resource utilization** mode, then you can see the remaining time before your DMS data collector completes the monitoring.

On the **Data collection** page, you can see the collection status for each object. If something isn't working properly, a message appears showing how many issues occurred. To help determine a fix to an issue, you can check details. The following tabs list potential issues:

- **Summary by query** – Shows status of tests like the Ping test. You can filter results in the **Status** column. The **Status** column provides a message indicating how many failures occurred during data collection.
- **Summary by a monitored object** – Shows overall status per object.
- **Summary by query type** – Shows status for type of collector query, such as SQL, Secure Shell (SSH), or Windows Management Instrumentation (WMI) calls.
- **Summary by issue** – Shows all unique issues that occurred, with issue names and the number of times that each issue occurs.



To export the collection results, choose **Export to CSV**.

After identifying issues and resolving them, choose **Start collection** and rerun the data collection process. After performing data collection, the data collector uses secure connections to upload collected data to a DMS Fleet Advisor inventory. DMS Fleet Advisor stores information in your Amazon S3 bucket. For information about configuring credentials for data forwarding, see Configuring credentials for data forwarding.

Collecting capacity and resource utilization metrics with AWS DMS Fleet Advisor

You can collect metadata and performance metrics in two modes: single run or ongoing monitoring. Depending on the option that you select, your DMS data collector tracks different metrics in your data environment. During a single run, your DMS data collector tracks only metadata metrics from your database and OS servers. During ongoing monitoring, your DMS data collector tracks the actual utilization of your resources.

AWS DMS gathers the following metadata and metrics during a single run of your DMS data collector.

- Available memory on your OS servers
- Available storage on your OS servers

- Database version and edition
- Number of CPUs on your OS servers
- Number of schemas
- Number of stored procedures
- Number of tables
- Number of triggers
- Number of views
- Schema structure

DMS Fleet Advisor uses these metrics to build an inventory of your database and OS servers. Also, DMS Fleet Advisor uses these metadata and metrics to analyze your source database schemas.

DMS Fleet Advisor can generate target recommendations using the metrics collected during a single run of the data collector. However, in this case for your overprovisioned source databases, the target recommendation is also overprovisioned. Thus, you incur additional costs on the maintenance of your resources in the AWS Cloud. For underprovisioned source databases, the target recommendation is also underprovisioned, which might lead to performance issues. We recommend to collect the data using ongoing monitoring by choosing the metadata, database capacity, and resource utilization mode for the DMS data collector.

AWS DMS gathers the following metrics during ongoing monitoring. You can run your DMS data collector for a period of 1 to 60 days.

- I/O throughput on your database servers
- Input/output operations per second (IOPS) on your database servers
- Number of CPUs that your OS servers use
- Memory usage on your OS servers
- Number of active database and OS server connections

DMS Fleet Advisor uses these metrics to generate accurate target recommendations, so your target databases meet your performance needs. This can prevent additional cost incurred on the maintenance of your resources in the AWS Cloud.

How does the AWS DMS Fleet Advisor collect capacity and resource utilization metrics?

DMS Fleet Advisor collects performance metrics every minute.

For Oracle and SQL Server, DMS Fleet Advisor runs SQL queries to capture values for each database metric.

For MySQL and PostgreSQL, DMS Fleet Advisor collects performance metrics from the OS server where your database runs. In Windows, DMS Fleet Advisor runs WMI Query Language (WQL) scripts and receives WMI data. In Linux, DMS Fleet Advisor runs commands that capture the OS server metrics.

> **Important**
>
> Running remote SQL scripts can impact the performance of your production databases. However, the data collection queries don't contain any calculation logic. Thus, the data collection process isn't likely to use more than 1 percent of your database resources.

You can view all queries that the data collector runs to collect metrics. To do so, open the `DMSCollector.Collections.json` file. You can find this file in the `etc` folder located in the same folder where you installed the data collector. The default path is `C:\ProgramData\Amazon\AWS DMS Collector\etc\DMSCollector.Collections.json`.

The DMS data collector uses the local file system as the temporary storage for all collected data. The DMS data collector stores the collected data in JSON format. You can use the local collector in an offline mode and manually check or verify the collected files before you configure data forwarding. You can see all collected files in the `out` folder located in the same folder where you installed the DMS data collector. The default path is `C:\ProgramData\Amazon\AWS DMS Collector\out`.

> **Important**
>
> If you run your DMS data collector in an offline mode and store the collected data on your server for more that 14 days, then you can't use Amazon CloudWatch to display these metrics. However, DMS Fleet Advisor still uses this data to generate recommendations. For more information about CloudWatch charts, see Recommendation details.

You can also check or verify the collected data files in an online mode. The DMS data collector forwards all data to the Amazon S3 bucket that you specified in the DMS data collector settings.

You can use your DMS data collector to collect data from on-premises databases. Also, you can collect data from Amazon RDS and Aurora databases. However, you can't successfully run all DMS data collector queries in the cloud because of the differences between Amazon RDS or Aurora and on-premises DB instances. Because the DMS data collector gathers utilization metrics for MySQL and PostgreSQL databases from the host OS, this approach won't work with Amazon RDS and Aurora.

Troubleshooting for DMS data collector

> **Important**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

In the following list, you can find actions to take when you encounter specific issues while collecting data with your data collector.

Topics

Troubleshooting data collection issues related to network and server connections in AWS DMS

> **Important**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS

DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

If you run into issues related to network and server connections with the DMS data collector, try the following actions.

NET: An exception occurred during a ping request.

Check the name of the computer to see if it's in a state where it can't be resolved to an IP address.

For example, check if the computer is switched off, disconnected from the network, or decommissioned.

NET: Timed Out

Turn on the inbound firewall rule "File and Printer Sharing (Echo Request - ICMPv4-In)". For example:

```
* Inbound ICMPv4
```

NET: DestinationHostUnreachable

Check the IP address of the computer. Specifically, check if it's on the same subnet as the computer running DMS data collector and whether it responds to Address Resolution Protocol (ARP) requests.

If the computer is on a different subnet, then the IP address of the gateway can't be resolved to the media access control (MAC) address.

Also, check if the computer is switched off, disconnected from the network, or decommissioned.

Troubleshooting data collection issues related to Windows Management Instrumentation in AWS DMS

Important

End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS

DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see [AWS DMS Fleet Advisor end of support](#).

If you run into issues related to Windows Management Instrumentation with the DMS data collector, try the following actions.

WMI: The RPC server is unavailable. (Exception from HRESULT: 0x800706BA)

Turn on the inbound firewall rule "Windows Management Instrumentation (DCOM–In)". For example:

`* Inbound TCP/IP at local port 135.`

Also, turn on the inbound firewall rule "Windows Management Instrumentation (WMI–In)". For example:

`* Inbound TCP/IP at local port 49152 – 65535` for Windows Server 2008 and higher versions.

`* Inbound TCP/IP at local port 1025 – 5000` for Windows Server 2003 and lower versions.

WMI: Access is denied. (Exception from HRESULT: 0x80070005 (E_ACCESSDENIED))

Try the following:

- Add the DMS data collector user to the Windows group, Distributed COM Users or Administrators.
- Start the Windows Management Instrumentation service and set its start-up type to Automatic.
- Make sure that your DMS data collector user name is in the \ format.

WMI: Access denied

Add Remote Enable permission to the DMS data collector user on the root WMI namespace.

Use Advanced settings and make sure that permissions apply to "This namespace and subnamespaces."

WMI: The call was canceled by the message filter. (Exception from HRESULT: 0x80010002...)

Restart the Windows Management Instrumentation service.

Troublehsooting data collection issues related to Windows webpage composer

Important

End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

If you run into issues related to Windows webpage composer with the DMS data collector, try the following actions.

WPC: The network path was not found

Turn on the inbound firewall rule "File and Printer Sharing (SMB–In)". For example:

```
* Inbound TCP/IP at local port 445.
```

Also, start the Remote Registry service and set its start-up type to Automatic.

WPC: Access is denied

Add the DMS data collector user to the Performance Monitor Users or Administrators group.

WPC: Category does not exist

Run `loader /r` to rebuild the performance counter cache, then restart your computer.

Note

For information about troubleshooting issues when migrating data using AWS Database Migration Service (AWS DMS), see Troubleshooting and diagnostic support.

Troubleshooting data collection issues related to SSL in AWS DMS

Important

End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS

DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see [AWS DMS Fleet Advisor end of support](#).

If you run into issues related to SSL with the DMS data collector, try the following actions.

SSL errors

Your database requires a secure SSL connection, and you have not turned on the **Verify CA** and **Use SSL** options for the connection. Turn on these options and ensure that your local OS has the Certificate Authority installed that your database uses. For more information, see Setting up SSL.

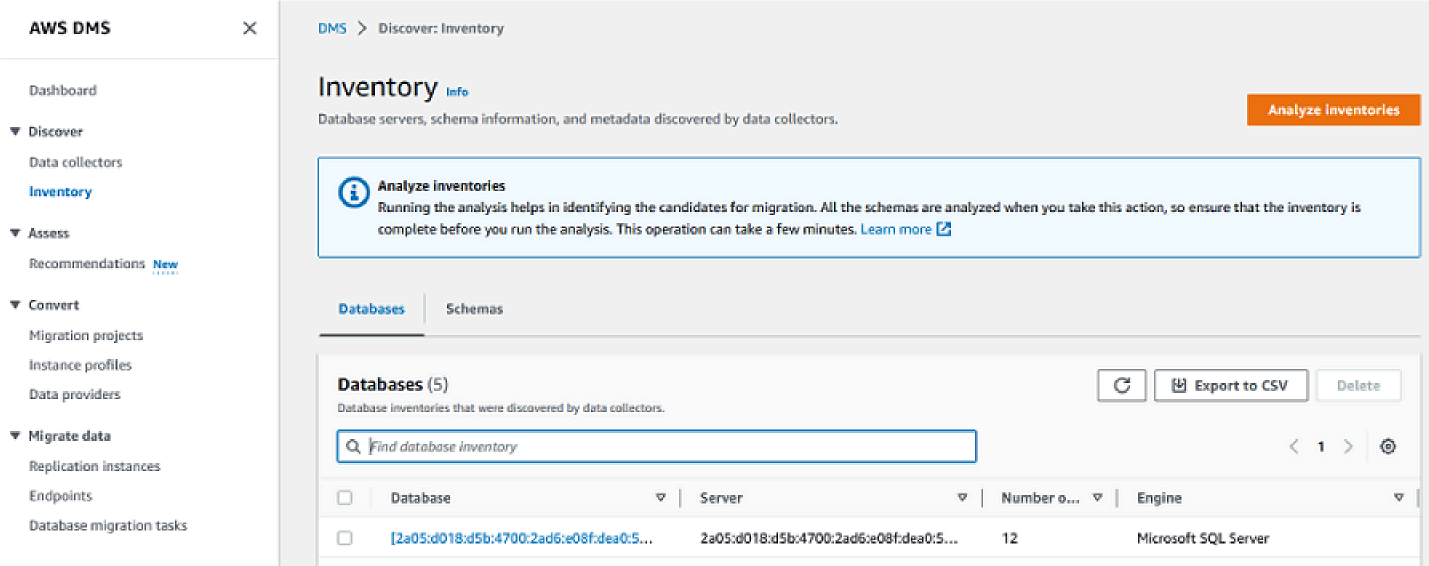
Using inventories for analysis in AWS DMS Fleet Advisor

Important

End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

To check the feasibility of potential database migrations, you can work with inventories of discovered databases and schemas. You can use the information in these inventories to understand which databases and schemas are good candidates for migration.

You can access database and schema inventories on the console. To do so, choose **Inventory** on the console.



DMS Fleet Advisor analyzes your database schemas to determine the similarity of different schemas. This analysis doesn't compare the actual code for objects. DMS Fleet Advisor compares only the names of schema objects, such as functions and procedures, to identify similar objects in different database schemas.

Topics

- Using a database inventory for analysis in AWS DMS
- Using a schema inventory for analysis in AWS DMS

Using a database inventory for analysis in AWS DMS

> **Important**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

To view a list of all databases on all the discovered servers within your network from which data was collected, use the following procedure.

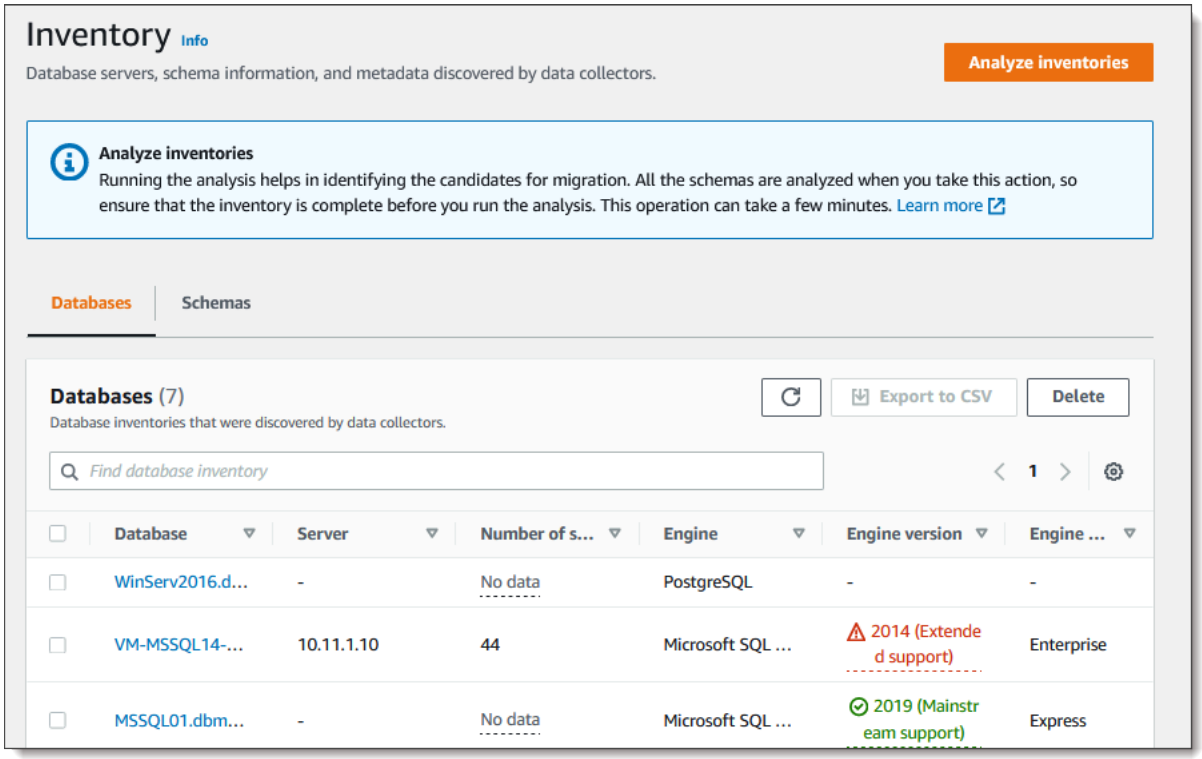
To view a list of databases on your network servers that data was collected from

1. Choose **Inventory** on the console.

   The **Inventory** page opens.

2. Choose the **Databases** tab.

   A list of discovered databases appears.



3. Choose **Analyze inventories** to determine schema properties, such as similarity and complexity. The amount of time the process takes depends on the number of objects to analyze, but it won't take more than one hour. Results from the analysis are found on the **Schemas** tab located on the **Inventory** page.

   DMS Fleet Advisor analyzes schemas across all discovered databases to define the intersection of their objects. The analysis result is expressed in percentage. DMS Fleet Advisor considers schemas with intersections of more than 50 percent as duplicates. Original schema is identified as the schema to which there are duplicates found. This helps to identify original schemas to convert or migrate first.

   The entire inventory is analyzed together to identify duplicate schemas.

Using a schema inventory for analysis in AWS DMS

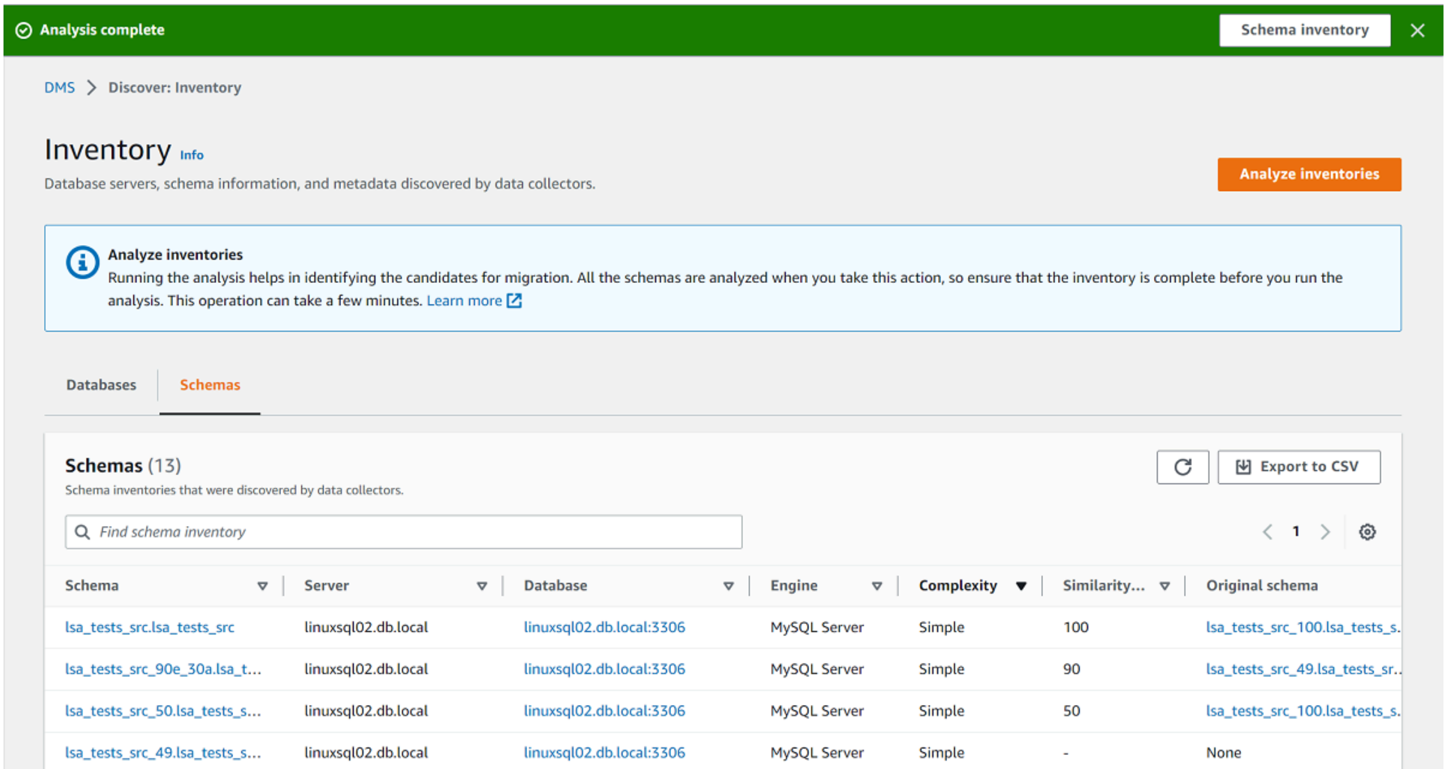
> **Important**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

You can view a list of database schemas discovered on servers within your network from which the data was collected. Perform the following procedure.

To view a list of schemas on your network servers that data was collected from

1. Choose **Inventory** on the console. The **Inventory** page opens.
2. Choose the **Schemas** tab. A list of schemas appears.
3. Select a schema in the list to view information about it, including server, database, size, and complexity.

   For each schema, you can view an object summary that provides information about object types, number of objects, object size, and lines of code.
4. (Optional) Choose **Analyze inventories** to identify duplicate schemas. DMS Fleet Advisor analyzes database schemas to define the intersection of their objects.
5. You can export inventory information to a `.csv` file for further review.



To identify schemas to migrate and determine the migration target, you can use AWS Schema Conversion Tool (AWS SCT) or DMS Schema Conversion. For more information, see Using a new project wizard in AWS SCT.

After you have identified schemas to migrate, you can convert schemas using AWS SCT or DMS Schema Conversion. For more information about DMS Schema Conversion, see Converting database schemas using DMS Schema Conversion.

Using the AWS DMS Fleet Advisor Target Recommendations feature

> **Important**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

To explore and choose an optimal migration target, you can generate target recommendations for your source on-premises databases in DMS Fleet Advisor. A *recommendation* includes one or more possible AWS target engines that you can choose for the migration of your source on-premises database. From these possible target engines, DMS Fleet Advisor suggests a single target engine as the right-sized migration destination and indicates this target as **DMS recommended**. To determine this right-sized migration destination, DMS Fleet Advisor uses the inventory metadata and metrics collected by your data collector.

You can use recommendations before the start of a migration to discover migration options, save costs, and reduce risks. You can export recommendations as a comma separated values (CSV) file, and share it with key stakeholders to facilitate decision making. You can export recommendations into the AWS Pricing Calculator to further optimize maintenance costs. For more information, see https://calculator.aws/#/.

You can't modify target recommendations in DMS Fleet Advisor. Thus, you can't use DMS Fleet Advisor for what-if analysis. *What-if analysis* is the process of changing the target parameters to see how those changes affect the pricing estimate of your recommendation. You can run what-if analysis in the AWS Pricing Calculator using the recommended target parameters as the starting point in the AWS Pricing Calculator. For more information, see https://calculator.aws/#/.

We recommend that you consider the DMS Fleet Advisor recommendation is a starting point in your migration planning. You can then decide to change the recommended instance parameters to optimize the cost or performance of your database workloads.

Topics

Recommended target instances

For target recommendations, DMS Fleet Advisor considers the following general purpose, memory-optimized, and burstable performance Amazon RDS DB instances.

- db.m5
- db.m6i
- db.r5
- db.r6i
- db.t3
- db.x1
- db.x1e
- db.z1d

For more information about Amazon RDS DB instance classes, see DB instance classes in the *Amazon RDS User Guide*.

How does DMS Fleet Advisor determine target instance specifications for the recommendation?

DMS Fleet Advisor can generate recommendations based on either database capacity or utilization.

- If you choose to generate the recommendation based on database capacity, then DMS Fleet Advisor maps the existing database capacity to the specifications of the closest instance class.
- If you choose to generate the recommendation based on resource utilization, then DMS Fleet Advisor determines the 95th percentile value for such metrics as CPU, memory, IO throughput, and IOPS. 95th percentile means that 95 percent of the collected data is lower than this value. Then, DMS Fleet Advisor maps these values to the specifications of the closest instance class.

To determine the size of the target database, DMS Fleet Advisor collects information about the size of your source database. Then, DMS Fleet Advisor recommends using the same size for the target storage. If your source database storage is overprovisioned, then the recommended size of the target storage will also be overprovisioned.

If you want to migrate data using AWS DMS, then you might need to increase IOPS provisioning for your target DB instance. When DMS Fleet Advisor generates target recommendations, the service considers only your source database metrics. DMS Fleet Advisor doesn't consider additional IOPS that you might need to run data migration tasks. For more information, see Migration tasks run slowly.

To estimate the IOPS costs, DMS Fleet Advisor uses a one-to-one mapping of your source IOPS usage as a baseline. DMS Fleet Advisor considers the peak load as the baseline value and 100% utilization for IOPS pricing.

For PostgreSQL and MySQL source databases, DMS Fleet Advisor can include Aurora and Amazon RDS DB instances in the target recommendations. If an Aurora configuration maps to the source requirements, then DMS Fleet Advisor marks this option as recommended.

Generating target recommendations with AWS DMS Fleet Advisor

> **Important**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see [AWS DMS Fleet Advisor end of support](#).

After you complete data collection and inventory of your database and analytics fleet, you can generate target recommendations in DMS Fleet Advisor. To do so, choose source databases and configure the settings that the DMS Fleet Advisor Target Recommendations feature uses to determine the size of target instances. Also, the DMS Fleet Advisor Target Recommendations feature uses the capacity and utilization metrics collected from your source databases.

To generate target recommendations

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.

   Make sure that you choose the AWS Region where you use the DMS Fleet Advisor.
2. In the navigation pane, choose **Recommendations** under **Assess**, and then choose **Generate recommendations**.
3. In the **Select source databases** panel, select the check boxes for the names of databases that you want to migrate to the AWS Cloud.

   For **Search source databases**, enter the name of your database to filter your inventory.

   DMS Fleet Advisor can generate recommendations for up to 100 databases at one time.
4. For **Availability and durability**, choose the preferred deployment option.

To calculate target recommendations for your production databases, choose **Production (Multi-AZ)**. DMS Fleet Advisor includes two DB instances in different Availability Zones in your target recommendation. This Multi-AZ deployment option provides high availability, data redundancy, and failover support.

If Aurora is the recommended target engine and **Availability and Durability** is a Multi-AZ deployment, the target recommendation includes a reader and writer DB instance.

To calculate target recommendations for databases that you use for development or testing, choose **Dev/Test (Single-AZ)**. DMS Fleet Advisor includes a single DB instance in your target recommendation. This Single-AZ deployment option reduces maintenance costs.

5. For **Target instance sizing**, choose the preferred option which DMS Fleet Advisor uses to calculate target recommendations.

   To calculate target recommendations based on your source database or OS server configuration, choose **Total capacity**. DMS Fleet Advisor uses such metrics as total CPU, memory, and disk capacity of your source databases or OS servers to generate target recommendations. Then, DMS Fleet Advisor maps your database capacity metrics to the specifications of the closest Amazon RDS DB instance class.

   To calculate target recommendations based on the actual utilization of your source database or OS server, choose **Resource utilization**. DMS Fleet Advisor uses utilization metrics of the CPU, memory, and disk capacity of your source databases or OS servers to generate target recommendations. From the utilization metrics, DMS Fleet Advisor computes the 95th percentile for each metric. 95th percentile means that 95 percent of the data within the period is lower than this value. Then, DMS Fleet Advisor maps these values to the closest Amazon RDS DB instance class.

   We recommend that you use the **Resource utilization** option for more accurate recommendations. To do so, make sure that you have collected the total capacity and resource utilization metrics.

6. Choose **Generate**.

DMS Fleet Advisor generates target recommendations for the selected databases. For successfully generated recommendations, DMS Fleet Advisor sets the status to **Computed**. Also, DMS Fleet Advisor uses the AWS Pricing Calculator to determine the estimated monthly cost for the recommended target DB instance. Now, you can explore the generated recommendations in detail. For more information, see Recommendation details.

To estimate the total monthly cost for your data inventory, select the check boxes for databases that you plan to move to the cloud. DMS Fleet Advisor displays the total estimated monthly cost and the summary of your target databases in the AWS Cloud. DMS Fleet Advisor uses the AWS Price List Query API to provide pricing details for your information only. Your actual fees depend on a variety of factors, including your actual usage of AWS services. For more information about AWS service pricing, see Cloud Services Pricing.

Exploring details of target recommendations with AWS DMS Fleet Advisor

Important

End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

After DMS Fleet Advisor generates target recommendations, you can view the key parameters of the recommended migration target in the **Recommendations** table. These key parameters include the target engine, instance class, number of virtual CPUs, memory, storage, and storage type. In addition to these parameters, DMS Fleet Advisor displays the estimated monthly cost of this recommended migration target.

Each recommendation might include one or more possible AWS target engines. If your recommendation includes several target engines, then AWS DMS marks one of them as recommended. Also, AWS DMS displays the parameters and estimated monthly cost for this recommended option in the **Recommendations** table.

To compare target recommendations to the utilization and capacity of your source database, explore your recommendations in detail. Also, you can view the migration limitations for a selected recommendation. These limitations include unsupported database features, action items, and other migration considerations.

To explore the recommendation in detail

1. Generate target recommendations with DMS Fleet Advisor. For more information, see Generating target recommendations.
2. Choose the recommendation name from the **Recommendations** table. The recommendation page opens.
3. If your recommendation includes more than one target options, then for **Target recommendations**, choose the target option.
4. Expand the **Source utilization and capacity** section. DMS Fleet Advisor displays resource utilization charts for the following metrics.

   - Number of CPUs
   - Memory
   - I/O throughput
   - Input/output operations per second (IOPS)
   - Storage
   - Number of active database server connections

   Use these charts to compare your source database metrics from your DMS data collector to the metrics of the selected target engine.

   If you can't see charts after you expand the **Source utilization and capacity** section, make sure that you granted your IAM user with permissions to view Amazon CloudWatch dashboards. For more information, see Using Amazon CloudWatch dashboards in the *Amazon CloudWatch User Guide*.
5. Choose the link with the name of your selected target engine. The **Target detail** page opens.
6. To export the target recommendations to CSV, choose **Export to CSV** option from the **Actions** dropdown.
7. To export the target recommendations to AWS Pricing Calculator, choose **Optimize cost with AWS Pricing Calculator** option from the **Actions** dropdown.
8. In the **Configuration** section, compare values of your source database parameters to the parameters of the target engine. For the target engine, DMS Fleet Advisor displays the estimated monthly costs for your cloud resources. DMS Fleet Advisor uses the AWS Price List Query API to provide pricing details for your information only. Your actual fees depend on a

variety of factors, including your actual usage of AWS services. For more information about AWS service pricing, see https://aws.amazon.com/pricing/Cloud Services Pricing.

9. In the **Migration limitations** section, view the migration limitations. We recommend that you consider these limitations when you migrate your source database to the AWS Cloud.

Exporting target recommendations with AWS DMS Fleet Advisor

> **Important**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

After you generate target recommendations, you can save a copy of the list of recommendations as a comma-separated value (CSV) file.

To generate target recommendations

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.

   Make sure that you choose the AWS Region where you use the DMS Fleet Advisor.
2. In the navigation pane, choose **Recommendations** under **Assess**, and then select the recommendations to include in your CSV file.
3. Choose **Export to CSV**, enter the file name and choose the folder on your PC where to save this file.
4. Open the CSV file.

The CSV file with recommendations contains the following information.

- **CreatedDate** – The date when DMS Fleet Advisor created the target engine recommendation.
- **DatabaseId** – The identifier of the source database for which DMS Fleet Advisor created this recommendation.
- **DeploymentOption** – The deployment option for the recommended Amazon RDS DB instance.

- **EngineEdition** – The recommended target Amazon RDS engine edition.
- **EngineName** – The name of the target engine.
- **InstanceMemory** – The amount of memory on the recommended Amazon RDS DB instance.
- **InstanceSizingType** – The size of your target instance.
- **InstanceType** – The recommended target Amazon RDS instance type.
- **InstanceVcpu** – The number of virtual CPUs on the recommended Amazon RDS DB instance.
- **Preferred** – A Boolean flag that indicates that this target option is recommended.
- **Status** – The status of the target engine recommendation.
- **StorageIops** – The number of I/O operations completed each second (IOPS) on the recommended Amazon RDS DB instance.
- **StorageSize** – The storage size of the recommended Amazon RDS DB instance.
- **StorageType** – The storage type of the recommended Amazon RDS DB instance.
- **WorkloadType** – The deployment option for your target engine such as Multi-AZ or Single-AZ deployment.

Discovering and analyzing migration limitations with AWS DMS Fleet Advisor

> **Important**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

You can use the DMS data collector to discover database features that your target engine doesn't support. To choose the right migration target, you should consider these limitations.

The DMS data collector discovers specific source database features. Then, DMS Fleet Advisor analyses source features from a migration standpoint to the specified target and provides additional information about the limitation and includes recommended actions to address or avoid this limitation. Also, DMS Fleet Advisor calculates the impact of these limitations.

The list of limitations is available in the **Target engine details** page. Navigate to this page from the **Recommendations** page in the left navigation menu. From the list of targets, choose the target engine to examine. The list of limitations is at the bottom of the page.

The following table includes MySQL database features that Amazon RDS for MySQL doesn't support.

Limitation	Description	Impact
Authentication plugins	Amazon RDS doesn't support MySQL authentication plugins.	Low
Error logging to the system log	Amazon RDS doesn't support writing the error log to the system log.	Low
Global transaction identifiers	You can use global transacti on identifiers with all RDS for MySQL 5.7 versions, and RDS for MySQL version 8.0.26 and higher MySQL 8.0 versions.	Low
Group Replication	Amazon RDS doesn't support the MySQL Group Replication plugin.	Low
InnoDB tablespace encryption	Amazon RDS doesn't support the InnoDB tablespace encryption.	Low
InnoDB reserved word	InnoDB is a reserved word for Amazon RDS for MySQL. You can't use this name for a MySQL database.	Low
Keyring plugin	Amazon RDS doesn't support the MySQL keyring plugin.	Low

Limitation	Description	Impact
Password validation plugin	Amazon RDS doesn't support the MySQL `validate_password` plugin.	Low
Persisted system variables	Amazon RDS doesn't support MySQL persisted system variables.	Low
Restricted access	Amazon RDS restricts access to certain system procedures and tables that require advanced permissions. Also, Amazon RDS doesn't allow direct host access to a DB instance by using Telnet, Secure Shell (SSH), or Windows Remote Desktop Connection.	Low
Rewriter query rewrite plugin	Amazon RDS doesn't support the MySQL Rewriter query rewrite plugin.	Low
Semisynchronous replication	Amazon RDS doesn't support the MySQL semisynchronous replication.	Low
Transportable Tablespaces	Amazon RDS doesn't support the MySQL Transportable Tablespaces feature.	Low
X Plugin	Amazon RDS doesn't support the MySQL X Plugin.	Low

The following table includes Oracle database features that Amazon RDS for Oracle doesn't support.

Limitation	Description	Impact
Active Data Guard	You can't use Active Data Guard with Oracle multitena nt container databases (CDB).	Medium
Automatic Storage Management	Amazon RDS doesn't support Oracle Automatic Storage Management (Oracle ASM).	Medium
Database Activity Streams	Amazon RDS doesn't support Oracle Database Activity Streams for the single-tenant architecture.	High
File size limit	The maximum size of a single file on RDS for Oracle DB instances is 16 TiB.	Medium
FTP and SFTP	Amazon RDS doesn't support FTP and SFTP.	Medium
Hybrid partitioned tables	Amazon RDS doesn't support Oracle hybrid partitioned tables.	High
Oracle Data Guard	Amazon RDS doesn't Oracle Data Guard for the single-te nant architecture.	High
Oracle Database Vault	Amazon RDS doesn't support Oracle Database Vault.	High
Oracle DBA privileges Vault	Amazon RDS has limitatio ns for Oracle DBA privilege s. For more information, see [Limitations for Oracle DBA privileges]().	High

Limitation	Description	Impact
Oracle Enterprise Manager	Amazon RDS doesn't Oracle Enterprise Manager for the single-tenant architecture.	High
Oracle Enterprise Manager Agent	Amazon RDS doesn't Oracle Enterprise Manager Agent for the single-tenant architect ure.	Medium
Oracle Enterprise Manager Cloud Control Management Repository	You can't use an Amazon RDS for Oracle DB instance for Oracle Enterprise Manager Cloud Control Management Repository.	High
Oracle Flashback Database	Amazon RDS doesn't support the Oracle Flashback Database feature.	High
Oracle Label Security	Amazon RDS doesn't support Oracle Label Security for the single-tenant architecture. You can use Oracle Label Security only with multitena nt container databases (Oracle CDB).	High
Oracle Messaging Gateway	Amazon RDS doesn't support Oracle Messaging Gateway.	High
Oracle Snapshot Standby databases	Amazon RDS doesn't support Oracle Snapshot Standby databases.	High

Limitation	Description	Impact
Public synonyms	Amazon RDS doesn't support public synonyms for Oracle-supplied schemas.	Medium
Schemas for unsupported features	Amazon RDS doesn't support schemas for Oracle features and components that require system privileges.	High
Pure unified auditing	Amazon RDS doesn't support the pure unified auditing. You can use the unified auditing in mixed mode.	Medium
Workspace Manager	Amazon RDS doesn't support the Oracle Database Workspace Manager `WMSYS` schema.	High

The following table includes PostgreSQL database features that Amazon RDS for PostgreSQL doesn't support.

Limitation	Description	Impact
Concurrent connections	The maximum number of concurrent connections to your RDS for PostgreSQ L instance is limited by the `max_connections` parameter.	Low
Newest versions	Amazon RDS doesn't apply major version upgrades automatically. To perform a major version upgrade,	Low

Limitation	Description	Impact
	modify your DB instance manually. For more informati on, see [Choosing a major version upgrade for PostgreSQL]().	
Reserved connections	Amazon RDS reserves up to 3 connections for system maintenance. If you specify a value for the user connectio ns parameter, add 3 to the number of connections that you expect to use.	Low
Supported extensions	RDS for PostgreSQL supports a limited number of extension s for the PostgreSQL database engine. You can find a list of supported extension s in the default DB parameter group for your PostgreSQ L version. You can also see the current extensions list using `psql` by showing the `rds.extensions` parameter.	Low
Tablespace splitting or isolation	You can't use tablespaces for I/O splitting or isolation . In RDS for PostgreSQL, all storage is on a single logical volume.	Low

The following table includes SQL Server database features that Amazon RDS for SQL Server doesn't support.

Limitation	Description	Impact
Backing up to Microsoft Azure Blob Storage	RDS for SQL Server doesn't support backing up to Microsoft Azure Blob Storage.	Medium
Buffer pool extension	RDS for SQL Server doesn't support the buffer pool extension.	High
Custom password policies	RDS for SQL Server doesn't support custom password policies.	Medium
Data Quality Services	RDS for SQL Server doesn't support SQL Server Data Quality Services (DQS).	High
Database Log Shipping	RDS for SQL Server doesn't support database Log Shipping.	High
Database names	Database names have the following limitations: can't start with rdsadmin; can't start or end with a space or a tab; can't contain any of the characters that create a new line; can't contain a single quotation mark (').	Medium
Database snapshots	RDS for SQL Server doesn't support database snapshots. You can use only DB instance snapshots in Amazon RDS.	Medium
Extended stored procedures	RDS for SQL Server doesn't support extended stored	High

Limitation	Description	Impact
	procedures, including `xp_cmdshell` .	
File tables	RDS for SQL Server doesn't support file tables.	Medium
FILESTREAM support	RDS for SQL Server doesn't provide FILESTREAM support.	Medium
Linked servers	RDS for SQL Server provides limited support for linked servers.	High
Machine Learning and R Services	RDS for SQL Server doesn't support Machine Learning and R Services because you need OS access to install these services.	High
Maintenance plans	RDS for SQL Server doesn't support maintenance plans.	High
Performance Data Collector	RDS for SQL Server doesn't support Performance Data Collector.	High
Policy-Based Management	RDS for SQL Server doesn't support Policy-Based Management.	Medium
PolyBase	RDS for SQL Server doesn't support PolyBase.	High
Replication	RDS for SQL Server doesn't support replication.	Medium

Limitation	Description	Impact
Resource Governor	RDS for SQL Server doesn't support Resource Governor.	High
Server-level triggers	RDS for SQL Server doesn't support server-level triggers.	Medium
Service Broker endpoints	RDS for SQL Server doesn't support Service Broker endpoints.	High
SSAS	Consider the limitations that apply to running SQL Server Analysis Services (SSAS) on RDS for SQL Server. For more information, see Limitations.	Low
SSIS	Consider the limitations that apply to running SQL Server Integration Services (SSIS) on RDS for SQL Server. For more information, see Limitations.	Low
SSRS	Consider the limitations that apply to running SQL Server Reporting Services (SSRS) on RDS for SQL Server. For more information, see Limitations.	Low

Limitation	Description	Impact
Storage size for SQL Server DB instances	The maximum storage size for SQL Server General Purpose (SSD) storage and Provisioned IOPS storage instances is 16 TiB. The maximum storage size for SQL Server Magnetic storage instances is 1 TiB.	High
Stretch Database	RDS for SQL Server doesn't support the SQL Server Stretch Database feature.	Medium
T-SQL endpoints	RDS for SQL Server doesn't support all operations that use `CREATE ENDPOINT`.	High
TRUSTWORTHY database property	RDS for SQL Server doesn't support the `TRUSTWORTHY` database property because it requires the `sysadmin` role.	Medium

The following table includes a list of recommendation issues. DMS Fleet Advisor analyses source and target database features and provides these migration limitations. The limitation with Blocker impact means that DMS Fleet Advisor can't generate target recommendations for the source database.

Limitation	Description	Impact
Appropriate instance is not found	AWS DMS can't find a target instance that can work as a right-sized migration destination for a combinati	Blocker

Limitation	Description	Impact
	on of your source database metrics.	
Appropriate instance is not found by IOPS	The source database uses a number of IOPS, which exceeds the maximum number of IOPS for the possible target DB instances.	Blocker
Appropriate instance is not found by RAM	The source database uses a number of GB of RAM, which exceeds the maximum size of RAM for the possible target DB instances.	Blocker
Appropriate instance is not found by storage size	The source database uses a number of TB of storage, which exceeds the maximum storage size for the possible target DB instances.	Blocker
Appropriate instance is not found by edition	The source database has an edition, which is not supported by Amazon RDS.	Blocker
Appropriate instance is not found by CPU cores	The source database has a number of CPU cores, which exceeds the maximum number of CPU cores for the possible target DB instances.	Blocker
Appropriate instance is not found by version	Your source database has version, which AWS DMS doesn't recognize.	Blocker

Limitation	Description	Impact
CPU parameter is undefined	The DMS data collector didn't collect information about the CPU that your source database uses. Make sure that you collected the required metrics and configured credentials for data forwardin g in your data collector. See Configuring credentials for data forwarding.	Blocker
Memory parameter is undefined	The DMS data collector didn't collect information about the memory that your source database uses. Make sure that you collected the required metrics and configured credentials for data forwardin g in your data collector. See Configuring credentials for data forwarding.	Blocker
Storage size parameter is undefined	The DMS data collector didn't collect information about the storage size that your source database uses. Make sure that you collected the required metrics and configured credentials for data forwardin g in your data collector. See Configuring credentials for data forwarding.	Blocker

Limitation	Description	Impact
Storage IOPS parameter is undefined	The DMS data collector didn't collect the storage IOPS metrics for your source database uses. Make sure that you collected the required metrics and configured credentials for data forwardin g in your data collector.	Blocker
Not Enough Data	The DMS data collector didn't collect enough data to generate a target recommend ation. Make sure that you configured credentials for data forwarding in your data collector. See Configuring credentials for data forwardin g.	Blocker
Database edition is undefined	The DMS data collector didn't collect information about your source database edition. Make sure that you collected the required metrics and configured credentials for data forwarding in your data collector. See Configuring credentials for data forwardin g.	Blocker
Unknown Error	DMS Fleet Advisor can't generate target recommend ations for your source database.	Blocker

Limitation	Description	Impact
Database version is undefined	The DMS Fleet Advisor didn't collect information about your source database version. DMS Fleet Advisor recommends that you use the latest database version for your source database. If you choose this recommend ation, then you must upgrade your database version. Review the generated target recommendations for your source database and make sure that these recommend ations meet your requireme nts.	High
Increase the number of database connections in RDS settings	Your source database requires certain number of connectio ns. By default, the number of available connections for Amazon RDS database instances is different. Make sure that you change this default value when you create your RDS database instance. To do so, update the `max_connections` parameter value in Parameter Groups.	Medium

Limitation	Description	Impact
Target edition is compatible	The target recommendation for your source database uses a different database edition. Your source database edition supports the same features as the recommended target edition. However, choosing this new database edition might increase your expenses.	Medium
Storage throughput parameter is undefined	The DMS data collector didn't collect the storage throughput metrics for your source database uses. Review the generated target recommendations for your source database and make sure that these recommend ations meet your requireme nts.	Medium
Database connection number parameter is undefined	The DMS data collector didn't collect information about the number of connections that your source database uses. Review the generated target recommendations for your source database and make sure that these recommend ations meet your requireme nts. Alternatively, request a quota increase.	Medium

Limitation	Description	Impact
Database downgrade version	Your source database runs on a higher version than the Amazon RDS database. To downgrade your database version, make sure that you don't use the features that aren't implemented in lower version. Alternatively, use Amazon EC2 as a migration target.	Medium
Target edition is different	The target recommendation for your source database uses a different database edition. Your source database edition is compatible with the recommended target edition. However, the recommend ed target database edition doesn't support some features of your source database edition. Choosing this new database edition might increase your expenses.	Medium

Limitation	Description	Impact
Upgrade from an unsupport ed version	Your source database has reached the end of support stage. To use the latest DB engine version as a target, upgrade your database before the migration. Alternati vely, use Amazon EC2 as a migration target. Depending on the database engine, use one of the following links for Learn more: Upgrading MySQL Upgrade SQL Server Upgrade OracleDB Upgrade PostgreSQL	Medium

Troubleshooting for target recommendations with AWS DMS Fleet Advisor

> **Important**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

In the following list, you can find actions to take when you encounter issues with the DMS Fleet Advisor Target Recommendations feature.

Topics

I can't see price estimates for target recommendations

If you see the **No data** for the **Estimated monthly cost** for a recommendation with a status of **Success**, then make sure that you granted your IAM user with permissions to access the AWS Price List Service API. To do so, you must create the policy that includes the `pricing:GetProducts` permission and add it to your IAM user as described in Create IAM resources.

DMS Fleet Advisor doesn't calculate the estimated monthly cost for recommendations with a status of **Failed**.

I can't see resource utilization charts

If you see the **Failed to load metrics** message after you expand the **Source utilization and capacity** section, then make sure that you granted your IAM user with permissions to view Amazon CloudWatch dashboards. To do so, you must add the required policy to your IAM user as described in Create IAM resources.

Alternatively, you can create a custom policy which includes the `cloudwatch:GetDashboard`, `cloudwatch:ListDashboards`, `cloudwatch:PutDashboard`, and `cloudwatch:DeleteDashboards` permissions. For more information, see Using Amazon CloudWatch dashboards in the *Amazon CloudWatch User Guide*.

I can't see the metrics collection status

If you see the **No data available** for **Metrics collection** when you choose **Generate recommendations**, then make sure that you collected data. For more information, see Collecting data for AWS DMS Fleet Advisor.

If you have this issue after you collected data, then make sure that you granted your IAM user with the `cloudwatch:Get*` permission to access Amazon CloudWatch. DMS Fleet Advisor uses a service-linked role to publish the collected database performance metrics to CloudWatch on your behalf. Make sure to create a service-linked role to use with DMS Fleet Advisor. For more information, see Create IAM resources.

Understanding DMS Fleet Advisor limitations

> **Important**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.

Limitations when using the DMS Fleet Advisor include the following:

- DMS Fleet Advisor generates one-to-one recommendations. For each source database, DMS Fleet Advisor determines a single target engine. DMS Fleet Advisor doesn't handle multitenant servers and doesn't provide recommendations for running several databases on a single target DB instance.
- DMS Fleet Advisor doesn't provide recommendations about available database version upgrades.
- DMS Fleet Advisor generates recommendations for up to 100 databases at one time.
- If you install DMS data collector, which is a Windows application, make sure that you also install .NET Framework 4.8 and PowerShell 6.0 and higher. For the hardware requirements, see Installing a data collector.
- The DMS data collector requires permissions to run requests using LDAP protocol on your domain server.
- The DMS data collector requires the sudo SSH script running in Linux.
- The DMS data collector requires permissions to run remote PowerShell, Windows Management Instrumentation (WMI), WMI Query Language (WQL), and registry scripts in Windows.
- For MySQL and PostgreSQL, DMS Fleet Advisor can't collect performance metrics from your database. Instead, DMS Fleet Advisor collects the OS server metrics. Therefore, you can't generate recommendations based on utilization metrics for MySQL and PostgreSQL databases that run on Amazon RDS and Aurora.

Converting database schemas using DMS Schema Conversion

Note

DMS Schema Conversion with generative AI feature is now available. For more information, see Viewing your database migration assessment report for DMS Schema Conversion and Converting database schemas in DMS Schema Conversion: step-by-step guide.

DMS Schema Conversion in AWS Database Migration Service (AWS DMS) makes database migrations between different types of databases more predictable. Use DMS Schema Conversion to assess the complexity of your migration for your source data provider, and to convert database schemas and code objects. You can then apply the converted code to your target database.

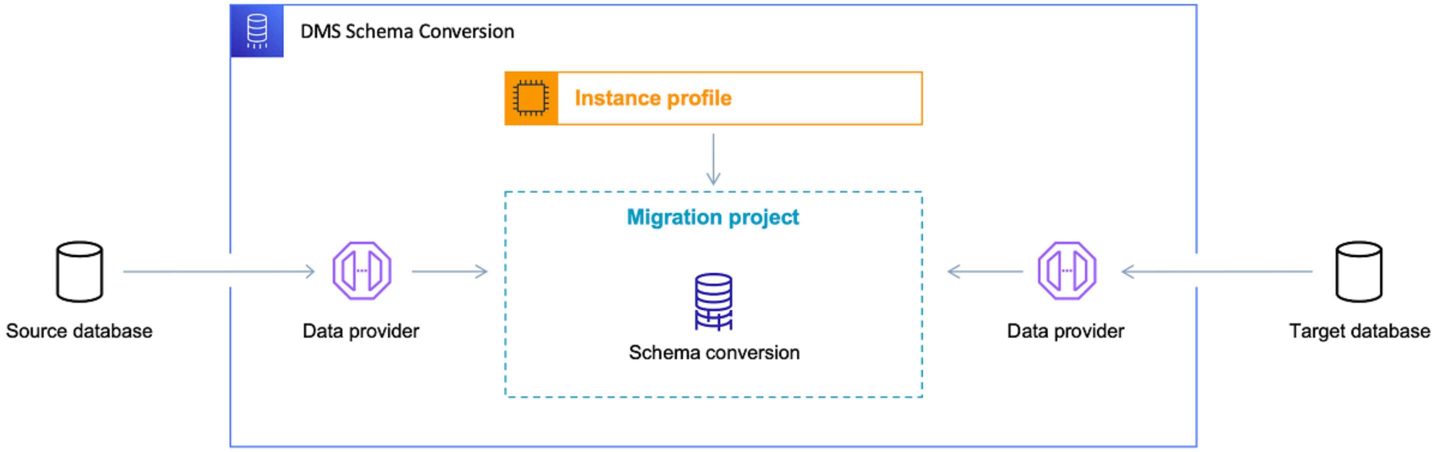
DMS Schema Conversion automatically converts your source database schemas and most of the database code objects to a format compatible with the target database. This conversion includes tables, views, stored procedures, functions, data types, synonyms, and so on. Any objects that DMS Schema Conversion can't convert automatically are clearly marked. To complete the migration, you can convert these objects manually.

At a high level, DMS Schema Conversion operates with the following three components: instance profiles, data providers, and migration projects. An *instance profile* specifies network and security settings. A *data provider* stores database connection credentials. A *migration project* contains data providers, an instance profile, and migration rules. AWS DMS uses data providers and an instance profile to design a process that converts database schemas and code objects.

For the list of supported source databases, see Sources for DMS Schema Conversion.

For the list of supported target databases, see Targets for DMS Schema Conversion.

The following diagram illustrates the DMS Schema Conversion process.



Use the following topics to better understand how to use DMS Schema Conversion.

Topics

Supported AWS Regions

You can create a DMS Schema Conversion migration project in the following AWS Regions. In other Regions, you can use the AWS Schema Conversion Tool. For more information about AWS SCT, see the AWS Schema Conversion Tool User Guide.

Region Name	Region
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Hyderabad)	ap-south-2
Asia Pacific (Tokyo)	ap-northeast-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Jakarta)	ap-southeast-3
Asia Pacific (Melbourne)	ap-southeast-4
Canada (Central)	ca-central-1
Canada West (Calgary)	ca-west-1
Europe (Frankfurt)	eu-central-1
Europe (Zurich)	eu-central-2
Europe (Stockholm)	eu-north-1
Europe (Milan)	eu-south-1
Europe (Spain)	eu-south-2
Europe (Ireland)	eu-west-1
Europe (Paris)	eu-west-3
Israel (Tel Aviv)	il-central-1

Region Name	Region
Middle East (UAE)	me-central-1
South America (São Paulo)	sa-east-1
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2

Schema conversion features

DMS Schema Conversion provides the following features:

- DMS Schema Conversion automatically manages the AWS Cloud resources that are required for your database migration project. These resources include instance profiles, data providers, and AWS Secrets Manager secrets. They also include AWS Identity and Access Management (IAM) roles, Amazon S3 buckets, and migration projects.
- You can use DMS Schema Conversion to connect to your source database, read the metadata, and create database migration assessment reports. You can then save the report to an Amazon S3 bucket. With these reports, you get a summary of your schema conversion tasks and the details for items that DMS Schema Conversion can't automatically convert to your target database. Database migration assessment reports help evaluate how much of your migration project DMS Schema Conversion can automate. These reports also help to estimate the amount of manual effort that is required to complete the conversion. For more information, see Creating database migration assessment reports with DMS Schema Conversion.
- After you connect to your source and target data providers, DMS Schema Conversion can convert your existing source database schemas to the target database engine. You can choose any schema item from your source database to convert. After you convert your database code in DMS Schema Conversion, you can review your source code and the converted code. You can save the converted SQL code to an Amazon S3 bucket.
- Before you convert your source database schemas, you can set up transformation rules. You can use transformation rules to change the data type of columns, move objects from one schema

to another, and change the names of objects. You can apply transformation rules to databases, schemas, tables, and columns. For more information, see Setting up transformation rules.

- You can change conversion settings to improve the performance of the converted code. These settings are specific for each conversion pair and depend on the features of the source database that you use in your code. For more information, see Specifying schema conversion settings.
- In some cases, DMS Schema Conversion can't convert source database features to equivalent Amazon RDS features. For these cases, DMS Schema Conversion creates an extension pack in your target database to emulate the features that weren't converted. For more information, see Using extension packs.
- You can apply the converted code and the extension pack schema to your target database. For more information, see Applying your converted code.
- DMS Schema Conversion supports all of the features in the latest AWS SCT release. For more information, see The latest release notes for AWS SCT .
- You can edit converted SQL code before DMS migrates it to the target database. For more information, see Editing and saving your converted SQL code.

Schema conversion limitations

DMS Schema Conversion is a web-version of the AWS Schema Conversion Tool (AWS SCT). DMS Schema Conversion supports fewer database platforms and provides more limited functionality compared to the AWS SCT desktop application. To convert data warehouse schemas, big data frameworks, application SQL code, and ETL processes, use AWS SCT. For more information about AWS SCT, see the AWS Schema Conversion Tool User Guide.

The following limitations apply when you use DMS Schema Conversion for database schema conversion:

- You can't save a migration project and use it in an offline mode.
- You can't edit SQL code for the source in a migration project for DMS Schema Conversion. To edit the SQL code of your source database, use your regular SQL editor. Choose **Refresh from database** to add the updated code in your migration project.
- Migration rules in DMS Schema Conversion don't support changing column collation. You can't use migration rules to move objects to a new schema.
- You can't apply filters to your source and target database trees to display only those database objects that meet the filter clause.

- DMS Schema Conversion extension pack doesn't include AWS Lambda functions that emulate email sending, job scheduling, and other features in your converted code.
- DMS Schema Conversion doesn't use customer-managed KMS keys for access to any customer AWS resources. For example, DMS Schema Conversion doesn't support using a customer-managed KMS key to access customer data in Amazon S3.

Getting started with DMS Schema Conversion

To get started with DMS Schema Conversion, use the following tutorial. In it, you can learn to set up DMS Schema Conversion, create a migration project, and connect to your data providers. Then, you can learn to assess the complexity of your migration, and convert your source database to a format compatible with your target database. In addition, you can learn to apply the converted code to your target database.

The following tutorial covers the prerequisite tasks and demonstrates the conversion of an Amazon RDS for SQL Server database to Amazon RDS for MySQL. You can use any of the supported source and target data providers. For more information, see Source data providers for DMS Schema Conversion.

For more information about DMS Schema Conversion, read the step-by-step migration walkthroughs for Oracle to PostgreSQL and SQL Server to MySQL migrations.

This video introduces the DMS Schema Conversion user interface and helps you get familiar with the core components of this service.

Topics

Complete prerequisites for working with DMS Schema Conversion

To set up DMS Schema Conversion, complete the following tasks. Then you can set up an instance profile, add data providers, and create a migration project.

Topics

Create a VPC based on Amazon VPC

In this step, you create a virtual private cloud (VPC) in your AWS account. This VPC is based on the Amazon Virtual Private Cloud (Amazon VPC) service and contains your AWS resources.

To create a VPC for DMS Schema Conversion

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. Choose **Create VPC**.
3. On the **Create VPC** page, enter the following settings:
   - **Resources to create – VPC and more**
   - **Name tag auto-generation** – Choose **Auto-generate** and enter a globally unique name. For example, enter **`sc-vpc`**.
   - **IPv4 CIDR block – `10.0.1.0/24`**
   - **NAT gateways – In 1 AZ**
   - **VPC endpoints – None**
4. Keep the rest of the settings as they are, and then choose **Create VPC**.
5. Choose **Subnets**, and take a note of your public and private subnet IDs.

   To connect to your Amazon RDS databases, create a subnet group that includes public subnets.

To connect to your on-premises databases, create a subnet group that includes private subnets. For more information, see Create an instance profile for DMS Schema Conversion.

6. Choose **NAT gateways**. Choose your **NAT gateway** and take a note of your **Elastic IP address**.

   Configure your network to make sure that AWS DMS can access your source on-premises database from this NAT gateway's public IP address. For more information, see Using an internet connection to a VPC.

Use this VPC when you create your instance profile and target databases on Amazon RDS.

Create an Amazon S3 bucket

To store information from your migration project, create an Amazon S3 bucket. DMS Schema Conversion uses this Amazon S3 bucket to save items such as assessment reports, converted SQL code, information about database schema objects, and so on.

To create an Amazon S3 bucket for DMS Schema Conversion

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose **Create bucket**.
3. On the **Create bucket** page, select a globally unique name for your S3 bucket. For example, enter **`sc-s3-bucket`**.
4. For **AWS Region**, choose your Region.
5. For **Bucket Versioning**, choose **Enable**.
6. Keep the rest of the settings as they are, and then choose **Create bucket**.

> **Note**
>
> DMS Schema Conversion (SC) works only with S3 buckets that use Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3). Other encryption configurations, including Server-Side Encryption with AWS KMS (SSE-KMS), Customer-Provided Keys (SSE-C), or Client-Side Encryption, are not currently supported by SC and prevents it from accessing the S3 bucket. If using a different encryption method, you must create a separate S3 bucket with SSE-S3 for use with DMS Schema Conversion.

Store database credentials in AWS Secrets Manager

Store your source and target database credentials in AWS Secrets Manager. Make sure that you replicate these secrets to your AWS Region. DMS Schema Conversion uses these secrets to connect to your databases in the migration project.

To store your database credentials in AWS Secrets Manager

1. Sign in to the AWS Management Console and open the AWS Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Choose **Store a new secret**.
3. The **Choose secret type** page opens. For **Secret type**, choose the type of database credentials to store:
   - **Credentials for Amazon RDS database** – Choose this option to store credentials for your Amazon RDS database. For **Credentials**, enter the credentials for your database. For **Database**, choose your database.
   - **Credentials for other database** – Choose this option to store credentials for your source Oracle or SQL Server databases. For **Credentials**, enter the credentials for your database.
   - **Other type of secret** – Choose this option to store only the user name and password to connect to your database. Choose **Add row** to add two key-value pairs. Make sure that you use **`username`** and **`password`** for key names. For values related to these keys, enter the credentials for your database.
4. For **Encryption key**, choose the AWS KMS key that Secrets Manager uses to encrypt the secret value. Choose **Next**.
5. On the **Configure secret** page, enter a descriptive **Secret name**. For example, enter **`sc-source-secret`** or **`sc-target-secret`**.
6. Choose **Replicate secret** and then for **AWS Region** choose your Region. Choose **Next**.
7. On the **Configure rotation** page, choose **Next**.
8. On the **Review** page, review your secret details, and then choose **Store**.

To store credentials for your source and target databases, repeat these steps.

Create IAM policies

To create an IAM policy for DMS Schema Conversion to access Amazon S3

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. On the **Select service** page, choose **Amazon S3** from the list.
5. In the **Actions allowed**, choose `PutObject`, `GetObject`, `GetObjectVersion`, `GetBucketVersioning`, `GetBucketLocation`, `ListBucket`.
6. In the **Resources** specify the ARN of the bucket that you created in the previous section. Choose **Next**.
7. On the **Review and create** page, enter a descriptive name. For example: `sc-s3-policy`. Then, choose **Create policy**.

To create an IAM policy for DMS Schema Conversionto access AWS Secrets Manager

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. On the **Select service** page, choose **Secrets Manager** from the list.
5. Choose **Next**. The **Add permissions** page opens.
6. In the **Actions allowed**, choose: `GetSecretValue` and `DescribeSecret`.
7. On the **Review and create** page, enter a descriptive name. For example: `sc-secrets-manager-policy`. Then, choose **Create policy**.

Create IAM roles

Create AWS Identity and Access Management (IAM) roles to use in your migration project. DMS Schema Conversion uses these IAM roles to access your Amazon S3 bucket and database credentials stored in AWS Secrets Manager.

To create an IAM role that provides access to your Amazon S3 bucket

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. On the **Select trusted entity** page, choose **AWS service**. Choose **DMS**.
5. Choose **Next**. The **Add permissions** page opens.
6. For **Filter policies**, enter **S3**. Choose the **sc-s3-policy** policy that you created in the previous section.
7. Choose **Next**. The **Name, review, and create** page opens.
8. For **Role name**, enter a descriptive name. For example, enter **`sc-s3-role`**. Choose **Create role**.
9. On the **Roles** page, enter **`sc-s3-role`** for **Role name**. Choose **sc-s3-role**.
10. On the **sc-s3-role page**, choose the **Trust relationships** tab. Choose **Edit trust policy**.
11. Thes AWS DMS regional service principal has the following format:

    ```
    dms.region-name.amazonaws.com
    ```

    Replace `region-name` with the name of your Region, such as `us-east-1`: The following code example shows the principal for the `us-east-1` Region:

    ```
    dms.us-east-1.amazonaws.com
    ```

    The following code example shows a trust policy for accessing AWS DMS schema conversion:

    JSON

    ```
    {
        "Version":"2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": "dms.us-east-1.amazonaws.com"
    ```

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

To create an IAM role that provides access to AWS Secrets Manager

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. On the **Select trusted entity** page, choose **AWS service**. Choose **DMS**.
5. Choose **Next**. The **Add permissions** page opens.
6. For **Filter policies**, enter `s3`. Choose the **sc-secrets-manager-policy** that you created in the previous section.
7. Choose **Next**. The **Name, review, and create** page opens.
8. For **Role name**, enter a descriptive name. For example, enter **`sc-secrets-manager-role`**. Choose **Create role**.
9. On the **Roles** page, enter **`sc-secrets-manager-role`** for **Role name**. Choose **sc-secrets-manager-role**.
10. On the **sc-secrets-manager-role** page, choose the **Trust relationships tab**. Choose **Edit trust policy**.
11. On the **Edit trust policy** page, edit the trust relationships for the role to use `schema-conversion.dms.amazonaws.com` and your AWS DMS regional service principal as the trusted entities. This AWS DMS regional service principal has the following format:

    ```
    dms.region-name.amazonaws.com
    ```

    Replace `region-name` with the name of your Region, such as `us-east-1`: The following code example shows the principal for the `us-east-1` Region:

    ```
    dms.us-east-1.amazonaws.com
    ```

    The following code example shows a trust policy for accessing AWS DMS schema conversion:

JSON

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

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 following IAM policy. Name the JSON file `dmsAssumeRolePolicyDocument.json`.

   JSON

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

   Then, 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 following IAM policy. Name the JSON file `dmsAssumeRolePolicyDocument2.json`.

   JSON

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

   Then, 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
```

Create an instance profile for DMS Schema Conversion

Before you create an instance profile, configure a subnet group for your instance profile. For more information about creating a subnet group for your AWS DMS migration project, see Creating a subnet group.

You can create an instance profile as described in the following procedure. In this instance profile, you specify network and security settings for your DMS Schema Conversion project.

To create an instance profile

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. In the navigation pane, choose **Instance profiles**, and then choose **Create instance profile**.
3. For **Name**, enter a unique name for your instance profile. For example, enter **`sc-instance`**.
4. For **Network type**, choose **IPv4** to create an instance profile that supports only IPv4 addressing. To create an instance profile that supports IPv4 and IPv6 addressing, choose **Dual-stack mode**.
5. For **Virtual private cloud (VPC)**, choose the VPC that you created in the prerequisites step.
6. For **Subnet group**, choose the subnet group for your instance profile. To connect to Amazon RDS databases, use a subnet group that includes public subnets. To connect to on-premises databases, use a subnet group that includes private subnets.
7. Choose **Create instance profile**.

To create a migration project, use this instance profile.

Configure your data providers for DMS Schema Conversion

Next, you create data providers that describe your source and target databases. For each data provider, you specify a data store type and location information. You don't store your database credentials in a data provider.

To create a data provider for an on-premises database

1. Sign in to the AWS Management Console, and open the AWS DMS console.
2. In the navigation pane, choose **Data providers**, and then choose **Create data provider**.
3. For **Name**, enter a unique name for your source data provider. For example, enter `sc-dp`.
4. For **Purpose**, select **Schema Conversion**.
5. For **Engine type**, choose the type of database engine for your data provider.
6. Turn on **Virtual mode** if you want to use schema conversion without connecting to a target database. For more information, see Virtual data provider.
7. If **Virtual Mode** is turned on, connection information will be preset automatically using defaults. Note that a virtual data provider can only be used as a target in a migration project or in schema conversion. For more information on virtual mode, see Virtual data provider.

   If **Virtual mode** is off, provide your connection information for the source database. The connection parameters depend on your database engine. For more information, see Creating data providers.
8. Select **Engine configuration**. Then select **Enter manually**. Provide your connection information for the source database. The connection parameters depend on your source database engine. For more information, see Creating data providers.
9. For **Secure Socket Layer (SSL) mode**, choose the type of SSL enforcement.
10. Choose **Create data provider**.

To create a data provider for an Amazon RDS database

1. Sign in to the AWS Management Console, and open the AWS DMS console.
2. In the navigation pane, choose **Data providers**, and then choose **Create data provider**.
3. For **Name**, enter a unique name for your source data provider. For example, enter `sc-dp`.
4. For **Purpose**, select **Schema Conversion**.
5. For **Engine type**, choose the type of database engine for your data provider.
6. Turn **Virtual Mode** on, if you intend to use this data provider for schema conversion without connecting to a target database. For more information on virtual mode, see Virtual data provider.
7. Select **Engine configuration**. Select **Choose RDS database instance from list**.

8. For **Database from RDS**, choose **Browse**, and choose your database. DMS Schema Conversion automatically retrieves the information about the engine type, server name, and port.
9. For **Database name**, enter the name of your database.
10. For **Secure Socket Layer (SSL) mode**, choose the type of SSL enforcement.
11. Choose **Create data provider**.

Create a migration project in DMS Schema Conversion

Now you can create a migration project. In the migration project, you specify your source and target data providers, and your instance profile.

To create a migration project

1. Choose **Migration projects**, and then choose **Create migration project**.
2. For **Name**, enter a unique name for your migration project. For example, enter **`sc-project`**.
3. For **Instance profile**, choose **`sc-instance`**.
4. For **Source**, choose **Browse**, and then choose **`sc-source`**.
5. For **Secret ID**, choose **`sc-source-secret`**.
6. For **IAM role**, choose **`sc-secrets-manager-role`**.
7. For **Target**, choose **Browse**, and then choose **`sc-target`**.
8. For **Secret ID**, choose **`sc-target-secret`**.
9. For **IAM role**, choose **`sc-secrets-manager-role`**.
10. Choose **Create migration project**.

Create an assessment report in DMS Schema Conversion

To assess the complexity of the migration, create the database migration assessment report. This report includes the list of all database objects that DMS Schema Conversion can't convert automatically.

To create an assessment report

1. Choose **Migration projects**, and then choose **`sc-project`**.
2. Choose **Schema conversion**, and then choose **Launch schema conversion**.

3. In the source database pane, choose the database schema to assess. Also, select the check box for the name of this schema.
4. In the source database pane, choose **Assess** in the **Actions** menu. The **Assess** dialog box appears.
5. Choose **Assess** in the dialog box to confirm your choice.

   The **Summary** tab shows the number of items that DMS Schema Conversion can automatically convert for database storage objects and database code objects.
6. Choose **Action items** to see the list of all database objects that DMS Schema Conversion can't convert automatically. Review the recommended actions for each item.
7. To save a copy of your assessment report, choose **Export results**. Next, choose one of the following formats: **CSV** or **PDF**. The **Export** dialog box appears.
8. Choose **Export** to confirm your choice.
9. Choose **S3 bucket**. The Amazon S3 console opens.
10. Choose **Download** to save your assessment report.

Convert your source code in DMS Schema Conversion

You can convert your source database schema using the following procedure. Then you can save the converted code as SQL scripts in a text file.

To convert your database schema

1. In the source database pane, choose the database schema to convert. Also, select the check box for the name of this schema.
2. In the source database pane, choose **Convert** in the **Actions** menu. The **Convert** dialog box appears.
3. Choose **Convert** in the dialog box to confirm your choice.
4. Choose a database object in the source database pane. DMS Schema Conversion displays the source code and the converted code for this object. You can edit the converted SQL code for a database object using the Edit SQL feature. For more information, see Editing and saving your converted SQL code.
5. In the target database pane, choose the converted database schema. Also, select the check box for the name of this schema.
6. For **Actions**, choose **Save as SQL**. The **Save** dialog box appears.

7. Choose **Save as SQL** to confirm your choice.
8. Choose **S3 bucket**. The Amazon S3 console opens.
9. Choose **Download** to save your SQL scripts.

Apply the converted code for DMS Schema Conversion

DMS Schema Conversion doesn't immediately apply the converted code to your target database. To update your target database, you can use the SQL scripts that you created in the previous step. Alternatively, use the following procedure to apply the converted code from DMS Schema Conversion.

To apply the converted code

1. In the target database pane, choose the converted database schema. Also, select the check box for the name of this schema.
2. For **Actions**, choose **Apply changes**. The **Apply changes** dialog box appears.
3. Choose **Apply** to confirm your choice.

Clean up and troubleshoot to work with DMS Schema Conversion

You can use Amazon CloudWatch to review or share your DMS Schema Conversion logs.

To review DMS Schema Conversion logs

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Logs**, **Log groups**.

   The name of your DMS Schema Conversion log group starts with `dms-tasks-sct`. You can sort the log groups by **Creation time** to find DMS Schema Conversion log group.

   Also, the name of your log group includes the Amazon Resource Name (ARN) of your migration project. You can see the ARN of your project on the **Migration projects** page in DMS Schema Conversion. Make sure that you choose **ARN** in **Preferences**.
3. Choose the name of your log group, and then choose the name of your log stream.
4. For **Actions**, choose **Export results** to save your DMS Schema Conversion log.

After you've finished your schema conversion in DMS Schema Conversion, clean up your resources.

To clean up your DMS Schema Conversion resources

1. Sign in to the AWS Management Console and open the AWS DMS console.
2. In the navigation pane, choose **Migration projects**.
   a. Choose **`sc-project`**.
   b. Choose **Schema conversion**, and then choose **Close schema conversion**.
   c. Choose **Delete** and confirm your choice.
3. In the navigation pane, choose **Instance profiles**.
   a. Choose **`sc-instance`**.
   b. Choose **Delete** and confirm your choice.
4. In the navigation pane, choose **Data providers**.
   a. Select **`sc-source`** and **`sc-target`**.
   b. Choose **Delete** and confirm your choice.

Also, make sure that you clean up other AWS resources that you created, such as your Amazon S3 bucket, database secrets in AWS Secrets Manager, IAM roles, and virtual private cloud (VPC).

Setting up a network for DMS Schema Conversion

DMS Schema Conversion creates a schema conversion instance in a virtual private cloud (VPC) based on the Amazon VPC service. When you create your instance profile, you specify the VPC to use. You can use your default VPC for your account and AWS Region, or you can create a new VPC.

You can use different network configurations to set up interaction for your source and target databases with DMS Schema Conversion. These configurations depend on the location of your source data provider and your network settings. The following topics provide descriptions of common network configurations.

Topics

- Using a single VPC for source and target data providers
- Using multiple VPCs for source and target data providers
- Using Direct Connect or a VPN to configure a network to a VPC

- Using an internet connection to a VPC

Using a single VPC for source and target data providers

The simplest network configuration for DMS Schema Conversion is a single VPC configuration. Here, your source data provider, instance profile, and the target data provider are all located in the same VPC. You can use this configuration to convert your source database on an Amazon EC2 instance.

To use this configuration, make sure that the VPC security group used by the instance profile has access to the data providers. For example, you can allow either a VPC Classless Inter-Domain Routing (CIDR) range or the Elastic IP address for your Network Address Translation (NAT) gateway.

Using multiple VPCs for source and target data providers

If your source and target data providers are in different VPCs, you can create your instance profile in one of the VPCs. You can then link these two VPCs by using VPC peering. You can use this configuration to convert your source database on an Amazon EC2 instance.

A *VPC peering connection* is a networking connection between two VPCs that activates routing using the private IP address of each VPC\ 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*.

To implement VPC peering, follow the instructions in Work with VPC peering connections in the *Amazon VPC User Guide*. Make sure that the route table of one VPC contains the CIDR block of the other. For example, suppose that VPC A is using destination 10.0.0.0/16 and VPC B is using destination 172.31.0.0. In this case, the route table of VPC A should contain 172.31.0.0, and the route table of VPC B must contain 10.0.0.0/16. For more detailed information, see Update your route tables for VPC peering connection in the *Amazon VPC Peering Guide*.

Using Direct Connect or a VPN to configure a network to a VPC

Remote networks can connect to a VPC using several options, such as Direct Connect or a software or hardware VPN connection. You can use these options to integrate existing on-site services by extending an internal network into the AWS Cloud. You might integrate on-site services such as monitoring, authentication, security, data, or other systems. By using this type of network extension, you can seamlessly connect on-site services to resources hosted by AWS, such as a VPC. You can use this configuration to convert your source on-premises database.

In this configuration, the VPC security group must include a routing rule that sends traffic destined for a VPC CIDR range or specific IP address 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 your VPC CIDR range or security group into the NAT instance. For more information, see Create a Site-to-Site VPN connection in the *AWS Site-to-Site VPN User Guide*.

Using an internet connection to a VPC

If you don't use a VPN or Direct Connect to connect to AWS resources, you can use an internet connection. This configuration involves a private subnet in a VPC with an internet gateway. The gateway contains the target data provider and the instance profile. You can use this configuration to convert your source on-premises database.

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 data provider appears to come from the public IP address of your NAT gateway. For more information, see VPC Route Tables in the *Amazon VPC User Guide*.

Creating source data providers in DMS Schema Conversion

You can use a Microsoft SQL Server, Oracle, PostgreSQL, MySQL, IBM DB2 for z/OS and SAP ASE (Sybase ASE) database as a source data provider in migration projects for DMS Schema Conversion.

Make sure that you configure the network to permit interaction between your source data provider and DMS Schema Conversion. For more information, see Setting up a network for DMS Schema Conversion.

Topics

- Using a PostgreSQL database as a source in DMS Schema Conversion
- Using a MySQL database as a source in DMS Schema Conversion
- Using an IBM Db2 for z/OS database as a source in DMS Schema Conversion
- Using a SAP ASE (Sybase ASE) database as a source in AWS DMS Schema Conversion

Using a Microsoft SQL Server database as a source in DMS Schema Conversion

You can use SQL Server databases as a migration source in DMS Schema Conversion.

You can use DMS Schema Conversion to convert database code objects from SQL Server to the following targets:

- Aurora MySQL
- Aurora PostgreSQL
- RDS for MySQL
- RDS for PostgreSQL

For information about the supported SQL Server database versions, see Source data providers for DMS Schema Conversion.

For more information about using DMS Schema Conversion with a source SQL Server database, see the SQL Server to MySQL migration step-by-step walkthrough.

Privileges for Microsoft SQL Server as a source

View the following list of privileges required for Microsoft SQL Server as a source:

- VIEW DEFINITION
- VIEW DATABASE STATE

The `VIEW DEFINITION` privilege enables users that have public access to see object definitions. DMS Schema Conversion uses the `VIEW DATABASE STATE` privilege to check the features of the SQL Server Enterprise edition.

Repeat the grant for each database whose schema you are converting.

In addition, grant the following privileges on the master database:

- VIEW SERVER STATE
- VIEW ANY DEFINITION

DMS Schema Conversion uses the VIEW SERVER STATE privilege to collect server settings and configuration. Make sure that you grant the VIEW ANY DEFINITION privilege to view data providers.

To read information about Microsoft Analysis Services, run the following command on the master database.

```
EXEC master..sp_addsrvrolemember @loginame = N'<user_name>', @rolename = N'sysadmin'
```

In the preceding example, replace the *<user_name>* placeholder with the name of the user who you previously granted with the required privileges.

To read information about SQL Server Agent, add your user to the SQLAgentUser role. Run the following command on the msdb database.

```
EXEC sp_addrolemember <SQLAgentRole>, <user_name>;
```

In the preceding example, replace the *<SQLAgentRole>* placeholder with the name of the SQL Server Agent role. Then replace the *<user_name>* placeholder with the name of the user who you previously granted with the required privileges. For more information, see Adding a user to the SQLAgentUser role in the *Amazon RDS User Guide*.

To detect log shipping, grant the SELECT on dbo.log_shipping_primary_databases privilege on the msdb database.

To use the notification approach of the data definition language (DDL) replication, grant the RECEIVE ON *<schema_name>*.*<queue_name>* privilege on your source databases. In this example, replace the *<schema_name>* placeholder with the schema name of your database. Then, replace the *<queue_name>* placeholder with the name of a queue table.

Using an Oracle database as a source in DMS Schema Conversion

You can use Oracle databases as a migration source in DMS Schema Conversion.

To connect to your Oracle database, use the Oracle System ID (SID). To find the Oracle SID, submit the following query to your Oracle database:

```
SELECT sys_context('userenv','instance_name') AS SID FROM dual;
```

You can use DMS Schema Conversion to convert database code objects from Oracle Database to the following targets:

- Aurora MySQL
- Aurora PostgreSQL
- RDS for MySQL
- RDS for PostgreSQL

For information about the supported Oracle database versions, see Source data providers for DMS Schema Conversion.

For more information about using DMS Schema Conversion with a source Oracle database, see the Oracle to PostgreSQL migration step-by-step walkthrough.

Privileges for Oracle as a source

The following privileges are required for Oracle as a source:

- CONNECT
- SELECT_CATALOG_ROLE
- SELECT ANY DICTIONARY
- SELECT ON SYS.ARGUMENT$

Using an Oracle Data Warehouse database as a source in DMS Schema Conversion

You can use Oracle Data Warehouse databases as a migration source in DMS Schema Conversion to convert database code objects and application code to Amazon Redshift.

For information about supported Oracle database versions, see Source data providers for DMS Schema Conversion. For more information about using DMS Schema Conversion with a source Oracle database, see the Oracle to PostgreSQL migration step-by-step walkthrough.

Privileges for using an Oracle Data Warehouse database as a source

The following privileges are required for Oracle Data Warehouse as a source:

- CONNECT
- SELECT_CATALOG_ROLE
- SELECT ANY DICTIONARY

Oracle Data Warehouse to Amazon Redshift conversion settings

For information about editing DMS Schema Conversion settings, see Specifying schema conversion settings for migration projects.

Oracle Data Warehouse to Amazon Redshift conversion settings include the following:

- **Add comments in the converted code for the action items of selected severity and higher**: This setting limits the number of comments with action items in the converted code. DMS adds comments in the converted code for action items of the selected severity and higher.

  For example, to minimize the number of comments in your converted code, choose **Errors only**. To include comments for all action items in your converted code, choose **All messages**.
- **The maximum number of tables for the target Amazon Redshift cluster**: This setting sets the maximum number of tables that DMS can apply to your target Amazon Redshift cluster. Amazon Redshift has quotas that limit the use tables for different cluster node types. This setting supports the following values:
  - **Auto**: DMS determines the number of tables to apply to your target Amazon Redshift cluster depending on the node type.
  - **Set a value**: Set the number of tables manually.

  DMS converts all your source tables, even if the number of tables is more than your Amazon Redshift cluster can store. DMS stores the converted code in your project and doesn't apply it to the target database. If you reach the Amazon Redshift cluster quota for the tables when you apply the converted code, DMS displays a warning message. Also, DMS applies tables to your target Amazon Redshift cluster until the number of tables reaches the limit.

  For information about Amazon Redshift table quotas, see Quotas and limits in Amazon Redshift.
- **Use the UNION ALL view**: This setting lets you set the maximum number of target tables that DMS can create for a single source table.

Amazon Redshift doesn't support table partitioning. To emulate table partitioning and make queries run faster, DMS can migrate each partition of your source table to a separate table in Amazon Redshift. Then, DMS creates a view that includes data from all of the target tables it creates.

DMS automatically determines the number of partitions in your source table. Depending on the type of source table partitioning, this number can exceed the quota for the tables that you can apply to your Amazon Redshift cluster. To avoid reaching this quota, enter the maximum number of target tables that DMS can create for partitions of a single source table. The default option is 368 tables, which represents a partition for 366 days of a year, plus two tables for NO RANGE and UNKNOWN partitions.

- **Datetype format elements that you use in the Oracle code are similar to datetime format strings in Amazon Redshift**: Use this setting to convert data type formatting functions such as `TO_CHAR`, `TO_DATE`, and `TO_NUMBER` with datetime format elements that Amazon Redshift doesn't support. By default, DMS uses extension pack functions to emulate these unsupported format elements in the converted code.

  The datetime format model in Oracle includes more elements than the datetime format strings in Amazon Redshift. When your source code includes only datetime format elements that Amazon Redshift supports, set this value to avoid extension pack functions in the converted code. Avoiding the extension functions makes the converted code run faster.
- **Numeric format elements that you use in the Oracle code are similar to numeric format strings in Amazon Redshift**: Use this setting to convert numeric data type formatting functions that Amazon Redshift doesn't support. By default, DMS uses extension pack functions to emulate these unsupported format elements in the converted code.

  The numeric format model in Oracle includes more elements than the numeric format strings in Amazon Redshift. When your source code includes only numeric format elements that Amazon Redshift supports, set this value to avoid extension pack functions in the converted code. Avoiding the extension functions makes the converted code run faster.
- **Use the NVL function to emulate the behavior of Oracle LEAD and LAG functions**: If your source code doesn't use the default values for offset in the `LEAD` and `LAG` functions, DMS can emulate these functions with the `NVL` function. By default, DMS raises an action item for each use of the `LEAD` and `LAG` functions. Emulating these functions using `NVL` makes the converted code run faster.

- **Emulate the behavior of primary and unique keys**: Set this setting to cause DMS to emulate the behavior of primary and unique key constraints on the target Amazon Redshift cluster. Amazon Redshift doesn't enforce primary and unique key constraints, and uses them for informational purposes only. If your source code uses primary or unique key constraints, set this setting to ensure that DMS emulates their behavior.
- **Use compression encoding**: Set this setting to apply compression encoding to Amazon Redshift table columns. DMS assigns compression encoding automatically using the default Redshift algorithm. For information about compression encoding, see [Compression encodings](#) in the *Amazon Redshift Database Developer Guide*.

  Amazon Redshift doesn't apply compression by default to columns that are defined as sort and distribution keys. To apply compression to these columns, set **Use compression encoding for KEY columns**. You can only select this option when you set **Use compression encoding**.

Using a PostgreSQL database as a source in DMS Schema Conversion

You can use PostgreSQL databases as a migration source in DMS Schema Conversion.

You can use DMS Schema Conversion to convert database code objects from PostgreSQL database to the following targets:

- MySQL
- Aurora MySQL

The privileges required for PostgreSQL as a source are as follows:

- CONNECT ON DATABASE <database_name>
- USAGE ON SCHEMA <database_name>
- SELECT ON ALL TABLES IN SCHEMA <database_name>
- SELECT ON ALL SEQUENCES IN SCHEMA <database_name>

Using a MySQL database as a source in DMS Schema Conversion

You can use MySQL databases as a migration source in DMS Schema Conversion.

You can use DMS Schema Conversion to convert database code objects from MySQL Database to the following targets:

- PostgreSQL
- Aurora PostgreSQL

The privileges required for MySQL as a source are as follows:

- `SELECT ON *.*`
- `SHOW VIEW ON *.*`

MySQL to PostgreSQL conversion settings

For information about editing DMS Schema Conversion settings, see Specifying schema conversion settings for migration projects.

MySQL to PostgreSQL conversion settings include the following:

- **Comments in converted SQL code**: Set this setting to add comments in the converted code for the action items of the selected severity and higher.

  Valid values:

  - **Errors only**
  - **Errors and warnings**
  - **All messages**

Using an IBM Db2 for z/OS database as a source in DMS Schema Conversion

You can use an IBM Db2 for z/OS databases as a migration source in DMS Schema Conversion.

You can use DMS Schema Conversion to convert database code objects from Db2 for z/OS Database to the following targets:

- Amazon RDS for Db2

For more information regarding the supported IBM Db2 for z/OS database versions, see Source data providers for DMS Schema Conversion.

Prerequisites for IBM Db2 for z/OS as a source database

The IBM Db2 for z/OS version 12 function level 100 database version does not support most new capabilities of IBM Db2 for z/OS version 12. This database version provides support for fallback to Db2 version 11 and data sharing with Db2 version 11. To avoid the conversion of unsupported features of Db2 version 11, we recommend that you use an IBM Db2 for z/OS database function level 500 or higher as a source for AWS DMS SC.

You can use the following code example to check the version of your source IBM Db2 for z/OS database:

```
SELECT GETVARIABLE('SYSIBM.VERSION') as version FROM SYSIBM.SYSDUMMY1;
```

Ensure that this code returns version `DSN12015` or higher.

You can use the following code example to check the value of the `APPLICATION COMPATIBILITY` special register in your source IBM Db2 for z/OS database:

```
SELECT CURRENT APPLICATION COMPATIBILITY as version FROM SYSIBM.SYSDUMMY1;
```

Ensure that this code returns version `V12R1M500` or higher.

Privileges for IBM Db2 for z/OS as a source database

The privileges needed to connect to a Db2 for z/OS database and read system catalogs and tables are as follows:

```
SELECT ON SYSIBM.LOCATIONS
SELECT ON SYSIBM.SYSCHECKS
SELECT ON SYSIBM.SYSCOLUMNS
SELECT ON SYSIBM.SYSDATABASE
SELECT ON SYSIBM.SYSDATATYPES
SELECT ON SYSIBM.SYSDUMMY1
SELECT ON SYSIBM.SYSFOREIGNKEYS
SELECT ON SYSIBM.SYSINDEXES
SELECT ON SYSIBM.SYSKEYCOLUSE
SELECT ON SYSIBM.SYSKEYS
SELECT ON SYSIBM.SYSKEYTARGETS
SELECT ON SYSIBM.SYSJAROBJECTS
SELECT ON SYSIBM.SYSPACKAGE
```

```
SELECT ON SYSIBM.SYSPARMS
SELECT ON SYSIBM.SYSRELS
SELECT ON SYSIBM.SYSROUTINES
SELECT ON SYSIBM.SYSSEQUENCES
SELECT ON SYSIBM.SYSSEQUENCESDEP
SELECT ON SYSIBM.SYSSYNONYMS
SELECT ON SYSIBM.SYSTABCONST
SELECT ON SYSIBM.SYSTABLES
SELECT ON SYSIBM.SYSTABLESPACE
SELECT ON SYSIBM.SYSTRIGGERS
SELECT ON SYSIBM.SYSVARIABLES
SELECT ON SYSIBM.SYSVIEWS
```

Using a SAP ASE (Sybase ASE) database as a source in AWS DMS Schema Conversion

You can use SAP ASE (Sybase ASE) databases as a migration source in DMS Schema Conversion.

You can use DMS Schema Conversion to convert database code objects from SAP ASE (Sybase ASE) Database to the following targets:

- Aurora PostgreSQL
- RDS for PostgreSQL

For information about the supported SAP ASE (Sybase ASE) database versions, see Source data providers for DMS Schema Conversion

Privileges for SAP ASE (Sybase ASE) as a source database

The following privileges are required when using SAP ASE (Sybase ASE) as a source database:

- USE master
- select on dbo.spt_values
- select on asehostname

For each database to be migrated, grant the following privileges:

- USE db_name *(Replace db_name with the name of the database being migrated)*
- select on dbo.sysusers

- select on dbo.sysobjects
- select on dbo.sysindexes
- select on dbo.syscolumns
- select on dbo.sysreferences
- select on dbo.syscomments
- select on dbo.syspartitions
- select on dbo.syspartitionkeys
- select on dbo.sysconstraints
- select on dbo.systypes
- select on dbo.sysqueryplans

Creating and setting target data providers in DMS Schema Conversion

You can use MySQL, PostgreSQL, Amazon Redshift and Amazon RDS for Db2 databases as a target data provider in migration projects for DMS Schema Conversion.

Topics

Using a MySQL database as a target in DMS Schema Conversion

You can use MySQL databases as a migration target in DMS Schema Conversion.

For information about supported target databases, see Target data providers for DMS Schema Conversion.

Privileges for MySQL as a target

The following privileges are required for MySQL as a target:

- `CREATE ON *.*`
- `ALTER ON *.*`
- `DROP ON *.*`
- `INDEX ON *.*`
- `REFERENCES ON *.*`
- `SELECT ON *.*`
- `CREATE VIEW ON *.*`
- `SHOW VIEW ON *.*`
- `TRIGGER ON *.*`
- `CREATE ROUTINE ON *.*`
- `ALTER ROUTINE ON *.*`
- `EXECUTE ON *.*`
- `CREATE TEMPORARY TABLES ON *.*`
- `AWS_LAMBDA_ACCESS (Only valid for Aurora MySQL)`
- `INSERT, UPDATE ON AWS_ORACLE_EXT.*`
- `INSERT, UPDATE, DELETE ON AWS_ORACLE_EXT_DATA.*`
- `INSERT, UPDATE ON AWS_SQLSERVER_EXT.*`
- `INSERT, UPDATE, DELETE ON AWS_SQLSERVER_EXT_DATA.*`
- `CREATE TEMPORARY TABLES ON AWS_SQLSERVER_EXT_DATA.*`

You can use the following code example to create a database user and grant the privileges.

```
CREATE USER 'user_name' IDENTIFIED BY 'your_password';
GRANT CREATE ON *.* TO 'user_name';
GRANT ALTER ON *.* TO 'user_name';
GRANT DROP ON *.* TO 'user_name';
GRANT INDEX ON *.* TO 'user_name';
GRANT REFERENCES ON *.* TO 'user_name';
GRANT SELECT ON *.* TO 'user_name';
GRANT CREATE VIEW ON *.* TO 'user_name';
GRANT SHOW VIEW ON *.* TO 'user_name';
GRANT TRIGGER ON *.* TO 'user_name';
GRANT CREATE ROUTINE ON *.* TO 'user_name';
GRANT ALTER ROUTINE ON *.* TO 'user_name';
```

```
GRANT EXECUTE ON *.* TO 'user_name';
GRANT CREATE TEMPORARY TABLES ON *.* TO 'user_name';
GRANT AWS_LAMBDA_ACCESS TO 'user_name'; (Only valid for Aurora MySQL)
GRANT INSERT, UPDATE ON AWS_ORACLE_EXT.* TO 'user_name';
GRANT INSERT, UPDATE, DELETE ON AWS_ORACLE_EXT_DATA.* TO 'user_name';
GRANT INSERT, UPDATE ON AWS_SQLSERVER_EXT.* TO 'user_name';
GRANT INSERT, UPDATE, DELETE ON AWS_SQLSERVER_EXT_DATA.* TO 'user_name';
GRANT CREATE TEMPORARY TABLES ON AWS_SQLSERVER_EXT_DATA.* TO 'user_name';
```

In the preceding example, replace *user_name* with the name of your user. Then, replace *your_password* with a secure password.

To use Amazon RDS for MySQL or Aurora MySQL as a target, set the `lower_case_table_names` parameter to 1. This value means that the MySQL server handles identifiers of such object names as tables, indexes, triggers, and databases as case insensitive. If you have turned on binary logging in your target instance, then set the `log_bin_trust_function_creators` parameter to 1. In this case, you don't need to use the `DETERMINISTIC`, `READS SQL DATA` or `NO SQL` characteristics to create stored functions. To configure these parameters, create a new DB parameter group or modify an existing DB parameter group.

Using a PostgreSQL database as a target in DMS Schema Conversion

You can use PostgreSQL databases as a migration target in DMS Schema Conversion.

For information about supported target databases, see [Target data providers for DMS Schema Conversion](#).

Privileges for PostgreSQL as a target

To use PostgreSQL as a target, DMS Schema Conversion requires the `CREATE ON DATABASE` privilege. Create a user and grant this user with this privilege for each database that you want to use in migration project for DMS Schema Conversion.

To use Amazon RDS for PostgreSQL as a target, DMS Schema Conversion requires the `rds_superuser` role.

To use the converted public synonyms, change the database default search path using the following command.

```
ALTER DATABASE <db_name> SET SEARCH_PATH = "$user", public_synonyms, public;
```

In this example, replace the *<db_name>* placeholder with the name of your database.

In PostgreSQL, only the schema owner or a `superuser` can drop a schema. The owner can drop a schema and all objects that this schema includes, even if the owner of the schema doesn't own some of its objects.

When you use different users to convert and apply different schemas to your target database, you may encounter an error message when DMS Schema Conversion can't drop a schema. To avoid this error message, use the `superuser` role.

Using an Amazon Redshift cluster as a target in DMS Schema Conversion

You can use Amazon Redshift databases as a migration target in DMS Schema Conversion. For information about supported target databases, see Target data providers for DMS Schema Conversion.

Privileges for Amazon Redshift as a target

Using Amazon Redshift as a target for DMS Schema Conversion requires the following privileges:

- **CREATE ON DATABASE**: Allows DMS to create new schemas in the database.
- **CREATE ON SCHEMA**: Allows DMS to create objects in the database schema.
- **GRANT USAGE ON LANGUAGE**: Allows DMS to create new functions and procedures in the database.
- **GRANT SELECT ON ALL TABLES IN SCHEMA pg_catalog**: Provides the user system information about the Amazon Redshift cluster.
- **GRANT SELECT ON pg_class_info**: Provides the user information about the table distribution style.

You can use the following code example to create a database user and grant it permissions. Replace the example values with your values.

```
CREATE USER user_name PASSWORD your_password;
GRANT CREATE ON DATABASE db_name TO user_name;
GRANT CREATE ON SCHEMA schema_name TO user_name;
GRANT USAGE ON LANGUAGE plpythonu TO user_name;
```

```
GRANT USAGE ON LANGUAGE plpgsql TO user_name;
GRANT SELECT ON ALL TABLES IN SCHEMA pg_catalog TO user_name;
GRANT SELECT ON pg_class_info TO user_name;
GRANT SELECT ON sys_serverless_usage TO user_name;
GRANT SELECT ON pg_database_info TO user_name;
GRANT SELECT ON pg_statistic TO user_name;
```

Repeat the `GRANT CREATE ON SCHEMA` operation for each target schema where you will apply the converted code or migrate data.

You can apply an extension pack on your target Amazon Redshift database. An extension pack is an add-on module that emulates source database functions that are required when converting objects to Amazon Redshift. For more information, see Using extension packs in DMS Schema Conversion.

Using Amazon RDS for Db2 database as a target in DMS Schema Conversion

You can use Amazon RDS for Db2 databases as a migration target in DMS Schema Conversion.

For more information regarding supported target databases, see Target data providers for DMS Schema Conversion.

Privileges for Amazon RDS for Db2 as a target

To use Amazon RDS for Db2 as a target, DMS Schema Conversion requires the `master_user_role` role. For more information, see Amazon RDS for Db2 default roles in the *Amazon Relational Database Service User Guide*.

Virtual data provider

AWS Database Migration Service (DMS) offers Virtual Mode for data providers in schema conversion. This feature allows you to perform schema conversion without connecting to a target database, reducing infrastructure costs and providing flexibility for migration planning. With Virtual Mode, you can start conversion work immediately to plan optimal migration strategy before committing resources. You can evaluate compatibility, convert and review schema code, and even test different target options. Then you can connect to a database when you are ready. Virtual Mode supports all target databases compatible with AWS DMS Schema Conversion, including MySQL, PostgreSQL, Amazon Redshift, and Amazon RDS for Db2.

Create virtual data provider

To create a virtual data provider, simply enable Virtual Mode in the form for creating the data provider. For more information, see Configure your data providers for DMS Schema Conversion.

Virtual data provider usage

To use a virtual data provider, create a new migration project or modify an existing migration project. Then set the project's target data provider to the virtual data provider you've created.

To use a virtual provider for the Secret and IAM role for reading that secret's fields, use any secret that is granted to the IAM role to successfully setup the migration project.

After target provider is set, choose the **Schema conversion** tab. Then choose the **Launch schema conversion** button. Wait until the project is started.

You can now use schema conversion with the virtual target data provider just as you would with a real target data provider. Actions that require connection to a real target database will be disabled, but **Save as SQL** will be available for the target tree.

Transition from a virtual data provider to a real data provider

When you are ready to proceed with the actual migration, you can make the transition to a real data provider .

To transition from a virtual data provider to a real data provider, follow these steps.

1. Sign in to the AWS Management Console, and open the AWS DMS console.
2. In the navigation pane, choose **Data providers**. Then choose the data provider you wish to modify.
3. Go to the section **Associated migration projects** , to see all the migration projects using this virtual data provider.
4. Choose the first associated project.
5. Choose the **Schema conversion** tab.
6. Check the option **Details section - Status**.
7. If the status is **Open** and the button **Close schema conversion** is not greyed out, choose it and wait until the project is closed.

8. Repeat the previous steps for all remaining associated migration projects.
9. Confirm that there are no remaining open projects.
10. Return to the data provider and choose **Modify**.
11. Turn **Virtual Mode** off.
12. Fill the connection settings correctly. The connection parameters depend on your database engine. For more information, see Creating data providers.
13. Choose **Save changes**.

   Note

   After changes have been saved, it won't be possible to turn Virtual Mode back to on.

14. Return to the AWS DMS console.
15. In the navigation pane, choose **Migration Projects**. Then choose the migration project you want to change the data provider for.
16. Choose **Modify**.
17. Fill the Secret to be used to connect to target data provider. Use the correct secret, containing the credentials needed to connect to the database.
18. Fill the IAM role to use to read the target secret. Check that the IAM role specified here is correct, is granted read on the connection credentials secret, and is available to AWS DMS Schema Conversion service.
19. Choose **Save changes**.
20. Choose **Schema conversion** tab.
21. Choose **Launch schema conversion** button.
22. Wait until the project is started.

Important

Once you disable Virtual Mode and save this change the data provider, this action cannot be undone. You will be required to provide actual database connection details for the data provider, and working Secret and IAM role to access the Secret in the associated migration project.

On the very first start of the project containing the transited data provider, you will have conversion results. Only these actions will be available:

- **On node schemas** - Refresh from database.
- **On database objects** - Apply changes, then save as SQL.

The **Apply** action will apply converted objects to the real database.

The **Refresh from database** action loads database objects from the real database. Any unsaved conversion objects will be lost.

Managing migration projects in DMS Schema Conversion

After you create an instance profile and compatible data providers for schema conversion, create a migration project. For more information, see Creating migration projects.

To use this new project in DMS Schema Conversion, on the **Migration projects** page, choose your project from the list. Next, on the **Schema conversion** tab, choose **Launch schema conversion**.

The first launch of DMS Schema Conversion requires some setup. AWS Database Migration Service (AWS DMS) starts a schema conversion instance, which takes up to five minutes. This process also reads the metadata from the source and target databases. After a successful first launch, you can access DMS Schema Conversion faster.

Amazon terminates the schema conversion instance that your migration project uses in three days after you complete the project. You can retrieve your converted schema and assessment report from the Amazon S3 bucket that you use for DMS Schema Conversion.

Specifying migration project settings for DMS Schema Conversion

After you create your migration project and launch schema conversion, you can specify migration project settings. You can change conversion settings to improve the performance of converted code. Also, you can customize your schema conversion view.

Conversion settings depend on your source and target database platforms. For more information, see Creating source data providers and Creating and setting target data providers.

To specify what schemas and databases you want to see in the source and target database panes, use the tree view settings. You can hide empty schemas, empty databases, system databases, and user-defined databases or schemas.

To hide databases and schemas in tree view

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Migration projects**. The **Migration projects** page opens.
3. Choose your migration project, and on the **Schema conversion** tab choose **Launch schema conversion**.
4. Choose **Settings**. The **Settings** page opens.
5. In the **Tree view** section, do the following:

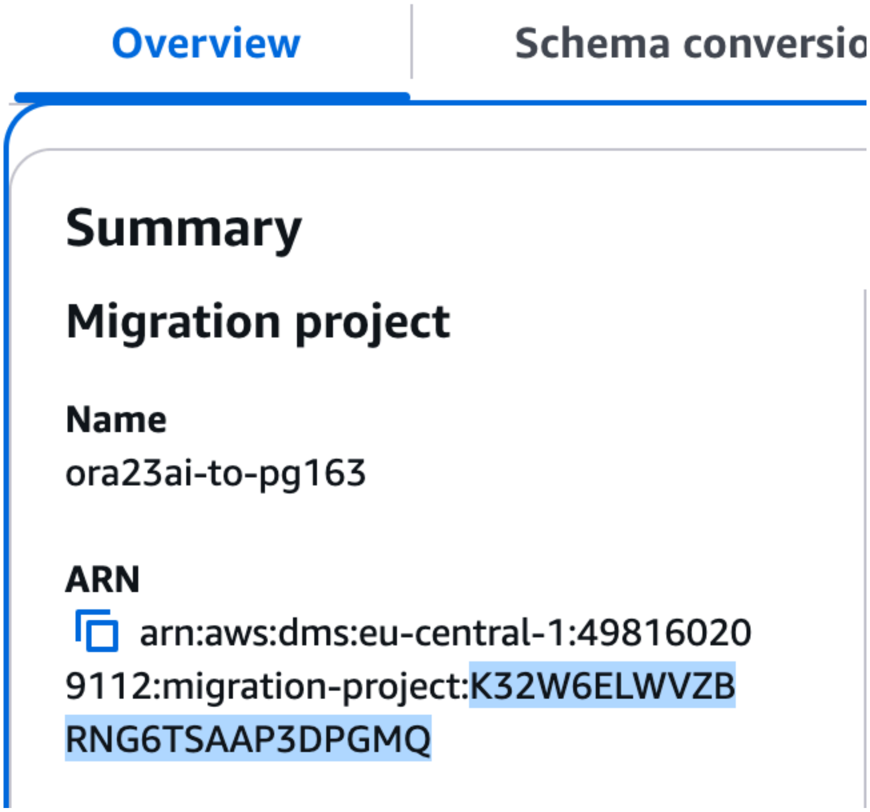
   - Choose **Hide empty schemas** to hide empty schemas.
   - Choose **Hide empty databases** to hide empty databases.
   - For **System databases or schemas**, choose system databases and schemas by name to hide them.
   - For **User-defined databases or schemas**, enter the names of user-defined databases and schemas that you want to hide. Choose **Add**. The names are case-insensitive.

     To add multiple databases or schemas, use a comma to separate their names. To add multiple objects with a similar name, use the percent (%) as a wildcard. This wildcard replaces any number of any symbols in the database or schema name.

   Repeat these steps for the **Source** and **Target** sections.
6. Choose **Apply**, and then choose **Schema conversion**.

Access logs for AWS DMS Schema Conversion

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/.
2. Choose **Migration projects**. The **Migration projects** page opens.
3. Choose your migration project, and on the **Overview** tab copy migration project id from the **ARN** field.



4. Open **CloudWatch** service.
5. Choose **Log groups** and enter `dms-tasks-sct-{migration_project_id}` where `{migration_project_id}` is the `id` from Step 3.
6. Inside the **Log group** you can find **Log stream** with logs.

Creating database migration assessment reports with DMS Schema Conversion

An important part of DMS Schema Conversion is the report that it generates to help you convert your schema. This *database migration assessment report* summarizes all of the schema conversion tasks. It also details the action items for schema that can't be converted to the DB engine of your target DB instance. You can view the report in the AWS DMS console or save a copy of this report as a PDF or comma-separated value (CSV) files.

The migration assessment report includes the following:

- An executive summary

- Recommendations, including conversion of server objects, backup suggestions, and linked server changes

When you have items that DMS Schema Conversion can't convert automatically, the report provides estimates showing how much effort is required to write the equivalent code for your target DB instance.

Topics

Creating a database migration assessment report for DMS Schema Conversion

After you create a migration project, use the following procedure to create a database migration assessment report.

To create a database migration assessment report

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Migration projects**. The **Migration projects** page opens.
3. Choose your migration project, and then choose **Schema conversion**.
4. Choose **Launch schema conversion**. The **Schema conversion** page opens.
5. In the source database pane, choose the database schema or schema items that you want to assess. To include multiple objects in the report, make sure that you select all items.
6. After you select the check boxes for all schema objects that you want to assess, you must choose the parent node for the selected objects. The **Actions** menu in the source database pane is now available.
7. Choose **Assess** in the **Actions** menu. A confirmation dialog box appears.
8. Choose **Assess** in the dialog box to confirm your choice.

Viewing your database migration assessment report for DMS Schema Conversion

After you create an assessment report, DMS Schema Conversion adds information in the following tabs:

- **Summary**
- **Action items**

The **Summary** tab shows the number of items that DMS Schema Conversion can automatically convert.

The **Action items** tab shows items that DMS Schema Conversion can't convert automatically, and provides recommendations about how to manage these items.

Assessment report summary

The **Summary** tab displays the summary information from the database migration assessment report. It shows the number of items that DMS Schema Conversion can automatically convert for database storage objects and database code objects.

In most cases, DMS Schema Conversion can't automatically convert all schema items to the target database engine. The **Summary** tab provides an estimate of the required effort to create schema items in your target DB instance that are equivalent to those in your source.

To see the conversion summary for database storage objects such as tables, sequences, constraints, data types, and so on, choose **Database storage objects**.

To see the conversion summary for database code objects such as procedures, functions, views, triggers, and so on, choose **Database code objects**.

To change the scope of the assessment report, select the required node in the source database tree. DMS Schema Conversion updates the assessment report summary to match the selected scope.

Assessment report action items

The **Action items** tab contains a list of items that DMS Schema Conversion can't automatically convert to a format compatible with the target database engine. For each action item, DMS

Schema Conversion provides the description of the issue and the recommended action. DMS Schema Conversion groups similar action items and displays the number of occurrences.

To view the code for the related database object, select an action item in the list.

Assessment report with generative AI

DMS Schema Conversion with generative AI streamlines the database migration process by offering recommendations to help you convert previously unconverted code objects that typically require complex manual conversion. Currently, this feature is available for Oracle to PostgreSQL/Aurora PostgreSQL and SQL Server to PostgreSQL/Aurora PostgreSQL conversions.

The assessment report which database code objects are candidates for conversion using generative AI. You can export this report to a PDF for a short list of five objects per action item. You can export the report to a CSV file to see the full list of candidates.

When you enable **Convert schema using generative AI** and perform a conversion, the updated assessment report identifies the objects that converted successfully using AI. These objects are clearly marked with a corresponding action item, making it easy to identify and track the AI-generated conversion recommendations.

To quickly locate objects that are converted by generative AI:

1. Navigate to the **Action Items** tab.
2. In the top-right corner, locate the drop-down menu.
3. Select **Generated by AI** from the options.

This filtering method allows you to efficiently view all database objects that were successfully converted using this feature.

Saving your database migration assessment report for DMS Schema Conversion

After you create a database migration assessment report, you can save a copy of this report as a PDF or comma-separated value (CSV) files.

To save a database migration assessment report as a PDF file

1. Choose **Export**, then choose **PDF**. Review the dialog box, and choose **Export to PDF**.

2. DMS Schema Conversion creates an archive with your PDF file and stores this archive in your Amazon S3 bucket. To change the Amazon S3 bucket, edit the schema conversion settings in your instance profile.
3. Open the assessment report file in your Amazon S3 bucket.

To save a database migration assessment report as CSV files

1. Choose **Export**, then choose **CSV**. Review the dialog box, and choose **Export to CSV**.
2. DMS Schema Conversion creates an archive with CSV files and stores this archive in your Amazon S3 bucket. To change the Amazon S3 bucket, edit the schema conversion settings in your instance profile.
3. Open the assessment report files in your Amazon S3 bucket.

The PDF file contains both the summary and action item information.

When you export your assessment report to CSV, DMS Schema Conversion creates three CSV files.

The first CSV file contains the following information about action items:

- Category
- Occurrence
- Action item
- Subject
- Group
- Description
- Documentation references
- Recommended action
- Line
- Position
- Source
- Target
- Server IP address and port
- Database
- Schema

The second CSV file includes the `Action_Items_Summary` suffix in its name and contains the following information:

- Schema
- Action item
- Number of occurrences
- Learning curve efforts, which is the amount of effort required to design an approach to converting each action item
- Efforts to convert an occurrence of the action item, which shows the effort required to convert each action item, following the designed approach
- Action item description
- Recommended action

The values that indicate the level of required efforts are based on a weighted scale, ranging from low (least) to high (most).

The third CSV file includes `Summary` in its name and contains the following information:

- Category
- Number of objects
- Objects automatically converted
- Objects with simple actions
- Objects with medium-complexity actions
- Objects with complex actions
- Total lines of code

Using DMS Schema Conversion

DMS Schema Conversion converts your existing database schemas and a majority of the database code objects to a format compatible with the target database.

DMS Schema Conversion automates much of the process of converting your online transaction processing (OLTP) database schemas to Amazon RDS for MySQL or RDS for PostgreSQL. The source and target database engines contain many different features and capabilities, and DMS Schema

Conversion attempts to create an equivalent schema wherever possible. For database objects where direct conversion isn't possible, DMS Schema Conversion provides a list of actions for you to take.

To convert your database schema, use the following process:

- Before you convert your database schemas, set up transformation rules that change the names of your database objects during conversion.
- Create a database migration assessment report to estimate the complexity of the migration. This report provides details about the schema elements that DMS Schema Conversion can't convert automatically.
- Convert your source database storage and code objects. DMS Schema Conversion creates a local version of the converted database objects. You can access these converted objects in your migration project.
- Save the converted code to SQL files to review, edit, or address conversion action items. Optionally, apply the converted code directly to your target database.

Topics

Setting up transformation rules in DMS Schema Conversion

Before you convert your database schema with DMS Schema Conversion, you can set up transformation rules. *Transformation rules* can do such things as change an object name to lowercase or uppercase, add or remove a prefix or suffix, and rename objects. For example, suppose that you have a set of tables in your source schema named `test_TABLE_NAME`. You can set up a rule that changes the prefix `test_` to the prefix `demo_` in the target schema.

You can create transformation rules that perform the following tasks:

- Add, remove, or replace a prefix
- Add, remove, or replace a suffix
- Change the data type of a column
- Change the object name to lowercase or uppercase
- Rename objects

You can create transformation rules for the following objects:

- Schema
- Table
- Column

Creating transformation rules

DMS Schema Conversion stores transformation rules as part of your migration project. You can set up transformation rules when you create your migration project, or edit them later.

You can add multiple transformation rules in your project. DMS Schema Conversion applies transformation rules during conversion in the same order as you added them.

To create transformation rules

1. On the **Create migration project** page, choose **Add transformation rule**. For more information, see [Creating migration projects](#).
2. For **Rule target**, choose the type of database objects to which this rule applies.
3. For **Source schema**, choose **Enter a schema**. Then, enter the names of your source schemas, tables, and columns to which this rule applies. You can enter an exact name to select one object, or you can enter a pattern to select multiple objects. Use the percent (%) as a wildcard to replace any number of any symbols in the database object name.
4. For **Action**, choose the task to perform.
5. Depending on the rule type, enter one or two additional values. For example, to rename an object, enter the new name of the object. To replace a prefix, enter the old prefix and the new prefix.
6. Choose **Add transformation rule** to add another transformation rule.

   After you are done adding rules, choose **Create migration project**.

To duplicate an existing transformation rule, choose **Duplicate**. To edit an existing transformation rule, choose the rule from the list. To delete an existing transformation rule, choose **Remove**.

Editing transformation rules

You can add new, remove, or edit existing transformation rules in your migration project. Because DMS Schema Conversion applies the transformation rules during the launch of schema conversion, make sure that you close schema conversion and launch it again after you edit your rules.

To edit transformation rules

1. Sign in to the AWS Management Console, and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Migration projects**, and then choose your migration project.
3. Choose **Schema conversion**, and then choose **Close schema conversion**.
4. After AWS DMS closes schema conversion, choose **Modify** to edit your migration project settings.
5. For **Transformation rules**, choose one of the following actions:
   - Choose **Duplicate** to duplicate an existing transformation rule and add it in the end of the list.
   - Choose **Remove** to remove an existing transformation rule.
   - Choose the existing transformation rule to edit it.
6. After you are done editing rules, choose **Save changes**.
7. On the **Migration projects** page, choose your project from the list. Choose **Schema conversion**, then choose **Launch schema conversion**.

Converting database schemas in DMS Schema Conversion: step-by-step guide

After you create the migration project and connect to your source and target databases, you can convert your source database objects to a format compatible with your target database. DMS Schema Conversion displays your source database schema in the left panel in a tree-view format.

Each node of the database tree is *lazy loaded*. When you choose a node in the tree view, DMS Schema Conversion requests the schema information from your source database at that time. To load the schema information faster, choose your schema, and then choose **Load metadata** from the **Actions** menu. DMS Schema Conversion then reads the database metadata and stores the information on an Amazon S3 bucket. You can now browse the database objects faster.

You can convert the whole database schema, or you can choose any schema item from your source database to convert. If the schema item that you choose depends on a parent item, then DMS Schema Conversion also generates the schema for the parent item. For example, when you choose a table to convert, DMS Schema Conversion creates the converted table and the database schema that the table is in.

Converting database objects

You can use DMS Schema Conversion to convert an entire database schema or separate database schema objects.

To convert an entire database schema

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Migration projects**. The **Migration projects** page opens.
3. Choose your migration project, and then choose **Schema conversion**.
4. Choose **Launch schema conversion**. The **Schema conversion** page opens.
5. In the source database pane, select the check box for the schema name.
6. Choose this schema in the left pane of the migration project. DMS Schema Conversion highlights the schema name in blue and activates the **Actions** menu.
7. For **Actions**, choose **Convert**. The conversion dialog box appears.
8. Choose **Convert** in the dialog box to confirm your choice.

To convert your source database objects

1. Sign in to the AWS Management Console, and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Migration projects**. The **Migration projects** page opens.
3. Choose your migration project, and then choose **Schema conversion**.
4. Choose **Launch schema conversion**. The **Schema conversion** page opens.
5. In the source database pane, select your source database objects.

6. After you select all check boxes for the objects that you want to convert, choose the parent node for all selected objects in your left panel.

   DMS Schema Conversion highlights the parent node in blue and activates the **Actions** menu.

7. For **Actions**, choose **Convert**. The conversion dialog box appears.

8. Choose **Convert** in the dialog box to confirm your choice.

For example, to convert two out of 10 tables, select the check boxes for the two tables that you want to convert. Notice that the **Actions** menu is inactive. After you choose the **Tables** node, DMS Schema Conversion highlights its name in blue and activates the **Actions** menu. Then you can choose **Convert** from this menu.

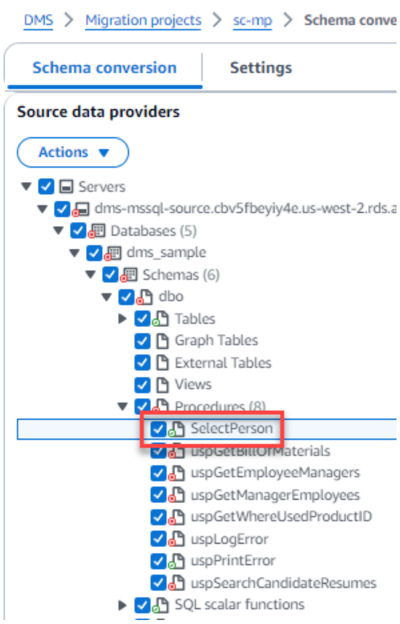
Likewise, to convert two tables and three procedures, select the check boxes for the object names. Then, choose the schema node to activate the **Actions** menu, and choose **Convert schema**.

Editing and saving your converted SQL code

The **Schema conversion** page allows you to edit converted SQL code in your database objects. Use the following procedure to edit your converted SQL code, apply the changes, and then save them.

To edit, apply changes to, and save your converted SQL code

1. In the **Schema conversion** page, open the tree view in the **Source data providers** pane to display a code object.



2. From the **Source data providers** pane, choose **Actions**, **Convert**. Confirm the action.
3. When the conversion completes, to view the converted SQL, expand the center pane if needed. To edit the converted SQL, choose the edit icon in the **Target SQL** pane.

4. After you edit the target SQL, confirm your changes by choosing the check icon at the top of the page. Confirm the action.
5. In the **Target data providers** pane, choose **Actions**, **Apply changes**. Confirm the action.
6. DMS writes the edited procedure to the target data store.

Reviewing converted database objects

After you have converted your source database objects, you can choose an object in the left pane of your project. You can then view the source and converted code for that object. DMS Schema Conversion automatically loads the converted code for the object that you selected in the left pane. You can also see the properties or parameters of the object that you selected.

DMS Schema Conversion automatically stores the converted code as part of your migration project. It doesn't apply these code changes to your target database. For more information about applying converted code to your target database, see Applying your converted code. To remove the converted code from your migration project, select your target schema in the right pane, and then choose **Refresh from database** from **Actions**.

After you have converted your source database objects, you can see the conversion summary and action items in the lower-center pane. You can see the same information when you create an assessment report. The assessment report is useful for identifying and resolving schema items that

DMS Schema Conversion can't convert. You can save the assessment report summary and the list of conversion action items in CSV files. For more information, see Database migration assessment reports.

Converting database objects with generative AI

The DMS Schema Conversion with generative AI feature streamlines the database migration process by offering recommendations to help you convert previously unconverted code objects that typically require complex manual conversion. This feature is available for Oracle to PostgreSQL/Aurora PostgreSQL, SQL Server to PostgreSQL/Aurora PostgreSQL and SAP ASE (Sybase ASE) to PostgreSQL/Aurora PostgreSQL conversions. You can convert an entire database schema or individual database schema objects.

> **Warning**
>
> Generative AI features in DMS Schema Conversion use cross-region inference. For more information, see Cross-region inference in DMS Schema Conversion.

To convert your source database objects with generative AI, follow steps 1 to 6 in Converting database objects then continue with one of these two methods:

- Method 1: From the **Actions** menu, select **Convert**. In the conversion dialog box that appears, enable the **Convert schema with Generative AI** option and click **Convert**.
- Method 2: Click **Convert schema with Generative AI** in the top right corner. In the conversion dialog box, ensure the option is enabled and click **Convert**.

To manually adjust this setting at any time in DMS Schema Conversion console:

- Navigate to the **Settings** tab.
- In the **Conversion settings section**, enable the **Generative AI** option to approve the use of generative AI.

> **Note**
>
> Supported regions:
>
> - Asia Pacific (Tokyo) (ap-northeast-1)

- Asia Pacific (Osaka) (ap-northeast-3)
- Asia Pacific (Sydney) (ap-southeast-2)
- Canada (Central) (ca-central-1)
- Europe (Frankfurt) (eu-central-1)
- Europe (Stockholm) (eu-north-1)
- Europe (Ireland) (eu-west-1)
- Europe (London) (eu-west-2)
- Europe (Paris) (eu-west-3)
- US East (N. Virginia) (us-east-1)
- US East (Ohio) (us-east-2)
- US West (Oregon) (us-west-2)

Note

Supported conversion paths:

- Oracle to Amazon RDS for PostgreSQL
- Oracle to Amazon Aurora PostgreSQL
- Microsoft SQL Server to Amazon RDS for PostgreSQL
- Microsoft SQL Server to Amazon Aurora PostgreSQL
- SAP ASE (Sybase ASE) to Amazon RDS for PostgreSQL
- SAP ASE (Sybase ASE) to Amazon Aurora PostgreSQL

Scope of Generative AI conversion

Generative AI-assisted schema conversion focuses on specific SQL elements with designated action items. All other SQL elements are converted using default rule-based approaches. The SQL elements within the extended scope of Generative AI conversion include:

Conversion direction	Action item	Message	Syntax element
Oracle to Amazon RDS for PostgreSQL and Oracle to Amazon Aurora PostgreSQL	5578	AWS DMS Schema Conversion cannot convert the SELECT statement	All occurrences except limitations
	30415	Your MERGE statement contains a filtering condition in the WHERE clause that is based on a value in a target table column	All occurrences except limitations
	5591	AWS DMS Schema Conversion cannot convert system objects	All occurrences except limitations
	5029	AWS DMS Schema Conversion cannot convert the usage of objects with unsupported data types	Usage of objects with the unsupported STANDARD.BFILE data type in function and procedure arguments.
	5031	AWS DMS Schema Conversion cannot convert CURSOR expressions	All occurrences except limitations
	5043	AWS DMS Schema Conversion cannot convert hierarchical queries with	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
		an asterisk in the SELECT clause	
	5073	PostgreSQL does not support hierarchi cal queries with pseudocolumns	All occurrences except limitations
	5102	PostgreSQL does not support MERGE statements	All occurrences except limitations
	5585	AWS DMS Schema Conversion cannot convert outer joins into correlated subqueries	All occurrences except limitations
	5608	AWS DMS Schema Conversion cannot convert UPDATE statements that have a subquery that returns multiple columns in the SET clause	All occurrences except limitations
	5619	AWS DMS Schema Conversion cannot convert system objects	All occurrences except limitations
	5852	PostgreSQL supports only tables as a target in the MERGE statement.	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	5853	AWS DMS Schema Conversion supports only tables, views or sub-queries as a source in the USING clause of the MERGE statement	All occurrences except limitations
	5855	Your MERGE statement contains a filtering condition in the WHERE clause that is based on a value in a target table column	All occurrences except limitations
	9996	Internal Converter error occurred	All occurrences except limitations
	9993	Unable to transform statement due to references to unresolved object	All occurrences except limitations
	5598	PostgreSQL does not support ROWID	All occurrences except limitations
	5340	AWS DMS Schema Conversion cannot convert functions	All occurrences except limitations
	5071	PostgreSQL does not support the INSERT statement for subqueries	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	5068	PostgreSQL does not support the DELETE statement for subqueries	All occurrences except limitations
	5065	PostgreSQL does not support the UPDATE statement for subqueries	All occurrences except limitations
	5586	AWS DMS Schema Conversion cannot convert queries with the NOCYCLE clause	All occurrences except limitations
	5351	AWS DMS Schema Conversion cannot convert objects	All occurrences except limitations
	5077	PostgreSQL does not support the PIVOT clause for SELECT statements	All occurrences except limitations
	5126	PostgreSQL does not support MODEL statements	All occurrences except limitations
	5121	PostgreSQL does not support FORALL statements	All occurrences except limitations
	5141	AWS DMS Schema Conversion does not support this type of conversion	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	5142	AWS DMS Schema Conversion cannot convert nested calls of the same method	All occurrences except limitations
	5245	PostgreSQL does not support views with nested table columns	All occurrences except limitations
	5500	AWS DMS Schema Conversion cannot convert database mail sending	All occurrences except limitations
	5501	AWS DMS Schema Conversion cannot convert scheduled jobs	All occurrences except limitations
	5645	PostgreSQL does not support BULK COLLECT INTO clauses for several object table targets.	All occurrences except limitations
	5665	PostgreSQL does not support the collectio n data type defined with PRAGMA AUTONOMOU S_TRANSACTION	All occurrences except limitations
	5637	PostgreSQL does not support bulk collect into VARRAY of VARRAY	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	5594	AWS DMS Schema Conversion cannot convert date time expressions	All occurrences except limitations
	5622	AWS DMS Schema Conversion converts the dbms_tran saction.local_tran saction_id function with the parameter set to true	All occurrences except limitations
	5643	PostgreSQL does not support BULK COLLECT INTO clauses for multileve l collection types in SELECT statements.	All occurrences except limitations
	5649	PostgreSQL does not support multiset operators for multilevel collection types.	All occurrences except limitations
	5651	AWS DMS Schema Conversion cannot convert pipelined table functions	All occurrences except limitations
	5793	AWS DMS Schema Conversion creates the queue with the GRANT ALL option	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	5794	PostgreSQL sets the queue mode to ENABLE by default	All occurrences except limitations
	5795	Amazon Simple Queue Service does not support queues in the DISABLE mode	All occurrences except limitations
Microsoft SQL Server to Amazon RDS for PostgreSQL and Microsoft SQL Server to Amazon Aurora PostgreSQL	7610	AWS DMS Schema Conversion cannot convert unsupported DDL statements	All occurrences except limitations
	7622	AWS DMS Schema Conversion cannot convert the DELETE statement using complex inline functions	All occurrences except limitations
	7624	AWS DMS Schema Conversion cannot convert the DELETE statement from an inline function for tables without primary keys	All occurrences except limitations
	7626	AWS DMS Schema Conversion cannot convert the UPDATE statement using complex inline functions	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	7627	AWS DMS Schema Conversion cannot convert this syntax element	All occurrences except limitations
	7628	PostgreSQL does not support GOTO statements	All occurrences except limitations
	7637	PostgreSQL does not support global cursors	All occurrences except limitations
	7639	PostgreSQL does not support dynamic cursors	All occurrences except limitations
	7644	PostgreSQL does not support the %s clause	All occurrences except limitations
	7645	PostgreSQL does not support running pass-through commands on linked servers	All occurrences except limitations
	7653	PostgreSQL does not support GROUP BY ROLLUP clauses	All occurrences except limitations
	7654	PostgreSQL does not support GROUP BY CUBE clauses	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	7655	PostgreSQL does not support GROUP BY GROUPING SETS clauses	All occurrences except limitations
	7672	PostgreSQL does not support EXECUTE statements that run a character string	All occurrences except limitations
	7683	MERGE is not supported if the target is a view, a materialized view, or an external table	All occurrences except limitations
	7687	PostgreSQL does not support CONTAINS predicates	All occurrences except limitations
	7688	PostgreSQL does not support FREETEXT predicates	All occurrences except limitations
	7691	PostgreSQL does not support the WAITFOR TIME feature	All occurrences except limitations
	7695	PostgreSQL does not support the call of a procedure as a variable	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	7696	AWS DMS Schema Conversion cannot convert the object because the %s object is not created	All occurrences except limitations
	7708	AWS DMS Schema Conversion cannot convert the usage of the unsupported %s data type	All occurrences except limitations
	7709	AWS DMS Schema Conversion cannot convert the usage of a symmetric key	All occurrences except limitations
	7773	AWS DMS Schema Conversion cannot convert arithmetic operations with dates	All occurrences except limitations
	7774	AWS DMS Schema Conversion cannot convert arithmeti c operations with mixed types of operands	All occurrences except limitations
	7794	PostgreSQL does not support user-defined data types	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	7796	PostgreSQL does not support TOP clauses in UPDATE statements	All occurrences except limitations
	7797	PostgreSQL does not support the DELETED column prefix for OUTPUT clauses in UPDATE statements	All occurrences except limitations
	7798	PostgreSQL does not support TOP clauses in DELETE statements	All occurrences except limitations
	7799	PostgreSQL does not support TOP clauses in INSERT operators	All occurrences except limitations
	7804	PostgreSQL does not support the bitwise exclusive OR operator	All occurrences except limitations
	7805	PostgreSQL does not support the !<	All occurrences except limitations
	7806	PostgreSQL does not support the !> (not greater than) operator	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	7811	PostgreSQL does not support the %s function. AWS DMS Schema Conversion skips this unsupport ed function in the converted code	All occurrences except limitations, excluding DDL
	7816	PostgreSQL does not support methods for the XML data type	All occurrences except limitations
	7817	PostgreSQL does not support the FOR XML PATH option in SQL queries	All occurrences except limitations
	7818	PostgreSQL does not support arithmeti c operations with binary data types	All occurrences except limitations
	7819	PostgreSQL does not support INSERT... EXECUTE statements	All occurrences except limitations
	7820	PostgreSQL does not support the VALUE() method	All occurrences except limitations
	7824	RECURSIVE CTE is not supported for MERGE statement	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	7829	AWS DMS Schema Conversion cannot convert variable assignments with UPDATE statements	All occurrences except limitations
	7830	AWS DMS Schema Conversion cannot convert arithmetic operations with the CASE operand	All occurrences except limitations
	7832	AWS DMS Schema Conversion cannot convert INSTEAD OF triggers on views	All occurrences except limitations
	7833	AWS DMS Schema Conversion cannot convert the `@@rowcount` function in the current context	All occurrences except limitations
	7836	PostgreSQL does not support write operations for binary data	All occurrences except limitations
	7840	AWS DMS Schema Conversion cannot convert Database Console Command statements	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	7904	AWS DMS Schema Conversion cannot convert the %s system object	All occurrences except limitations
	7905	PostgreSQL does not support PIVOT clauses for SELECT statements	All occurrences except limitations
	7906	PostgreSQL does not support UNPIVOT clauses for SELECT statements	All occurrences except limitations
	7909	AWS DMS Schema Conversion cannot convert UPDATE(column) OR COLUMNS_UPDATED statements	All occurrences except limitations
	7916	AWS DMS Schema Conversion cannot emulate the MERGE statement using the INSERT ON CONFLICT statement	All occurrences except limitations
	7917	PostgreSQL does not support the %s function	All occurrences except limitations
	7918	PostgreSQL does not support table-valued functions	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	7919	PostgreSQL does not support FOR XML with the %s directive	All occurrences except limitations
	7920	PostgreSQL does not support EXPLICIT mode with FOR XML	All occurrences except limitations
	7925	PostgreSQL does not support the percent character for OPENXML flags	All occurrences except limitations
	7927	PostgreSQL does not support OUTER joins for self-refe renced tables without a primary key	All occurrences except limitations
	7929	AWS DMS Schema Conversion cannot convert INSERT from EXEC statements	All occurrences except limitations
	7939	AWS DMS Schema Conversion cannot convert the %s JSON system function	All occurrences except limitations
	7940	AWS DMS Schema Conversion cannot convert OPENJSON system table-valued functions	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	7941	AWS DMS Schema Conversion cannot convert all open datasets because you have multiple open datasets	All occurrences except limitations
	9996	Internal Converter error occurred	All occurrences except limitations
SAP ASE (Sybase ASE) to Amazon RDS for PostgreSQL and SAP ASE (Sybase ASE) to Amazon Aurora PostgreSQL	3014	Unable to convert functions	All occurrences except limitations
	3016	PostgreSQL does not support TOP option in the DML operator	All occurrences except limitations
	3021	Unable to perform an automated migration of the arithmetic operation	All occurrences except limitations
	3023	PostgreSQL does not support arithmeti c operations with binary data types	All occurrences except limitations
	3025	Date/time format can be not matched	All occurrences except limitations
	3026	Automatic conversio n of the operator WAITFOR with a variable is not supported	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	3027	PostgreSQL does not support WAITFOR TIME feature	All occurrences except limitations
	3028	PostgreSQL does not support WAITFOR with instruction	All occurrences except limitations
	3061	Unable to convert system objects	All occurrences except limitations
	3064	In PostgreSQL you should not repeat the target table in the FROM clause of an UPDATE statement	All occurrences except limitations
	3065	DELETE statement with self-refe rence table in the FROM clause and OUTER JOIN can not be transformed automatically	All occurrences except limitations
	3069	Unable to convert statement	All occurrences except limitations
	3081	DMS SC can't convert unsupported DDL statements	All occurrences except limitations
	3088	PostgreSQL does not support global cursors	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	3089	PostgreSQL does not support dynamic cursors	All occurrences except limitations
	3121	DMS SC can't convert the usage of an unsupported data type	All occurrences except limitations
	3122	DMS SC can't convert arithmetic operations with dates	All occurrences except limitations
	3123	DMS SC can't convert arithmetic operations with mixed types of operands	All occurrences except limitations
	3146	PostgreSQL does not support the bitwise exclusive OR operator	All occurrences except limitations
	3147	PostgreSQL does not support the !< (not less than) operator	All occurrences except limitations
	3148	PostgreSQL does not support the !> (not greater than) operator	All occurrences except limitations
	3150	DMS SC can't convert functions	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	3156	PostgreSQL does not support arithmeti c operations with binary data types	All occurrences except limitations
	3162	DMS SC can't convert variable assignmen ts with UPDATE statements	All occurrences except limitations
	3163	DMS SC can't convert arithmetic operation s with the CASE operand	All occurrences except limitations
	3168	PostgreSQL does not support write operations for binary data	All occurrences except limitations
	3172	DMS SC can't convert Database Console Command statement s	All occurrences except limitations
	3177	DMS SC can't convert system objects	All occurrences except limitations
	3182	DMS SC can't convert UPDATE(column) OR COLUMNS_UPDATED statements	All occurrences except limitations
	3190	DMS SC can't convert functions	All occurrences except limitations

Conversion direction	Action item	Message	Syntax element
	3191	PostgreSQL does not support table-valued functions	All occurrences except limitations
	9996	Internal Converter error occurred	All occurrences except limitations

Limitations

The Converting database objects with generative AI feature has the following limitations:

- Database endpoints supporting generative AI conversion are not visible in the AWS Console. You can view them only by exporting the assessment report as a PDF or CSV file.
- As a probabilistic system, generative AI-assisted Schema Conversion may not achieve 100 percent accuracy in all conversions. It can also produce different results for the same SQL statements over a period of time. You must review and validate all conversion outputs.
- Generative AI conversion is not supported for:
  - DEFAULT constraint in a table
  - DEFAULT value for a function or procedure parameter
  - COMPUTE COLUMN in a table
  - TRIGGER
  - COLUMN DATA TYPE
  - Dynamic SQL
  - INDEX
  - CONSTRAINT
- If the source statement is converted with multiple action items and at least one action item is processed using generative AI, then all Action Items are replaced by one action item 5444 on a target for Oracle and 7744 for Microsoft SQL Server. The action item 9997 is an exception that is saved after processed using generative AI.

> **Warning**
>
> Conversion using generative AI takes longer than basic conversion.

Every AWS account have a per-minute quota limiting the number of SQL statements that can be converted using generative AI. Statements exceeding this limit are queued for processing in subsequent minutes. The quota is as follows:

Region	SQL Statements per AWS account per minute
Asia Pacific (Tokyo) (ap-northeast-1) Asia Pacific (Osaka) (ap-northeast-3) Asia Pacific (Sydney) (ap-southeast-2) Canada (Central) (ca-central-1) Europe (Stockholm) (eu-north-1) Europe (Ireland) (eu-west-1) Europe (London) (eu-west-2) Europe (Paris) (eu-west-3) US East (Ohio) (us-east-2)	Up to 24 statements
Europe (Frankfurt) (eu-central-1) US East (N. Virginia) (us-east-1) US West (Oregon) (us-west-2)	Up to 40 statements

Specifying schema conversion settings for migration projects

After you create a migration project, you can specify conversion settings in DMS Schema Conversion. Configuring your schema conversion settings improves the performance of the converted code.

To edit conversion settings

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Migration projects**. The **Migration projects** page opens.
3. Choose your migration project. Choose **Schema conversion**, then **Launch schema conversion**.
4. Choose **Settings**. The **Settings** page opens.
5. In the **Conversion** section, change the settings.
6. Choose **Apply**, and then choose **Schema conversion**.

For all conversion pairs, you can limit the number of comments with action items in the converted code. To limit the number of comments in the converted code, open the conversion settings in your migration project.

For the **Comments in converted SQL code**, choose the severity level of action items. DMS Schema Conversion adds comments in the converted code for action items of the selected severity and higher. For example, to minimize the number of comments in your converted code, choose **Errors only**.

To include comments for all action items in your converted code, choose **All messages**.

Other conversion settings are different for each pair of source and target databases.

Topics

- Understanding Oracle to MySQL conversion settings
- Understanding Oracle to PostgreSQL conversion settings
- Understanding SQL Server to MySQL conversion settings
- Understanding SQL Server to PostgreSQL conversion settings
- Understanding PostgreSQL to MySQL conversion settings
- Understanding IBM Db2 for z/OS to Amazon RDS for Db2 conversion settings
- Understanding SAP ASE (Sybase ASE) to PostgreSQL conversion settings

Understanding Oracle to MySQL conversion settings

Oracle to MySQL conversion settings in DMS Schema Conversion include the following:

- Your source Oracle database can use the ROWID pseudocolumn. MySQL doesn't support similar functionality. DMS Schema Conversion can emulate the ROWID pseudocolumn in the converted code. To do so, turn on the **Generate row ID** option.

  If your source Oracle code doesn't use the ROWID pseudocolumn, turn off the **Generate row ID** option. In this case, the converted code works faster.
- Your source Oracle code can include the TO_CHAR, TO_DATE, and TO_NUMBER functions with parameters that MySQL doesn't support. By default, DMS Schema Conversion emulates the usage of these parameters in the converted code.

  You can use native MySQL TO_CHAR, TO_DATE, and TO_NUMBER functions when your source Oracle code lacks parameters that are unsupported by MySQL. In this case, the converted code works faster. To do so, select the following values:
  - **Use a native MySQL TO_CHAR function**
  - **Use a native MySQL TO_DATE function**
  - **Use a native MySQL TO_NUMBER function**
- Your database and applications can run in different time zones. By default, DMS Schema Conversion emulates time zones in the converted code. However, you don't need this emulation when your database and applications use the same time zone. In this case, select **Improve the performance of the converted code where the database and applications use the same time zone**.

Understanding Oracle to PostgreSQL conversion settings

Oracle to PostgreSQL conversion settings in DMS Schema Conversion include the following:

- To convert applicable database objects using generative AI, enable the **Generative AI** setting. Objects successfully converted using generative AI will be clearly identified with **Action Item 5444**, which states: "This conversion uses machine learning models that generate predictions based on patterns in data. For more information, see the section called “Converting database objects with generative AI”.
  - You can convert schemas for different supported versions of your target database, not just the latest version. To specify a version, use the **Target engine version** setting. Available conversion options will vary depending on the engine version you select. Currently, PostgreSQL versions 14 and 15 are supported (they have identical settings). Note that using a target engine version

during schema conversion that differs from the version specified in this setting may lead to compatibility issues and errors when performing the Apply to the target operation.

- AWS DMS can convert Oracle materialized views to tables or materialized views on PostgreSQL. For **Materialized views**, choose how to convert your source materialized views.
- Your source Oracle database can use the `ROWID` pseudocolumn. PostgreSQL doesn't support similar functionality. DMS Schema Conversion can emulate the `ROWID` pseudocolumn in the converted code using the `bigint` or `character varying` data type. To do so, choose **Use the bigint data type to emulate the ROWID pseudocolumn** or **Use the character varying data type to emulate the ROWID pseudocolumn** for **Row ID**.
  - To convert NUMBER data type to the best fitting numeric data type, depending on the precision and scale you can select Use the optimized data type mapping for columns of the NUMBER data type.
- If your source Oracle code doesn't use the `ROWID` pseudocolumn, choose **Don't generate**. In this case, the converted code works faster.
- Your source Oracle code can include the `TO_CHAR`, `TO_DATE`, and `TO_NUMBER` functions with parameters that PostgreSQL doesn't support. By default, DMS Schema Conversion emulates the usage of these parameters in the converted code.

  You can use native PostgreSQL `TO_CHAR`, `TO_DATE`, and `TO_NUMBER` functions when your source Oracle code lacks parameters that are unsupported by PostgreSQL. In this case, the converted code works faster. To do so, select the following values:
  - **Use a native PostgreSQL TO_CHAR function**
  - **Use a native PostgreSQL TO_DATE function**
  - **Use a native PostgreSQL TO_NUMBER function**
- Your database and applications can run in different time zones. By default, DMS Schema Conversion emulates time zones in the converted code. However, you don't need this emulation when your database and applications use the same time zone. In this case, select **Improve the performance of the converted code where the database and applications use the same time zone**.
- To continue using sequences in your converted code, select **Populate converted sequences with the last value generated on the source side**.
- In some cases, your source Oracle database might store only integer values in the primary or foreign key columns of the `NUMBER` data type. In these cases, AWS DMS can convert these columns to the `BIGINT` data type. This approach improves the performance of your converted code. To do so, select **Convert primary and foreign key columns of the NUMBER data type to**

the BIGINT data type. Make sure that your source doesn't include floating point values in these columns to avoid data loss.

- To skip deactivated triggers and constraints in your source code, choose **Convert only active triggers and constraints**.
- You can use DMS Schema Conversion to convert string variables that are called as dynamic SQL. Your database code can change the values of these string variables. To make sure that AWS DMS always converts the latest value of this string variable, select **Convert the dynamic SQL code that is created in called routines**.
- PostgreSQL versions 10 and earlier don't support procedures. If you aren't familiar with using procedures in PostgreSQL, AWS DMS can convert Oracle procedures to PostgreSQL functions. To do so, select **Convert procedures to functions**.
- To see additional information about the occurred action items, you can add specific functions to the extension pack. To do so, select **Add extension pack functions that raise user-defined exceptions**. Then choose severity levels to raise user-defined exceptions. Make sure that you apply the extension pack schema after you convert your source database objects. For more information about extension packs, see Using extension packs.
- Your source Oracle database can include constraints with the automatically generated names. If your source code uses these names, make sure that you select **Keep the names of system generated constraints**. If your source code uses these constraints, but doesn't use their names, clear this option to increase the conversion speed.
- If your source and target databases run in different time zones, the function that emulates the `SYSDATE` built-in Oracle function returns different values compared to the source function. To make sure that your source and target functions return the same values, choose **Set the time zone of your source database**.
- You can use the functions from the orafce extension in your converted code. To do so, for **Orafce built-in routines**, select the functions to use. For more information about orafce, see orafce on GitHub.

Understanding SQL Server to MySQL conversion settings

SQL Server to MySQL conversion settings in DMS Schema Conversion include the following:

- Your source SQL Server database can store the output of `EXEC` in a table. DMS Schema Conversion creates temporary tables and an additional procedure to emulate this feature. To use this emulation, select **Create additional routines to handle open datasets**.

Understanding SQL Server to PostgreSQL conversion settings

SQL Server to PostgreSQL conversion settings in DMS Schema Conversion include the following:

- To convert applicable database objects using generative AI enable the **Generative AI** setting. Objects successfully converted using generative AI will be clearly identified with **Action Item 7744**, which states: "This conversion uses machine learning models that generate predictions based on patterns in data." For more information, see the section called “Converting database objects with generative AI”.
- In SQL Server, you can use indexes with the same name in different tables. However, in PostgreSQL, all index names that you use in the schema must be unique. To make sure that DMS Schema Conversion generates unique names for all your indexes, select **Generate unique names for indexes**.
- PostgreSQL versions 10 and earlier don't support procedures. If you aren't familiar with using procedures in PostgreSQL, AWS DMS can convert SQL Server procedures to PostgreSQL functions. To do so, select **Convert procedures to functions**.
- Your source SQL Server database can store the output of EXEC in a table. DMS Schema Conversion creates temporary tables and an additional procedure to emulate this feature. To use this emulation, select **Create additional routines to handle open datasets**.
- You can define the template to use for the schema names in the converted code. For **Schema names**, choose one of the following options:
  - **DB –** Uses the SQL Server database name as a schema name in PostgreSQL.
  - **SCHEMA –** Uses the SQL Server schema name as a schema name in PostgreSQL.
  - **DB_SCHEMA –** Uses a combination of the SQL Server database and schema names as a schema name in PostgreSQL.
- You can keep the letter case in the names of source operands. To avoid conversion of operand names to lowercase, select **Avoid casting to lowercase for case-sensitive operations**. This option is applicable only if the case-sensitivity feature is enabled in the source database.
- You can keep the parameter names from your source database. DMS Schema Conversion can add double quotation marks to the names of parameters in the converted code. To do so, select **Keep original parameter names**.
- You can keep a length of routine parameters from your source database. DMS Schema Conversion creates domains and uses them to specify a length of routine parameters. To do so, select **Preserve parameter lengths**.

- To convert unsupported built-in objects to stub objects, enable the **Convert unsupported built-in objects to stub objects** setting:
  - When enabled, DMS SC replaces unsupported built-in objects with corresponding stub objects in the target database. This feature converts code sections that would normally be enclosed by migration issue 7811 or 7904. It creates stub objects based on the type of the source built-in objects `PROCEDURE` for procedures, `VIEW` for views or tables.

    Conversion of a source database object with a call of an unsupported object results in a call of a stub object and migration issue 7822.

    You can choose to create stub objects in a separate schema by enabling the **Create stub objects in a separate schema** option. When selected, stub objects are created in a special schema named `aws_sqlserver_stub` in the target database. If not selected, they are created in the same schema as the calling objects.
  - Stub routines are named based on the fully qualified name of the original built-in. For stub views, the naming convention includes the system schema name `system_schema_name $builtin_view_name`.

    During re-conversion, DMS SC checks for existing stub routines in the target database. If a routine with the same name and input parameters already exists, it is not overwritten.

    After conversion, review and implement custom code for stub routines as needed.
- `CITEXT` datatype for all string datatypes setting in DMS Schema Conversion include the following:
  - To use the `CITEXT` datatype for case-insensitive string operations when converting from SQL Server to PostgreSQL, enable the **Use CITEXT for all string datatypes** setting. This option helps maintain consistent behavior when migrating from a case-insensitive SQL Server to a case-sensitive PostgreSQL environment.
  - When enabled, DMS SC converts all relevant string datatypes from the source SQL Server database to `CITEXT` in PostgreSQL. This eliminates the need for explicit LOWER () function calls in conditions and automatically casts string expressions in conditional operations to `CITEXT`.
  - To determine if your SQL Server instance is case-sensitive, run the following query:

    ```
    SELECT SERVERPROPERTY('COLLATION');
    ```

    A result containing 'CI' indicates case-insensitive, while 'CS' indicates case-sensitive.

- The `CITEXT` conversion may not apply in scenarios where explicit case-sensitive collate settings are used at the server, database, or column level.

  To use this feature, ensure that the `CITEXT` module is installed and available in your target PostgreSQL database.
- When using the `CITEXT` datatype conversion, consider the following best practices:
  - Enable this feature when migrating from a case-insensitive SQL Server to maintain consistent behavior in PostgreSQL.
  - Review your application code to ensure it doesn't rely on case-sensitive string operations.
  - Thoroughly test your application after migration to verify that case-insensitive behavior is maintained as expected.

Understanding PostgreSQL to MySQL conversion settings

PostgreSQL to MySQL conversion settings in DMS Schema Conversion include the following:

- **Comments in converted SQL code**: This setting includes comments in the converted code for the action items of the selected severity and higher. This setting supports the following values:
  - Errors only
  - Errors and warnings
  - All messages

Understanding IBM Db2 for z/OS to Amazon RDS for Db2 conversion settings

IBM for z/OS to Amazon RDS for DB2 conversion settings in DMS Schema Conversion include the following:

- **Comments in converted SQL code**: This setting includes comments in the converted code for the action items of the selected severity and higher. This setting supports the following values:
  - Errors only
  - Errors and warnings
  - All messages

Understanding SAP ASE (Sybase ASE) to PostgreSQL conversion settings

SAP ASE (Sybase ASE) to PostgreSQL conversion settings in DMS Schema Conversion include the following:

- To convert applicable database objects using generative AI enable the **Generative AI** setting. Objects successfully converted using generative AI will be clearly identified with **Action Item 3077**, which states: "This conversion uses machine learning models that generate predictions based on patterns in data." For more information, see Converting database objects with generative AI.
- **Comments in converted SQL code:** This setting includes comments in the converted code for the action items of the selected severity and higher. This setting supports the following values:
  - Errors only
  - Errors and warnings
  - All messages
- You can define the template to use for the schema names in the converted code. For **Schema names**, choose one of the following options:
  - **DB** – Uses the SAP ASE (Sybase ASE) database name as a schema name in PostgreSQL.
  - **SCHEMA** – Uses the SAP ASE (Sybase ASE) schema name as a schema name in PostgreSQL.
  - **DB_SCHEMA** – Uses a combination of the SAP ASE (Sybase ASE) database and schema names as a schema name in.
- You can keep the exact case of object names from your source database. To do so, select Treat source database object names as case sensitive. When this option is turned off, all database object names are converted to lowercase.
- You can avoid casting operands to lowercase during case-insensitive operations. DMS Schema Conversion will not apply the LOWER function to operands in the converted code. To do so, select **Avoid casting operands to lowercase for case-insensitive operations**. When this option is not selected (default), DMS Schema Conversion automatically applies the LOWER function to convert operands to lowercase before performing case-insensitive comparisons.
- In SAP ASE (Sybase ASE), you can create indexes with identical names across different tables. PostgreSQL requires all index names within a schema to be unique. To ensure AWS Schema Conversion creates unique index names during migration, select **Generate unique names for indexes**. This option automatically adds prefixes to index names to prevent naming conflicts in your target PostgreSQL database.

Refreshing your database schemas in DMS Schema Conversion

After you create a migration project, DMS Schema Conversion stores the information about your source and target schemas in this project. DMS Schema Conversion uses *lazy loading* to load metadata only as it is needed, such as when you choose a node in your database tree. You can use *eager loading* to load the schema information faster. To do so, choose your schema, and then choose **Load metadata** from **Actions**.

After you automatically or manually load the object to your migration project, DMS Schema Conversion doesn't use lazy loading again. So when you change objects, such as tables and procedures in your database, make sure to refresh them in your migration project.

To refresh schemas from the database, select the objects that you want to refresh, and choose **Refresh from database** from **Actions**. You can refresh database objects in your source and target database schemas:

- **Source –** If you update your source database schema, choose **Refresh from database** to replace the schema in your project with the latest schema from your source database.
- **Target –** If you update the schema for your target database, DMS Schema Conversion replaces the schema in your project with the latest schema from your target database. DMS Schema Conversion replaces your converted code with the code from your target database. Make sure that you applied the converted code to your target database before you choose **Refresh from database**. Otherwise, convert your source database schema again.

Saving and applying your converted code in DMS Schema Conversion

After DMS Schema Conversion converts your source database objects, it doesn't immediately apply the converted code to your target database. Instead, DMS Schema Conversion stores the converted code in your project until you are ready to apply it to your target database.

Before you apply the converted code, you can update your source database code and convert the updated objects again to address the existing action items. For more information about items that DMS Schema Conversion can't convert automatically, see Creating database migration assessment reports with DMS Schema Conversion. For more information about refreshing your source database objects in migration project for DMS Schema Conversion, see Refreshing your database schemas.

Instead of applying the converted code directly to your database in DMS Schema Conversion, you can save the code to a file as a SQL script. You can review these SQL scripts, edit them where necessary, and then manually apply these SQL scripts to your target database.

Saving your converted code to a SQL file

You can save your converted schema as SQL scripts in a text file. You can modify the converted code to address action items that DMS Schema Conversion can't convert automatically. You can then run your updated SQL scripts on your target database to apply the converted code to your target database.

To save your converted schema as SQL scripts

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Migration projects**. The **Migration projects** page opens.
3. Choose your migration project, and then choose **Schema conversion**.
4. Choose **Launch schema conversion**. The **Schema conversion** page opens.
5. In the right pane, choose the target database schema or select the converted objects that you want to save. Make sure that DMS Schema Conversion highlights the parent node name in blue and activates the **Actions** menu for the target database.
6. Choose **Save as SQL** for **Actions**. The **Save** dialog box appears.
7. Choose **Save as SQL** to confirm your choice.

   DMS Schema Conversion creates an archive with SQL files and stores this archive in your Amazon S3 bucket.
8. (Optional) Change the S3 bucket for the archive by editing the schema conversion settings in your instance profile.
9. Open the SQL scripts from your S3 bucket.

Applying your converted code

When you are ready to apply your converted code to your target database, choose the database objects in the right pane of your project. You can apply changes to an entire database schema or selected database schema objects.

After you select the database objects, DMS Schema Conversion highlights the name of the selected node or the parent node in blue. It then activates the **Actions** menu. Choose **Apply changes** for **Actions**. In the dialog box that appears, choose **Apply** to confirm your choice and apply the converted code to your target database.

Applying the extension pack schema

When you apply your converted schema to your target database for the first time, DMS Schema Conversion might also apply the extension pack schema. The extension pack schema emulates system functions of the source database that are required to run your converted code for your target database. If your converted code uses the functions of the extension pack, make sure that you apply the extension pack schema.

To apply the extension pack to your target database manually, choose **Apply changes** for **Actions**. In the dialog box that appears, choose **confirm** to apply the extension pack to your target database.

We recommend that you don't modify the extension pack schema to avoid unexpected results in the converted code.

For more information, see Using extension packs in DMS Schema Conversion.

Converting embedded SQL in Java applications

When you use AWS DMS and DMS Schema Conversion to migrate a database, you might need to convert the embedded SQL in your application to be compatible with your target database. Rather than converting it manually, you can use Amazon Q in the IDE to automate the conversion. Amazon Q uses metadata from a DMS Schema Conversion to convert embedded SQL in your application to a version that is compatible with your target database. Amazon Q will detect Oracle SQL code in your application and convert them into PostgreSQL syntax. For more information, see Converting embedded SQL in Java applications with Amazon Q Developer.

Downloading metadata file for embedded SQL conversion in a Java Application

1. Once you complete the conversion, close the project and go to the S3 bucket where your project is stored.
2. Open the folder and find the project folder (“sct-project”).
3. Download the object inside the project folder. This will be a zip file.

4. Use the downloaded zip file as an input to your transformation with Amazon Q: Converting embedded SQL in Java applications with Amazon Q Developer.

Using extension packs in DMS Schema Conversion

An *extension pack* in DMS Schema Conversion is an add-on module that emulates source database functions that aren't supported in the target database. Use an extension pack to make sure that the converted code produces the same results as the source code. Before you can install an extension pack, convert your database schemas.

Each extension pack includes a database schema. This schema includes SQL functions, procedures, tables, and views for emulating specific online transaction processing (OLTP) objects or unsupported built-in functions from the source database.

When you convert your source database, DMS Schema Conversion adds an additional schema to your target database. This schema implements SQL system functions of the source database that are required to run your converted code on your target database. This additional schema is called the extension pack schema.

The extension pack schema is named according to your source database as follows:

- Microsoft SQL Server – `aws_sqlserver_ext`
- Oracle – `aws_oracle_ext`

You can apply extension packs in two ways:

- DMS Schema Conversion can automatically apply an extension pack when you apply your converted code. DMS Schema Conversion applies the extension pack before it applies all other schema objects.
- You can apply an extension pack manually. To do so, choose the extension pack schema in your target database tree, and then choose **Apply**, then **Apply extension pack**.

AWS IAM actions to API mapping for DMS Schema Conversion and Common Studio Framework (CSF)

When setting up access control and writing IAM permissions policies for DMS Schema Conversion and Common Studio Framework, it is important to understand how API actions map to IAM permissions. While some actions share identical names across both interfaces, others differ significantly.

The following table lists the correct mapping between API operations and IAM actions:

API to IAM Mapping

Service	API	IAM
Common Studio Framework (CSF)	CreateMigrationProject	dms:CreateMigrationProject
	DeleteMigrationProject	dms:DeleteMigrationProject
	ModifyMigrationProject	dms:UpdateMigrationProject
	DescribeMigrationProjects	dms:ListMigrationProjects
	CreateInstanceProfile	dms:CreateInstanceProfile
	DeleteInstanceProfile	dms:DeleteInstanceProfile
	ModifyInstanceProfile	dms:UpdateInstanceProfile
	DescribeInstanceProfiles	dms:ListInstanceProfiles
	CreateDataProvider	dms:CreateDataProvider
	DeleteDataProvider	dms:DeleteDataProvider
	ModifyDataProvider	dms:UpdateDataProvider
	DescribeDataProviders	dms:ListDataProviders
DMS Schema Conversion	ModifyConversionConfiguration	dms:dms.UpdateConversionConfiguration
	DescribeConversionConfiguration	dms:DescribeConversionConfiguration
	StartMetadataModelImport	dms:StartMetadataModelImport

Service	API	IAM
	ExportMetadataModelAssessment	dms:ExportMetadataModelAssessment
	StartMetadataModelConversion	dms:StartMetadataModelConversion
	StartMetadataModelExportAsScript	dms:StartMetadataModelExportAsScripts
	StartMetadataModelExportToTarget	dms:StartMetadataModelExportToTarget
	StartExtensionPackAssociation	dms:AssociateExtensionPack
	DescribeMetadataModelConversions	dms:ListMetadataModelConversions
	DescribeMetadataModelExportsToTarget	dms:ListMetadataModelExports
	DescribeMetadataModelExportsAsScript	dms:ListMetadataModelExports
	DescribeMetadataModelImports	dms:DescribeMetadataModelImports
	DescribeMetadataModelAssessments	dms:ListMetadataModelAssessments
	DescribeExtensionPackAssociations	dms:ListExtensionPacks

Migrating databases to their Amazon RDS equivalents with AWS DMS

Homogeneous data migrations in AWS Database Migration Service (AWS DMS) simplify the migration of self-managed, on-premises databases to their Amazon Relational Database Service (Amazon RDS) equivalents. For example, you can use homogeneous data migrations to migrate an on-premises PostgreSQL database to Amazon RDS for PostgreSQL or Aurora PostgreSQL. For homogeneous data migrations, AWS DMS uses native database tools to provide easy and performant like-to-like migrations.

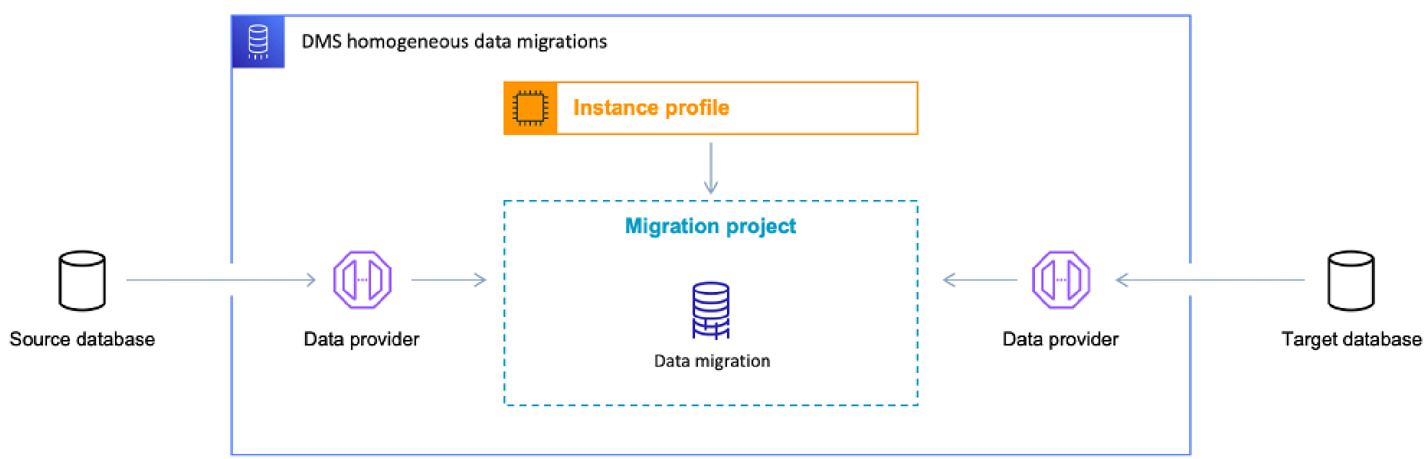
Homogeneous data migrations are serverless, which means that AWS DMS automatically scales the resources that are required for your migration. With homogeneous data migrations, you can migrate data, table partitions, data types, and secondary objects such as functions, stored procedures, and so on.

At a high level, homogeneous data migrations operate with instance profiles, data providers, and migration projects. When you create a migration project with the compatible source and target data providers of the same type, AWS DMS deploys a serverless environment where your data migration runs. Next, AWS DMS connects to the source data provider, reads the source data, dumps the files on the disk, and restores the data using native database tools. For more information about instance profiles, data providers, and migration projects, see Working with data providers, instance profiles, and migration projects in AWS DMS.

For the list of supported source databases, see Sources for DMS homogeneous data migrations.

For the list of supported target databases, see Targets for DMS homogeneous data migrations.

The following diagram illustrates how homogeneous data migrations work.



The following sections provide information about using homogeneous data migrations.

Topics

- Supported AWS Regions
- Features
- Limitations for homogeneous data migrations
- Overview of the homogeneous data migration process in AWS DMS
- Setting up homogeneous data migrations in AWS DMS
- Creating source data providers for homogeneous data migrations in AWS DMS
- Creating and setting a target database to work with AWS DMS schema conversion
- Running homogeneous data migrations in AWS DMS
- Troubleshooting for homogeneous data migrations in AWS DMS

Supported AWS Regions

You can run homogeneous data migrations in the following AWS Regions.

Region Name	Region
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Canada (Central)	ca-central-1
Canada West (Calgary)	ca-west-1
South America (São Paulo)	sa-east-1
Asia Pacific (Tokyo)	ap-northeast-1
Asia Pacific (Seoul)	ap-northeast-2

Region Name	Region
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Jakarta)	ap-southeast-3
Asia Pacific (Melbourne)	ap-southeast-4
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Hyderabad)	ap-south-2
Europe (Frankfurt)	eu-central-1
Europe (Zurich)	eu-central-2
Europe (Stockholm)	eu-north-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Milan)	eu-south-1
Europe (Spain)	eu-south-2
Middle East (UAE)	me-central-1
Middle East (Bahrain)	me-south-1
Israel (Tel Aviv)	il-central-1
Africa (Cape Town)	af-south-1

Features

Homogeneous data migrations provide the following features:

- AWS DMS automatically manages the compute and storage resources in the AWS Cloud that are required for homogeneous data migrations. AWS DMS deploys these resources in a serverless environment when you start a data migration.
- AWS DMS uses native database tools to initiate a fully-automated migration between the databases of the same type.
- You can use homogeneous data migrations to migrate your data as well as the secondary objects such as partitions, functions, stored procedures, and so on.
- You can run homogeneous data migrations in the following three migration modes: full load, ongoing replication, and full load with ongoing replication.
- For homogeneous data migrations, you can use on-premises, Amazon EC2, Amazon RDS databases as a source. You can choose Amazon RDS or Amazon Aurora as a migration target for homogeneous data migrations.
- Homogeneous data migrations only support target table preparation mode for PostgreSQL, MongoDB, and Amazon DocumentDB migrations. For more information, see the section called “Target table preparation mode”.
- You can use homogeneous data migrations to migrate your data from a MySQL-based read-replica to Amazon RDS or Aurora instance

Limitations for homogeneous data migrations

The following limitations apply when you use homogeneous data migrations:

- The Support for selection rules in AWS DMS homogeneous data migrations depends on the source database engine and migration type. PostgreSQL and MongoDB-compatible sources support selection rules for all migration types, while MySQL sources only support selection rules for the Full Load migration type.
- Homogeneous data migrations don't provide a built-in tool for data validation.
- When using homogeneous data migrations with PostgreSQL, AWS DMS migrates views as tables to your target database.

- Homogeneous data migrations capture schema-level changes during an ongoing data replication only for MySQL engine. For other engines, if you create a new table in your source database, then AWS DMS can't migrate this table. To migrate this new table, restart your data migration.
- You can't use homogeneous data migrations in AWS DMS to migrate data from a higher database version to a lower database version.
- Homogeneous data migrations don't support establishing a connection with database instances in VPC secondary CIDR ranges.
- You can't use the 8081 port for homogeneous migrations from your data providers.
- Homogeneous data migrations migrate encrypted MySQL databases and tables as unencrypted on the target database. This is because RDS for MySQL does not support encryption using Keyring plugin. For more information, see MySQL Keyring Plugin not supported documentation in the Amazon RDS User Guide.

Overview of the homogeneous data migration process in AWS DMS

You can use homogeneous data migrations in AWS DMS to migrate data between two databases of the same type. Use the following workflow to create and run a data migration.

1. Create the required AWS Identity and Access Management (IAM) policy and role. For more information, see Creating IAM resources.
2. Configure your source and target databases and create database users with the minimum permissions required for homogeneous data migrations in AWS DMS. For more information, see Creating source data providers and Create and set a target database.
3. Store your source and target database credentials in AWS Secrets Manager. For more information, see Step 1: Create the secret in the *AWS Secrets Manager User Guide*.
4. Create a subnet group, an instance profile, and data providers in the AWS DMS console. For more information, see Creating a subnet group, Creating instance profiles, and Creating data providers.
5. Create a migration project by using the resources that you created in the previous step. For more information, see Creating migration projects.
6. Create, configure, and start a data migration. For more information, see Creating a data migration.

7. After you complete the full load or ongoing replication, you can cut over to start using your new target database.
8. Clean up your resources. Amazon terminates your data migration in your migration project in three days after you complete the migration. However, you need to manually delete such resources as instance profile, data providers, IAM policy and role, and secrets in AWS Secrets Manager.

For more information about homogeneous data migrations in AWS DMS, read the step-by-step migration walkthough for PostgreSQL to Amazon RDS for PostgreSQL migrations.

This video introduces the homogeneous data migrations in AWS DMS and helps you get familiar with this feature.

Setting up homogeneous data migrations in AWS DMS

To set up homogeneous data migrations in AWS DMS, complete the following prerequisite tasks.

Topics

- Creating required IAM resources for homogeneous data migrations in AWS DMS
- Setting up a network for homogeneous data migrations in AWS DMS
- VPC peering network configurations

Creating required IAM resources for homogeneous data migrations in AWS DMS

To run homogeneous data migrations, you must create an IAM policy and an IAM role in your account to interact with other AWS services. In this section, you create these required IAM resources.

Topics

- Creating an IAM policy for homogeneous data migrations in AWS DMS
- Creating an IAM role for homogeneous data migrations in AWS DMS

Creating an IAM policy for homogeneous data migrations in AWS DMS

To access your databases and to migrate data, with AWS DMS, you can create a serverless environment for homogeneous data migrations. Also, AWS DMS stores logs, metrics, and progress for each data migration in Amazon CloudWatch. To create a data migration project, AWS DMS needs access to these services.

In this step, you create an IAM policy that provides AWS DMS with access to Amazon EC2 and CloudWatch resources. Next, create an IAM role and attach this policy.

To create an IAM policy for homogeneous data migrations in AWS DMS

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. In the **Create policy** page, choose the **JSON** tab.
5. Paste the following JSON into the editor.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeVpcs"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:dms-data-migration-*"
        },
        {
            "Effect": "Allow",
```

```
                "Action": [
                    "logs:CreateLogStream",
                    "logs:PutLogEvents"
                ],
                "Resource": "arn:aws:logs:*:*:log-group:dms-data-migration-*:log-
stream:dms-data-migration-*"
            },
            {
                "Effect": "Allow",
                "Action": "cloudwatch:PutMetricData",
                "Resource": "*"
            }
        ]
}
```

6. Choose **Next**.

7. Enter **`HomogeneousDataMigrationsPolicy`** for **Policy name**, and choose **Create policy**.

Creating an IAM role for homogeneous data migrations in AWS DMS

In this step, you create an IAM role that provides AWS DMS with access to AWS Secrets Manager, Amazon EC2, and CloudWatch.

When creating an IAM role, you must also create a `dms-vpc-role`. For more information, see [Creating an IAM role for AWS DMS to manage Amazon VPC](#) in the *Amazon Relational Database Service User Guide*.

To create an IAM role for homogeneous data migrations in AWS DMS

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Roles**.

3. Choose **Create role**.

4. On the **Select trusted entity** page, for **Trusted entity type**, choose **AWS Service**. For **Use cases for other AWS services**, choose **DMS**.

5. Select the **DMS** check box and choose **Next**.

6. On the **Add permissions** page, choose **HomogeneousDataMigrationsPolicy** that you created before.

7. On the **Name, review, and create** page, enter **`HomogeneousDataMigrationsRole`** for **Role name**, and choose **Create role**.
8. Choose **Update policy**.

Setting up a network for homogeneous data migrations in AWS DMS

With AWS DMS, you can create a serverless environment for homogeneous data migrations which uses networking connectivity model that relies on network interfaces. For each data migration, AWS DMS assigns a private IP within one of the subnets defined in the instance profile DMS subnet group. Additionally, a non-static public IP may be assigned if the instance profile is configured for public access. The subnets used in the instance profile should provide access to both source and target hosts, as defined in the data providers. This access can be within the local VPC or established through VPC peering, Direct Connect, VPN, etc.

Also, for ongoing data replication, you must set up interaction between your source and target databases. These configurations depend on the location of your source data provider and your network settings. The following sections provide descriptions of common network configurations.

Topics

Configuring a network using a single virtual private cloud (VPC)

In this configuration, AWS DMS connects to your source and target data providers within the private network.

To configure a network when your source and target data providers are in the same VPC

1. Create the subnet group in the AWS DMS console with the VPC and subnets that your source and target data providers use. For more information, see Creating a subnet group.
2. Create the instance profile in the AWS DMS console with the VPC and the subnet group that you created. Also, choose VPC security groups that your source and target data providers use. For more information, see Creating instance profiles.

3. Ensure that the security group used for the source and target database allows connections from the security group attached to instance profile used by data migration or CIDR block of subnets, specified in replication subnet group.

This configuration doesn't require you to use the public IP address for data migrations.

Configuring a network using different virtual private clouds (VPCs)

In this configuration, AWS DMS uses a private network to connect to your source or target data provider. For another data provider, AWS DMS uses a public network. Depending on which data provider you have in the same VPC as your instance profile, choose one of the following configurations.

To connect through a private network

1. Create the subnet group in the AWS DMS console with the VPC and subnets that your source data provider uses. For more information, see Creating a subnet group.
2. Create the instance profile in the AWS DMS console with the VPC and the subnet group that you created. Also, choose VPC security groups that your source data provider uses. For more information, see Creating instance profiles.
3. Configure VPC peering connection between source and target database VPCs. For more information see, Work with VPC peering connections.
4. Make sure to enable DNS resolution for both directions if you plan to use endpoints instead of private IPs directly. For more information see, Enable DNS resolution for a VPC peering connection.
5. Allow access from the CIDR block of source database's VPC for target database security group. For more information, see Controlling access with security groups.
6. Allow access from the CIDR block of target database's VPC for target database security group. For more information, see Controlling access with security groups.

To connect through a Public network

If your database accepts connections from any address:

1. Create the subnet group in the AWS DMS console with the VPC and public subnets. For more information, see Creating a subnet group.

2. Create the instance profile in the AWS DMS console with the VPC and the subnet group that you created. Set the **Publicly Available** option to **On** for the instance profile.

If you require a persistent public IP address that can be associated to the data migration:

1. Create the subnet group in the AWS DMS console with the VPC and private subnets. For more information, see Creating a subnet group.
2. Create the instance profile in the AWS DMS console with the VPC and the subnet group that you created. Set the **Publicly Available** option to **Off** for the instance profile.
3. Setup NAT Gateway. For more information see Work with NAT gateways.
4. Setup Routing table for NAT gateway. For more information see NAT gateway use cases.
5. Allow access from the public IP address of your NAT Gateway in your database security group. For more information, see Controlling access with security groups.

Using Direct Connect or a VPN to configure a network to a VPC

You can connect remote networks to your VPC through Direct Connect or VPN connections (software or hardware). These options enable you to extend your internal network into AWS Cloud and integrate existing on-premises services such as monitoring, authentication, security, and data systems with your AWS resources. For this configuration, your VPC security group must include a routing rule that directs traffic to a host capable of bridging VPC traffic to your on-premises VPN. This traffic can be designated using either your VPC CIDR range or specific IP addresses. The NAT host must have its own security group configured to allow traffic from your VPC CIDR range or security group into the NAT instance, ensuring seamless communication between your VPC and on-premises infrastructure. For more information, see step 5 for Get started with AWS Site-to-Site VPN procedure in the AWS Site-to-Site VPN User Guide.

Resolving domain endpoints using DNS

For DNS resolution in AWS DMS homogeneous migrations, the service primarily uses the Amazon ECS DNS resolver to resolve domain endpoints. If you need additional DNS resolution capabilities, Amazon Route 53 Resolver is available as an alternative solution. For more information, see Getting started with Route 53 Resolver in the Amazon Route 53 user guide. For more information, regarding configuring endpoint resolution using your on-premises name server with Amazon Route 53 Resolver, see the section called “Using your own on-premises name server”.

> **Note**
>
> If your data migration log shows the message "Initiating connection - Networking model: VPC Peering", refer to the section called "VPC peering network configurations" topic.

VPC peering network configurations

With AWS DMS, you can create a serverless environment for homogeneous data migrations in a virtual private cloud (VPC) based on the Amazon VPC service. When you create your instance profile, you specify the VPC to use. You can use your default VPC for your account and AWS Region, or you can create a new VPC.

For each data migration, AWS DMS establishes a VPC peering connection with the VPC that you use for your instance profile. Next, AWS DMS adds the CIDR block in the security group that is associated with your instance profile. Because AWS DMS attaches a public IP address to your instance profile, all your data migrations that use the same instance profile have the same public IP address. When your data migration stops or fails, AWS DMS deletes the VPC peering connection.

To avoid CIDR block overlapping with the VPC of your instance profile VPC, AWS DMS uses the /24 prefix from one of the following CIDR blocks: `10.0.0.0/8`, `172.16.0.0/12`, and `192.168.0.0/16`. For example, if you run three data migrations in parallel, AWS DMS uses the following CIDR blocks to establish a VPC peering connection.

- `192.168.0.0/24` – for the first data migration
- `192.168.1.0/24` – for the second data migration
- `192.168.2.0/24` – for the third data migration

You can use different network configurations to set up interaction between your source and target databases with AWS DMS. Also, for ongoing data replication, you must set up interaction between your source and target databases. These configurations depend on the location of your source data provider and your network settings. The following sections provide descriptions of common network configurations.

Topics

- Configuring a network using a single virtual private cloud (VPC)
- Configuring a network using different virtual private clouds (VPCs)

- Using an on-premises source data provider
- Configuring ongoing data replication

Configuring a network using a single virtual private cloud (VPC)

In this configuration, AWS DMS connects to your source and target data providers within the private network.

To configure a network when your source and target data providers are in the same VPC

1. Create the subnet group in the AWS DMS console with the VPC and subnets that your source and target data providers use. For more information, see Creating a subnet group.
2. Create the instance profile in the AWS DMS console with the VPC and the subnet group that you created. Also, choose VPC security groups that your source and target data providers use. For more information, see Creating instance profiles.

This configuration doesn't require you to use the public IP address for data migrations.

Configuring a network using different virtual private clouds (VPCs)

In this configuration, AWS DMS uses a private network to connect to your source or target data provider. For another data provider, AWS DMS uses a public network. Depending on which data provider you have in the same VPC as your instance profile, choose one of the following configurations.

To configure a private network for your source data provider and a public network for your target data provider

1. Create the subnet group in the AWS DMS console with the VPC and subnets that your source data provider uses. For more information, see Creating a subnet group.
2. Create the instance profile in the AWS DMS console with the VPC and the subnet group that you created. Also, choose VPC security groups that your source data provider uses. For more information, see Creating instance profiles.
3. Open your migration project. On the **Data migrations** tab, choose your data migration. Take a note of the **public IP address** under **Connectivity and security** on the **Details** tab.

4. Allow access from the public IP address of your data migration in your target database security group. For more information, see Controlling access with security groups in the *Amazon Relational Database Service User Guide*.

To configure a public network for your source data provider and a private network for your target data provider

1. Create the subnet group in the AWS DMS console with the VPC and subnets that your target data provider uses. For more information, see Creating a subnet group.
2. Create the instance profile in the AWS DMS console with the VPC and the subnet group that you created. Also, choose VPC security groups that your target data provider uses. For more information, see Creating instance profiles.
3. Open your migration project. On the **Data migrations** tab, choose your data migration. Take a note of the **public IP address** under **Connectivity and security** on the **Details** tab.
4. Allow access from the public IP address of your data migration in your source database security group. For more information, see Controlling access with security groups in the *Amazon Relational Database Service User Guide*.

Using an on-premises source data provider

In this configuration, AWS DMS connects to your source data provider within the public network. AWS DMS uses a private network to connect to your target data provider.

> **Note**
>
> For homogeneous data migrations, AWS DMS connects to your source database within the public network. However, connectivity to a source database within a public network is not always possible. For more information, see Migrate an on-premises MySQL database to Amazon Aurora MySQL over a private network using AWS DMS homogeneous data migration and Network Load Balancer .

To configure a network for your source on-premises data provider

1. Create the subnet group in the AWS DMS console with the VPC and subnets that your target data provider uses. For more information, see Creating a subnet group.

2. Create the instance profile in the AWS DMS console with the VPC and the subnet group that you created. Also, choose VPC security groups that your target data provider uses. For more information, see Creating instance profiles.
3. Open your migration project. On the **Data migrations** tab, choose your data migration. Take a note of the **public IP address** under **Connectivity and security** on the **Details** tab.
4. Allow access to your source database from the public IP address of your data migration in AWS DMS.

AWS DMS creates inbound or outbound rules in in VPC security groups. Make sure that you don't delete these rules because this action can lead to a failure of your data migration. You can configure your own rules in VPC security groups. We recommended that you add a description to your rules so that you can manage them.

Configuring ongoing data replication

To run data migrations of the **Full load and change data capture (CDC)** or **Change data capture (CDC)** type, you must allow connection between your source and target databases.

To configure a connection between your publicly accessible source and target databases

1. Take a note of the public IP addresses of your source and target databases.
2. Allow access to your source database from the public IP address of your target database.
3. Allow access to your target database from the public IP address of your source database.

To configure a connection between your source and target databases that are privately accessible in a single VPC

1. Take a note of the private IP addresses of your source and target databases.

   > **Important**
   >
   > If your source and target databases are in different VPCs or in different networks, then you can only use public IP addresses for your source and target databases. You can only use public hostnames or IP addresses in data providers.

2. Allow access to your source database from the security group of your target database.
3. Allow access to your target database from the security group of your source database.

Creating source data providers for homogeneous data migrations in AWS DMS

AWS DMS Schema Conversion supports the following databases as source data providers for Homogeneous data migrations projects: Microsoft SQL Server, Oracle, PostgreSQL, MySQL, IBM DB2 for z/OS, and SAP ASE (Sybase ASE).

For supported database versions, see Source data providers for DMS homogeneous data migrations.

Your source data provider can be an on-premises, Amazon EC2, or Amazon RDS database.

Topics

Using a MySQL compatible database as a source for homogeneous data migrations in AWS DMS

You can use a MySQL-compatible database (MySQL or MariaDB) as a source for Homogeneous data migrations in AWS DMS. In this case, your source data provider can be an on-premises, Amazon EC2, or RDS for MySQL or MariaDB database.

To run homogeneous data migrations, you must use a database user with the `SELECT` privileges for the all source tables and secondary objects for replication. For change data capture (CDC) tasks, this user must also have the `REPLICATION CLIENT` (`BINLOG MONITOR` for MariaDB versions later than 10.5.2) and `REPLICATION SLAVE` privileges. For a full load data migration, you don't need these two privileges.

Use the following script to create a database user with the required permissions in your MySQL database. Run the `GRANT` queries for all databases that you migrate to AWS.

```
CREATE USER 'your_user'@'%' IDENTIFIED BY 'your_password';

GRANT REPLICATION SLAVE, REPLICATION CLIENT  ON *.* TO 'your_user'@'%';
```

```
GRANT SELECT, RELOAD, LOCK TABLES, SHOW VIEW, EVENT, TRIGGER ON *.* TO 'your_user'@'%';

GRANT BACKUP_ADMIN ON *.* TO 'your_user'@'%';
```

In the preceding example, replace each *user input placeholder* with your own information. If your source MySQL database version is lower than 8.0, then you can skip the GRANT BACKUP_ADMIN command.

Use the following script to create a database user with the required permissions in your MariaDB database. Run the GRANT queries for all databases that you migrate to AWS.

```
CREATE USER 'your_user'@'%' IDENTIFIED BY 'your_password';
GRANT SELECT, RELOAD, LOCK TABLES, REPLICATION SLAVE, BINLOG MONITOR, SHOW VIEW ON  *.*
 TO 'your_user'@'%';
```

In the preceding example, replace each *user input placeholder* with your own information.

The following sections describe specific configuration prerequisites for self-managed and AWS-managed MySQL databases.

Topics

Using a self-managed MySQL compatible database as a source for homogeneous data migrations

This section describes how to configure your MySQL compatible databases that are hosted on-premises or on Amazon EC2 instances.

Check the version of your source MySQL or MariaDB database. Make sure that AWS DMS supports your source MySQL or MariaDB database version as described in Sources for DMS homogeneous data migrations.

To use CDC, make sure to enable binary logging. To enable binary logging, configure the following parameters in the `my.ini` (Windows) or `my.cnf` (UNIX) file of your MySQL or MariaDB database.

Parameter	Value
server-id	Set this parameter to a value of 1 or greater.
log-bin	Set the path to the binary log file, such as log-bin=E:\MySql_Logs\BinLog . Don't include the file extension.
binlog_format	Set this parameter to ROW. We recommend this setting during replication because in certain cases when binlog_format is set to STATEMENT , it can cause inconsistency when replicating data to the target. The database engine also writes similar inconsistent data to the target when binlog_format is set to MIXED, because the database engine automatically switches to STATEMENT -based logging.
expire_logs_days	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.
binlog_checksum	Set this parameter to NONE.
binlog_row_image	Set this parameter to FULL.
log_slave_updates	Set this parameter to TRUE if you are using a MySQL or MariaDB replica as a source.

Using an AWS-managed MySQL compatible database as a source for homogeneous data migrations in AWS DMS

This section describes how to configure your Amazon RDS for MySQL and Amazon RDS for MariaDB database instances.

When you use an AWS-managed MySQL or MariaDB database as a source for homogeneous data migrations in AWS DMS, make sure that you have the following prerequisites for CDC:

- To enable binary logs for RDS for MySQL and MariaDB, enable automatic backups at the instance level. To enable binary logs for an Aurora MySQL cluster, change the variable binlog_format in the parameter group. You don't need to enable automatic backups for an Aurora MySQL cluster.

Next, set the `binlog_format` parameter to `ROW`.

For more information about setting up automatic backups, see [Enabling automated backups](#) in the *Amazon RDS User Guide*.

For more information about setting up binary logging for an Amazon RDS for MySQL or MariaDB database, see [Setting the binary logging format](#) in the *Amazon RDS User Guide*.

For more information about setting up binary logging for an Aurora MySQL cluster, see [How do I turn on binary logging for my Amazon Aurora MySQL cluster?](#).

- Ensure that the binary logs are available to AWS DMS. Because AWS-managed MySQL and MariaDB 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_row_image` parameter to `Full`.
- Set the `binlog_checksum` parameter to `NONE`.
- If you are using an Amazon RDS MySQL or MariaDB replica as a source, enable backups on the read replica, and ensure the `log_slave_updates` parameter is set to `TRUE`.

Limitations for using a MySQL compatible database as a source for homogeneous data migrations

The following limitations apply when using a MySQL compatible database as a source for homogeneous data migrations:

- MariaDB objects such as sequences are not supported in homogeneous migration tasks.
- Migration from MariaDB to Amazon RDS MySQL/Aurora MySQL might fail due to incompatible object differences.
- The username you use to connect to your data source has the following limitations:
  - Can be 2 to 64 characters in length.
  - Can't have spaces.
  - Can include the following characters: a-z, A-Z, 0-9, underscore (_).
  - Must start with a-z or A-Z.

- The password you use to connect to your data source has the following limitations:
  - Can be 1 to 128 characters in length.
  - Can't contain any of the following: single quote ('), double quote ("), semicolon (;) or space.
- AWS DMS homogeneous data migrations creates unencrypted MySQL and MariaDB objects on the target Amazon RDS instances even if the source objects were encrypted. RDS for MySQL doesn't support the MySQL keyring_aws AWS Keyring Plugin required for encrypted objects. Refer to the MySQL Keyring Plugin not supported documentation in the Amazon RDS User Guide
- AWS DMS does not use Global Transaction Identifiers (GTIDs) for for data replication even if the source data contains them.

Using a PostgreSQL database as a source for homogeneous data migrations in AWS DMS

You can use a PostgreSQL database as a source for Homogeneous data migrations in AWS DMS. In this case, your source data provider can be an on-premises, Amazon EC2, or RDS for PostgreSQL database.

To run homogeneous data migrations, grant superuser permissions for the database user that you specified in AWS DMS for your PostgreSQL source database. The database user needs superuser permissions to access replication-specific functions in the source. For a full load data migration, your database user needs `SELECT` permissions on tables to migrate them.

Use the following script to create a database user with the required permissions in your PostgreSQL source database. Run the `GRANT` query for all databases that you migrate to AWS.

```
CREATE USER your_user WITH LOGIN PASSWORD 'your_password';
ALTER USER your_user WITH SUPERUSER;
GRANT SELECT ON ALL TABLES IN SCHEMA schema_name TO your_user;
```

In the preceding example, replace each *`user input placeholder`* with your own information.

AWS DMS supports 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. Here, you can use additional configuration settings and use table replica identity as a workaround. However, this approach can generate extra logs. We recommend that you use table replica identity as a workaround only after careful testing. For more information, see Additional configuration settings when using a PostgreSQL database as a DMS source.

The following sections describe specific configuration prerequisites for self-managed and AWS-managed PostgreSQL databases.

Topics

Using a self-managed PostgreSQL database as a source for homogeneous data migrations in AWS DMS

This section describes how to configure your PostgreSQL databases that are hosted on-premises or on Amazon EC2 instances.

Check the version of your source PostgreSQL database. Make sure that AWS DMS supports your source PostgreSQL database version as described in Sources for DMS homogeneous data migrations.

Homogeneous data migrations support change data capture (CDC) using logical replication. To turn on logical replication on a self-managed PostgreSQL source database, set the following parameters and values in the `postgresql.conf` configuration file:

- Set `wal_level` to `logical`.
- Set `max_replication_slots` to a value greater than 1.

  Set the `max_replication_slots` value according to the number of tasks that you want to run. For example, to run five tasks you 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. Make sure 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.

- The `wal_sender_timeout` parameter ends replication connections that are inactive longer than the specified number of milliseconds. The default is 60000 milliseconds (60 seconds). Setting the value to 0 (zero) disables the timeout mechanism, and is a valid setting for DMS.

Some parameters are static, and you can only set them at server start. Any changes to their entries in the configuration file are ignored until the server is restarted. For more information, see the PostgreSQL documentation.

Using an AWS-managed PostgreSQL database as a source for homogeneous data migrations in AWS DMS

This section describes how to configure your Amazon RDS for PostgreSQL database instances.

Use the AWS master user account for the PostgreSQL DB instance as the user account for the PostgreSQL source data provider for homogeneous data migrations in 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, then 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.

Use the following code example grant the `rds_superuser` and `rds_replication` roles.

```
GRANT rds_superuser to your_user;
GRANT rds_replication to your_user;
```

In the preceding example, replace *`your_user`* with the name of your database user.

To turn on logical replication, 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.

Limitations for using a PostgreSQL compatible database as a source for homogeneous data migrations

The following limitations apply when using a PostgreSQL compatible database as a source for homogeneous data migrations:

- The username you use to connect to your data source has the following limitations:
  - Can be 2 to 64 characters in length.

  - Can't have spaces.
  - Can include the following characters: a-z, A-Z, 0-9, underscore (_).
  - Must start with a-z or A-Z.
- The password you use to connect to your data source has the following limitations:
  - Can be 1 to 128 characters in length.
  - Can't contain any of the following: single quote ('), double quote ("), semicolon (;) or space.

Using a MongoDB compatible database as a source for homogeneous data migrations in AWS DMS

You can use a MongoDB-compatible database as a source for Homogeneous data migrations in AWS DMS. In this case, your source data provider can be an on-premises, Amazon EC2 for MongoDB database or Amazon DocumentDB (with MongoDB compatibility) database.

For supported database versions, see Source data providers for DMS homogeneous data migrations.

The following sections describe specific configuration prerequisites for self-managed MongoDB databases and AWS-managed Amazon DocumentDB databases.

Topics

- Using a self-managed MongoDB database as a source for homogeneous data migrations in AWS DMS
- Using an Amazon DocumentDB database as a source for homogeneous data migrations in AWS DMS
- Features for using a MongoDB-compatible database as a source for homogeneous data migrations
- Limitations for using a MongoDB-compatible database as a source for homogeneous data migrations
- Best practices for using a MongoDB-compatible database as a source for homogeneous data migrations

Using a self-managed MongoDB database as a source for homogeneous data migrations in AWS DMS

This section describes how to configure your MongoDB databases that are hosted on-premises or on Amazon EC2 instances.

Check the version of your source MongoDB database. Make sure that AWS DMS supports your source MongoDB database version as described in Source data providers for DMS homogeneous data migrations.

To run homogeneous data migrations 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. For more information about user creation, see Permissions needed when using MongoDB as a source for AWS DMS.

To use ongoing replication or CDC with MongoDB, AWS DMS requires access to the MongoDB operations log (oplog). For more information, see Configuring a MongoDB replica set for CDC.

For information about MongoDB authentication methods, see Security requirements when using MongoDB as a source for AWS DMS.

For MongoDB as a source, homogeneous data migrations supports all of the datatypes that Amazon DocumentDB supports.

For MongoDB as a source, to store user credentials in Secrets Manager, you need to provide them in plain text, using the **Other type of secrets** type. For more information, see Using secrets to access AWS Database Migration Service endpoints.

The following code sample demonstrates how to store database secrets using plain text.

```
{
  "username": "dbuser",
  "password": "dbpassword"
}
```

Using an Amazon DocumentDB database as a source for homogeneous data migrations in AWS DMS

This section describes how to configure your Amazon DocumentDB database instances for use as a source for homogeneous data migrations.

Use the master username for the Amazon DocumentDB instance as the user account for the MongoDB-compatible source data provider for homogeneous data migrations in 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, then the account must have the root role. For more information on the user creation as a root account, see Setting permissions to use Amazon DocumentDB as a source.

To turn on logical replication, set the `change_stream_log_retention_duration` parameter in your database parameter group to a setting appropriate for your transaction workload. Changing this static parameter requires you to reboot your DB instance to take effect. Before starting data migration for all the task types including Full Load Only, enable Amazon DocumentDB change streams for all collections within a given database, or only for selected collections. For more information about enabling change streams for Amazon DocumentDB, see Enabling Change Streams in the *Amazon DocumentDB developer guide*.

> **Note**
>
> AWS DMS uses the Amazon DocumentDB change stream to capture changes during ongoing replication. If Amazon DocumentDB flushes out the records from the change stream before DMS reads them, your tasks will fail. We recommend setting the `change_stream_log_retention_duration` parameter to retain changes for at least 24 hours.

To use Amazon DocumentDB for homogeneous data migration, store user credentials in Secrets Manager under **Credentials for Amazon DocumentDB database**.

Features for using a MongoDB-compatible database as a source for homogeneous data migrations

- You can migrate all the secondary indexes that Amazon DocumentDB supports during the Full load phase.
- AWS DMS migrates collections in parallel. homogeneous data migrations calculates segments at runtime based on the average size of each document in the collection for maximum performance.
- DMS can replicate the secondary indexes that you create in the CDC phase. DMS supports this feature in MongoDB version 6.0.

- DMS supports documents with a nesting level greater than 97.

Limitations for using a MongoDB-compatible database as a source for homogeneous data migrations

- Documents can't have field names with a $ prefix.
- AWS DMS doesn't support time series collection migration.
- AWS DMS doesn't support `create`, `drop`, or `rename collection` DDL events during the CDC phase.
- AWS DMS doesn't suport inconsistent datatypes in the collection for the `_id` field. For example, the following unsupported collection has multiple data types for the `_id` field.

```
rs0 [direct: primary] test> db.collection1.aggregate([
...   {
...     $group: {
...       _id: { $type: "$_id" },
...       count: { $sum: 1 }
...     }
...   }
... ])
[ { _id: 'string', count: 6136 }, { _id: 'objectId', count: 848033 } ]
```

- For CDC-only tasks, AWS DMS only supports the `immediate` start mode.
- AWS DMS doesn't support documents with invalid UTF8 characters.
- AWS DMS doesn't support sharded collections.

Best practices for using a MongoDB-compatible database as a source for homogeneous data migrations

- For multiple large databases and collections hosted on same MongoDB instance, we recommend you use selection rules for each database and collection to split the task between multiple data migration tasks and projects. You can tune your database and collection divisions for maximum performance.

Creating and setting a target database to work with AWS DMS schema conversion

You can use MySQL-compatible, PostgreSQL, and Amazon DocumentDB databases as a target data provider for homogeneous data migrations in AWS DMS.

For supported database versions, see Target data providers for DMS homogeneous data migrations.

Your target data provider can be an Amazon RDS DB instance or an Amazon Aurora DB cluster. Note that the database version of your target data provider must be equal or higher than the database version of your source data provider.

Topics

Using a MySQL compatible database as a target for homogeneous data migrations in AWS DMS

You can use a MySQL compatible database as a migration target for homogeneous data migrations in AWS DMS.

AWS DMS requires certain permissions to migrate data into your target Amazon RDS for MySQL or MariaDB or Amazon Aurora MySQL database. Use the following script to create a database user with the required permissions in your MySQL target database.

In this example, replace each *user input placeholder* with your own information. If your target MariaDB database version is lower than 10.5, then you can skip the `GRANT SLAVE MONITOR` command.

```
CREATE USER 'your_user'@'%' IDENTIFIED BY 'your_password';

GRANT ALTER, CREATE, DROP, INDEX, INSERT, UPDATE, DELETE, SELECT, CREATE VIEW, CREATE
 ROUTINE, ALTER ROUTINE, EVENT, TRIGGER, EXECUTE, REFERENCES ON *.* TO 'your_user'@'%';
GRANT REPLICATION SLAVE, REPLICATION CLIENT  ON *.* TO 'your_user'@'%'; GRANT SLAVE
 MONITOR  ON *.* TO 'your_user'@'%';
```

In the preceding example, replace each *user input placeholder* with your own information.

Use the following script to create a database user with the required permissions in your MariaDB database. Run the GRANT queries for all databases that you migrate to AWS.

```
CREATE USER 'your_user'@'%' IDENTIFIED BY 'your_password';
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, INDEX, ALTER, CREATE VIEW, CREATE
 ROUTINE, ALTER ROUTINE, EVENT, TRIGGER, EXECUTE,SLAVE MONITOR, REPLICATION SLAVE ON
 *.* TO 'your_user'@'%';
```

In the preceding example, replace each *user input placeholder* with your own information.

> **Note**
>
> In Amazon RDS, when you turn on automated backup for a MySQL/Maria database instance, you also turn on binary logging. When these settings are enabled, your data migration task may fail with the following error while creating secondary objects such as functions, procedures, and triggers on the target database. If your target database has binary logging enabled, then set `log_bin_trust_function_creators` to `true` in the database parameter group before starting the task.
>
> ```
> ERROR 1419 (HY000): You don't have the SUPER privilege and binary logging is
>  enabled (you might want to use the less safe log_bin_trust_function_creators
>  variable)
> ```

Limitations for using a MySQL compatible database as a target for homogeneous data migrations

The following limitations apply when using a MySQL compatible database as a target for homogeneous data migrations:

- The username you use to connect to your data source has the following limitations:
  - Can be 2 to 64 characters in length.
  - Can't have spaces.
  - Can include the following characters: a-z, A-Z, 0-9, underscore (_).
  - Can't include a hyphen (-).

- Must start with a-z or A-Z.
- The password you use to connect to your data source has the following limitations:
  - Can be 1 to 128 characters in length.
  - Can't contain any of the following: single quote ('), double quote ("), semicolon (;) or space.

Using a PostgreSQL database as a target for homogeneous data migrations in AWS DMS

You can use a PostgreSQL database as a migration target for homogeneous data migrations in AWS DMS.

AWS DMS requires certain permissions to migrate data into your target Amazon RDS for PostgreSQL or Amazon Aurora PostgreSQL database. Use the following script to create a database user with the required permissions in your PostgreSQL target database.

```
CREATE USER your_user WITH LOGIN PASSWORD 'your_password';
GRANT USAGE ON SCHEMA schema_name TO your_user;
GRANT CONNECT ON DATABASE db_name to your_user;
GRANT CREATE ON DATABASE db_name TO your_user;
GRANT CREATE ON SCHEMA schema_name TO your_user;
GRANT UPDATE, INSERT, SELECT, DELETE, TRUNCATE ON ALL TABLES IN SCHEMA schema_name
 TO your_user;
            #For "Full load and change data capture (CDC)" and "Change data capture
 (CDC)" data migrations, setting up logical replication requires rds_superuser
 privileges
GRANT rds_superuser TO your_user;
```

In the preceding example, replace each *`user input placeholder`* with your own information.

To turn on logical replication for your RDS for PostgreSQL target, set the `rds.logical_replication` parameter in your DB parameter group to 1. This static parameter requires a reboot of the DB instance or DB cluster to take effect. Some parameters are static, and you can only set them at server start. AWS DMS ignores changes to their entries in the DB parameter group until you restart the server.

PostgreSQL uses triggers to implement foreign key constraints. During the full load phase, AWS DMS loads each table one at a time. We recommend that you turn off foreign key constraints on your target database during the full load. To do so, use one of the following methods.

- Temporarily turn off all triggers for your instance, and finish the full load.
- Change the value of 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 you set the `session_replication_role` parameter to `replica`, only triggers in the `replica` state are active. Otherwise, the triggers remain inactive.

Limitations for using a PostgreSQL compatible database as a target for homogeneous data migrations

The following limitations apply when using a PostgreSQL compatible database as a target for homogeneous data migrations:

- The username you use to connect to your data source has the following limitations:
  - Can be 2 to 64 characters in length.
  - Can't have spaces.
  - Can include the following characters: a-z, A-Z, 0-9, underscore (_).
  - Must start with a-z or A-Z.
- The password you use to connect to your data source has the following limitations:
  - Can be 1 to 128 characters in length.
  - Can't contain any of the following: single quote ('), double quote ("), semicolon (;) or space.

Using an Amazon DocumentDB database as a target for homogeneous data migrations in AWS DMS

You can use an Amazon DocumentDB (with MongoDB compatibility) database and DocumentDB Elastic cluster as a migration target for homogeneous data migrations in AWS DMS.

To run homogeneous data migrations for an Amazon DocumentDB target, you can create either a user account with administrator privileges, or a user with read/write permissions only on the database to migrate.

Homogeneous data migrations supports all of the BSON data types that Amazon DocumentDB supports. For a list of these data types, see [Data Types](#) in the *Amazon DocumentDB Developer Guide*.

To use shard features of DocumentDB Elastic cluster for migrating non-sharded collection from the source, create a shard collection to migrate prior to starting the data migration task. For more information about shard collection in an Amazon DocumentDB Elastic cluster, see [Step 5: Shard your collection] in the *Amazon DocumentDB Developer Guide*.

For an Amazon DocumentDB target, AWS DMS supports the `none` or `require` SSL modes.

Running homogeneous data migrations in AWS DMS

You can use Homogeneous data migrations in AWS DMS to migrate data from your source database to the equivalent engine on Amazon Relational Database Service (Amazon RDS) , Amazon Aurora, or Amazon DocumentDB. AWS DMS automates the data migration process by using native database tools in your source and target databases.

After you create an instance profile and compatible data providers for homogeneous data migrations, create a migration project. For more information, see Creating migration projects.

The following sections describe how to create, configure, and run homogeneous data migrations.

Topics

Creating a data migration in AWS DMS

After you create a migration project with compatible data providers of the same type, you can use this project for homogeneous data migrations. For more information, see Creating migration projects.

To start using homogeneous data migrations, create a new data migration. You can create several homogeneous data migrations of different types in a single migration project.

AWS DMS has the maximum number of homogeneous data migrations that you can create for your AWS account. See the following section for information about AWS DMS service quotas Quotas for AWS Database Migration Service.

Before you create a data migration, make sure that you set up the required resources such as your source and target databases, an IAM policy and role, an instance profile, and data providers. For more information, see Creating IAM resources, Creating instance profiles, and Creating data providers.

Also, we recommend that you don't use homogeneous data migrations to migrate data from a higher database version to a lower database version. Check the versions of databases that you use for source and target data providers, and upgrade your target database version, if needed.

To create a data migration

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Migration projects**. The **Migration projects** page opens.
3. Choose your migration project, and on the **Data migrations** tab, choose **Create data migration**.
4. For **Name**, enter a name for your data migration. Make sure that you use a unique name for your data migration so that you can easily identify it.
5. For **Replication type**, choose the type of data migration that you want to configure. You can choose one of the following options.

   - **Full load** — Migrates your existing source data.
   - **Full load and change data capture (CDC)** — Migrates your existing source data and replicates ongoing changes.
   - **Change data capture (CDC)** — Replicates ongoing changes.
6. Select the check box for **Turn on CloudWatch logs** to store data migration logs in Amazon CloudWatch. If you don't choose this option, then you can't see the log files when your data migration fails.
7. (Optional) Expand **Advanced settings**. For **Number of jobs**, enter the number of parallel threads that AWS DMS can use to migrate your source data to the target.

8. For **IAM service role**, choose the IAM role that you created in prerequisites. For more information, see Creating an IAM role for homogeneous data migrations in AWS DMS.
9. Configure the **Start mode** for data migrations of the **Change data capture (CDC)** type. You can choose one of the following options.
   - **Immediately** — Starts the ongoing replication when you start your data migration.
   - **Using a native start point** — Starts the ongoing replication from the specified point.

     For PostgreSQL databases, enter the name of the logical replication slot for **Slot name** and enter the transaction log sequence number for **Native start point**.

     For MySQL databases, enter the transaction log sequence number for **Log sequence number (LSN)**.
10. Configure the **Stop mode** for data migrations of the **Change data capture (CDC)** or **Full load and change data capture (CDC)** type. You can choose one of the following options.
    - **Don't stop CDC** — AWS DMS continues the ongoing replication until you stop your data migration.
    - **Using a server time point** — AWS DMS stops the ongoing replication at the specified time.

      If you choose this option, then for **Stop date and time**, enter the date and time when you want to automatically stop the ongoing replication.
11. Choose **Create data migration**.

AWS DMS creates your data migration and adds it to the list on the **Data migrations** tab in your migration project. Here you can see the status of your data migration. For more information, see Migration statuses.

Important

For data migrations of the **Full load** and **Full load and change data capture (CDC)** type, AWS DMS deletes all data, tables, and other database objects on your target database. Make sure you have a backup of your target database.

After AWS DMS creates your data migration, the status of this data migration is set to **Ready**. To migrate your data, you must start the data migration manually. To do so, choose your data migration from the list. Next, for **Actions**, choose **Start**. For more information, see Managing data migrations.

The first launch of a homogeneous data migration requires some setup. AWS DMS creates a serverless environment for your data migration. This process takes up to 15 minutes. After you stop and restart your data migration, AWS DMS doesn't create the environment again, and you can access your data migration faster.

Selection rules for homogeneous data migrations

You can use selection rules to choose the schema, tables, or both that you want to include in your replication.

When creating data migration task, choose **Add selection rule**.

For the rule settings, provide the following values:

- **Schema**: Choose **Enter a schema**.
- **Schema name**: Provide the name of the schema you want to replicate, or use `%` as a wildcard.
- **Table name**: : Provide the name of the table you want to replicate, or use `%` as a wildcard.

By default, the only rule-action that DMS supports is `Include`, and the only wildcard character that DMS supports is `%`.

> **Note**
>
> The support for selection rules AWS DMS for homogeneous data migrations varies based on the combination of the source database engine and the migration type chosen. PostgreSQL and MongoDB-compatible sources allow selection rules for all migration types, while MySQL sources only support selection rules for the Full Load migration type.

Example Migrate all tables in a schema

The following example migrates all tables from a schema named `dmsst` in your source to your target endpoint.

```
{
    "rules": [
```

```
        {
            "rule-type": "selection",
            "rule-action": "include",
            "object-locator": {
                "schema-name": "dmsst",
                "table-name": "%"
            },
            "filters": [],
            "rule-id": "1",
            "rule-name": "1"
        }
    ]
}
```

Example Migrate some tables in a schema

The following example migrates all tables with a name starting with `collectionTest`, from a schema named `dmsst` in your source to your target endpoint.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-action": "include",
            "object-locator": {
                "schema-name": "dmsst",
                "table-name": "collectionTest%"
            },
            "filters": [],
            "rule-id": "1",
            "rule-name": "1"
        }
    ]
}
```

Example Migrate specific tables from multiple schemas

The following example migrates some of the tables from multiple schemas named `dmsst` and `Test` in your source to your target endpoint.

```
{
    "rules": [
        {
```

```
            "rule-type": "selection",
            "rule-action": "include",
            "object-locator": {
                "schema-name": "dmsst",
                "table-name": "collectionTest1"
            },
            "filters": [],
            "rule-id": "1",
            "rule-name": "1"
        },
        {
            "rule-type": "selection",
            "rule-action": "include",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "products"
            },
            "filters": [],
            "rule-id": "2",
            "rule-name": "2"
        }
    ]
}
```

Managing data migrations in AWS DMS

After you create a data migration, AWS DMS doesn't automatically start migrating data. You start a data migration manually when needed.

Before you start a data migration, you can modify all settings of your data migration. After you start your data migration, you can't change the replication type. To use another replication type, create a new data migration.

To start a data migration

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Migration projects**. The **Migration projects** page opens.
3. Choose your migration project. On the **Data migrations** tab, choose your data migration. The **Summary** page for your data migration opens.
4. For **Actions**, choose **Start**.

After this, AWS DMS creates a serverless environment for your data migration. This process takes up to 15 minutes.

After you start a data migration, AWS DMS sets its status to **Starting**. The next status that AWS DMS uses for your data migration, depends on the type of replication that you choose in the data migration settings. For more information, see Migration statuses.

To modify a data migration

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Migration projects**. The **Migration projects** page opens.
3. Choose your migration project. On the **Data migrations** tab, choose your data migration. The **Summary** page for your data migration opens.
4. Choose **Modify**.
5. Configure the settings for your data migration.

> **Important**
>
> If you have started a data migration, then you can't change the replication type.

6. To view your data migration logs in Amazon CloudWatch, select the check box for **Turn on CloudWatch logs**.
7. Choose **Save changes**.

After AWS DMS starts a data migration, you can stop it. To do so, choose your data migration on the **Data migrations** tab. Next, for **Actions**, choose **Stop**.

After you stop a data migration, AWS DMS sets its status to **Stopping**. Next, AWS DMS sets the status of this data migration to **Stopped**. After AWS DMS stops a data migration, you can modify, resume, restart, or delete your data migration.

To continue the data replication, choose the data migration that you stopped on the **Data migrations** tab. Next, for **Actions**, choose **Resume processing**.

To restart the data load, choose the data migration that you stopped on the **Data migrations** tab. Next, for **Actions**, choose **Restart**. AWS DMS deletes all data from your target database and starts the data migration from scratch.

You can delete a data migration that you have stopped or that you haven't started. To delete a data migration, choose it on the **Data migrations** tab. Next, for **Actions**, choose **Delete**. To delete your migration project, stop and delete all data migrations.

Monitoring data migrations in AWS DMS

After you start your homogeneous data migration, you can monitor its status and progress. Data migrations of large data sets such as hundreds of gigabytes take hours to complete. To maintain the reliability, availability, and high performance of your data migration, monitor its progress regularly.

To check the status and progress of your data migration

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Migration projects**. The **Migration projects** page opens.
3. Choose your migration project and navigate to the **Data migrations** tab.
4. For your data migration, see the **Status** column. For more information about values in this column, see Migration statuses.
5. For a running data migration, the **Migration progress** column displays the percentage of migrated data.

To check the details of your data migration

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Migration projects**. The **Migration projects** page opens.
3. Choose your migration project. On the **Data migrations** tab, choose your data migration.
4. On the **Details** tab, you can see the migration progress. Particularly, you can see the following metrics.

   - **Public IP address –** The public IP address of your data migration. You need this value to configure a network. For more information, see Setting up a network.

- **Tables loaded** – The number of successfully loaded tables.
- **Tables loading** – The number of tables currently loading.
- **Tables queued** – The number of tables currently waiting to be loaded.
- **Tables errored** – The number of tables that failed to load.
- **Elapsed time** – The amount of time that passed after the start of your data migration.
- **CDC latency** – The average time that passes between when a change occurs on a source table and when AWS DMS applies this change to the target table.
- **Migration started** – The time when you started this data migration.
- **Migration stopped** – The time when you stopped this data migration.

5. To view the log files for your data migration, choose **View CloudWatch logs** under **Homogeneous data migration settings**. You can **Turn on CloudWatch logs** when you create or modify a data migration. For more information, see Creating a data migration and Managing data migrations.

You can use Amazon CloudWatch alarms or events to closely track your data migration. For more information, 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.

For homogeneous data migrations, AWS DMS includes the following metrics in Amazon CloudWatch.

Metric	Description
OverallCDCLatency	The overall latency during the CDC phase. For MySQL databases, this metric shows the number of seconds that passes between the change in the source binary log and the replication of this change. For PostgreSQL databases, this metric shows the number of seconds that passes between `last_msg_receipt_time` and `last_msg_send_time` from the `pg_stat_subscription` view. Units: Seconds

Metric	Description
StorageConsumption	The storage that your data migration consumes. Units: Bytes

Statuses of homogeneous data migrations in AWS DMS

For each data migration that you run, AWS DMS displays the **Status** in the AWS DMS console. The following list includes the available statuses.

- `Creating` – AWS DMS is creating the data migration.
- `Ready` – The data migration is ready to start.
- `Starting` – AWS DMS is creating the serverless environment for your data migration. This process takes up to 15 minutes.
- `Load running` – AWS DMS is performing the full load migration.
- `Load complete, replication ongoing` – AWS DMS completed the full load and now replicates ongoing changes. AWS DMS uses this status only for data migrations of the full load and change data capture (CDC) type.
- `Replication ongoing` – AWS DMS is replicating ongoing changes. AWS DMS uses this status only for migrations of the change data capture (CDC) type.
- `Reloading target` – AWS DMS is restarting a data migration and performs the specified migration type.
- `Stopping` – AWS DMS is stopping the data migration. AWS DMS sets this status after you choose to stop the data migration on the **Actions** menu.
- `Stopped` – AWS DMS has stopped the data migration.
- `Failed` – The data migration has failed. For more information, see the log files.

  To view the log files, choose your data migration on the **Data migrations** tab. Next, choose **View CloudWatch logs** under **Homogeneous data migration settings**.

> **Important**
>
> You can view log files if you select the check box for **Turn on CloudWatch logs** when you create your data migration.

- `Deleting` – AWS DMS is deleting the data migration. AWS DMS sets this status after you choose to delete the data migration on the **Actions** menu.
- `Maintenance` – AWS DMS puts a task in maintenance mode status when new image is deployed on the underlying serverless container associated with your data migration task.

Migrating data from MySQL databases with homogeneous data migrations in AWS DMS

You can use Homogeneous data migrations to migrate a self-managed MySQL database to RDS for MySQL or Aurora MySQL. AWS DMS creates a serverless environment for your data migration. For different types of data migrations, AWS DMS uses different native MySQL database tools.

For homogeneous data migrations of the **Full load** type, AWS DMS uses mydumper to read data from your source database and store it on the disk attached to the serverless environment. After AWS DMS reads all your source data, it uses myloader in the target database to restore your data.

For homogeneous data migrations of the **Full load and change data capture (CDC)** type, AWS DMS uses mydumper to read data from your source database and store it on the disk attached to the serverless environment. After AWS DMS reads all your source data, it uses myloader in the target database to restore your data. After AWS DMS completes the full load, it sets up the binlog replication with the binlog position set to the start of the full load.

For homogeneous data migrations of the **Change data capture (CDC)** type, AWS DMS requires the **Native CDC start point** to start the replication. If you provide the native CDC start point, then AWS DMS captures changes from that point. Alternatively, choose **Immediately** in the data migration settings to automatically capture the start point for the replication when the actual data migration starts.

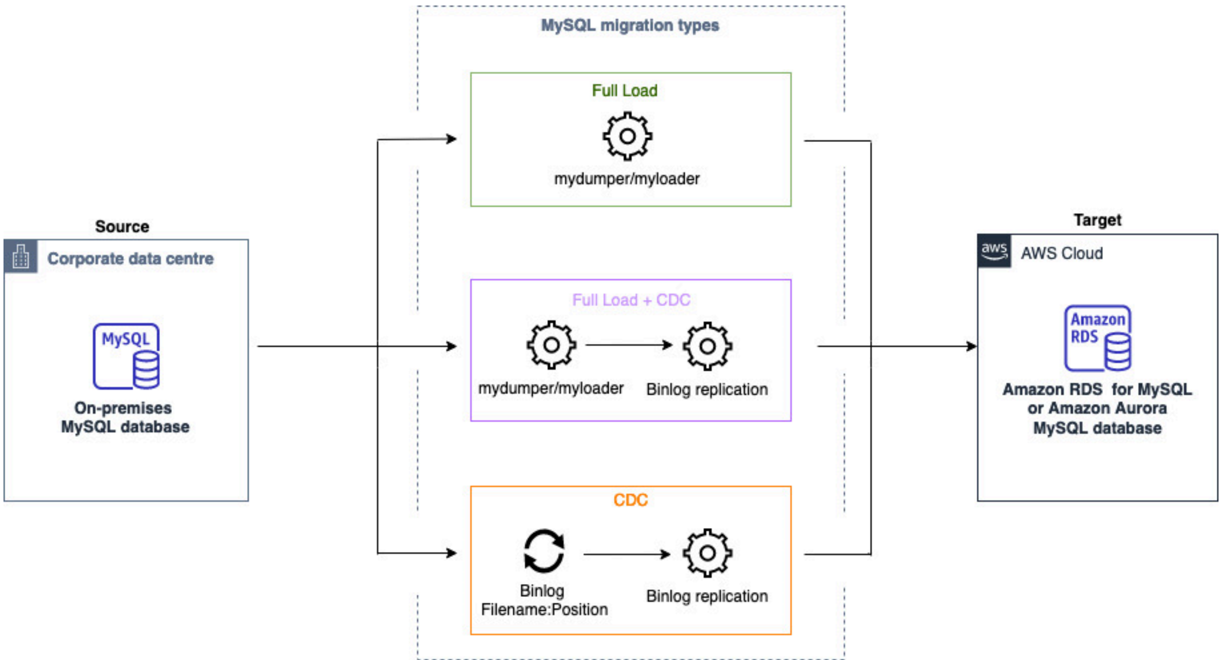
> **Note**
>
> For a CDC-only migration to work properly, all source database schemas and objects must already be present on the target database. The target may have objects that are not present on the source, however.

You can use the following code example to get the current log sequence number (LSN) in your MySQL database.

```
show master status
```

This query returns a binlog file name and the position. For the native start point, use a combination of the binlog file name and the position. For example, `mysql-bin-changelog.000024:373`. In this example, `mysql-bin-changelog.000024` is the binlog file name and `373` is the position where AWS DMS starts capturing changes.

The following diagram shows the process of using homogeneous data migrations in AWS DMS to migrate a MySQL database to RDS for MySQL or Aurora MySQL.



Migrating data from PostgreSQL databases with homogeneous data migrations in AWS DMS

You can use Homogeneous data migrations to migrate a self-managed PostgreSQL database to RDS for PostgreSQL or Aurora PostgreSQL. AWS DMS creates a serverless environment for your data migration. For different types of data migrations, AWS DMS uses different native PostgreSQL database tools.

For homogeneous data migrations of the **Full load** type, AWS DMS uses pg_dump to read data from your source database and store it on the disk attached to the serverless environment. After AWS DMS reads all your source data, it uses pg_restore in the target database to restore your data.

For homogeneous data migrations of the **Full load and change data capture (CDC)** type, AWS DMS uses pg_dump to read schema objects without table data from your source database and store them on the disk attached to the serverless environment. It then uses `pg_restore` in the target database to restore your schema objects. After AWS DMS completes the `pg_restore` process, it automatically switches to a publisher and subscriber model for logical replication with the `Initial Data Synchronization` option to copy initial table data directly from the source database to the target database, and then initiates ongoing replication. In this model, one or more subscribers subscribe to one or more publications on a publisher node.

For homogeneous data migrations of the **Change data capture (CDC)** type, AWS DMS requires the native start point to start the replication. If you provide the native start point, then AWS DMS captures changes from that point. Alternatively, choose **Immediately** in the data migration settings to automatically capture the start point for the replication when the actual data migration starts.

> **Note**
>
> For a CDC-only migration to work properly, all source database schemas and objects must already be present on the target database. The target may have objects that are not present on the source, however.

You can use the following code example to get the native start point in your PostgreSQL database.

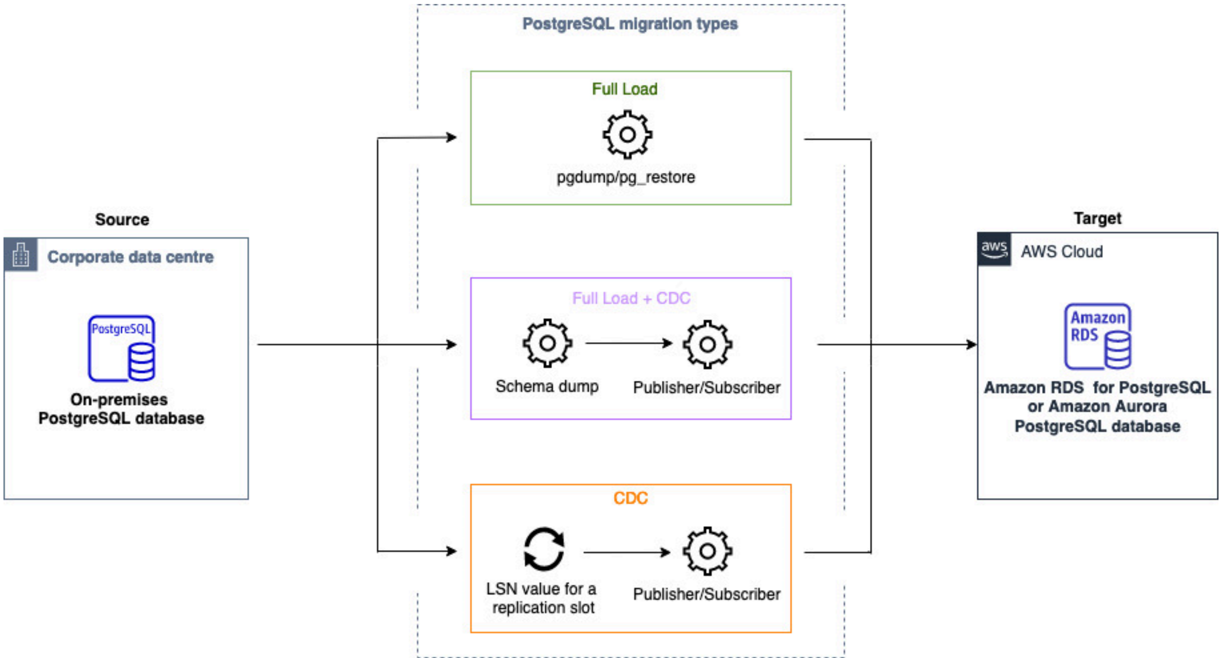
```
select confirmed_flush_lsn from pg_replication_slots where
 slot_name=‘migrate_to_target';
```

This query uses the `pg_replication_slots` view in your PostgreSQL database to capture the log sequence number (LSN) value.

After AWS DMS sets the status of your PostgreSQL homogeneous data migration to **Stopped**, **Failed**, or **Deleted**, the publisher and replication aren't removed. If you don't want to resume the migration, then delete the replication slot and the publisher by using the following command.

```
SELECT pg_drop_replication_slot('migration_subscriber_{ARN}');
            DROP PUBLICATION publication_{ARN};
```

The following diagram shows the process of using homogeneous data migrations in AWS DMS to migrate a PostgreSQL database to RDS for PostgreSQL or Aurora PostgreSQL.



Best practices for using a PostgreSQL database as a source for homogeneous data migrations

- To speed up initial data syncing on the subscriber side for FLCDC task, you must adjust `max_logical_replication_workers` and `max_sync_workers_per_subscription`. Increasing these values enhances table synchronization speed.
  - **max_logical_replication_workers** – Specifies maximum number of logical replication workers. This includes both the apply workers on the subscriber side and the table synchronization workers.
  - **max_sync_workers_per_subscription** – Increasing `max_sync_workers_per_subscription` only affects the number of tables that are synchronized in parallel, not the number of workers per table.

> **Note**
>
> `max_logical_replication_workers` should not exceed `max_worker_processes`, and `max_sync_workers_per_subscription` should be less than or equal to `max_logical_replication_workers`.

- For migrating large tables, consider dividing them into separate tasks using selection rules. For example, you can divide large tables into separate individual tasks and small tables into another single task.

- Monitor disk and CPU usage on the subscriber side to maintain optimal performance.

Migrating data from MongoDB databases with homogeneous data migrations in AWS DMS

You can use Homogeneous data migrations to migrate a self-managed MongoDB database to Amazon DocumentDB. AWS DMS creates a serverless environment for your data migration. For different types of data migrations, AWS DMS uses different native MongoDB database tools.

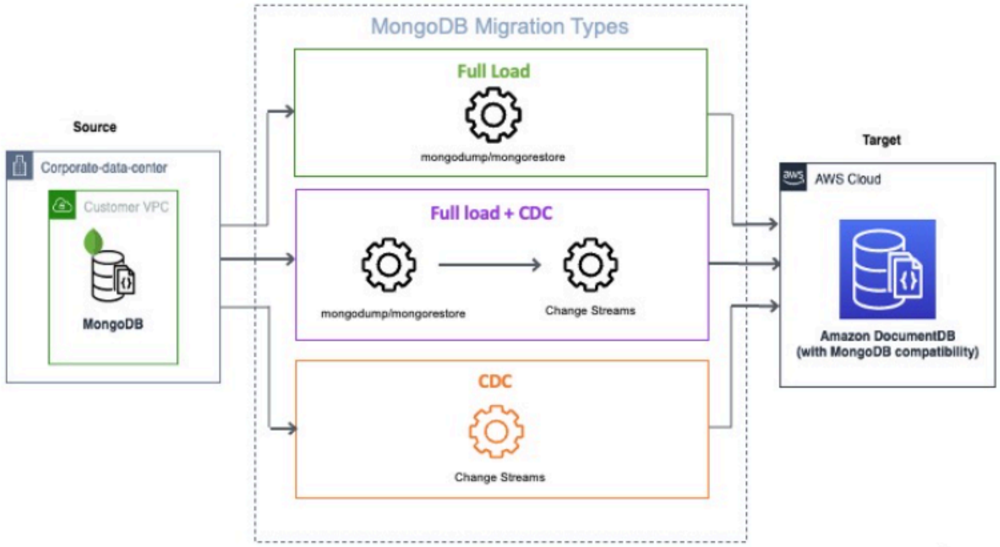
For homogeneous data migrations of the **Full load** type, AWS DMS uses `mongodump` to read data from your source database and store it on the disk attached to the serverless environment. After AWS DMS reads all your source data, it uses `mongorestore` in the target database to restore your data.

For homogeneous data migrations of the **Full load and change data capture (CDC)** type, AWS DMS uses `mongodump` to read data from your source database and store it on the disk attached to the serverless environment. After AWS DMS reads all your source data, it uses `mongorestore` in the target database to restore your data. After AWS DMS completes the full load, it automatically switches to a publisher and subscriber model for logical replication. In this model, we recommend sizing the oplog to retain changes for at least 24 hours.

For homogeneous data migrations of the **Change data capture (CDC)** type, choose `immediately` in the data migration settings to automatically capture the start point for the replication when the actual data migration starts.

> **Note**
>
> For any new or renamed collection, you need to create a new data migration task for those collections as homogeneous data migrations. For a MongoDB-compatible source, AWS DMS doesn't support `create`, `rename` and `drop collection` operations.

The following diagram shows the process of using homogeneous data migrations in AWS DMS to migrate a MongoDB database to Amazon DocumentDB.



Target table preparation mode

You can select Target table preparation mode when you choose to a create data migration task under the **Advanced Settings** tab in the AWS DMS console for PostgreSQL, MongoDB, and Amazon DocumentDB migrations.

Drop tables on target

In Drop tables on target mode, AWS DMS homogeneous migration drops the target tables and recreates them before starting the migration. This approach ensures that the target tables are empty at the start of the migration. During homogeneous migrations, AWS DMS creates all secondary objects, including indexes defined in the source table metadata, before loading the data to ensure efficient data migration.

When using the Drop tables on target mode, you might need to configure the target database. For example, with a PostgreSQL target, AWS DMS cannot create a schema user for security reasons. In this case, you must pre-create the schema user to match the source, allowing AWS DMS to create the tables and assign them to a similar role as the source when the migration begins.

Truncate

In Truncate mode, AWS DMS homogeneous migration truncates all existing target tables before the migration begins. This preserves the table structure. This mode is suitable for full load or full load plus CDC migrations where the target schema is pre-created. For an Amazon DocumentDB target, if the collection does not exist, AWS DMS creates the collection without indexes during the migration.

Do nothing

In Do nothing mode, AWS DMS homogeneous migration assumes that the target tables are pre-created. If the target tables are not empty, data conflicts may occur during migration, potentially causing a DMS task error. In this mode, the table structure remains unchanged, and any existing data is preserved. Do nothing mode is suitable for CDC-only tasks when the target tables have been backfilled from the source, and ongoing replication is used to synchronize the source and target. For an Amazon DocumentDB target, if the collection does not exist, AWS DMS creates the collection without secondary indexes. Additionally, Do nothing mode can be used during the Full Load phase when migrating data from a MongoDB sharded collection to Amazon DocumentDB.

Troubleshooting for homogeneous data migrations in AWS DMS

In the following list, you can find actions to take when you encounter issues with homogeneous data migrations in AWS DMS.

Topics

I can't create a homogeneous data migration in AWS DMS

If you get an error message that says that AWS DMS can't connect to your data providers after you choose **Create data migration**, then make sure that you have configured the required IAM role. For more information, see Creating an IAM role.

If you have configured the IAM role and still get this error message, then add this IAM role to your key user in the AWS KMS key configuration. For more information, see Allows key users to use the KMS key in the *AWS Key Management Service Developer Guide*.

I can't start a homogeneous data migration in AWS DMS

If you get the `Failed` status when you start a data migration in your migration project, check the versions of your source and target data providers. To do so, run the `SELECT VERSION();` query in your MySQL or PostgreSQL database. Make sure that you use the supported database version.

For the list of supported source databases, see Sources for DMS homogeneous data migrations.

For the list of supported target databases, see Targets for DMS homogeneous data migrations.

If you use an unsupported database version, then upgrade your source or target database, and try again.

Check the error message for your data migration in the AWS DMS console. To do so, open your migration project, and choose your data migration. On the **Details** tab, check the **Last failure message** under **General**.

Finally, analyze the CloudWatch log. To do so, open your migration project, and choose your data migration. On the **Details** tab, choose **View CloudWatch logs**.

I can't connect to the target database when running a data migration in AWS DMS

If you get the **Unable to connect to target** error message, then perform the following actions.

1. Make sure that the security group that is attached to your source and target databases contains a rule for any inbound and outbound traffic. For more information, see the section called "Ongoing replication".
2. Verify the network access control list (ACL) and route table rules.
3. Your database must be accessible from the VPC that you created. Add public IP addresses in VPC security groups, and allow input connections in your firewall.
4. On the **Data migrations** tab of your migration project, choose your data migration. Take a note of the **public IP address** under **Connectivity and security** on the **Details** tab. Next, allow access from the public IP address of your data migration in your source and target databases.
5. For ongoing data replication, make sure that your source and target databases can communicate with each other.

For more information, see Control traffic to resources using security groups in the *Amazon Virtual Private Cloud User Guide*.

AWS DMS migrates views as tables in PostgreSQL

Homogeneous data migration doesn't support migrating views as views in PostgreSQL. For PostgreSQL, AWS DMS migrates views as tables.

Working with data providers, instance profiles, and migration projects in AWS DMS

When you use DMS Schema Conversion and homogeneous data migrations in AWS Database Migration Service, you work with migration projects. In turn, AWS DMS migration projects use subnet groups, instance profiles, and data providers.

A *subnet* is a range of IP addresses in your VPC. A replication *subnet group* includes subnets from different Availability Zones which your instance profile can use. Note that a replication *subnet group* is a DMS resource, and is distinct from subnet groups that Amazon VPC and Amazon RDS use.

An *instance profile* specifies network and security settings for the serverless environment where your migration project runs.

A *data provider* stores a data store type and the location information about your database. After you add a data provider to your migration project, you provide the database credentials from AWS Secrets Manager. AWS DMS uses this information to connect to your database.

After you create data providers, your instance profile, and other AWS resources, you can create a migration project. A *migration project* describes your instance profile, source and target data providers, and secrets from AWS Secrets Manager. You can create multiple migration projects for different source and target data providers.

You perform most of your work in the migration project. For DMS Schema Conversion, you use a migration project to assess the objects of your source data provider and convert them to a format compatible with the target database. Then, you can apply converted code to your target data provider or save it as a SQL script. For homogeneous data migrations, you use a migration project to migrate data from your source database to a target database of the same type in the AWS Cloud.

Migration projects in AWS DMS are serverless only. AWS DMS automatically provisions the cloud resources for your migration projects.

AWS DMS has the maximum number of instance profiles, data providers, and migration projects that you can create for your AWS account. See the following section for information about AWS DMS service quotas [Quotas for AWS Database Migration Service](#).

Topics

Creating a subnet group for an AWS DMS migration project

Before you create an instance profile, configure a subnet group for your instance profile.

To create a subnet group

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. In the navigation pane, choose **Subnet groups**, and then choose **Create subnet group**.
3. For **Name**, enter a unique name of your subnet group.
4. For **Description**, enter a brief description of your subnet group.
5. For **VPC**, choose a VPC that has at least one subnet in at least two Availability Zones.
6. For **Add subnets**, choose subnets to include in the subnet group. You must choose subnets in at least two Availability Zones.

   To connect to Amazon RDS databases, add public subnets into your subnet group. To connect to on-premises databases, add private subnets into your subnet group.
7. Choose **Create subnet group**.

Creating instance profiles for AWS Database Migration Service

You can create multiple instance profiles in the AWS DMS console. Make sure that you select an instance profile to use for each migration project that you create in AWS DMS.

To create an instance profile

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. In the navigation pane, choose **Instance profiles**.

3. Choose **Create instance profile**.
4. On the **Create instance profile** page, enter a descriptive value for **Name** for your instance profile.
5. For **Network type**, choose **Dual-stack mode** to create an instance profile that supports IPv4 and IPv6 addressing. Keep the default option to create an instance profile that supports only IPv4 addressing.
6. Next, choose **Virtual private cloud (VPC)** to run your instance of the selected network type. Then choose a **Subnet group** and **VPC security groups** for your instance profile.

   To connect to Amazon RDS databases, use the subnet configuration appropriate for your RDS setup, whether public or private. For connecting to on-premises databases, If use a private subnet group and ensure that your network is configured to allow AWS DMS access to the source on-premises database through the NAT gateway's public IP address. For more information, see Create a VPC based on Amazon VPC.
7. (Optional) In you create a migration project for DMS Schema Conversion, then for **Schema conversion settings - optional**, choose an Amazon S3 bucket to store information from your migration project. Then choose the AWS Identity and Access Management (IAM) role that provides access to this Amazon S3 bucket. For more information, see Create an Amazon S3 bucket.
8. Choose **Create instance profile**.

After you create your instance profile, you can modify or delete it.

To modify an instance profile

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Instance profiles**. The **Instance profiles** page opens.
3. Choose your instance profile, and then choose **Modify**.
4. Update the name of your instance profile, edit the VPC or Amazon S3 bucket settings.
5. Choose **Save changes**.

To delete an instance profile

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Instance profiles**. The **Instance profiles** page opens.
3. Choose your instance profile, and then choose **Delete**.
4. Choose **Delete** to confirm your choice.

Creating data providers in AWS Database Migration Service

You can create data providers and use them in AWS DMS migration projects. Your data provider can be a self-managed engine running on-premises or on an Amazon EC2 instance. Also, your data provider can be a fully managed engine, such as Amazon Relational Database Service (Amazon RDS) or Amazon Aurora.

For each database, you can create a single data provider. You can use a single data provider in multiple migration projects.

Before creating a migration project, make sure that you have created at least two data providers. One of your data providers must be on an AWS service. You can't use AWS DMS to convert your schemas or migrate your data to an on-premises database.

The following procedure shows you how to create data providers in the AWS DMS console wizard.

To create a data provider

1. Sign in to the AWS Management Console, then open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Data providers**. The **Data providers** page opens.
3. Choose **Create data provider**. The following table describes the settings.

Option	Action
Configuration	Choose whether to enter the information about your data provider manually or to use the Amazon RDS DB instance.

Option	Action
Name	Enter a name for your data provider. Make sure that you use a unique name for your data provider so that you can easily identify it.
Engine type	Choose the type of the database engine for your data provider.
Server name	Enter the Domain Name Service (DNS) name or IP address of your database server. The server name for a data provider used for a homogeneous replication must start with an alphanumeric character, and can only contain alphanumeric characters, hyphens (-), periods (.), or underscores (_).
Port	Enter the port used to connect to your database server.
Service ID (SID) or service name	Enter the Oracle System ID (SID). To find the Oracle SID, submit the following query to your Oracle database: `SELECT sys_context('userenv','instance_name') AS SID FROM dual;`
Database name	Enter the name of the database for this data provider. The database name for a data provider used for a homogeneo us replication can be up to 63 characters and can't contain spaces.
Secure Socket Layer (SSL) mode	Choose an SSL mode if you want to turn on connection encryption for this data provider. Depending on the mode that you select, you might need to provide certificate and server certificate information. For further details, see Using SSL with AWS Database Migration Service.
Authentication mode	For a MongoDB source, the authentication mode that AWS DMS uses to authenticate the endpoint connection.

Option	Action
Authentication source	For a MongoDB source, the name of the MongoDB database to use to validate your credentials for authentication.
Authentication mechanism	For a MongoDB source, the authentication method that MongoDB uses to encrypt the password.

4. Choose **Create data provider**.

After you create a data provider, make sure that you add database connection credentials in AWS Secrets Manager.

Creating migration projects in AWS Database Migration Service

Before you create a migration project in AWS DMS, make sure that you create the following resources:

- Data providers that describe your source and target databases
- Secrets with database credentials stored in AWS Secrets Manager
- The AWS Identity and Access Management (IAM) role that provides access to Secrets Manager
- An instance profile that includes network and security settings

To create a migration project

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Migration projects**. The **Migration projects** page opens.
3. Choose **Create migration project**. The following table describes the settings.

Option	Action
Name	Enter a name for your migration project. Make sure that you use a unique name for your migration project so that you can easily identify it.

Option	Action
Instance profile	Choose your instance profile to use for your migration project.
Source	Choose **Browse**, and then choose your source data provider.
Secret ID	Choose the Amazon Resource Name (ARN) of your secret in Secrets Manager that stores your source database credentia ls.
IAM role	Choose an IAM role to provide access to your source database credentials in Secrets Manager.
Target	Choose **Browse**, and then choose your target data provider.
Secret ID	Choose the ARN of your secret in Secrets Manager that stores your target database credentials.
IAM role	Choose an IAM role to provide access to your target database credentials in Secrets Manager.
Transformation rules	(Optional) If you create a migration project for DMS Schema Conversion, then choose **Add transformation rule** to set up transformation rules. Transformation rules make it possible for you to change the object names according to the rule that you specify. For more information, see [Setting up transformation rules]().

4. Choose **Create migration project**.

After AWS DMS creates your migration project, you can use this project in DMS Schema Conversion or homogeneous data migrations. To start working with your migration project, on the **Migration projects** page, choose your project from the list.

Managing migration projects in AWS Database Migration Service

After you create your migration project, you can modify or delete it. For example, to change the source or target data provider, modify your migration project.

You can modify or delete your migration project only after you close the schema conversion or data migration operations. To do so, choose your migration project from the list, and choose **Schema conversion** or **Data migrations**. Next, choose **Close schema conversion** for DMS Schema Conversion and confirm your choice. For homogeneous data migrations, choose your data migration, then choose **Stop** on the **Actions** menu. After you edit your migration project, you can launch schema conversion or start your data migration again.

To modify a migration project

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Migration projects**. The **Migration projects** page opens.
3. Choose your migration project, and then choose **Modify**.
4. Update the name of your project, edit the instance profile, or change source and target data providers. Optionally, add or edit migration rules that change the object names during conversion.
5. Choose **Save changes**.

To delete a migration project

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. Choose **Migration projects**. The **Migration projects** page opens.
3. Choose your migration project, and then choose **Delete**.
4. Choose **Delete** to confirm your choice.

AWS zero-ETL integration for self-managed database sources

AWS zero-ETL integration is a fully managed solution that makes transactional and operational data available in Amazon Redshift, Amazon S3, and Amazon S3 Tables from multiple operational and transactional database sources. Using zero-ETL, you can replicate data from your self-managed source databases such as MySQL, PostgreSQL, SQL Server, and Oracle to Amazon Redshift through existing AWS Database Migration Service (AWS DMS) source endpoints. The automatic synchronization avoids the traditional extract, transform, and load (ETL) process. It also enables real-time analytics and AI workloads. For more information, see zero-ETL integrations in the *Amazon Redshift Management Guide*.

Zero-ETL integration provides the following benefits:

- **Real-time data replication –** Continuous data synchronization from Oracle databases to Amazon Redshift with minimal latency.
- **Elimination of complex ETL pipelines –** No need to build and maintain custom data integration solutions.
- **Reduced operational overhead –** Automated setup and management through AWS APIs.
- **Simplified data integration architecture –** Seamless integration between self-managed databases and AWS analytics services.
- **Enhanced security –** Built-in encryption and IAM access controls.

How zero-ETL integration works for self-managed database sources

You can use existing AWS DMS endpoints previously created for self-managed databases or create new ones.

- Use the AWS Glue console or CLI to create the zero-ETL integration with Amazon Redshift as a target in the AWS Glue catalog. You can specify schema and table filter when creating zero-ETL integrations.
- Additional read-only resources related to the integrations are automatically created within the AWS DMS service. These resources including the zero-ETL engine are used to initiate full

load and continuous data change processes, to sync up data with the Amazon Redshift target database.

- You control the encryption of your data when you create the integration source, when you create the zero-ETL integration, and when you create the Amazon Redshift data warehouse.
- The integration monitors the health of the data pipeline and recovers from issues when possible.
- You can create integrations from sources of the same type into a single Amazon Redshift data warehouse to derive holistic insights across multiple applications.

Once the data is replicated you can use the analytics capabilities of Amazon Redshift. For example, built-in machine learning (ML), materialized views, data sharing, and direct access to multiple data stores and data lakes. For data engineers, zero-ETL integrations provide access to time-sensitive data that otherwise can get delayed by intermittent errors in complex data pipelines. You can run analytical queries and ML models on transactional data to derive timely insights for time-sensitive events and business decisions.

You can create an Amazon Redshift event notification subscriptions to be automatically notified when issues occur for any zero-ETL integration. To view the list of integration-related event notifications, see Zero-ETL integration event notifications with Amazon EventBridge. The simplest way to create a subscription is with the Amazon Simple Notification Service console. For information on creating an Amazon SNS topic and subscribing to it, see Getting started with Amazon SNS in the *Amazon Simple Notification Service Developer Guide*.

Source database configuration before creating integration

Before configuring zero-ETL integrations, you must properly configure your source database according to the requirements for your database engine. Each engine has specific configuration requirements and limitations.

SQL Server

- For configuration requirements, see Using a Microsoft SQL Server database as a source for AWS DMS.
- For change data capture (CDC) requirements, see Capturing data changes for ongoing replication from SQL Server.

> **Note**
>
> RDS SQL Server and Azure SQL Server cannot be used as sources for self-managed zero-ETL integrations.

Oracle

- For configuration requirements and limitations, see Using an Oracle database as a source for AWS DMS.

MySQL

- For configuration requirements and limitations, see Using a MySQL-compatible database as a source for AWS DMS.

PostgreSQL

- For configuration requirements and limitations, see Using a PostgreSQL database as an AWS DMS source.
- When using a self-managed PostgreSQL database as a source, the following limitation applies:
  - For self-managed PostgreSQL sources, you must set `CaptureDdls` to `false` in the PostgreSQL endpoint settings when creating the source endpoint. To set this parameter, use the following when creating your source endpoint:

    ```
    --postgre-sql-settings '{"CaptureDdls": false}'
    ```

    If you do not set CaptureDdls to false, the task may fail to start successfully.
  - When `CaptureDdls` is set to `false`, DDL operations (such as CREATE TABLE, ALTER TABLE, and DROP TABLE) performed on the source database during replication will not be captured or replicated to the target.

Setting up IAM permissions and encryption for zero-ETL integration

To create and manage zero-ETL integrations, you need to configure appropriate IAM permissions, AWS Key Management Service (AWS KMS) encryption keys, and resource policies. This section provides guidance on setting up the required security components.

Prerequisites

Before creating a zero-ETL integration, ensure you have the following:

- An IAM user or role with appropriate permissions to create and manage integrations
- Configure AWS DMS source endpoints for your self-managed databases. For more information, see the section called "Source database configuration before creating integration".
- An Amazon Redshift provisioned cluster or serverless namespace as the target
- Network configuration including VPC subnets and security groups

Creating a KMS key

First, create a customer managed AWS KMS key to encrypt data in your zero-ETL integration. The following example creates a symmetric encryption key:

```
aws kms create-key \
  --description "On-prem Zero-ETL Integration Encryption Key" \
  --key-usage ENCRYPT_DECRYPT \
  --key-spec SYMMETRIC_DEFAULT \
  --region region
```

Note the `KeyId` and `Arn` from the response, as you will need them when configuring the key policy and creating the integration.

Example output:

```
{
    "KeyMetadata": {
        "AWSAccountId": "account-id",
        "KeyId": "4e2c14f8-7abe-4aec-851a-379f6ed973a8",
```

```
        "Arn": "arn:aws:kms:region:account-
id:key/4e2c14f8-7abe-4aec-851a-379f6ed973a8",
        "CreationDate": 1763155061.148,
        "Enabled": true,
        "Description": "Zero-ETL Integration Encryption Key",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "Origin": "AWS_KMS",
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "KeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ],
        "MultiRegion": false
    }
}
```

Configuring the KMS key policy

After creating the KMS key, configure the key policy to allow Amazon Redshift and AWS Glue services to use the key for encryption and decryption operations. The key policy must grant the necessary permissions to the service principals and include the encryption context that will be used during integration creation.

The following example shows a key policy for zero-ETL integrations:

```
{
    "Version": "2012-10-17",
    "Id": "key-default-1",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::account-id:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allows the Redshift and glue service principal to add a grant to a
 KMS key",
```

```
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "redshift.amazonaws.com",
                    "glue.amazonaws.com"
                ]
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
               "kms:EncryptionContext:context-key": "context-value"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant",
                        "GenerateDataKeyWithoutPlaintext",
                        "ReEncryptTo"
                    ]
                }
            }
        }
    ]
}
```

The `kms:EncryptionContext` condition must match the additional encryption context you specify when creating the integration. You can update the key policy using the AWS KMS console or the following CLI command:

```
aws kms put-key-policy \
  --key-id key-id \
  --policy-name default \
  --policy file://kms-key-policy.json
```

Creating an IAM user and configuring AWS CLI

Create an IAM user that will be used to manage zero-ETL integrations and configure the AWS CLI with appropriate credentials.

1. Create an IAM user:

```
aws iam create-user --user-name cli-user
```

2. Create access keys for the user:

```
aws iam create-access-key --user-name cli-user
```

Save the `AccessKeyId` and `SecretAccessKey` from the output, as you will need them to configure the AWS CLI.

Creating and attaching an IAM policy

Create an IAM policy that grants permissions for zero-ETL integration operations. The policy should include permissions for AWS Glue, AWS DMS, Amazon Redshift, AWS KMS, and .

Save the following policy to a file (for example, `/tmp/zetl-policy.json`):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ZetlGlueIntegrationAccess",
            "Effect": "Allow",
            "Action": [
                "glue:CreateIntegration",
                "glue:ModifyIntegration",
                "glue:DeleteIntegration",
                "glue:DescribeIntegrations",
                "glue:DescribeInboundIntegrations"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DMSIntegrationAccess",
            "Effect": "Allow",
            "Action": [
                "dms:CreateOutboundIntegration",
                "dms:ModifyOutboundIntegration",
                "dms:CreateEndpoint",
                "dms:DescribeEndpoints",
                "dms:ModifyEndpoint",
```

```
                "dms:DeleteEndpoint",
                "dms:TestConnection"
            ],
            "Resource": "*"
        },
        {
            "Sid": "ZetlRedshiftFullAccess",
            "Effect": "Allow",
            "Action": [
                "redshift:*",
                "redshift-serverless:*",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAddresses",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:DescribeInternetGateways",
                "sns:CreateTopic",
                "sns:Get*",
                "sns:List*",
                "cloudwatch:Describe*",
                "cloudwatch:Get*",
                "cloudwatch:List*",
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:EnableAlarmActions",
                "cloudwatch:DisableAlarmActions",
                "tag:GetResources",
                "tag:UntagResources",
                "tag:GetTagValues",
                "tag:GetTagKeys",
                "tag:TagResources"
            ],
            "Resource": "*"
        },
        {
            "Sid": "ZetlRedshiftDataAPI",
            "Effect": "Allow",
            "Action": [
                "redshift-data:ExecuteStatement",
                "redshift-data:CancelStatement",
                "redshift-data:ListStatements",
                "redshift-data:GetStatementResult",
                "redshift-data:DescribeStatement",
```

```
                "redshift-data:ListDatabases",
                "redshift-data:ListSchemas",
                "redshift-data:ListTables",
                "redshift-data:DescribeTable"
            ],
            "Resource": "*"
        },
        {
            "Sid": "ZetlKMSAccess",
            "Effect": "Allow",
            "Action": [
                "kms:CreateKey",
                "kms:DescribeKey",
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:GenerateDataKey",
                "kms:ListKeys",
                "kms:CreateAlias",
                "kms:ListAliases",
                "kms:CreateGrant"
            ],
            "Resource": "*"
        },
        {
            "Sid": "ZetlSecretsManagerAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "secretsmanager:CreateSecret",
                "secretsmanager:UpdateSecret",
                "secretsmanager:DeleteSecret",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecrets",
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:PutResourcePolicy",
                "secretsmanager:ValidateResourcePolicy"
            ],
            "Resource": "*"
        }
    ]
}
```

Create the policy and attach it to the IAM user:

```
aws iam create-policy \
  --policy-name ZetlCustomPolicy \
  --policy-document file:///tmp/zetl-policy.json

aws iam attach-user-policy \
  --policy-arn arn:aws:iam::account-id:policy/ZetlCustomPolicy \
  --user-name cli-user
```

Configuring the AWS CLI profile

Configure an AWS CLI profile with the user credentials created in the previous steps:

```
aws configure set aws_access_key_id ACCESS_KEY_ID --profile cli-user
aws configure set aws_secret_access_key SECRET_ACCESS_KEY --profile cli-user
aws configure set region region --profile cli-user
aws configure set output json --profile cli-user
```

Test the profile configuration:

```
aws sts get-caller-identity --profile cli-user
```

The output should display the user's account ID, user ID, and ARN, confirming that the profile is configured correctly.

Creating a Redshift resource policy

Create an Amazon Redshift resource policy to authorize the AWS DMS source endpoint to create inbound integrations with your Amazon Redshift namespace. This policy is attached to the Amazon Redshift namespace and controls which sources can replicate data into it.

The following example shows how to create a resource policy for an Amazon Redshift namespace:

```
aws redshift put-resource-policy \
  --policy '{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
```

```
        "Principal": {
          "Service": ["redshift.amazonaws.com"]
        },
        "Action": ["redshift:AuthorizeInboundIntegration"],
        "Condition": {
          "StringEquals": {
            "aws:SourceArn": "arn:aws:dms:region:account-id:endpoint:endpoint-id"
          }
        }
      },
      {
        "Effect": "Allow",
        "Principal": {
          "AWS": "account-id"
        },
        "Action": [
          "redshift:CreateInboundIntegration",
          "redshift:ModifyInboundIntegration"
        ]
      }
    ]
  }' \
  --resource-arn arn:aws:redshift:region:account-id:namespace:namespace-id \
  --region region
```

Replace the following placeholders:

- *region* – The AWS region where your resources are located
- *account-id* – Your AWS account ID
- *endpoint-id* – The ID of your AWS DMS source endpoint
- *namespace-id* – The ID of your Amazon Redshift namespace

Example: Creating a zero-ETL integration with encryption

After setting up the necessary permissions and encryption keys, you can create a zero-ETL integration using the AWS Glue API. The following example demonstrates creating an integration from a MySQL source to an Amazon Redshift target with KMS encryption:

```
aws glue create-integration \
  --integration-name mysql-onprem-integration \
```

```
  --source-arn arn:aws:dms:region:account-id:endpoint:source-endpoint-id \
  --target-arn arn:aws:redshift:region:account-id:namespace:namespace-id \
  --description "MySQL to Redshift integration" \
  --integration-config '{"SourceProperties":{"SubnetIds":"subnet-id1,subnet-id2,subnet-
id3","VpcSecurityGroupIds":"sg-id"}}' \
  --data-filter "include: mysql.*" \
  --kms-key-id arn:aws:kms:region:account-id:key/key-id \
  --additional-encryption-context '{"context-key": "context-value"}' \
  --profile cli-user \
  --region region
```

The command includes the following key parameters:

- `--integration-name` – A unique name for your integration
- `--source-arn` – The ARN of your AWS DMS source endpoint
- `--target-arn` – The ARN of your Amazon Redshift namespace
- `--integration-config` – Network configuration including subnet IDs and security groups
- `--data-filter` – Specifies which schemas and tables to replicate
- `--kms-key-id` – The ARN of the AWS KMS key for encryption
- `--additional-encryption-context` – Encryption context key-value pairs that must match the KMS key policy (e.g., `{"context-key": "context-value"}`)
- `--profile` – The AWS CLI profile to use (the cli-user profile created earlier)

Upon successful creation, the command returns the integration details including the integration ARN, status, and configuration parameters. Example output:

```
{
    "SourceArn": "arn:aws:dms:region:account-id:endpoint:endpoint-id",
    "TargetArn": "arn:aws:redshift:region:account-id:namespace:namespace-id",
    "IntegrationName": "mysql-onprem-integration",
    "IntegrationArn": "arn:aws:glue:region:account-id:integration:integration-id",
    "KmsKeyId": "arn:aws:kms:region:account-id:key/key-id",
    "AdditionalEncryptionContext": {
   "context-key": "context-value"
    },
    "Status": "CREATED",
    "CreateTime": 1763234086.001,
    "DataFilter": "include: mysql.*",
    "IntegrationConfig": {
```

```
        "SourceProperties": {
            "SubnetIds": "subnet-id1,subnet-id2,subnet-id3",
            "VpcSecurityGroupIds": "sg-id"
        }
    }
}
```

Security best practices

Follow these security best practices when setting up zero-ETL integrations:

- **Use least privilege access** – Grant only the minimum permissions required for creating and managing integrations. Consider using resource-level permissions where possible.
- **Enable encryption** – Always use customer managed AWS KMS keys to encrypt integration data. Specify encryption context for additional security.
- **Rotate credentials regularly** – If using IAM user access keys, rotate them regularly. Consider using IAM roles with temporary credentials instead.
- **Monitor access** – Use AWS CloudTrail to monitor API calls related to integration creation and management.
- **Restrict network access** – Configure VPC security groups to limit network access to only required resources.
- **Use resource policies** – Implement Amazon Redshift resource policies to control which sources can create integrations with your data warehouse.
- **Tag resources** – Apply tags to integrations and related resources for better organization and cost tracking.

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

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

- To connect your source and target databases to an AWS DMS replication instance, you configure a network. Doing this can be as simple as connecting two AWS resources in the same virtual private cloud (VPC) as your replication instance. It can range to more complex configurations such as connecting an on-premises database to an Amazon RDS DB instance over a virtual private network (VPN). For more information, see Network configurations for database migration.
- **Source and target endpoints** – Make sure that you 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. Depending on your source and target database engine, you might need to set up supplemental logging or modify other settings for a source or target database. For more information, see Sources for data migration and Targets for data migration.
- **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 convert your schema. To convert an existing schema to a different database engine, you can use the AWS Schema Conversion Tool (AWS SCT). It can create a target schema and 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 SCT, see the AWS SCT User Guide.
- **Unsupported data types** – Make sure that you can convert source data types into the equivalent data types for the target database. For more information on supported data types, see the source or target section for your data store.
- **Diagnostic support script results** – When you plan your migration, we recommend that you run diagnostic support scripts. With the results from these scripts, you can find advance information about potential migration failures.

  If a support script is available for your database, download it using the link in the corresponding script topic in the following section. After verifying and reviewing the script, you can run it according to the procedure described in the script topic in your local environment. When the script run is complete, you can review the results. We recommend running these scripts as a first step of any troubleshooting effort. The results can be useful while working with an AWS Support team. For more information, see Working with diagnostic support scripts in AWS DMS.
- **Premigration assessments** – 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. By using this assessment, you can identify potential problems before you

run a new or modified task. For more information on working with premigration assessments, see Enabling and working with premigration assessments for a task.

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 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 SCT 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.

Reviewing the AWS DMS public documentation

We highly recommended that you go through the AWS DMS public documentation pages for your source and target endpoints before your first migration. This documentation can help you to identify the prerequisites for the migration and understand the current limitations before you begin. For more information, see Working with AWS DMS endpoints.

During migration, the public documentation can help you to troubleshoot any issues with AWS DMS. Troubleshooting pages in the documentation can help you to resolve common issues using both AWS DMS and selected endpoint databases. For more information, see Troubleshooting migration tasks in AWS Database Migration Service.

Running a proof of concept

To help discover issues with your environment in early phases of your database migration, we recommend that you run a small test migration. Doing this can also help you to set a more realistic migration time line. In addition, you might need to run a full-scale test migration to measure whether AWS DMS can handle the throughput of your database over your network. During this time, we recommend to benchmark and optimize your initial full load and ongoing replication. Doing this can help you to understand your network latency and gauge overall performance.

At this point, you also have an opportunity to understand your data profile and how large your database is, including the following:

- How many tables are large, medium, and small in size.
- How AWS DMS handles data type and character-set conversions.
- How many tables having large object (LOB) columns.
- How long it takes to run a test migration.

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.

You can improve performance by using some or all of the best practices mentioned following. Whether you can use one of these practices depends on your specific use case. You can find some limitations following:

Provisioning a proper replication server

AWS DMS is a managed service that runs on an Amazon EC2 instance. This service connects to the source database, reads the source data, formats the data for consumption by the target database, and loads the data into the target database.

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. In the following sections, you can find what to consider when you choose your replication server.

CPU

AWS DMS is designed for heterogeneous migrations, but it also supports homogeneous migrations. To perform a homogeneous migration, first convert each source data type to its equivalent AWS DMS data type. Then convert each AWS DMS type data to the target data type. You can find references for these conversions for each database engine within the *AWS DMS User Guide*.

For AWS DMS to perform these conversions optimally, the CPU must be available when the conversions happen. Overloading the CPU and not having enough CPU resources can result in slow migrations, which can also cause other side effects.

Replication instance class

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, consider using one of the larger instances. A larger instance can be a good idea because the service consumes a fair amount of memory and CPU.

AWS DMS support for R5 and C5 instance classes

The R5 instance classes are memory-optimized instances that are designed to deliver fast performance for workloads that process large data sets in memory. Ongoing migrations or replications of high-throughput transaction systems using AWS DMS can, at times, consume large amounts of CPU and memory. R5 instances deliver 5 percent additional memory per vCPU than R4 and the largest size provides 768 GiB of memory. In addition, R5 instances deliver a 10 percent price per GiB improvement and a ~20% increased CPU performance over R4.

The C5 instance classes are optimized for compute-intensive workloads and deliver cost-effective high performance at a low price per compute ratio. They achieve significantly higher network performance. Elastic Network Adapter (ENA) provides C5 instances with up to 25 Gbps of network bandwidth and up to 14 Gbps of dedicated bandwidth to Amazon EBS. AWS DMS can be CPU-intensive, especially when performing heterogeneous migrations and replications such as migrating from Oracle to PostgreSQL. C5 instances can be a good choice for these situations.

Storage

Depending on the instance class, your replication server comes with either 50 GB or 100 GB of data storage. This storage is used for log files and any cached changes that are collected during the load. If your source system is busy or takes large transactions, you might need to increase your storage. If you're running multiple tasks on the replication server, you might also need a storage increase. However, the default amount is usually sufficient.

All storage volumes in AWS DMS are GP2 or General-Purpose solid-state drives (SSDs). GP2 volumes come with a base performance of three I/O operations per second (IOPS), with abilities to burst up to 3,000 IOPS on a credit basis. As a rule of thumb, check the `ReadIOPS` and `WriteIOPS` metrics for the replication instance. Make sure that the sum of these values doesn't cross the base performance for that volume.

Multi-AZ

Choosing a Multi-AZ instance can protect your migration from storage failures. Most migrations are transient and aren't intended to run for long periods of time. If you use AWS DMS for ongoing replication purposes, choosing a Multi-AZ instance can improve your availability should a storage issue occur.

When using a single AZ or Multi-AZ replication instance during a FULL LOAD and a failover or host replacement occurs, the full load task is expected to fail. You can restart the task from the point of failure for the remaining tables that didn't complete, or are in an error state.

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`.

Using parallel full load

You can use a parallel load from Oracle, Microsoft SQL Server, MySQL, Sybase, and IBM Db2 LUW sources based on partitions and subpartitions. Doing this can improve overall full load duration. In addition, while running an AWS DMS migration task, you can accelerate the migration of large or partitioned tables. To do this, split the table into segments and load the segments in parallel in the same migration task.

To use a parallel load, create a table mapping rule of type `table-settings` with the `parallel-load` option. Within the `table-settings` rule, specify the selection criteria for the table or tables that you want to load in parallel. To specify the selection criteria, set the `type` element for `parallel-load` to one of the following settings:

- `partitions-auto`
- `subpartitions-auto`
- `partitions-list`
- `ranges`
- `none`

For more information on these settings, see Table and collection settings rules and operations.

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 (change data capture, or 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. Or 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.

If your data volumes are relatively small and the additional migration time doesn't concern you, you can build primary key and secondary indexes before a full load task. Always turn off referential integrity constraints and triggers.

For a full load plus CDC task, we recommend that you add secondary indexes before the CDC phase. Because AWS DMS uses logical replication, make sure that secondary indexes that support DML operations are in place to prevent full table scans. You can pause the replication task before the CDC phase to build indexes and create referential integrity constraints before you resume the task.

You should enable triggers right before the cutover.

Turn off backups and transaction logging

When migrating to an Amazon RDS database, it's a good idea to turn off backups and Multi-AZ on the target until you're ready to cut over. Similarly, when migrating to systems other than Amazon RDS, turning off 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. Also, 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 turn these constraints off during the migration process and turn them on again as part of the cutover process.

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.

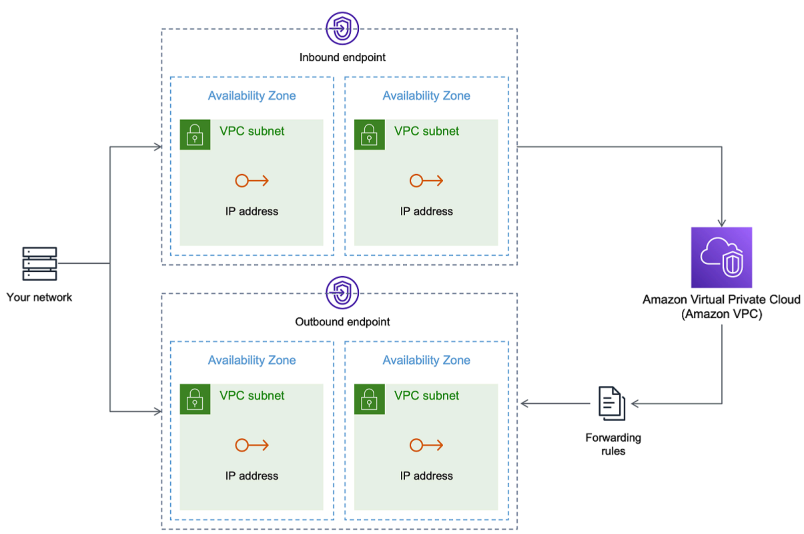
If you have inbound endpoints, you can use DNS queries that originate on-premises to resolve AWS-hosted domains. To configure the endpoints, assign IP addresses in each subnet that you want to provide a resolver. To establish connectivity between your on-premises DNS infrastructure and AWS, use Direct 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 set up 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 Direct Connect or VPN is required.

The following diagram shows the Route 53 Resolver architecture.



For more information about 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.

To create an on-premises name server for AWS 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 where 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*.

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, ensure that the **Max LOB size** parameter setting is correct. Set this parameter 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 chose **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

- Amazon S3

For endpoints that have full LOB support, you can also set a size limit for LOB data types.

Improved LOB performance

While migrating LOB data, you can specify the following different LOB optimization settings.

Per table LOB settings

Using per table LOB settings, you can override task-level LOB settings for some or all of your tables. To do this, define the `lob-settings` in your `table-settings` rule. Following is an example table that includes some large LOB values.

```
SET SERVEROUTPUT ON
CREATE TABLE TEST_CLOB
(
ID NUMBER,
C1 CLOB,
C2 VARCHAR2(4000)
);
DECLARE
bigtextstring CLOB := '123';
iINT;
BEGIN
WHILE Length(bigtextstring) <= 60000 LOOP
bigtextstring := bigtextstring || '000000000000000000000000000000000';
END LOOP;
INSERT INTO TEST_CLOB (ID, C1, C2) VALUES (0, bigtextstring,'AnyValue');
END;
/
SELECT * FROM TEST_CLOB;
COMMIT
```

Next, create a migration task and modify the LOB handling for your table using the new `lob-settings` rule. The `bulk-max-siz` value determines the maximum LOB size (KB). It's truncated if it's bigger than the size specified.

```
{
 "rules": [{
   "rule-type": "selection",
   "rule-id": "1",
```

```
   "rule-name": "1",
   "object-locator": {
    "schema-name": "HR",
    "table-name": "TEST_CLOB"
   },
   "rule-action": "include"
  },
  {
   "rule-type": "table-settings",
   "rule-id": "2",
   "rule-name": "2",
   "object-locator": {
    "schema-name": "HR",
    "table-name": "TEST_CLOB"
   },
   "lob-settings": {
    "mode": "limited",
    "bulk-max-size": "16"
   }
  }
 ]
}
```

Even if this AWS DMS task is created with `FullLobMode : true`, the per table LOB settings direct AWS DMS to truncate LOB data in this particular table to 16,000. You can check the task logs to confirm this.

```
721331968: 2018-09-11T19:48:46:979532 [SOURCE_UNLOAD] W: The value of column 'C' in
 table
'HR.TEST_CLOB' was truncated to length 16384
```

Inline LOB settings

When you create an AWS DMS task, the LOB mode determines how LOBs are handled.

With full LOB mode and limited LOB mode, each has its own benefits and disadvantages. Inline LOB mode combines the advantages of both full LOB mode and limited LOB mode.

You can use inline LOB mode when you need to replicate both small and large LOBs, and most of the LOBs are small. When you choose this option, during full load the AWS DMS task transfers the small LOBs inline, which is more efficient. The AWS DMS task transfers the large LOBs by performing a lookup from the source table.

During change processing, both small and large LOBs are replicated by performing a lookup from the source table.

When you use inline LOB mode, the AWS DMS task checks all of the LOB sizes to determine which ones to transfer inline. LOBs larger than the specified size are replicated using full LOB mode. Therefore, if you know that most of the LOBs are larger than the specified setting, it's better not to use this option. Instead, allow an unlimited LOB size.

You configure this option using an attribute in task settings, `InlineLobMaxSize`, which is only available when `FullLobMode` is set to true. The default value for `InlineLobMaxSize` is 0 and the range is 1 –102400 kilobytes (100 MB). Setting `InlineLobMaxSize` to a value larger than 32Kb can lead to memory pressure and performance degradation depending on the workload.

For example, you might use the following AWS DMS task settings. Here, setting `InlineLobMaxSize` to a value of 5 results in all LOBs smaller than or equal to 5 KiB (5,120 bytes) being transferred inline.

```
{
  "TargetMetadata": {
  "TargetSchema": "",
  "SupportLobs": true,
  "FullLobMode": true,
  "LobChunkSize": 64,
  "LimitedSizeLobMode": false,
  "LobMaxSize": 32,
  "InlineLobMaxSize": 5,
  "LoadMaxFileSize": 0,
  "ParallelLoadThreads": 0,
  "ParallelLoadBufferSize":0,
  "BatchApplyEnabled": false,
  "TaskRecoveryTableEnabled": false},
  . . .
}
```

Resource allocation

- Select the appropriate replication instance size, ensure to allocate sufficient storage for LOB processing. For more information, see Selecting the best size for a replication instance.
- Ensure the target has adequate free space for tables containing LOB columns.
- Create separate tasks for LOBs with different characteristics to optimize performance:

- Frequently updated LOBs vs. rarely modified LOBs
- Large LOBs vs. small LOBs
- Frequently accessed tables vs. infrequently accessed tables

Improving performance when migrating large tables using row filtering

To improve the performance when migrating a large table, 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 console:

1. Open the AWS Management Console.
2. Choose **Tasks**, and create a new task.
3. Choose the **Table mappings** tab, and expand **Selection rules**.
4. Choose **Add new selection rule**. You can now add a column filter with either a less than or equal to, greater than or equal to, equal to, or a range condition between two values. For more information on column filtering, see Specifying table selection and transformations rules from the console.

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 plus CDC task for the currently updated partition.

If your table has a single-column primary key or unique index, you can have your AWS DMS task segment the table using a parallel load of the ranges type to load the data in parallel. For more information, see Table and collection settings rules and operations.

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) statements. 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, set the **Multi-AZ** option when you create your replication instance. By choosing **Multi-AZ**, you get high availability and failover support for the replication instance. However, this option can have an impact on performance and can slow down replication while applying changes to the target system.

Before you upgrade your source or target databases, we recommend that you stop any AWS DMS tasks that are running on these databases. Resume the tasks after your upgrades are complete.

During ongoing replication, it's critical to identify the network bandwidth between your source database system and your AWS DMS replication instance. Make sure that the network doesn’t cause any bottlenecks during ongoing replication.

It's also important to identify the rate of change and archive log generation per hour on your source database system. Doing this can help you to understand the throughput that you might get during ongoing replication.

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. Also, each task that 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 that you're overburdening your source database, 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.

Reducing the bottlenecks on your target database

During the migration, try to remove any processes that compete for write resources on your target database:

- Turn off unnecessary triggers.
- Turn off secondary indexes during initial load and turn them back on later during ongoing replication.

- With Amazon RDS databases, it's a good idea to turn off backups and Multi-AZ until the cutover.
- While migrating to non-RDS systems, it's a good idea turn off any logging on the target until the cutover.

Using data validation during migration

To ensure that your data was migrated accurately from the source to the target, we highly recommend that you use data validation. If you turn on data validation for a task, AWS DMS begins comparing the source and target data immediately after a full load is performed for a table.

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-Compatible Edition
- Amazon Aurora PostgreSQL-Compatible Edition
- IBM Db2 LUW
- Amazon Redshift

For more information, see AWS DMS data validation.

Monitoring your AWS DMS tasks using metrics

You have several options for monitoring metrics for your tasks using the AWS DMS console:

Host metrics

> You can find host metrics on the **CloudWatch metrics** tab for each particular replication instance. Here, you can monitor whether your replication instance is sized appropriately.

Replication task metrics

Metrics for replication tasks, including incoming and committed changes, and latency between the replication host and source/target databases can be found on the **CloudWatch metrics** tab for each particular task.

Table metrics

You can find individual table metrics on the **Table statistics** tab for each individual task. These metrics include these numbers:

- Rows loaded during the full load.
- Inserts, updates, and deletes since the task started.
- DDL operations since the task started.

For more information on metrics monitoring, see Monitoring AWS DMS tasks.

Events and notifications

AWS DMS uses 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. These can include email messages, text messages, or calls to an HTTP endpoint.

For more information, see Working with Amazon SNS events and notifications in AWS Database Migration Service.

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 view the task log, configure Amazon CloudWatch as part of task creation.

For more information, see Monitoring replication tasks using Amazon CloudWatch.

Troubleshooting replication tasks with Time Travel

To troubleshoot AWS DMS migration issues, you can work with Time Travel. For more information about Time Travel, see Time Travel task settings.

When you work with Time Travel, be aware of the following considerations:

- To avoid overhead on a DMS replication instance, turn on Time Travel only for tasks that need debugging.
- When you use Time Travel to troubleshoot replication tasks that might run for several days, monitor replication instance metrics for resource overheads. This approach applies especially in cases where high transaction loads run on source databases for extended periods of time. For more details, see Monitoring AWS DMS tasks.
- When the Time Travel task setting `EnableRawData` is set to `true`, the task memory usage during DMS replication might be higher than when Time Travel isn't turned on. If you turn on Time Travel for extended periods of time, monitor your task.
- Currently, you can turn on Time Travel only at the task level. Changes to all tables are logged in Time Travel logs. If you are troubleshooting for specific tables in a database with high transaction volume, create a separate task .

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 connects to the Oracle target as `MASTER` and populates the `MASTER` schema with database objects from `PERFDATA`.

To override this behavior, provide a schema transformation. For example, to migrate the `PERFDATA` schema objects to a `PERFDATA` schema at the target endpoint, 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.

Changing table and index tablespaces for an Oracle target

When using Oracle as a target, AWS DMS migrates all tables and indexes to the default tablespace 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 tablespace.

To override this behavior, 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`.

```
{
   "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.

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 Endpoint settings when using Oracle as a source for AWS DMS.

Upgrading a replication instance version

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. It's critical to test the existing version of your AWS DMS replication instance running a production work load before you upgrade your replication instance to a later version. For more information on available version upgrades, see AWS DMS release notes.

Understanding your migration cost

AWS Database Migration Service helps you migrate databases to AWS easily and securely at a low cost. You only pay for your replication instances and any additional log storage. Each database migration instance includes storage sufficient for swap space, replication logs, and data cache for most replications and inbound data transfer is free.

You might need more resources during initial load or during peak load time. You can closely monitor replication instance resource utilization using cloud watch metrics. You can then scale up and scale down replication instance size based on usage.

For more information on estimating your migration costs, see:

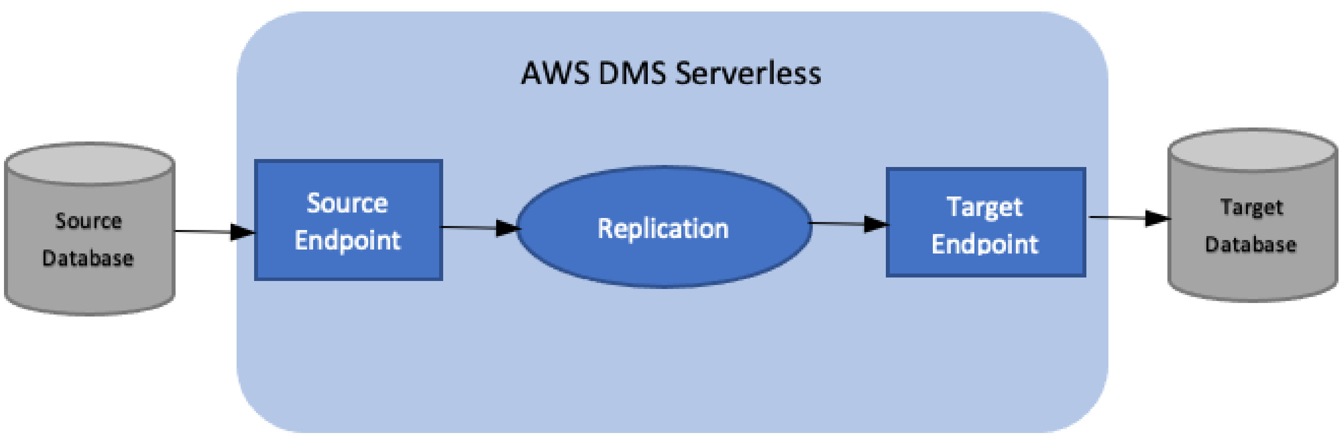
- AWS Database Migration Service pricing
- AWS Pricing Calculator

Working with AWS DMS Serverless

AWS DMS Serverless is a feature that provides automatic provisioning, scaling, built-in high availability, and a pay-for-use billing model, to increase operations agility and optimize your costs. The Serverless feature eliminates replication instance management tasks like capacity estimation, provisioning, cost optimization, and managing replication engine versions and patching.

With AWS DMS Serverless, similar to the current functionality of AWS DMS (referred to in this document as AWS DMS Standard), you create source and target connections using endpoints. After you create your source and target endpoints, you create a replication configuration, which includes configuration settings for the given replication. You can manage the replications by starting, stopping, modifying, or deleting them. Each replication has settings that you can configure according to the requirements of your database migration. You specify these settings using either a JSON file or the AWS DMS section of the AWS Management Console. For more information about replication settings, see [Working with AWS DMS endpoints](). After starting the replication, AWS DMS serverless connects to the source database and collects the database metadata to analyze the replication workload. Using this metadata, AWS DMS computes and provisions the required capacity and starts the data replication.

The following diagram shows the AWS DMS Serverless replication process.



> **Note**
>
> The current engine version for AWS DMS Serverless is [3.5.4]()

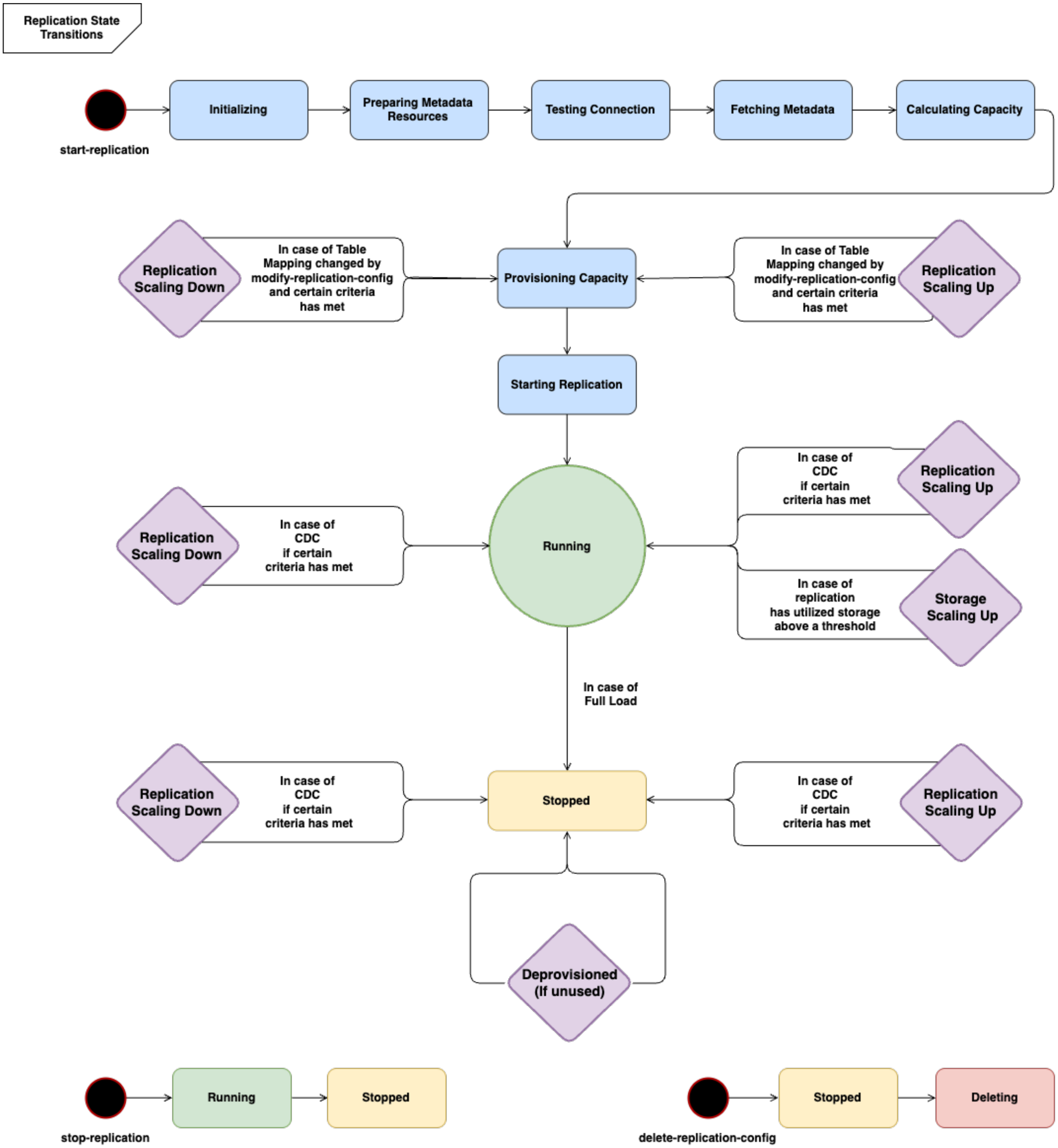
View the following topics to discover more details about AWS DMS Serverless.

Topics

AWS DMS Serverless components

To manage the resources needed to perform a replication, AWS DMS Serverless has granular states that reveal different internal actions taken by the service. When you start the replication, AWS DMS Serverless calculates the capacity load, provisions the calculated capacity, and starts the data replication according to the following replication states.

The following diagram shows the state transitions for an AWS DMS Serverless replication.



- The first state after you start the replication is **Initializing**. In this state, all the required parameters are initialized.

- The states immediately following include **Preparing Metadata Resources**, **Testing Connection**, and **Fetching Metadata**. In these states, AWS DMS Serverless connects to your source database to obtain the information needed to predict the capacity needed.
  - When the replication state is **Testing Connection**, AWS DMS Serverless verifies that the connection to your source and target databases are set up successfully.
  - The replication state following **Testing Connection** is **Fetching Metadata**. Here, AWS DMS retrieves the information needed to calculate capacity.
  - Once AWS DMS retrieves the necessary information, the next state is **Calculating Capacity**. Here, the system calculates the size of underlying resources required to perform the replication.
- The state transition following **Calculating Capacity** is **Provisioning Capacity**. While the replication is in this state, AWS DMS Serverless initializes the underlying compute resources.
- The replication state after all resources are successfully provisioned is **Replication Starting**. In this state, AWS DMS Serverless begins the replication of data. The phases of a replication include the following:
  - **Full load:** In this phase, DMS replicates the source data store as it was when the replication started.
  - **CDC (initial):** In this phase, DMS replicates the changes to the source data store that occurred during the Full Load phase. DMS only runs this phase if the `StopTaskCachedChangesNotApplied` task setting is `false`.
  - **CDC (ongoing):** After the initial CDC phase, DMS replicates changes on the source database as they occur. DMS only continues to run replication after the initial CDC phase if the `StopTaskCachedChangesApplied` task setting is `false`.
- The final state is **Running**. In the **Running** state, the replication of data is ongoing.
- A replication that you stop enters the **Stopped** state. A replication can go into a stopped state for full-load only replication tasks that are successfully completed. These circumstances need to be accounted for resuming or restarting replications in the stopped or failed state:
  - You cannot restart a replication that has not started in 48 hours as AWS DMS deprovisions the resources.

This topic contains the following sections.

- Modifying AWS DMS serverless replications
- Compute Config
- Understanding autoscaling in AWS DMS serverless
- Monitoring AWS DMS serverless replications
- Enhanced Throughput for Full-Load Oracle to Amazon Redshift and Amazon S3 Migrations
- Understanding Storage Autoscaling in AWS DMS Serverless

For AWS DMS Serverless, the left-hand navigation panel of the AWS DMS console has a new option, **Serverless replications**. For **Serverless Replications**, you specify *Replications* instead of replication instance types or tasks to define a replication. In addition, you specify the maximum and minimum DMS capacity units (DCUs) that you want DMS to provision for the replication. A DCU is 2GB of RAM. AWS DMS bills your account for each DCU that your replication is currently using. For information about AWS DMS pricing, see AWS Database Migration Service pricing.

AWS DMS then automatically provisions replication resources based on your table mappings and the predicted size of your workload. This capacity unit is a value in the range of the minimum and maximum capacity unit values that you specify.

Supported Endpoints

With AWS DMS Serverless, you do not need to choose and manage engine versions, as the service handles that setting. AWS DMS Serverless supports the following sources:

- MongoDB
- Amazon DocumentDB (with MongoDB compatibility)
- Microsoft SQL Server
- PostgreSQL-compatible databases
- MySQL-compatible databases
- MariaDB
- Oracle
- Amazon S3
- IBM Db2

AWS DMS Serverless supports the following targets:

- Microsoft SQL Server
- PostgreSQL
- MySQL-compatible databases
- Oracle
- Amazon S3
- Amazon Redshift
- Amazon DynamoDB
- Amazon Kinesis Data Streams
- Amazon Managed Streaming for Apache Kafka
- Amazon OpenSearch Service
- Amazon DocumentDB (with MongoDB compatibility)
- Amazon Neptune

As part of AWS DMS Serverless, you have access to console commands that allow you to create, configure, start, and manage AWS DMS serverless replications. To run these commands using the **Serverless replications** section of the console, you need to do one of the following:

- Set up a new AWS Identity and Access Management (IAM) policy and IAM role to attach that policy to.
- Use an AWS CloudFormation template to provide the access that you need.

AWS DMS Serverless requires a service linked role (SLR) to exist in your account. AWS DMS manages the creation and usage of this role. For more information about making sure that you have the necessary SLR, see Service-linked role for AWS DMS.

Creating a serverless replication

To create a serverless replication between two existing AWS DMS endpoints, do the following. For information about creating AWS DMS endpoints, see Creating source and target endpoints.

Creating a serverless replication

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.

2. On the navigation pane, choose **Serverless replications**, and then choose **Create replication**.
3. On the **Create replication** page, specify your serverless replication configuration:

Option	Action
Name	Enter a name to identify the replication, such as **`DMS-replication`** .
Descriptive Amazon Resource Name (ARN)- Optional	You can use this optional parameter to provide a description of the replication.
Source database endpoint	Choose existing endpoints in your account. Note that AWS DMS Serverless only supports a subset of the endpoint types that AWS DMS standard supports.
Target database endpoint	Choose existing endpoints in your account. Note that AWS DMS Serverless only supports a subset of the endpoint types that AWS DMS standard supports.
Replication type	Choose a replication type based on your requireme nts: • **Full load**: AWS DMS migrates existing data only. • **Full load and change data capture (CDC)**: AWS DMS migrates existing data and changes that occur during replication. • **Change data capture (CDC)**: AWS DMS only migrates changes that occur after you start replication.

In the **Settings** section, set the settings that your replication requires.

In the **Table mappings** section, set up table mapping to define rules to select and filter data that you are replicating. Before you specify your mapping, make sure that you review the documentation section on data type mapping for your source and your target database. For information about data type mapping for your source and target databases, see the data types section for your source and target endpoint types in the Working with AWS DMS endpoints topic.

In the **Compute settings** section, set the following settings. For information about Compute Config settings, see Compute Config.

Option	Action
VPC	Choose an existing VPC.
Subnet group	Choose an existing subnet group.
VPC security group(s)	Choose **default** if it isn't already chosen.
AWS KMS key	Choose an appropriate KMS key. For information about KMS keys, see Creating keys in the *AWS Key Management Service API Reference*.
Deployment	Leave as is.
Availability Zone	Leave as is.
Minimum DMS capacity units (DCU) - (Optional)	Leave blank to use the default value of 1 DCU.
Maximum DMS capacity units (DCU)	Choose **16 DCU**.

Leave the **Maintenance** settings as they are.

4. Choose **Create replication**.

AWS DMS creates a serverless replication to perform your migration.

Modifying AWS DMS serverless replications

To modify your replication configuration, use the `modify-replication-config` action. You can only modify an AWS DMS replication configuration that is in the `CREATED`, `STOPPED`, or `FAILED` states. For information about the `modify-replication-config` action, see [ModifyReplicationConfig](#) in the *AWS Database Migration Service API Reference*.

To modify a serverless replication configuration by using the AWS Management Console

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. In the navigation pane, choose **Serverless replications**.
3. Choose the replication you want to modify. The following table describes the modifications you can make based on the current state of the replication.

Setting	Description	Allowed States
Name	You can change the name of the replication. Enter a name for the replication 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 details to the name, such as including the AWS Region and task you are performing, for example: **`west2-mysql2mysql-config1`** .	`ReplicationState` is `CREATED`, `STOPPED`, or `FAILED`.
Source database endpoint	Choose a new existing source endpoint as the source for the replication.	`ReplicationState` is `CREATED`, or `FAILED` when `Provision State` is `null`.
Target database endpoint	Choose a new existing target endpoint as the target for the replication.	`ReplicationState` is `CREATED`, or `FAILED` when `Provision State` is `null`.

Setting	Description	Allowed States
Replication type	You can modify the type of a serverless replication.	`ReplicationState` is `CREATED`, or `FAILED` when `ProvisionState` is `null`.
Replication Settings	You can modify the replication settings, including the target table preparation mode, whether to include LOB columns in replication, maximum LOB size, validation, and logging. For more information, see Task settings.	`ReplicationState` is `CREATED`, `STOPPED`, or `FAILED`.
Table mappings	You can modify the table mapping settings for a serverless replication, including the selection rules and the transformation rules. For more information, see Table mapping.	`ReplicationState` is `CREATED`, `STOPPED`, or `FAILED`.

Setting	Description	Allowed States
Compute config	You can modify the compute configuration settings for a serverless replication, including the networking settings, scaling settings, and maintenance settings. For information about Compute Config settings, see [Compute Config]().	• You can modify the following scaling, maintenance, and network settings when the `ReplicationState` is `CREATED`, `STOPPED`, or `FAILED`: ◦ `MinCapacityUnits` ◦ `MaxCapacityUnits` ◦ `MultiAZ` ◦ `PreferredMaintenanceWindow` ◦ `VpcSecurityGroupIds` • You can modify the following networking and security settings when the `ReplicationState` is `CREATED`, or `FAILED` when `ProvisionState` is null: ◦ `AvailabilityZone` ◦ `DnsNameServers` ◦ `KmsKeyId`

Setting	Description	Allowed States
		• `ReplicationSubnetGroupId`

> **Note**
>
> You cannot modify endpoints associated to a DMS task when the task status is starting or running.

Compute Config

You configure your replication provisioning using the Compute Config parameter or console section. The fields in the Compute Config object include the following:

Option	Description
MinCapacityUnits	This is the minimum number of DMS Capacity Units (DCU) that AWS DMS will provision. This is also the minimum DCU that autoscaling could scale down to.
MaxCapacityUnits	This is the the maximum DMS Capacity Units (DCU) that AWS DMS can provision, depending on your replication's capacity prediction. This is also the maximum DCU that autoscaling could scale up to.
KmsKeyId	The encryption key to use to encrypt replication storage and connection information. If you choose **(Default) aws/dms**, AWS DMS uses the default KMS key associate d with your account and AWS Region. A description and your account number are shown, along with the key's ARN. For more information about using the encryption key, see Setting an encryption key and specifying AWS KMS permissions. For this tutorial, leave **(Default) aws/dms** chosen.

Option	Description
ReplicationSubnetGroupId	The replication subnet group in your selected VPC where you want the replication 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. For more information about replication subnet groups, see [Creating a replication subnet group]().
VpcSecurityGroupIds	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 is located.
PreferredMaintenanceWindow	This parameter defines a weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC). The default is 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.
MultiAZ	Setting this optional parameter creates a standby replica of your replication in another Availability Zone for failover support. If you intend to use change data capture (CDC) or ongoing replication, we recommend that you turn on this option.

Understanding autoscaling in AWS DMS serverless

After you provision a replication and it is in the `RUNNING` state, the AWS DMS service manages the capacity of the underlying resources to adapt to changing workloads. This management scales replication resources based on the following replication settings:

- `MinCapacityUnits`
- `MaxCapacityUnits`

Replications scale up after a period of exceeding an upper utilization threshold, and down when capacity utilization is below a minimum capacity utilization threshold for a longer period.

> **Note**
>
> Serverless replications can't autoscale down while a full load is in progress.

Tuning autoscaling in AWS DMS serverless

To tune your replication autoscaling parameters, we recommend that you set `MaxCapacityUnits` to the maximum value, and let AWS DMS manage provisioning of resources. It is recommended that you choose the largest DCU maximum capacity setting to allow the greatest benefit from auto-scaling, to accommodate spikes in transaction volume. The pricing calculator shows the maximum monthly cost if your replication continuously uses the maximum DCU. The maximum DCU does not represent the actual cost, as you only pay for the capacity used.

If your replication is not using its resources at full capacity, AWS DMS will gradually deprovision resources to save you costs. However, since provisioning and deprovisioning resources takes time, we recommend that you set your `MinCapacityUnits` setting to a value that can handle any sudden spikes you expect in your replication workload. This will keep your replication from being under-provisioned while AWS DMS provisions resources for the higher workload level.

If you under-provision your replication with a maximum capacity setting that is too low for data requirements, or a minimum capacity that is too low to handle sudden spikes in your replication workload, you might see your `CapacityUtilization` metric consistently at its maximum value. This can cause your replication to fail. If your replication fails due to under-provisioned resources, AWS DMS creates an out-of-memory event in your replication logs. When an out-of-memory condition occurs due to a sudden spike in replication workload or tuned configuration, the system has built-in auto-scaling capabilities to handle the situation and resume processing. However, this automatic recovery mechanism is not immediate and may take some time to become effective. For faster recovery, you can take manual action by modifying the task configuration, specifically by increasing the `MinCapacityUnits` value and then resuming the task. This manual intervention provides a quicker resolution to the out-of-memory error compared to waiting for the automatic auto-scaling process.

Monitoring AWS DMS serverless replications

AWS provides several tools for monitoring your AWS DMS serverless replications, and responding to potential incidents:

- AWS DMS serverless replication metrics
- AWS DMS serverless replication logs

AWS DMS serverless replication metrics

Serverless replication monitoring includes Amazon CloudWatch metrics for the following statistics. These statistics are grouped by each serverless replication.

Metric	Units	Description
CapacityUtilization	Percent	The percentage of memory used by the serverless replication
CDCIncomingChanges	Percent	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.
CDCLatencySource	Seconds	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.

Metric	Units	Description
		CDCLatencySource can be 0 when there is no replicati on lag between the source and the replication. CDCLatencySource can also become zero when the replication attempts to read the next event in the source's transaction log and there are no new events compared to the last time it read from the source. When this happens, the replication resets the CDCLatencySource to 0.
CDCLatenc yTarget	Seconds	The gap, in seconds, between the first event timestamp waiting to commit on the target and the current timestamp of the AWS DMS instance. Target latency is the difference between the replication instance server time and the oldest unconfirmed event id forwarded to a target component. In other words, target latency is the timestamp difference between the replication instance and the oldest event applied but unconfirmed by TRG endpoint (99%). When CDCLatencyTarget is high, it indicates the process of applying change events to the target is delayed. To identify latency in an ongoing replicati on, you can view this metric together with CDCLatenc ySource. If CDCLatencyTarget is high but CDCLatenc ySource isn't high, investigate if: • No primary keys or indexes are in the target • Resource bottlenecks occur in the target or replicati on instance • Network issues reside between replication and target

Metric	Units	Description
CDCThroughputBandwidthTarget	KB/ second	Outgoing data transmitted for the target in KB per second. CDCThroughputBandwidth records outgoing data transmitted on sampling points. If no network traffic is found, the value is zero. Because CDC does not issue long-running transactions, network traffic may not be recorded.
CDCThroughputRowsSource	Rows/ second	Incoming changes from the source in rows per second.
CDCThroughputRowsTarget	Rows/ second	Outgoing changes for the target in rows per second.
FullLoadThroughputBandwidthTarget	KB/ second	Outgoing data transmitted from a full load for the target in KB per second.
FullLoadThroughputRowsTarget	Rows/ second	Outgoing changes from a full load for the target in rows per second.

AWS DMS serverless replication logs

You can use Amazon CloudWatch to log replication information during an AWS DMS migration process. You enable logging when you select replication settings.

Serverless replications upload status logs to your CloudWatch account to provide increased visibility into the progress of the replication, and to assist with troubleshooting.

AWS DMS uploads serverless-linked logs to a dedicated log group with the prefix `dms-serverless-replication-`*`<your replication config resource ID>`*. Within this log group, there is a log stream called `dms-serverless-replication-orchestrator-`*`<your replication config resource ID>`*. This log stream reports the replication state of your replication, and an associated message providing further details on the work it is doing in this stage. For examples of log entries, see Serverless replication log examples following.

> **Note**
>
> AWS DMS does not create either the log group or stream until you run the replication. AWS DMS does not create the log group or stream if you only create the replication.

To view logs of a replication that ran, follow these steps:

1. Open the AWS DMS console, and choose **Serverless replications** from the navigation pane. The **Serverless replications** dialog appears.
2. Go to the **Configuration** section and choose **View serverless logs** in the General column. The CloudWatch log group opens.

If your replication fails, AWS DMS creates a log entry with a replication state of `failed`, and a message describing the reason for the failure. You should check your CloudWatch logs as the first step in troubleshooting a failed replication.

> **Note**
>
> As with AWS DMS Standard, you have the option to enable more granular logging on the progress of the data migration itself; that is, the logs emitted by the underlying replication task. You can enable these logs in your replication settings by setting `EnableLogging` in the `Logging` field to `true` , as in the following JSON example:
>
> ```
> {
>   "Logging": {
>     "EnableLogging": true
>   }
> }
> ```
>
> If you enable these logs, they only begin appearing during the `running` stage of your serverless replication. They will appear under the same log group as the previous log stream, but will be under the new log stream `dms-serverless-serv-res-id-`*`{unique identifier}`*. See the following section for information about how to interpret serverless replication logs.

Serverless replication log examples

This section includes examples of log entries for serverless replications.

Example: Replication start

When you run a serverless replication, AWS DMS creates a log entry similar to the following:

```
{'replication_state':'initializing', 'message': 'Initializing the replication
 workflow.'}
```

Example: Replication failure

If one of the endpoints of the replication is not configured correctly, AWS DMS creates a log entry similar to the following:

```
{'replication_state':'failed', 'message': 'Test connection failed for endpoint X.',
 'failure_message': 'X'}
```

If you see this message in your log after a failure, make sure that the specified endpoint is healthy and configured correctly.

Enhanced Throughput for Full-Load Oracle to Amazon Redshift and Amazon S3 Migrations

AWS DMS provides significantly improved throughput performance for full-load migrations from Oracle to Amazon Redshift and Amazon S3. DMS automatically enables this feature for tables without the custom `parallel-load` option in its table mappings. For tables with customized parallel-load options, DMS serverless distributes the table load based on the given table mapping configurations. To use enhanced throughput, do the following:

- Provide selection rules that don't reference partitions or boundaries. For example, if the table settings in the table mappings contains `parallel-load`, DMS Serverless won't use the enhanced throughput feature. For more information, see Selection rules and actions.
- Set `MaxFileSize` and `WriteBufferSize` to 64 MB. For more information, see Endpoint settings when using Amazon Redshift as a target for AWS DMS.
- We recommend setting `CompressCsvFiles` to `true` for a data store with sparse data, and `false` for a data store with dense data.

- Set the following task settings to `0`:
  - `ParallelLoadThreads`
  - `ParallelLoadQueuesPerThread`
  - `ParallelApplyThreads`
  - `ParallelApplyQueuesPerThread`
  - `ParallelLoadBufferSize`
- Set `MaxFullLoadSubTasks` to 49 to support parallel data migration.
- Set `LOB mode` to `inline`. For more information, see Setting LOB support for source databases in an AWS DMS task.

AWS DMS does not provide enhanced throughput performance for the following replications:

- Replications with tables using parallel-load. For more information, see Using parallel load for selected tables, views, and collections.
- Replications with data transformation rules.
- Replications with filter rules.
- Replications with the transformation rule.
- Replications with Amazon Redshift Serverless as a target.

Understanding Storage Autoscaling in AWS DMS Serverless

When you start a replication process, AWS DMS Serverless allocates 100GB of initial storage for the replication. 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 Serverless does not use a significant amount of storage. Some exceptions include the following:

- Very large tables that incur a significant transaction load. Loading a large table can take some time, so cached transactions are more likely to be written to disk during a large table load.
- Tasks that are configured to pause 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 storage might be consumed by the cached transactions.
- Tasks configured with tables being loaded into Amazon Redshift. This configuration is not an issue when Amazon Aurora is the target.

Hence, AWS DMS Serverless monitors the storage utilisation every 15 minutes. Once the allocated storage is utilised by 90 percent, AWS DMS Serverless scales up the replication with additional storage. In case the 100 percent storage of the replication is utilised and replication tasks fail before or during the scalling process, DMS Serverless resumes the tasks once the scaling is completed successfully.

Note

- Full-load operations restart from the beginning for any incomplete tables when you resume a previously stopped task.
- There is no impact on the DMS task performance during storage scaling event.
- There is no cooling period between two storage auto scaling events.

AWS DMS Serverless limitations

AWS DMS Serverless has the following limitations:

- You can only modify an AWS DMS replication configuration that is in the `CREATED`, `STOPPED`, or `FAILED` states. For details about which settings you can change under which conditions, see [Modifying AWS DMS serverless replications](#).
- You can only delete an AWS DMS replication configuration that is in the `STOPPED`, or `FAILED` states.
- Unlike replication instances, AWS DMS Serverless replications do not have a public IP address for management tasks. You manage serverless replications using the console.
- This release of AWS DMS serverless does not support all the source and target endpoint types that AWS DMS standard supports. For a list of supported engine types, see [AWS DMS Serverless components](#).
- Serverless replications need to access dependencies by using VPC endpoints. You must use VPC endpoints to access the following endpoint types:
  - Amazon Amazon S3
  - Amazon Kinesis
  - AWS Secrets Manager
  - Amazon DynamoDB

  - Amazon Redshift
  - Amazon OpenSearch Service

  For information about setting up VPC endpoints, see Configuring VPC endpoints for AWS DMS.
- AWS DMS serverless does not support views.
- AWS DMS Serverless does not support SSL connections for DB2 endpoints.
- AWS DMS Serverless does not support setting custom CDC start points.
- When a replication task is in deprovisioned state, the metadata related to the table and the replication statistics are lost.

AWS DMS Serverless premigration.

AWS DMS Serverless includes pre-migration assessment capabilities that help identify potential issues before starting your database migration. By running a pre-migration assessment, you can detect and resolve configuration problems or compatibility issues that might prevent successful serverless replication. For more information see Individual assessments.

Unlike AWS DMS Standard, AWS DMS Serverless automatically stores pre-migration assessment results in a system-managed Amazon S3 bucket, eliminating the need for you to specify a custom bucket.

AWS DMS Serverless provides the following optional settings to support pre-migration assessment:

- **`ResultLocationFolder`**: The folder within an Amazon S3 bucket where you want AWS DMS to store the results of this assessment run.
- **`ResultEncryptionMode`**: The supported values are `SSE_KMS` and `SSE_S3`. If these values are not provided, then the files are not encrypted at rest. For more information, see Creating AWS KMS keys to encrypt Amazon S3 target objects.
- **`ResultKmsKeyArn`**: The ARN of a customer KMS encryption key that you specify when you set `ResultEncryptionMode` to `SSE_KMS`.
- **`IncludeOnly`**: A space-separated list of names for specific individual assessments that you want to include. These names come from the default list of individual assessments that AWS DMS supports for the associated migration.
- **`Exclude`**: A space-separated list of names for specific individual assessments that you want to exclude. These names come from the default list of individual assessments that AWS DMS supports for the associated migration.

- **FailOnAssessmentFailure**: A configurable setting you can set to `true` (the default setting) or `false`. Use this setting to to stop the replication from starting automatically if the assessment fails. This can help you evaluate the issue that is preventing the replication from running successfully.

Using KMS key to encrypt files

To configure `SSE-KMS` for DMS Serverless premigration assessment with a customer-managed key, add a policy statement that grants the DMS service-linked role on your KMS key, enabling secure encryption and decryption of data during the assessment process. You must configure the `kms:GenerateDataKey` and `kms:Decrypt` permissions. See the example below:

```
{
      "Sid": "AccessForDMSServerlessPremigration",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::<CustomerAccountId>:role/aws-service-role/
dms.amazonaws.com/AWSServiceRoleForDMSServerless"
      },
      "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey*"
      ],
      "Resource": "*"
    }
```

Limitations

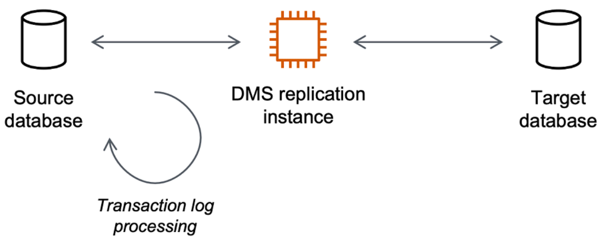
Serverless premigrations has the following limitations:

- AWS DMS Serverless retains only the most recent pre-migration assessment results when you call the describe-replications API. While older assessment runs are removed from the immediate display, the corresponding result files remain accessible in the S3 results bucket.
- Custom S3 buckets cannot be chosen to store the assessment results.
- Transformations on remap schema, table, columns are not supported by preflight.

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	Region	Endpoint	Protocol
US East (Ohio)	us-east-2	dms.us-east-2.amazonaws.com dms-fips.us-east-2.amazonaws.com	HTTPS HTTPS
US East (N. Virginia)	us-east-1	dms.us-east-1.amazonaws.com dms-fips.us-east-1.amazonaws.com	HTTPS HTTPS
US West (N.	us-west-1	dms.us-west-1.amazonaws.com	HTTPS

Region Name	Region	Endpoint	Protocol
California)		dms-fips.us-west-1.amazonaws.com	HTTPS
US West (Oregon)	us-west-2	dms.us-west-2.amazonaws.com dms-fips.us-west-2.amazonaws.com	HTTPS HTTPS
Africa (Cape Town)	af-south-1	dms.af-south-1.amazonaws.com	HTTPS
Asia Pacific (Hong Kong)	ap-east-1	dms.ap-east-1.amazonaws.com	HTTPS
Asia Pacific (Hyderabad)	ap-south-2	dms.ap-south-2.amazonaws.com	HTTPS
Asia Pacific (Jakarta)	ap-southeast-3	dms.ap-southeast-3.amazonaws.com	HTTPS
Asia Pacific (Malaysia)	ap-southeast-5	dms.ap-southeast-5.amazonaws.com	HTTPS
Asia Pacific (Melbourne)	ap-southeast-4	dms.ap-southeast-4.amazonaws.com	HTTPS

Region Name	Region	Endpoint	Protocol
Asia Pacific (Mumbai)	ap-south-1	dms.ap-south-1.amazonaws.com	HTTPS
Asia Pacific (New Zealand)	ap-southeast-6	dms.ap-southeast-6.amazonaws.com	HTTPS
Asia Pacific (Osaka)	ap-northeast-3	dms.ap-northeast-3.amazonaws.com	HTTPS
Asia Pacific (Seoul)	ap-northeast-2	dms.ap-northeast-2.amazonaws.com	HTTPS
Asia Pacific (Singapore)	ap-southeast-1	dms.ap-southeast-1.amazonaws.com	HTTPS
Asia Pacific (Sydney)	ap-southeast-2	dms.ap-southeast-2.amazonaws.com	HTTPS
Asia Pacific (Taipei)	ap-east-2	dms.ap-east-2.amazonaws.com	HTTPS
Asia Pacific (Thailand)	ap-southeast-7	dms.ap-southeast-7.amazonaws.com	HTTPS

Region Name	Region	Endpoint	Protocol
Asia Pacific (Tokyo)	ap-northeast-1	dms.ap-northeast-1.amazonaws.com	HTTPS
Canada (Central)	ca-central-1	dms.ca-central-1.amazonaws.com dms-fips.ca-central-1.amazonaws.com	HTTPS HTTPS
Canada West (Calgary)	ca-west-1	dms.ca-west-1.amazonaws.com dms-fips.ca-west-1.amazonaws.com	HTTPS HTTPS
Europe (Frankfurt)	eu-central-1	dms.eu-central-1.amazonaws.com	HTTPS
Europe (Ireland)	eu-west-1	dms.eu-west-1.amazonaws.com	HTTPS
Europe (London)	eu-west-2	dms.eu-west-2.amazonaws.com	HTTPS
Europe (Milan)	eu-south-1	dms.eu-south-1.amazonaws.com	HTTPS
Europe (Paris)	eu-west-3	dms.eu-west-3.amazonaws.com	HTTPS
Europe (Spain)	eu-south-2	dms.eu-south-2.amazonaws.com	HTTPS
Europe (Stockholm)	eu-north-1	dms.eu-north-1.amazonaws.com	HTTPS

Region Name	Region	Endpoint	Protocol
Europe (Zurich)	eu-central-2	dms.eu-central-2.amazonaws.com	HTTPS
Israel (Tel Aviv)	il-central-1	dms.il-central-1.amazonaws.com	HTTPS
Mexico (Central)	mx-central-1	dms.mx-central-1.amazonaws.com	HTTPS
Middle East (Bahrain)	me-south-1	dms.me-south-1.amazonaws.com	HTTPS
Middle East (UAE)	me-central-1	dms.me-central-1.amazonaws.com	HTTPS
South America (São Paulo)	sa-east-1	dms.sa-east-1.amazonaws.com	HTTPS
AWS GovCloud (US-East)	us-gov-east-1	dms.us-gov-east-1.amazonaws.com	HTTPS
AWS GovCloud (US-West)	us-gov-west-1	dms.us-gov-west-1.amazonaws.com	HTTPS

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

AWS DMS creates the replication instance on an Amazon EC2 instance. AWS DMS currently supports the T3, C5, C6i, R5, and R6i Amazon EC2 instance classes for replication instances:

- T3 instances are the next-generation burstable general-purpose instance type. This type provides a baseline level of CPU performance with the ability to burst CPU usage at any time for as long as required. T3 instances offer a balance of compute, memory, and network resources and are designed for applications with moderate CPU usage that experience temporary spikes in use. T3 instances accumulate CPU credits when a workload is operating below baseline threshold. Each earned CPU credit provides the T3 instance the opportunity to burst with the performance of a full CPU core for one minute when needed.

  T3 instances can burst at any time for as long as required in `unlimited` mode. For more information on `unlimited` mode, see Working with unlimited mode for burstable performance instances.

- C5 instances are the next-generation instance type to deliver cost-effective high performance at a low price per compute ratio for running advanced compute-intensive workloads. This includes workloads such as high-performance web servers, high-performance computing (HPC), batch processing, ad serving, highly scalable multiplayer gaming, and video encoding. Other workloads C5 instances are suited to include scientific modeling, distributed analytics, and machine and deep learning inference. The C5 instances are available with a choice of processors from Intel and AMD.
- C6i instances offer up to 15% better compute price performance over comparable Gen5 instances for a wide variety of workloads, and always-on memory encryption. C6i instances are an ideal fit for compute-intensive workloads such as batch processing, distributed analytics, high performance computing (HPC), ad serving, highly scalable multiplayer gaming, and video encoding.
- R5 instances are the next generation of memory-optimized instance types for Amazon EC2. R5 instances are well-suited for memory-intensive applications such as high performance databases, distributed web scale in-memory caches, midsize in-memory databases, real time big data analytics, and other enterprise applications. Ongoing migrations or replications of high-throughput transaction systems using AWS DMS can also consume large amounts of CPU and memory.
- R6i instances offer up to 15% better compute price performance over comparable Gen5 instances for a wide variety of workloads, and always-on memory encryption. R6i instances are SAP Certified and are ideal for workloads such as SQL and noSQL databases, distributed web scale in-memory caches like Memcached and Redis OSS, in-memory databases like SAP HANA, and real time big data analytics like Hadoop and Spark clusters.
- C7i instances offer better compute performance over comparable previous generation instances. For AWS DMS workloads, C7i instances excel at accelerating data transformation processes, handling compute-heavy schema conversions, and maintaining consistent throughput during high-volume migration tasks. These instances provide an ideal balance of compute performance that require sustained CPU performance.
- R7i instances offer better compute performance over comparable previous generation instances, combined with high memory capacity for memory-intensive workloads. For AWS DMS workloads, R7i instances are particularly well-suited for tasks involving large databases that process high volumes of concurrent database transactions, enabling efficient handling of memory-intensive replication scenarios and complex data validation processes that require substantial memory buffers.

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 service pricing page.

General Purpose Replication Instance Types

Type	vCPU	Memory (GiB)
dms.t3.micro	2	1
dms.t3.small	2	2
dms.t3.medium	2	4
dms.t3.large	2	8

Compute Optimized Replication Instance Types

Type	vCPU	Memory (GiB)
dms.c5.large	2	4
dms.c5.xlarge	4	8
dms.c5.2xlarge	8	16
dms.c5.4xlarge	16	32
dms.c5.9xlarge	36	72
dms.c5.12xlarge	48	96
dms.c5.18xlarge	72	144
dms.c5.24xlarge	96	192
dms.c6i.large	2	4
dms.c6i.xlarge	4	8

Type	vCPU	Memory (GiB)
dms.c6i.2xlarge	8	16
dms.c6i.4xlarge	16	32
dms.c6i.8xlarge	32	64
dms.c6i.12xlarge	48	96
dms.c6i.16xlarge	64	128
dms.c6i.24xlarge	96	192
dms.c6i.32xlarge	128	256
dms.c7i.large	2	4
dms.c7i.xlarge	4	8
dms.x7i.2xlarge	8	16
dms.x7i.4xlarge	16	32
dms.x7i.8xlarge	32	64
dms.x7i.12xlarge	48	96
dms.x7i.16xlarge	64	128
dms.x7i.24xlarge	96	192
dms.x7i.48xlarge	192	384

Memory Optimized Replication Instance Types

Type	vCPU	Memory (GiB)
dms.r5.large	2	16

Type	vCPU	Memory (GiB)
dms.r5.xlarge	4	32
dms.r5.2xlarge	8	64
dms.r5.4xlarge	16	128
dms.r5.8xlarge	32	256
dms.r5.12xlarge	48	384
dms.r5.16xlarge	64	512
dms.r5.24xlarge	96	768
dms.r6i.large	2	16
dms.r6i.xlarge	4	32
dms.r6i.2xlarge	8	64
dms.r6i.4xlarge	16	128
dms.r6i.8xlarge	32	256
dms.r6i.12xlarge	48	384
dms.r6i.16xlarge	64	512
dms.r6i.24xlarge	96	768
dms.r6i.32xlarge	128	1024
dms.r7i.large	2	16
dms.r7i.xlarge	4	32
dms.r7i.2xlarge	8	64
dms.r7i.4xlarge	16	128

Type	vCPU	Memory (GiB)
dms.r7i.8xlarge	32	256
dms.r7i.12xlarge	48	384
dms.r7i.16xlarge	64	512
dms.r7i.24xlarge	96	768
dms.r7i.48xlarge	192	1536

The tables above list all of the AWS DMS replication instance types, but the types that are available in your region might vary. To see the replication instance types available in your region, you can run the following AWS CLI command:

```
aws dms describe-orderable-replication-instances --region your_region_name
```

Topics

- Deciding what instance class to use
- Working with unlimited mode for burstable performance instances

Deciding what instance class to use

To help determine which replication instance class might work best for you, let's look at the change data capture (CDC) process that AWS DMS 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. Or 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 process 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 R5 and R6i instances useful. You can use R5 and R6i instances to hold a large number of transactions in memory and prevent memory-pressure issues during ongoing replications.

Working with unlimited mode for burstable performance instances

A burstable performance instance configured as `unlimited`, such as a T3 instance, can sustain high CPU utilization for any period of time whenever required. The hourly instance price can automatically cover all CPU usage spikes. It does so if the average CPU utilization of the instance is at or below the baseline over a rolling 24-hour period or the instance lifetime, whichever is shorter.

For the vast majority of general-purpose workloads, instances configured as `unlimited` give enough performance without any additional charges. If the instance runs at higher CPU utilization for a prolonged period, it can do so for a flat additional rate per vCPU-hour. For information about T3 instance pricing, see "T3 CPU Credits" in [AWS Database Migration Service](https://aws.amazon.com/dms/pricing/).

For more information on `unlimited` mode for T3 instances, see [Unlimited mode for burstable performance instances](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-performance-instances-unlimited-mode.html) in the *Amazon EC2 User Guide*.

> **Important**
>
> If you use a `dms.t3.micro` instance under the AWS Free Tier offer and use it in `unlimited` mode, charges might apply. In particular, charges might apply if your average utilization over a rolling 24-hour period exceeds the baseline utilization of the instance. For more information, see Baseline utilization in the *Amazon EC2 User Guide*.
> T3 instances launch as `unlimited` by default. If the average CPU usage over a 24-hour period exceeds the baseline, you incur charges for surplus credits. In some cases, you might launch T3 Spot Instances as `unlimited` and plan to use them immediately and for a short duration. If you do so with no idle time for accruing CPU credits, you incur charges for surplus credits. We recommend that you launch your T3 Spot Instances in standard mode to avoid paying higher costs. For more information, see Surplus credits can incur charges, T3 Spot Instances, and Standard mode for burstable performance instances in the *Amazon EC2 User Guide*.

Selecting the best 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](#).

Factors to consider

Memory and disk space are key factors in selecting an appropriate replication instance for your use case. Following, you can find a discussion of the use case characteristics to analyze to choose a replication instance.

- Database and table size

  Data volume helps determine the task configuration to optimize full load performance. For example, for two 1 TB schemas, you can partition tables into four tasks of 500 GB and run them in parallel. The possible parallelism depends on the CPU resource available in the replication instance. That's why it's a good idea understand the size of your database and tables to optimize full load performance. It helps determine the number of tasks that you can possibly have.

- Large objects

  The data types that are present in your migration scope can affect performance. Particularly, large objects (LOBs) impact performance and memory consumption. To migrate a LOB value, AWS DMS performs a two-step process. First, AWS DMS inserts the row into the target without the LOB value. Second, AWS DMS updates the row with the LOB value. This has an impact on the memory, so it's important to identify LOB columns in the source and analyze their size.

- Load frequency and transaction size

  Load frequency and transactions per second (TPS) influence memory usage. A high number of TPS or data manipulation language (DML) activities leads to high usage of memory. This happens because DMS caches the changes until they are applied to the target. During CDC, this leads to swapping (writing to the physical disk due to memory overflow), which causes latency.

- Table keys and referential integrity

  Information about the keys of the table determines the CDC mode (batch apply or transactional apply) that you use to migrate data. In general, transactional apply is slower than batch apply. For long-running transactions, there can be many changes to migrate. When you use transactional apply, AWS DMS might require more memory to store the changes compared to batch apply. If you migrate tables without primary keys, batch apply will fail and the DMS task moves to transactional apply mode. When referential integrity is active between tables during

CDC, AWS DMS uses transactional apply by default. For more information about batch apply compared to transactional apply, see [How can I use the DMS batch apply feature to improve CDC replication performance?](#) .

Use these metrics to determine if you need the replication instance to be compute optimized or memory optimized.

Common issues

You might face the following common issues that cause resource contention on the replication instance during migration. For information on the replication instance metrics, see [Replication instance metrics](#).

- If the memory in a replication instance becomes insufficient, this results in writing data to the disk. Reading from the disk can cause latency, which you can avoid by sizing the replication instance with enough memory.
- The disk size assigned to the replication instance can be smaller than required. The disk size is used when data in memory spills over; it's also used to store the task logs. The maximum IOPS depends on it too.
- Running multiple tasks or tasks with high parallelism affects CPU consumption of the replication instance. This slows down the processing of the tasks and results in latency.

Best practices

Consider these two most common best practices when sizing a replication instance. For more information, see [Best practices for AWS Database Migration Service](#).

1. Size your workload and understand if it's computer-intensive or memory-intensive. Based on this, you can determine the class and size of the replication instance:
   - AWS DMS processes LOBs in memory. This operation requires a fair amount of memory.
   - The number of tasks and the number of threads impact CPU consumption. Avoid using more than eight `MaxFullLoadSubTasks` during the full load operation.
2. Increase the disk space assigned to the replication instance when you have a high workload during full load. Doing this lets the replication instance use the maximum IOPS assigned to it.

The preceding guidelines don't cover all possible scenarios. It's important to consider the specifics of your particular use case when you determine the size of your replication instance.

The preceding tests show CPU and memory vary with different workloads. Particularly, LOBs affect memory, and task count or parallelism affect the CPU. After your migration is running, monitor the CPU, freeable memory, free storage, and IOPS of your replication instance. Based on the data you gather, you can size your replication instance up or down as needed.

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.

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.

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/v2/.
2. In the navigation pane, choose **Replication instances**.
3. Choose your replication engine, and then choose **Modify**.
4. For **Engine version**, choose the version number you want, and then choose **Modify**.

> **Note**
>
> We recommend that you stop all tasks before upgrading the Replication Instance. If you don't stop the task, AWS DMS will stop the task automatically before the upgrade. If

you stop the task manually, you will need to start the task manually after the upgrade is complete. 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:6EFQQO6U6EDPRCPKLNPL2SCEEY`

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

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

   In the output, 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: `3.4.5`

Note

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

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

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

Public and private replication instances

You can specify if 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 virtual private cloud (VPC) of the replication instance. The network can be connected to the VPC by using a virtual private network (VPN), Direct Connect, or VPC peering.

A *VPC peering* connection is a networking connection between two VPCs. It allows 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*.

A *public replication instance* can use the VPC security group of the replication instance, and the replication instance's public IP address or the NAT gateway's public IP address. These connections form a network that you use for data migration.

IP addressing and network types

AWS DMS always creates your replication instance in an Amazon Virtual Private Cloud (VPC). When you create your VPC, you can determine the IP addressing to use: either IPv4 or IPv6, or both. Then, when you create or modify a replication instance, you can specify use of an IPv4 address protocol or an IPv6 address protocol using *dual-stack mode*.

IPv4 addresses

When you create a VPC, you can specify a range of IPv4 addresses for the VPC in the form of a Classless Inter-Domain Routing (CIDR) block, such as 10.0.0.0/16. A subnet group defines the range of IP addresses in this CIDR block. These IP addresses can be private or public.

A private IPv4 address is an IP address that's not reachable over the internet. You can use private IPv4 addresses for communication between your replication instance and other resources, such as Amazon EC2 instances, in the same VPC. Each replication instance has a private IP address for communication in the VPC.

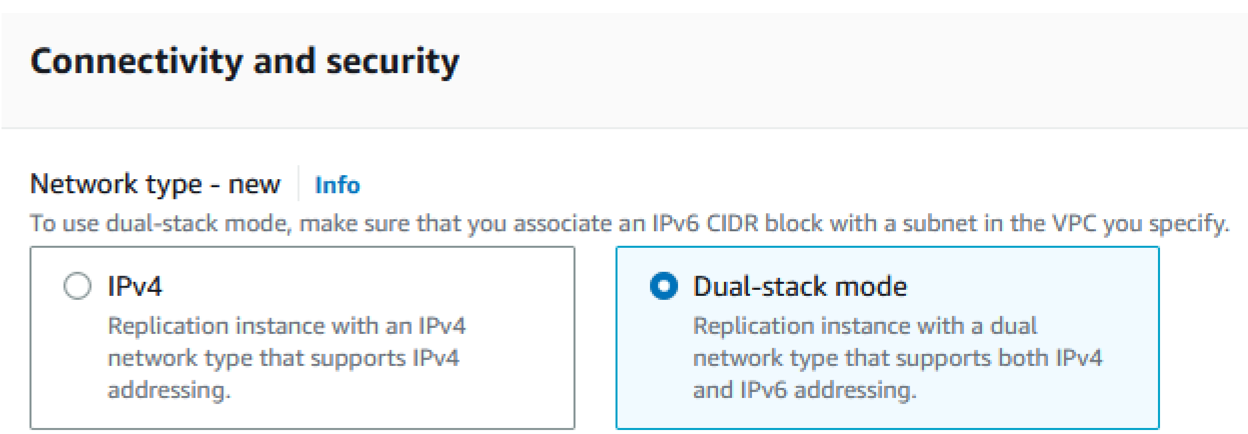
A public IP address is an IPv4 address that is reachable from the internet. You can use public addresses for communication between your replication instance and resources on the internet. You control whether your replication instance receives a public IP address.

Dual-stack mode and IPv6 addresses

When you have resources that must communicate with your replication instance over IPv6, use *dual-stack mode*. To use dual-stack mode, make sure that each subnet in the subnet group that you associate with the replication instance has an IPv6 CIDR block associated with it. You can create a new replication subnet group or modify an existing replication subnet group to meet this requirement. Each IPv6 address is globally unique. The IPv6 CIDR block for your VPC is automatically assigned from the Amazon pool of IPv6 addresses. You can't choose the range yourself.

DMS disables Internet Gateway access for IPv6 endpoints of private dual-stack mode replication instances. DMS does this to ensure that your IPv6 endpoints are private and can only be accessed from within your VPC.

You can use the AWS DMS Console to create or modify a replication instance, and specify dual-stack mode in the **Network type** section. The following image shows the **Network type** section in the console.



References

- For mode information about IPv4 and IPv6 addresses, see IP Addressing in the *Amazon VPC User Guide*.
- For more information about creating a replication instance using dual-stack mode, see Creating a replication instance.
- For mode information about modifying a replication instance, see Modifying a replication instance.

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.

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 ingress 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.

Note

Since an IP address can change as a result of changes to underlying infrastructure, we recommend you use a VPC CIDR range, or route your replication instance outbound traffic through a NAT GW associated Elastic IP. For more information about creating a VPC, including a CIDR block, see Work with VPCs and subnets in the *Amazon Virtual Private*

Cloud User Guide. For more information about Elastic IP addresses, see Elastic IP addresses in the *Amazon Elastic Compute Cloud User Guide*.

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.

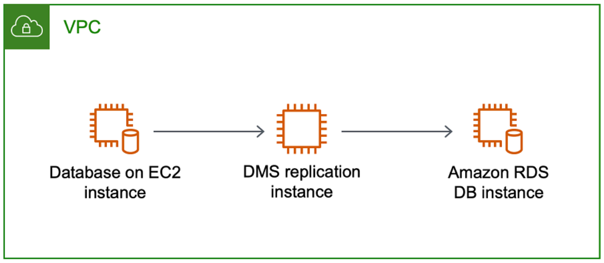
Topics

When practical, we recommend that you create a DMS replication instance in the same Region as your target endpoint, and in the same VPC or subnet as your target endpoint.

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.



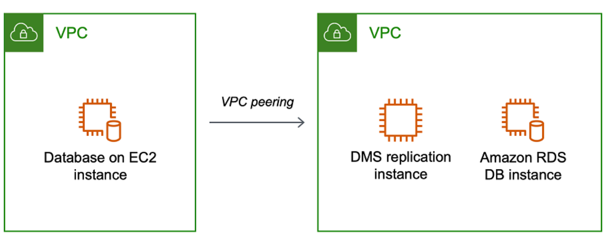
The VPC security group used in this configuration must allow ingress on the database port from the replication instance. You can do this in a couple of ways. You can ensure that the security group used by the replication instance has ingress to the endpoints. Or you can allow the VPC CIDR range, NAT GW Elastic IP, or private IP address of the replication instance if you are using one. But we do not recommend you use the private IP address of the replication instance, because it can break your replication if the replication IP address changes.

Configuration with multiple 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.



To implement VPC peering, follow the instructions in Work with VPC peering connections located in the *Amazon Virtual Private Cloud, VPC Peering* documentation. Be sure the route table of one VPC contains the CIDR block of the other. For example, if VPC A is using destination 10.0.0.0/16 and VPC B is using destination172.31.0.0, the route table of VPC A should contain 172.31.0.0, and route table of VPC B must contain 10.0.0.0/16. For more detailed information, see Update your route tables for VPC peering connection in the *Amazon Virtual Private Cloud, VPC Peering* documentation.

The VPC security groups used in this configuration must allow ingress on the database port from the replication instance, or it should allow ingress on the CIDR block for the VPC being peered.

Configuration with shared VPCs

AWS DMS treats subnets that are shared to a participating customer account in an organization just like regular subnets in the same account. Below is a description of how AWS DMS handles VPCs, subnets, and how you can use shared VPCs.

You can configure your network configuration to operate in custom subnets or VPCs by creating `ReplicationSubnetGroup` objects. When you create a `ReplicationSubnetGroup`, you can choose to specify subnets from a particular VPC in your account. The list of subnets you specify must include at least two subnets that are in separate availability zones, and all subnets must be in the same VPC. When creating a `ReplicationSubnetGroup`, customers only specify subnets. AWS DMS will determine the VPC on your behalf, as each subnet is linked to exactly one VPC.

When you create an AWS DMS `ReplicationInstance` or a AWS DMS `ReplicationConfig`, you can choose to specify a `ReplicationSubnetGroup` and/or a VPC Security Group in which the `ReplicationInstance` or Serverless Replication operates. If not specified, AWS DMS chooses the customer default `ReplicationSubnetGroup` (which AWS DMS creates on your behalf if not specified for all subnets in the default VPC) and the default VPC Security Group.

You can choose to run your migrations in an availability zone that you specify, or in any of the availability zones in your `ReplicationSubnetGroup`. When AWS DMS attempts to create a Replication Instance or start a Serverless Replication, it translates the availability zones of your subnets into availability zones in the core service account, to ensure that we launch instances in the correct Availability Zone even if Availability Zone mappings aren't identical between the two accounts.

If you use a shared VPC, you will need to ensure you create `ReplicationSubnetGroup` objects that map to the subnets you wish to use from a shared VPC. When you create a `ReplicationInstance` or a `ReplicationConfig`, you must specify a `ReplicationSubnetGroup` for the shared VPC, and specify a VPC security group that you created for your shared VPC with your Create request.

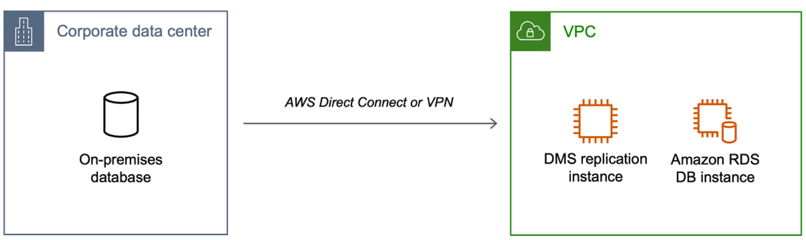
Note the following about using a shared VPC:

- The VPC owner can't share a resource with a participant, but the participant can create a service resource in the owner's subnet.
- The VPC owner can't access a resource (such as a replication instance) that the participant creates, because all resources are account-specific. However, as long as you create the replication instance in the shared VPC, it can access the resources in the VPC regardless of the owning account, as long as the replication endpoint or task had the correct permissions.
- Since resources are account-specific, other participants can't access resources owned by other accounts. There are no permissions that you can give other accounts to let them access resources created in the shared VPC with your account.

Configuration for a network to a VPC using 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.

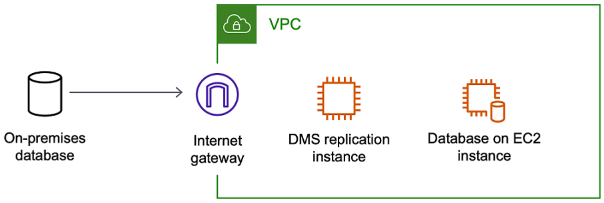
The following illustration shows a configuration where the source endpoint is an on-premises database in a corporate data center. It is connected by using Direct Connect or a VPN to a VPC that contains the replication instance and a target database on an Amazon RDS DB instance.



In this configuration, the VPC security group must include a routing rule that sends traffic destined for a VPC CIDR range or specific IP address 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 VPC CIDR range, or private IP address, or security group into the NAT instance. But we do not recommend you use the private IP address of the replication instance, because it can break your replication if the replication IP address changes.

Configuration for a network to a VPC using the internet

If you don't use a VPN or 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*.

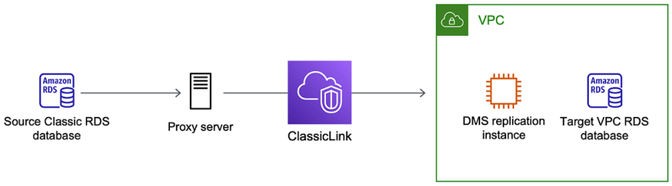
Configuration with an RDS DB instance not in a VPC to a DB instance in a VPC using ClassicLink

> We are retiring EC2-Classic on August 15, 2022. We recommend that you migrate from EC2-Classic to a VPC. For more information, see [Migrate from EC2-Classic to a VPC] in the *Amazon EC2 User Guide* and the blog [EC2-Classic Networking is Retiring – Here's How to Prepare].

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 DB 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 open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.

       If you're 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 about the permissions required for database migration, see [IAM permissions needed to use AWS DMS](#).

b. On the **Dashboard** page, choose **Replication Instance**. Follow the instructions at [Step 1: Create a replication instance using the AWS DMS console] to create a replication instance.

c. After you have created the AWS DMS replication instance, open the EC2 service console. C **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 2: Specify source and target endpoints. 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 3: Create a task and migrate data.

Configuration for a network connecting to AWS services

To connect with AWS services, use either an internet connection or Virtual Private Cloud (VPC) endpoints. This applies when:

Your source or target endpoints use AWS services like:

- AWS Secrets Manager
- Amazon Simple Storage Service

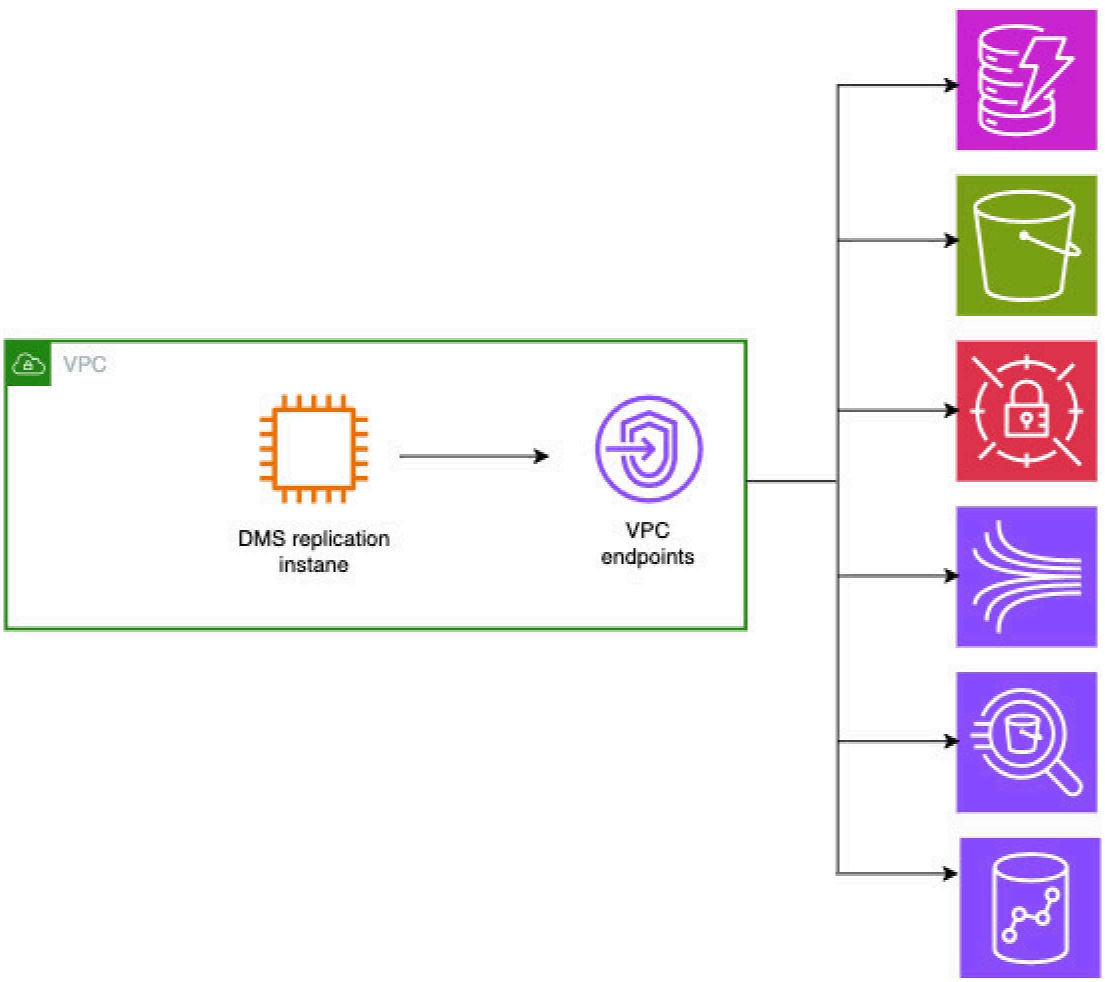
Your target endpoint is an AWS service such as:

- Amazon S3

- Amazon Kinesis
- Amazon DynamoDB
- Amazon Redshift
- Amazon OpenSearch Service
- Amazon Athena

Configuration for a network connecting to AWS services using VPC endpoints

VPC endpoints provide secure connections between your AWS resources, connecting VPC resources to AWS services without requiring internet access. Your applications in private subnets can access AWS services while staying within the AWS network, improving security and reducing latency. Please refer to the image below:



For more information, see Configuring VPC endpoints as AWS DMS source and target endpoints and Configuring AWS DMS secrets manager VPC Endpoint.

Configuration for a network connecting to AWS services using the internet

A replication instance needs internet access to connect with AWS resources during data migration.

For more information regarding private and public subnets within a VPC, see [Example: VPC with servers in private subnets and NAT] in the *Amazon Virtual Private Cloud User Guide*. You must ensure to test your network configuration for connectivity with any required service.

Creating a replication subnet group

As part of the network to use for database migration, you need to specify which 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.

When you create a replication instance or an instance profile in the AWS DMS console, you can use the subnet that you choose.

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

To create a replication subnet group

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.

   If you're signed in as an IAM user, make sure that you 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.
2. In the navigation pane, choose **Subnet groups**.
3. Choose **Create subnet group**.
4. On the **Create replication subnet group** page, specify your replication subnet group information. The following table describes the settings.

Option	Action
Name	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

Option	Action
	your account for the AWS Region that you selected. You can choose to add some intelligence to the name such as including the AWS Region and task you are performing, for example **`DMS-default-VPC`** .
Description	Enter a brief description of the replication subnet group.
VPC	Choose the VPC that 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.
Add subnets	Choose the subnets you want to include in the replication subnet group. You must choose subnets in at least two Availability Zones.

5. Choose **Create subnet group**.

Resolving domain endpoints using DNS

Usually, an AWS DMS replication instance uses the Domain Name System (DNS) resolver in an Amazon EC2 instance to resolve domain endpoints. If you require DNS resolution, you can use the Amazon Route 53 Resolver. For more information about using Route 53 DNS Resolver, see Getting started with Route 53 Resolver.

For information about how to use your own on-premises name server to resolve certain endpoints using the Amazon Route 53 Resolver, see Using your own on-premises name server.

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 AWS KMS key that is unique to your AWS account. You can view and manage this KMS key with AWS Key Management Service (AWS KMS). You can use the default KMS key in your account (`aws/dms`) or a KMS 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 AWS KMS. You can find AWS KMS in the AWS Management Console by searching for **KMS** on the navigation pane.

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. 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 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.

To create a replication instance by using the AWS console

1. Choose **Replication instances** in the navigation pane of the AWS DMS console and then choose **Create replication instance**.
2. On the **Create replication instance** page, specify your replication instance information. The following table describes the settings you can make.

Option	Action
Name	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 **`west2-mysql2mysql-instance1`** .
Descriptive Amazon Resource Name (ARN) - *Optional*	A friendly name to override the default DMS ARN. You can't modify it after creation.
Description	Enter a brief description of the replication instance.
Instance class	Choose an instance class with the configuration you need for your migration. Keep in mind that the instance must have enough storage, network, and processing power to successfully complete your migration. For more information on how to determine which instance class is best for your migration, see Working with an AWS DMS replication instance.
Engine version	In the AWS DMS console, you can choose any supported engine version that you want. From the AWS CLI, the replication instance runs the latest non-beta version of the AWS DMS replication engine unless you specify a different engine version in the AWS CLI.
High Availability	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 replicati on, you should turn on this option.

Option	Action
Allocated storage (GiB)	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 significa nt amount of storage. Some exceptions include the following: • Very large tables that incur a significant transacti on load. Loading a large table can take some time, so cached transactions are more likely to be written to disk during a large table load. • Tasks that are configured to pause before loading cached transactions. In this case, all transactions are cached until the full load completes for all tables. With this configuration, a fair amount of storage might be consumed by cached transacti ons. • Tasks configured with tables being loaded into Amazon Redshift. However, this configuration isn't an issue when Amazon Aurora is the target. In most cases, the default allocation of storage is sufficient. However, it's always a good idea to pay attention to storage-related metrics. Make sure to scale up your storage if you find you are consuming more than the default allocation.

Option	Action
Network type	DMS supports the **IPv4** addressing protocol network type, and supports both IPv4 and IPv6 addressing protocol network types in **Dual-stack** mode. When you have resources that must communicate with your replication instance using an IPv6 addressing protocol network type, use **Dual-stack** mode. For information about limitations in dual-stack mode, see Limitations for dual-stack network DB instances in the Amazon Relational Database Service userguide.
VPC	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.
Replication Subnet Group	Choose the replication subnet group in your selected VPC where you want the replication instance to be created. If your source database is in a VPC, choose the subnet group that contains the source database as the location for your replication instance. For more information about replication subnet groups, see Creating a replication subnet group.
Publicly accessible	Choose this option if you want the replication instance to be accessible from the internet. The default is publicly accessible, and once the option is chosen, you can't modify it after you create the replication instance.

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

Option	Action
Availability zone	Choose the Availability Zone where your source database is located.
VPC Security group(s)	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.
KMS key	Choose the encryption key to use to encrypt replication storage and connection information. If you choose **(Default) aws/dms**, the default AWS Key Management Service (AWS KMS) key associated with your account and AWS Region is used. A description and your account number are shown, along with the key's ARN. For more information on using the encryption key, see [Setting an encryption key and specifying AWS KMS permissions]().

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

Option	Action
Automatic version upgrade	AWS DMS doesn't differentiate between major and minor versions. For example, upgrading from version 3.4.x to 3.5.x isn't considered a major upgrade, so all changes should be backward-compatible. When **Automatic version upgrade** is enabled, DMS automatically upgrades the replication instance's version during the maintenance window if it is deprecated.

Option	Action
	When **AutoMinorVersionUpgrade** is enabled, DMS uses the current default engine version when you create a replication instance. For example, if you set **Engine version** to a lower version number than the current default version, DMS uses the default version. If **AutoMinorVersionUpgrade** isn't enabled when you create a replication instance, DMS uses the engine version specified by the **Engine version** parameter.
Maintenance window	Choose a weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC). Default: A 30-minute window selected at random from an 8-hour block of time per AWS Region, occurring on a random day of the week.

5. 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 open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
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.

Option	Action
Name	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 **`west2-mysql2mysql-instance1`** .
Description	Revise or enter a brief description of the replication instance.
Instance class	You can change the instance class. Choose an instance class with the configuration you need for your migration. Changing the instance class causes the replication instance to reboot. This reboot occurs during the next maintenance window or can occur immediately if you choose the **Apply changes immediately** option.

Option	Action
	For more information on how to determine which instance class is best for your migration, see Working with an AWS DMS replication instance.
Engine version	You can upgrade the engine version that is used by the replication instance. Upgrading the replication engine version causes the replication instance to shut down while it is being upgraded.
Multi-AZ	You can change this option to create a standby replica of your replication instance in another Availability Zone for failover support or remove this option. If you intend to use change data capture (CDC), ongoing replication, you should enable this option.

Option	Action
Allocated storage (GiB)	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 significa nt amount of storage. Some exceptions include the following: • Very large tables that incur a significant transacti on load. Loading a large table can take some time, so cached transactions are more likely to be written to disk during a large table load. • Tasks that are configured to pause before loading cached transactions. In this case, all transactions are cached until the full load completes for all tables. With this configuration, a fair amount of storage might be consumed by cached transacti ons. • Tasks configured with tables being loaded into Amazon Redshift. However, this configuration isn't an issue when Amazon Aurora is the target. In most cases, the default allocation of storage is sufficient. However, it's always a good idea to pay attention to storage related metrics and scale up your storage if you find you are consuming more than the default allocation.

Option	Action
Network type	DMS supports the **IPv4** addressing protocol network type, and supports both IPv4 and IPv6 addressing protocol network types in **Dual-stack** mode. When you have resources that must communicate to your replication instance using an IPv6 addressing protocol network type, choose **Dual-stack** mode. For information about limitations in dual-stack mode, see Limitations for dual-stack network DB instances in the Amazon Relational Database Service userguide.
VPC Security Group(s)	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.
Automatic version upgrade	AWS DMS doesn't differentiate between major and minor versions. For example, upgrading from version 3.4.x to 3.5.x isn't considered a major upgrade, so all changes should be backward-compatible. When **Automatic version upgrade** is enabled, DMS automatically upgrades the replication instance' s version during the maintenance window if it is deprecated. When **Automatic version upgrade** is enabled, DMS uses the current default engine version when you create a replication instance. For example, if you set **Engine version** to a lower version number than the current default version, DMS uses the default version. If **Automatic version upgrade** isn't enabled when you create a replication instance, DMS uses the engine version specified by the **Engine version** parameter.

Option	Action
Maintenance window	Choose a weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC). Default: A 30-minute window selected at random from an 8-hour block of time per AWS Region, occurring on a random day of the week.
Apply changes immediately	Choose this option to apply any modifications you made immediately. Depending on the settings you choose, choosing this option could cause an immediate reboot of the replication instance. If you choose **Test connection** while AWS DMS applies changes, then you will see an error message. After AWS DMS applies changes to your replication instance, choose **Test connection** again.
Apply changes during next scheduled maintenance window	Choose this option if you want DMS to wait until the next scheduled maintenance window to apply your changes.

Best practices when modifying a replication instance

When modifying a replication instance, following these best practices helps ensure a successful update with minimal impact to your migration workflows. Take the following key steps before, during, and after modifications to maintain data integrity and operational efficiency throughout the process.

Plan modification timing:

- You can either apply changes immediately or schedule them for the next maintenance window.
- Schedule during low-traffic periods to minimize impact.

Pre-modification steps:

- Stop all active replication tasks.

- Verify all tasks have successfully stopped.
- Document current configuration task settings.

During modification:

- Monitor the modification progress.
- Wait for "Available" status before proceeding.

Post-modification steps:

- Resume all previously stopped tasks.
- Confirm tasks are running correctly.

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 planned failover from one AWS Availability Zone to another when you reboot. When you force a planned failover of your AWS DMS instance, AWS DMS closes out active connections on the current instance prior to automatically switching to a standby instance in another Availability Zone. Rebooting with a planned failover helps you simulate a planned failover event of an AWS DMS instance, such as when scaling the replication instance class.

> **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 migration tasks are running on the replication instance when a reboot occurs, no data loss occurs but the task stops, and the task status changes to an error state.

If the tables in the migration task are in the middle of a bulk load (full load phase) and haven't yet started, they go into an error state. But tables that are complete at that time, remain in a complete state. When a reboot happens during the full load phase, we recommended that you perform either one of the steps below.

- Remove the tables that are in a complete state from the task, and restart the task with the remaining tables.
- Create a new task with tables in an error state, and with tables that are pending.

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 open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. In the navigation pane, choose **Replication instances**.
3. Choose the replication instance you want to reboot.
4. Choose **Reboot**. The **Reboot replication instance** dialog box opens.
5. Select the check box for **Reboot with planned 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 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 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-planned-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 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 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
&ForcePlannedFailover=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.

If your replication instance meets all the criteria for deletion, and it stays in the `DELETING` status for an extended period of time, contact support to troubleshoot the issue.

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 open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
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 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 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.

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 stops the task for a moment and then resumes it after 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 as part of a **migrate existing data** or **migrate existing data and replicate ongoing changes** task, DMS stops and then restarts the migration for all tables that are in full load phase while the patch is applied. DMS also stops and resumes all tables that are in CDC phase while the patch is applied.

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 console

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

To change the preferred maintenance window 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/.

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 endpoint settings. These settings can control logging, file size, and other parameters; for more information about endpoint settings, see the documentation section for your data store.

Following, you can find out more details about endpoints.

Topics

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 following procedure 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.

Configuration options	Configuration settings
Endpoint type	Choose whether this endpoint is the source or target endpoint.
Select RDS DB Instance	Choose this option if the endpoint is an Amazon RDS database instance.
Endpoint identifier	Type the name you want to use to identify the endpoint. You might want to include in the name the type of endpoint, such as **`oracle-source`** or **`PostgreSQL-target`** . The name must be unique for all replication instances.
Descriptive Amazon Resource Name (ARN) - *optional*	Provide a name to override the default DMS ARN. This setting is optional.
Source engine and **Target engine**	Choose the type of database engine that is the endpoint.

Configuration options	Configuration settings
Access to endpoint database	Choose the option you want to use to specify endpoint database credentials: • **Choose AWS Secrets Manager** – Use secrets defined in AWS Secrets Manager to secretly provide your credentials as shown following. For more information on creating these secrets and the secret access roles that enable AWS DMS to access them, see Using secrets to access AWS Database Migration Service endpoints. • **Provide access information manually** – Use clear-text credentials that you enter directly as shown following. • **IAM authentication** – IAM as the authentication type instead of username and password for your Amazon RDS database instance.
Choose AWS Secrets Manager	Set the following secret credentials.
Secret ID	Type the full Amazon Resource Name (ARN), partial ARN, or friendly name of a secret that you have created in the AWS Secrets Manager for endpoint database access.
IAM role	Type the ARN of a secret access role that you have created in IAM to provide AWS DMS access on your behalf to the secret identified by **Secret ID**. For information about creating a secret access role, see Using secrets to access AWS Database Migration Service endpoints.

Configuration options	Configuration settings
Secret ID for Oracle automatic storage management (ASM)	(For Oracle source endpoints using Oracle ASM only) Type the full Amazon Resource Name (ARN), partial ARN, or friendly name of a secret that you have created in the AWS Secrets Manager for Oracle ASM access. This secret is typically created to access Oracle ASM on the same server as the secret identified by **Secret ID**.
IAM role for Oracle ASM	(For Oracle source endpoints using Oracle ASM only) Type the ARN of a secret access role that you have created in IAM to provide AWS DMS access on your behalf to the secret identified by **Secret ID for Oracle automatic storage management (ASM)**.
Provide access information manually	Set the following clear-text credentials.
Server name	Type the server name. For an on-premises database, this can be the IP address or the public hostname. For an Amazon RDS DB instance, this can be the endpoint (also called the DNS name) for the DB instance, such as `mysqlsrvinst.abcd12345678.us-west-2.rds.amazonaws.com` .
Port	Type the port used by the database.
Secure Socket Layer (SSL) mode	Choose an SSL mode if you want to enable connectio n encryption for this endpoint. Depending on the mode you select, you might be asked to provide certificate and server certificate information.
User name	Type the user name with the permissions required to allow data migration. For information on the permissions required, see the security section for the source or target database engine in this user guide.

Configuration options	Configuration settings
Password	Type the password for the account with the required permissions. Passwords for AWS DMS source and target endpoints have character restrictions, depending on the database engine. For more information, see the following table.
Database name	For certain database engines, the name of the database you want to use as the endpoint database.

The following table lists the unsupported characters in endpoint passwords and secret manager secrets for the listed database engines. If you want to use commas (,) in your endpoint passwords, use the Secrets Manager support provided in AWS DMS to authenticate access to your AWS DMS instances. For more information, see Using secrets to access AWS Database Migration Service endpoints.

For this database engine	The following characters are unsupported in an endpoint password and secret manager secrets
All	`{ }`
Microsoft Azure, as a source only	`;`
Microsoft SQL Server	`, ;`
MySQL-compatible, including MySQL, MariaDB, and Amazon Aurora MySQL	`;`
Oracle	`,`
PostgreSQL, Amazon Aurora PostgreSQL-Compatible Edition, and Amazon Aurora Serverles s as a target only for Aurora PostgreSQL-Compatible Edition	`; + %`

For this database engine	The following characters are unsupported in an endpoint password and secret manager secrets
Amazon Redshift, as a target only	, ;

2. Choose **Endpoint settings** and **AWS KMS key** if you need them. You can test the endpoint connection by choosing **Run test**. The following table describes the settings.

Configuration options	Configuration settings
Endpoint settings	Select any additional connection parameters here. For more information about endpoint settings, see the documentation section for your **Source engine** or **Target engine** (specified in step 1). For an Oracle source endpoint that uses Oracle ASM, if you choose **Provide access information manually** in step 1, you might also need to type in endpoint setting to specify Oracle ASM user credentials. For more information on these Oracle ASM endpoint settings, see Using Oracle LogMiner or AWS DMS Binary Reader for CDC.
AWS KMS key	Choose the encryption key to use to encrypt replicati on storage and connection information. If you choose **(Default) aws/dms**, the default AWS Key Management Service (AWS KMS) key associated with your account and AWS Region is used. For more information on using the encryption key, see Setting an encryption key and specifying AWS KMS permissio ns.
Test endpoint connection (optional)	Add the VPC and replication instance name. To test the connection, choose **Run test**.

Using IAM authentication for Amazon RDS endpoint in AWS DMS

AWS Identity and Access Management (IAM) database authentication provides enhanced security for your Amazon RDS databases by managing database access through AWS IAM credentials. Instead of using traditional database passwords, IAM authentication generates short-lived authentication tokens, valid for 15 minutes, using AWS credentials. This approach significantly improves security by eliminating the need to store database passwords in application code, reducing the risk of credential exposure, and providing centralized access management through IAM. It also simplifies access management by leveraging existing AWS IAM roles and policies, enabling you to control database access using the same IAM framework you use for other AWS services.

AWS DMS now supports IAM authentication for replication instances running DMS version 3.6.1 or later when connecting to MySQL, PostgreSQL, Aurora PostgreSQL, Aurora MySQL, or MariaDB endpoints on Amazon RDS. When creating a new endpoint for these engines, you can select IAM authentication and specify an IAM role instead of providing database credentials. This integration enhances security by eliminating the need to manage and store database passwords for your migration tasks.

Configuring IAM authentication for Amazon RDS endpoint in AWS DMS

When creating an endpoint you can configure IAM authentication for your Amazon RDS database. To configure IAM authentication, do the following:

DMS Console

1. Ensure the Amazon RDS and the database user has IAM authentication enabled. For more information, see Enabling and disabling IAM database authentication in the *Amazon Relational Database Service user guide*.
2. Navigate to the IAM Console, create an IAM role with the below policies:

   Policy

   JSON

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

```
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": "*"
        }
    ]
}
```

Trust policy:

JSON

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

3. During the endpoint configuration in the AWS DMS console, navigate to the **Access to endpoint database** section and select **IAM authentication**.
4. In the **IAM role for RDS database authentication** dropdown menu, select the IAM role with appropriate permissions to access the database.

   For more information, see Creating source and target endpoints.

AWS CLI

1. Ensure the Amazon RDS and the database user has IAM authentication enabled. For more information, see [Enabling and disabling IAM database authentication] in the *Amazon Relational Database Service user guide*.
2. Navigate to the AWS CLI, create an IAM role, and allow DMS to assume the role:

   Policy:

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": "*"
        }
    ]
}
```

   Trust policy:

   JSON

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

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

3. Run the following command to import the certificate and download the PEM file. For more information, see [Download certificate bundles for Amazon RDS](#) in the *Amazon Relational Database Service user guide*.

```
aws dms import-certificate --certificate-identifier rdsglobal --certificate-pem
 file://~/global-bundle.pem
```

4. Run the following commands to create an IAM endpoint:

   - For PostgreSQL/Aurora PostgreSQL endpoints (When `sslmode` is set to `required`, `--certificate-arn` flag is not required):

```
aws dms create-endpoint --endpoint-identifier <endpoint-name> --endpoint-
type <source/target> --engine-name <postgres/aurora-postgres> --username <db
 username with iam auth privileges> --server-name <db server name> --port <port
 number> --ssl-mode <required/verify-ca/verify-full> --postgre-sql-settings
 "{\"ServiceAccessRoleArn\": \"role arn created from step 2 providing permissions
 for iam authentication\", \"AuthenticationMethod\": \"iam\", \"DatabaseName\":
 \"database name\"}" --certificate-arn <if sslmode is verify-ca/verify full use
 cert arn generated in step 3, otherwise this parameter is not required>
```

   - For MySQL, MariaDB, or Aurora MySQL endpoints:

```
aws dms create-endpoint --endpoint-identifier <endpoint-name> --endpoint-type
 <source/target> --engine-name <mysql/mariadb/aurora> --username <db username
 with iam auth privileges> --server-name <db server name> --port <port number>
 --ssl-mode <verify-ca/verify-full> --my-sql-settings "{\"ServiceAccessRoleArn
\": \"role arn created from step 2 providing permissions for iam authentication
\", \"AuthenticationMethod\": \"iam\", \"DatabaseName\": \"database name\"}" --
certificate-arn <cert arn from previously imported cert in step 3>
```

5. Run a test connection against your desired replication instance to create the instance endpoint association and verify everything is set up correctly:

```
aws dms test-connection --replication-instance-arn <replication instance arn> --
endpoint-arn <endpoint arn from previously created endpoint in step 4>
```

> **Note**
>
> When using IAM authentication, the replication instance provided in test-connection must be on AWS DMS version 3.6.1 or later.

Limitations

AWS DMS has following limitations when using IAM authentication with Amazon RDS endpoint:

- Currently Amazon RDS PostgreSQL and Amazon Aurora PostgreSQL instances do not support CDC connections with IAM authentication. For more information, see Limitations for IAM database authentication in the *Amazon Relational Database Service User Guide*.

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 AWS service such as Amazon RDS or Amazon S3.

For a comprehensive list of valid sources, see Sources for AWS DMS.

Topics

- Using a MySQL-compatible database as a source for AWS DMS
- Using an SAP ASE database as a source for AWS DMS
- Using MongoDB as a source for AWS DMS
- Using Amazon DocumentDB (with MongoDB compatibility) as a source for AWS DMS
- Using Amazon S3 as a source for AWS DMS
- Using IBM Db2 for Linux, Unix, Windows, and Amazon RDS database (Db2 LUW) as a source for AWS DMS
- Using IBM Db2 for z/OS databases as a source for AWS DMS

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.

AWS DMS supports the following Oracle database editions:

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

For information about versions of Oracle databases that AWS DMS supports as a source, see Sources for AWS DMS.

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 SSL support for an Oracle endpoint.

AWS DMS 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.

AWS supports the use of TLS version 1.2 and later with Oracle endpoints (and all other endpoint types), and recommends using TLS version 1.3 or later.

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 source database.
2. Create an Oracle source endpoint that conforms with your chosen Oracle database configuration. To create a full-load-only task, no further configuration is needed.
3. To create a task that handles change data capture (a CDC-only or full-load and CDC task), choose 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](#)

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

Topics

- Using Oracle LogMiner or AWS DMS Binary Reader for CDC
- Workflows for configuring a self-managed or AWS-managed Oracle source database for AWS DMS Configuring an Oracle source database
- Working with a self-managed Oracle database as a source for AWS DMS
- Working with an AWS-managed Oracle database as a source for AWS DMS
- Limitations on using Oracle as a source for AWS DMS
- SSL support for an Oracle endpoint
- Supported encryption methods for using Oracle as a source for AWS DMS
- Supported compression methods for using Oracle as a source for AWS DMS
- Replicating nested tables using Oracle as a source for AWS DMS
- Storing REDO on Oracle ASM when using Oracle as a source for AWS DMS
- Endpoint settings when using Oracle as a source for AWS DMS
- Source data types for Oracle

Using Oracle LogMiner or AWS DMS Binary Reader for CDC

In AWS DMS, there are 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 method that reads and parses the raw redo log files directly. These methods have the following features.

Feature	LogMiner	Binary Reader
Easy to configure	Yes	No
Lower impact on source system I/O and CPU	No	Yes
Better CDC performance	No	Yes
Supports Oracle table clusters	Yes	No
Supports all types of Oracle Hybrid Columnar Compressi on (HCC)	Yes	Partially Binary Reader does not support QUERY LOW for tasks with CDC. All other HCC types are fully supported.
LOB column support in Oracle 12c only	No (LOB Support is not available with LogMiner in Oracle 12c.)	Yes
Supports `UPDATE` statements that affect only LOB columns	No	Yes
Supports Oracle transparent data encryption (TDE)	Partially	Partially

Feature	LogMiner	Binary Reader
	When using Oracle LogMiner, AWS DMS does not support TDE encryption on column level for Amazon RDS for Oracle.	Binary Reader supports TDE only for self-managed Oracle databases.
Supports all Oracle compression methods	Yes	No
Supports XA transactions	No	Yes
RAC	Yes Not recommended, due to performance reasons, and some internal DMS limitations.	Yes Highly recommended

Note

By default, AWS DMS uses Oracle LogMiner for (CDC).
AWS DMS supports transparent data encryption (TDE) methods when working with an Oracle source database. If the TDE credentials you specify are incorrect, the AWS DMS migration task does not fail, which can impact ongoing replication of encrypted tables. For more information about specifying TDE credentials, see Supported encryption methods for using Oracle as a source for AWS DMS.

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 does not support all Oracle options, particularly compression and most options for encryption.
- LogMiner offers a simpler configuration, especially compared to Binary Reader 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 does not.

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 logs are mined directly rather than making multiple database queries.
- For migrations with a high volume of changes, CDC performance is usually much better when using Binary Reader compared with using Oracle LogMiner.
- Binary Reader supports CDC for LOBs in Oracle version 12c. LogMiner does not.

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, make sure to restart the CDC task.

Configuration for CDC on an Oracle source database

For an Oracle source endpoint to connect to the database for a change data capture (CDC) task, you might need to specify extra connection attributes. This can be true for either a full-load and CDC task or for a CDC-only task. The extra connection attributes that you specify depend 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 a 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. You don't have to specify additional extra connection attributes to use it.

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

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

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_numb
+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
```

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 following settings provide good results for most ASM configurations.

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

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

In addition, the performance of a CDC task with an Oracle source that uses 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. For more information on extra connection attributes for an Oracle source using ASM, see Endpoint settings when using Oracle as a source for AWS DMS

You also need to choose an appropriate CDC start point. Typically when you do this, you want to identify the point of transaction processing that captures the earliest open transaction to begin CDC from. 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 Performing replication starting from a CDC start point.

For more information on configuring CDC for a self-managed Oracle database as a source, see Account privileges required when using Oracle LogMiner to access the redo logs, Account privileges required when using AWS DMS Binary Reader to access the redo logs, and Additional account privileges required when using Binary Reader with Oracle ASM.

For more information on configuring CDC for an AWS-managed Oracle database as a source, see Configuring a CDC task to use Binary Reader with an RDS for Oracle source for AWS DMS and Using an Amazon RDS Oracle Standby (read replica) as a source with Binary Reader for CDC in AWS DMS.

Workflows for configuring a self-managed or AWS-managed Oracle source database for AWS DMS

Workflows for configuring a self-managed or AWS-managed Oracle source database for AWS DMS

To configure a self-managed source database instance, use the following workflow steps , depending on how you perform CDC.

For this workflow step	If you perform CDC using LogMiner, do this	If you perform CDC using Binary Reader, do this
Grant Oracle account privileges.	See User account privileges required on a self-managed Oracle source for AWS DMS.	See User account privileges required on a self-managed Oracle source for AWS DMS.
Prepare the source database for replication using CDC.	See Preparing an Oracle self-managed source database for CDC using AWS DMS.	See Preparing an Oracle self-managed source database for CDC using AWS DMS.

For this workflow step	If you perform CDC using LogMiner, do this	If you perform CDC using Binary Reader, do this
Grant additional Oracle user privileges required for CDC.	See Account privileges required when using Oracle LogMiner to access the redo logs.	See Account privileges required when using AWS DMS Binary Reader to access the redo logs.
For an Oracle instance with ASM, grant additional user account privileges required to access ASM for CDC.	No additional action. AWS DMS supports Oracle ASM without additional account privileges.	See Additional account privileges required when using Binary Reader with Oracle ASM.
If you haven't already done so, configure the task to use LogMiner or Binary Reader for CDC.	See Using Oracle LogMiner or AWS DMS Binary Reader for CDC.	See Using Oracle LogMiner or AWS DMS Binary Reader for CDC.
Configure Oracle Standby as a source for CDC.	AWS DMS does not support Oracle Standby as a source.	See Using a self-managed Oracle Standby as a source with Binary Reader for CDC in AWS DMS.

Use the following workflow steps to configure an AWS-managed Oracle source database instance.

For this workflow step	If you perform CDC using LogMiner, do this	If you perform CDC using Binary Reader, do this
Grant Oracle account privileges.	For more information, see User account privileges required on an AWS-managed Oracle source for AWS DMS.	For more information, see User account privileges required on an AWS-managed Oracle source for AWS DMS.
Prepare the source database for replication using CDC.	For more information, see Configuring an AWS-managed Oracle source for AWS DMS.	For more information, see Configuring an AWS-managed Oracle source for AWS DMS.

For this workflow step	If you perform CDC using LogMiner, do this	If you perform CDC using Binary Reader, do this
Grant additional Oracle user privileges required for CDC.	No additional account privileges are required.	For more information, see Configuring a CDC task to use Binary Reader with an RDS for Oracle source for AWS DMS.
If you haven't already done so, configure the task to use LogMiner or Binary Reader for CDC.	For more information, see Using Oracle LogMiner or AWS DMS Binary Reader for CDC.	For more information, see Using Oracle LogMiner or AWS DMS Binary Reader for CDC.
Configure Oracle Standby as a source for CDC.	AWS DMS does not support Oracle Standby as a source.	For more information, see Using an Amazon RDS Oracle Standby (read replica) as a source with Binary Reader for CDC in AWS DMS.

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 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 following privileges 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 dms_user;
```

```
GRANT SELECT ANY TRANSACTION TO dms_user;
GRANT SELECT ON V_$ARCHIVED_LOG TO dms_user;
GRANT SELECT ON V_$LOG TO dms_user;
GRANT SELECT ON V_$LOGFILE TO dms_user;
GRANT SELECT ON V_$LOGMNR_LOGS TO dms_user;
GRANT SELECT ON V_$LOGMNR_CONTENTS TO dms_user;
GRANT SELECT ON V_$DATABASE TO dms_user;
GRANT SELECT ON V_$THREAD TO dms_user;
GRANT SELECT ON V_$PARAMETER TO dms_user;
GRANT SELECT ON V_$NLS_PARAMETERS TO dms_user;
GRANT SELECT ON V_$TIMEZONE_NAMES TO dms_user;
GRANT SELECT ON V_$TRANSACTION TO dms_user;
GRANT SELECT ON V_$CONTAINERS TO dms_user;
GRANT SELECT ON ALL_INDEXES TO dms_user;
GRANT SELECT ON ALL_OBJECTS TO dms_user;
GRANT SELECT ON ALL_TABLES TO dms_user;
GRANT SELECT ON ALL_USERS TO dms_user;
GRANT SELECT ON ALL_CATALOG TO dms_user;
GRANT SELECT ON ALL_CONSTRAINTS TO dms_user;
GRANT SELECT ON ALL_CONS_COLUMNS TO dms_user;
GRANT SELECT ON ALL_TAB_COLS TO dms_user;
GRANT SELECT ON ALL_IND_COLUMNS TO dms_user;
GRANT SELECT ON ALL_ENCRYPTED_COLUMNS TO dms_user;
GRANT SELECT ON ALL_LOG_GROUPS TO dms_user;
GRANT SELECT ON ALL_TAB_PARTITIONS TO dms_user;
GRANT SELECT ON SYS.DBA_REGISTRY TO dms_user;
GRANT SELECT ON SYS.OBJ$ TO dms_user;
GRANT SELECT ON DBA_TABLESPACES TO dms_user;
GRANT SELECT ON DBA_OBJECTS TO dms_user; -- Required if the Oracle version is earlier
 than 11.2.0.3.
GRANT SELECT ON SYS.ENC$ TO dms_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.
GRANT SELECT ON GV_$TRANSACTION TO dms_user; -- Required if the source database is
 Oracle RAC in AWS DMS versions 3.4.6 and higher.
GRANT SELECT ON V_$DATAGUARD_STATS TO dms_user; -- Required if the source database
 is Oracle Data Guard and Oracle Standby is used in the latest release of DMS version
 3.4.6, version 3.4.7, and higher.
GRANT SELECT ON V_$DATABASE_INCARNATION TO dms_user;
```

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

```
GRANT SELECT on any-replicated-table to dms_user;
```

Grant the additional following privilege to use validation feature.

```
GRANT EXECUTE ON SYS.DBMS_CRYPTO TO dms_user;
```

Grant the additional following privilege if you use binary reader instead of LogMiner.

```
GRANT SELECT ON SYS.DBA_DIRECTORIES TO dms_user;
```

Grant the additional following privilege to expose views.

```
GRANT SELECT on ALL_VIEWS to dms_user;
```

To expose views, you must also add the `exposeViews=true` extra connection attribute to your source endpoint.

Grant the additional following privilege when using serverless replications.

```
GRANT SELECT on dba_segments to dms_user;
GRANT SELECT on v_$tablespace to dms_user;
GRANT SELECT on dba_tab_subpartitions to dms_user;
GRANT SELECT on dba_extents to dms_user;
```

For information about serverless replications, see Working with AWS DMS Serverless.

Grant the additional following privileges when using Oracle-specific premigration assessments.

```
GRANT SELECT on gv_$parameter  to dms_user;
GRANT SELECT on v_$instance to dms_user;
GRANT SELECT on v_$version to dms_user;
GRANT SELECT on gv_$ASM_DISKGROUP to dms_user;
GRANT SELECT on gv_$database to dms_user;
GRANT SELECT on dba_db_links to dms_user;
GRANT SELECT on gv_$log_History to dms_user;
GRANT SELECT on gv_$log to dms_user;
GRANT SELECT ON DBA_TYPES TO dms_user;
GRANT SELECT ON DBA_USERS to dms_user;
GRANT SELECT ON DBA_DIRECTORIES to dms_user;
GRANT EXECUTE ON SYS.DBMS_XMLGEN TO dms_user;
```

For information about Oracle-specific premigration assessments, see Oracle assessments.

Prerequisites for handling open transactions for Oracle Standby

When using AWS DMS versions 3.4.6 and higher, perform the following steps to handle open transactions for Oracle Standby.

1. Create a database link named, `AWSDMS_DBLINK` on the primary database. *DMS_USER* will use the database link to connect to the primary database. Note that the database link is executed from the standby instance to query the open transactions running on the primary database. See the following example.

```
CREATE PUBLIC DATABASE LINK AWSDMS_DBLINK
   CONNECT TO DMS_USER IDENTIFIED BY DMS_USER_PASSWORD
   USING '(DESCRIPTION=
            (ADDRESS=(PROTOCOL=TCP)(HOST=PRIMARY_HOST_NAME_OR_IP)(PORT=PORT))
            (CONNECT_DATA=(SERVICE_NAME=SID))
          )';
```

2. Verify the connection to the database link using *DMS_USER* is established, as shown in the following example.

```
select 1 from dual@AWSDMS_DBLINK
```

Preparing an Oracle self-managed source database for CDC using AWS DMS

Prepare your self-managed Oracle database as a source to run a CDC task by doing the following:

- Verifying that AWS DMS supports the source database version.
- Making sure that ARCHIVELOG mode is on.
- Setting up supplemental logging.

Verifying that AWS DMS supports the source database version

Run a query like the following to verify that the current version of the Oracle source database is supported by AWS DMS.

```
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. If this query runs without error, AWS DMS supports the current version of the database and you can continue with the migration. If the query raises an error, AWS DMS does not support the current version of the database. To proceed with migration, first convert the Oracle database to an version supported by AWS DMS.

Making sure that ARCHIVELOG mode is on

You can run Oracle in two different modes: the ARCHIVELOG mode and the NOARCHIVELOG mode. To run a CDC task, run the database in ARCHIVELOG mode. To know if the database is in ARCHIVELOG mode, execute the following query.

```
SQL> SELECT log_mode FROM v$database;
```

If NOARCHIVELOG mode is returned, set the database to ARCHIVELOG per Oracle instructions.

Setting up supplemental logging

To capture ongoing changes, AWS DMS requires that you enable minimal supplemental logging on your Oracle source database. In addition, you need to enable supplemental logging on each replicated table in the database.

By default, AWS DMS adds PRIMARY KEY supplemental logging on all replicated tables. To allow AWS DMS to add PRIMARY KEY supplemental logging, grant the following privilege for each replicated table.

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

You can disable the default PRIMARY KEY supplemental logging added by AWS DMS using the extra connection attribute addSupplementalLogging. For more information, see Endpoint settings when using Oracle as a source for AWS DMS.

Make sure to turn on supplemental logging if your replication task updates a table using a WHERE clause that does not reference a primary key column.

To manually set up supplemental logging

1. Run the following query to verify if supplemental logging is already enabled for the database.

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

If the result returned is YES or IMPLICIT, supplemental logging is enabled for the database.

If not, enable supplemental logging for the database by running the following command.

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

2. Make sure that the required supplemental logging is added for each replicated table.

   Consider the following:

   - If ALL COLUMNS supplemental logging is added to the table, you don't need to add more logging.
   - 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 itself, or by adding supplemental logging on the primary key columns on the database.

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

   - 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.

     ```
     ALTER TABLE TableName ADD SUPPLEMENTAL LOG GROUP LogGroupName
      (UniqueIndexColumn1[, UniqueIndexColumn2] ...) ALWAYS;
     ```

     Using SUPPLEMENTAL LOG DATA (UNIQUE INDEX) COLUMNS does not add the unique index columns to the log.

   - If no primary key exists and the table has multiple unique indexes, AWS DMS selects the first unique index in an alphabetically ordered ascending list. You need to add supplemental logging on the selected index's columns as in the previous item.

     Using SUPPLEMENTAL LOG DATA (UNIQUE INDEX) COLUMNS does not add the unique index columns to the log.

   - If no primary key exists and there is no unique index, add supplemental logging on all columns.

     ```
     ALTER TABLE TableName ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
     ```

In some cases, the target table primary key or unique index is different than the source table primary key or unique index. In such 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.

Consider the following:

- If `ALL COLUMNS` supplemental logging is added to the table, you don't need to add more 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. However, do so only if those columns are different from the primary key or unique index columns.
- If a transformation includes 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`.
- To set up supplemental logging on primary key or unique index columns and other columns that are filtered or transformed, you can set up `USER_LOG_GROUP` supplemental logging. Add this logging on both the primary key or unique index columns and any 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 following to create the log group supplemental logging.

  ```
  ALTER TABLE TEST.LOGGING ADD SUPPLEMENTAL LOG GROUP TEST_LOG_GROUP (ID, NAME) ALWAYS;
  ```

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

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

```
GRANT EXECUTE on DBMS_LOGMNR to dms_user;
GRANT SELECT on V_$LOGMNR_LOGS to dms_user;
GRANT SELECT on V_$LOGMNR_CONTENTS to dms_user;
GRANT LOGMINING to dms_user; -- Required only if the Oracle version is 12c or higher.
```

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 following privileges to the Oracle user specified in the Oracle endpoint connection settings.

```
GRANT SELECT on v_$transportable_platform to dms_user;   -- Grant this privilege if the
 redo logs are stored in Oracle Automatic Storage Management (ASM) and AWS DMS accesses
 them from ASM.
GRANT CREATE ANY DIRECTORY to dms_user;                  -- 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 does not have file-level access to the redo
 logs and the redo logs are on non-ASM storage.
GRANT EXECUTE on DBMS_FILE_TRANSFER to dms_user;         -- Grant this privilege to
 copy the redo log files to a temporary folder using the CopyToTempFolder method.
GRANT EXECUTE on DBMS_FILE_GROUP to dms_user;
```

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 are not 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.

> **Note**
>
> AWS DMS CDC does not support Active Dataguard Standby that is not configured to use automatic redo transport service.

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

Additional account privileges required when using Binary Reader with Oracle ASM

To access the redo logs in Automatic Storage Management (ASM) using Binary Reader, grant the following privileges 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
 higher, 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 following statements, 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
```

Using a self-managed Oracle Standby as a source with Binary Reader for CDC in AWS DMS

To configure an Oracle Standby instance as a source when using Binary Reader for CDC, start with the following prerequisites:

- AWS DMS currently supports only Oracle Active Data Guard Standby.
- Make sure that the Oracle Data Guard configuration uses:
  - Redo transport services for automated transfers of redo data.
  - Apply services to automatically apply redo to the standby database.

To confirm those requirements are met, execute the following query.

```
SQL> select open_mode, database_role from v$database;
```

From the output of that query, confirm that the standby database is opened in READ ONLY mode and redo is being applied automatically. For example:

```
OPEN_MODE             DATABASE_ROLE
--------------------  ----------------
READ ONLY WITH APPLY  PHYSICAL STANDBY
```

To configure an Oracle Standby instance as a source when using Binary Reader for CDC

1. Grant additional privileges required to access standby log files.

   ```
   GRANT SELECT ON v_$standby_log TO dms_user;
   ```

2. Create a source endpoint for the Oracle Standby by using the AWS Management Console or AWS CLI. When creating the endpoint, specify the following extra connection attributes.

   ```
   useLogminerReader=N;useBfile=Y;
   ```

   Note

   In AWS DMS, you can use extra connection attributes to specify if you want to migrate from the archive logs instead of the redo logs. For more information, see Endpoint settings when using Oracle as a source for AWS DMS.

3. Configure archived log destination.

   DMS binary reader for Oracle source without ASM uses Oracle Directories to access archived redo logs. If your database is configured to use Fast Recovery Area (FRA) as an archive log destination, the location of archive redo files isn't constant. Each day that archived redo logs are generated results in a new directory being created in the FRA, using the directory name format YYYY_MM_DD. For example:

   ```
   DB_RECOVERY_FILE_DEST/SID/archivelog/YYYY_MM_DD
   ```

When DMS needs access to archived redo files in the newly created FRA directory and the primary read-write database is being used as a source, DMS creates a new or replaces an existing Oracle directory, as follows.

```
CREATE OR REPLACE DIRECTORY dmsrep_taskid AS 'DB_RECOVERY_FILE_DEST/SID/archivelog/
YYYY_MM_DD';
```

When the standby database is being used as a source, DMS is unable to create or replace the Oracle directory because the database is in read-only mode. But, you can choose to perform one of these additional steps:

a. Modify `log_archive_dest_id_1` to use an actual path instead of FRA in such a configuration that Oracle won't create daily subdirectories:

```
ALTER SYSTEM SET log_archive_dest_1='LOCATION=full directory path'
```

Then, create an Oracle directory object to be used by DMS:

```
CREATE OR REPLACE DIRECTORY dms_archived_logs AS 'full directory path';
```

b. Create an additional archive log destination and an Oracle directory object pointing to that destination. For example:

```
ALTER SYSTEM SET log_archive_dest_3='LOCATION=full directory path';
CREATE DIRECTORY dms_archived_log AS 'full directory path';
```

Then add an extra connection attribute to the task source endpoint:

```
archivedLogDestId=3
```

c. Manually pre-create Oracle directory objects to be used by DMS.

```
CREATE DIRECTORY dms_archived_log_20210301 AS 'DB_RECOVERY_FILE_DEST/SID/
archivelog/2021_03_01';
CREATE DIRECTORY dms_archived_log_20210302 AS 'DB_RECOVERY_FILE_DEST>/SID>/
archivelog/2021_03_02';
...
```

d. Create an Oracle scheduler job that runs daily and creates the required directory.

4. Configure online log destination.

   Create Oracle directory that points to OS directory with standby redo logs:

```
CREATE OR REPLACE DIRECTORY STANDBY_REDO_DIR AS '<full directory path>';
GRANT READ ON DIRECTORY STANDBY_REDO_DIR TO <dms_user>;
```

Using a user-managed database on Oracle Cloud Infrastructure (OCI) as a source for CDC in AWS DMS

A user-managed database is a database that you configure and control, such as an Oracle database created on a virtual machine (VM), bare metal, or Exadata server. Or, databases that you configure and control that run on dedicated infrastructure, like Oracle Cloud Infrastructure (OCI). The following information describes the privileges and configurations you need when using an Oracle user-managed database on OCI as a source for change data capture (CDC) in AWS DMS.

To configure an OCI hosted user-managed Oracle database as a source for change data capture

1. Grant required user account privileges for a user-managed Oracle source database on OCI. For more information, see Account privileges for a self-managed Oracle source endpoint.
2. Grant account privileges required when using Binary Reader to access the redo logs. For more information, see Account privileges required when using Binary Reader.
3. Add account privileges that are required when using Binary Reader with Oracle Automatic Storage Management (ASM). For more information, see Additional account privileges required when using Binary Reader with Oracle ASM.
4. Set-up supplemental logging. For more information, see Setting up supplemental logging.
5. Set-up TDE encryption. For more information, see Encryption methods when using an Oracle database as a source endpoint.

The following limitations apply when replicating data from an Oracle source database on Oracle Cloud Infrastructure (OCI).

Limitations

- DMS does not support using Oracle LogMiner to access the redo logs.
- DMS does not support Autonomous DB.

Working with an AWS-managed Oracle database as a source for AWS DMS

An AWS-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 AWS-managed Oracle database with AWS DMS.

User account privileges required on an AWS-managed Oracle source for AWS DMS

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

> **Important**
>
> For all parameter values such as *dms_user* and *any-replicated-table*, Oracle assumes the value is all uppercase unless you specify the value with a case-sensitive identifier. For example, suppose that you create a *dms_user* value without using quotation marks, as in `CREATE USER` *myuser* or `CREATE USER MYUSER`. In this case, Oracle identifies and stores the value as all uppercase (`MYUSER`). If you use quotation marks, as in `CREATE USER "MyUser"` or `CREATE USER 'MyUser'`, Oracle identifies and stores the case-sensitive value that you specify (`MyUser`).

```
GRANT CREATE SESSION to dms_user;
GRANT SELECT ANY TRANSACTION to dms_user;
GRANT SELECT on DBA_TABLESPACES to dms_user;
GRANT SELECT ON any-replicated-table to dms_user;
GRANT EXECUTE on rdsadmin.rdsadmin_util to dms_user;
 -- For Oracle 12c or higher:
GRANT LOGMINING to dms_user; – Required only if the Oracle version is 12c or higher.
```

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', 'dms_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_TAB_PARTITIONS', 'dms_user',
 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_INDEXES', 'dms_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_OBJECTS', 'dms_user', 'SELECT');
exec rdsadmin.rdsadmin_util.grant_sys_object('ALL_TABLES', 'dms_user', 'SELECT');
```

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

-- (as of Oracle versions 12.1 and higher)
exec rdsadmin.rdsadmin_util.grant_sys_object('REGISTRY$SQLPATCH', 'dms_user',
 'SELECT');

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

-- (for transparent data encryption (TDE))

exec rdsadmin.rdsadmin_util.grant_sys_object('ENC$', 'dms_user', 'SELECT');

-- (for validation with LOB columns)
exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_CRYPTO', 'dms_user', 'EXECUTE');

-- (for binary reader)
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_DIRECTORIES','dms_user','SELECT');
```

```
-- Required when the source database is Oracle Data guard, and Oracle Standby is used
 in the latest release of DMS version 3.4.6, version 3.4.7, and higher.

exec rdsadmin.rdsadmin_util.grant_sys_object('V_$DATAGUARD_STATS', 'dms_user',
 'SELECT');
```

For more information on using Amazon RDS Active Dataguard Standby (ADG) with AWS DMS see Using an Amazon RDS Oracle Standby (read replica) as a source with Binary Reader for CDC in AWS DMS.

For more information on using Oracle TDE with AWS DMS, see Supported encryption methods for using Oracle as a source for AWS DMS.

Prerequisites for handling open transactions for Oracle Standby

When using AWS DMS versions 3.4.6 and higher, perform the following steps to handle open transactions for Oracle Standby.

1. Create a database link named, `AWSDMS_DBLINK` on the primary database. *DMS_USER* will use the database link to connect to the primary database. Note that the database link is executed from the standby instance to query the open transactions running on the primary database. See the following example.

```
CREATE PUBLIC DATABASE LINK AWSDMS_DBLINK
   CONNECT TO DMS_USER IDENTIFIED BY DMS_USER_PASSWORD
   USING '(DESCRIPTION=
            (ADDRESS=(PROTOCOL=TCP)(HOST=PRIMARY_HOST_NAME_OR_IP)(PORT=PORT))
            (CONNECT_DATA=(SERVICE_NAME=SID))
          )';
```

2. Verify the connection to the database link using *DMS_USER* is established, as shown in the following example.

```
select 1 from dual@AWSDMS_DBLINK
```

Configuring an AWS-managed Oracle source for AWS DMS

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

- Enable automatic backups. For more information about enabling automatic backups, see [Enabling automated backups](#) in the *Amazon RDS User Guide*.
- Set up supplemental logging.
- 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.

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 following command.

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

   > **Note**
   >
   > The commit is required for a change to take effect.

2. Make sure that your storage has enough space for the archived redo logs during the specified retention period. For example, if your retention period is 24 hours, calculate the total size of your accumulated archived redo logs over a typical hour of transaction processing and multiply that total by 24. Compare this calculated 24-hour total with your available storage space and decide if you have enough storage space to handle a full 24 hours transaction processing.

To set up supplemental logging

1. Run the following command to enable supplemental logging at the database level.

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

2. Run the following command to enable primary key supplemental logging.

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

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

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

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

Configuring a CDC task to use Binary Reader with an RDS for Oracle source for AWS DMS

You can configure AWS DMS to access the source Amazon RDS for Oracle instance redo logs using Binary Reader for CDC.

> **Note**
>
> To use Oracle LogMiner, the minimum required user account privileges are sufficient. For more information, see User account privileges required on an AWS-managed Oracle source for AWS DMS.

To use AWS DMS Binary Reader, specify additional settings and extra connection attributes for the Oracle source endpoint, depending on your AWS DMS version.

Binary Reader support is available in the following versions of Amazon RDS for Oracle:

- Oracle 11.2 – Versions 11.2.0.4V11 and higher
- Oracle 12.1 – Versions 12.1.0.2.V7 and higher
- Oracle 12.2 – All versions
- Oracle 18.0 – All versions
- Oracle 19.0 – All versions

To configure CDC using Binary Reader

1. Log in to your Amazon RDS for Oracle source database as the master user and run the following stored procedures to create the server-level directories.

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

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

2. Grant the following privileges to the Oracle user account that is used to access the Oracle source endpoint.

```
GRANT READ ON DIRECTORY ONLINELOG_DIR TO dms_user;
GRANT READ ON DIRECTORY ARCHIVELOG_DIR TO dms_user;
```

3. Set the following extra connection attributes on the Amazon RDS Oracle source endpoint:

   - For RDS Oracle versions 11.2 and 12.1, set the following.

   ```
   useLogminerReader=N;useBfile=Y;accessAlternateDirectly=false;useAlternateFolderForOnline=
   oraclePathPrefix=/rdsdbdata/db/{$DATABASE_NAME}_A/;usePathPrefix=/rdsdbdata/
   log/;replacePathPrefix=true;
   ```

   - For RDS Oracle versions 12.2, 18.0, and 19.0, set the following.

   ```
   useLogminerReader=N;useBfile=Y;
   ```

> **Note**
>
> Make sure there's no white space following the semicolon separator (;) for multiple attribute settings, for example `oneSetting;thenAnother`.

For more information configuring a CDC task, see Configuration for CDC on an Oracle source database.

Using an Amazon RDS Oracle Standby (read replica) as a source with Binary Reader for CDC in AWS DMS

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

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

After you do so, use the following procedure to use RDS for Oracle Standby as a source when using Binary Reader for CDC.

To configure an RDS for Oracle Standby as a source when using Binary Reader for CDC

1. Sign in to RDS for Oracle primary instance as the master user.
2. Run the following stored procedures 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, the preceding code 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/onlinelog
ONLINELOG_DIR_B	/rdsdbdata/db/ORCL_B/onlinelog

4. Grant the `Read` privilege on the preceding directories to the Oracle user account that is used to access the Oracle Standby.

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

5. Perform an archive log switch on the primary instance. Doing this makes sure that the changes to `ALL_DIRECTORIES` are also ported to the Oracle Standby.
6. Run an `ALL_DIRECTORIES` query on the Oracle Standby to confirm that the changes were applied.

7. Create a source endpoint for the Oracle Standby by using the AWS DMS Management Console or AWS Command Line Interface (AWS CLI). While creating the endpoint, specify the following extra connection attributes.

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

8. After creating the endpoint, use **Test endpoint connection** on the **Create endpoint** page of the 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 supports Oracle Extended data types in AWS DMS version 3.5.0 and higher.
- AWS DMS does not support long object names (over 30 bytes).
- AWS DMS does not 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, see the following topics:
  - For a self-managed Oracle source database, see Setting up supplemental logging.
  - For an AWS-managed Oracle source database, see Configuring an AWS-managed Oracle source for AWS DMS.
- AWS DMS does not support the multi-tenant container root database (CDB$ROOT). It does support a PDB using the Binary Reader.
- AWS DMS does not support deferred constraints.
- AWS DMS version 3.5.3 and above fully supports secure LOBs.
- AWS DMS supports the `rename table` *`table-name`* `to` *`new-table-name`* syntax for all supported Oracle versions 11 and higher. This syntax isn't supported for any Oracle version 10 source databases.
- AWS DMS does not 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`.
- When using AWS DMS version 3.4.7 or higher, to replicate changes that result from partition or subpartition operations, do the following before starting a DMS task.

  - Manually create the partitioned table structure (DDL);
  - Make sure the DDL is the same on both Oracle source and Oracle target;
  - Set the extra connection attribute `enableHomogenousPartitionOps=true`.

  For more information about `enableHomogenousPartitionOps`, see [Endpoint settings when using Oracle as a source for AWS DMS](). Also, note that on FULL+CDC tasks, DMS does not replicate data changes captured as part of the cached changes. In that use case, recreate the table structure on the Oracle target and reload the tables in question.

  Prior to AWS DMS version 3.4.7:

  DMS does not replicate data changes that result from partition or subpartition operations (`ADD`, `DROP`, `EXCHANGE`, and `TRUNCATE`). Such updates might cause the following errors 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 versions prior to 3.4 don'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.
- AWS DMS does not capture changes made by the Oracle `DBMS_REDEFINITION` package, for example the table metadata and the `OBJECT_ID` field.
- When Limited-size LOB mode is enabled, empty BLOB/CLOB columns on the Oracle source are replicated as NULL values. When Full LOB mode is enabled, they are replicated as an empty string (' ').
- 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 does not support batch updates to numeric columns defined as a primary key.

- AWS DMS does not support certain `UPDATE` commands. The following example 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 does not support full LOB mode for loading LONG and LONG RAW columns. Instead, you can use limited LOB mode for migrating these datatypes to an Oracle target. In limited LOB mode, AWS DMS truncates any data to 64 KB that you set to LONG or LONG RAW columns longer than 64 KB.
- AWS DMS does not support full LOB mode for loading XMLTYPE columns. Instead, you can use limited LOB mode for migrating XMLTYPE columns to an Oracle target. In limited LOB mode, DMS truncates any data larger than the user defined 'Maximum LOB size' variable. The maximum recommended value for 'Maximum LOB size' is 100MB.
- AWS DMS does not replicate tables whose names contain apostrophes.
- AWS DMS supports CDC from materialized views. But DMS does not support CDC from any other views.
- AWS DMS does not support CDC for index-organized tables with an overflow segment.
- AWS DMS does not support the `Drop Partition` operation for tables partitioned by reference with `enableHomogenousPartitionOps` set to `true`.
- When you use Oracle LogMiner to access the redo logs, AWS DMS has the following limitations:
  - For Oracle 12 only, AWS DMS does not replicate any changes to LOB columns.
  - AWS DMS does not support XA transactions in replication while using Oracle LogMiner.
  - Oracle LogMiner does not support connections to a pluggable database (PDB). To connect to a PDB, access the redo logs using Binary Reader.
  - SHRINK SPACE operations aren't supported.
- When you use Binary Reader, AWS DMS has these limitations:
  - It does not support table clusters.
  - It supports only table-level `SHRINK SPACE` operations. This level includes the full table, partitions, and sub-partitions.
  - It does not support changes to index-organized tables with key compression.
  - It does not support implementing online redo logs on raw devices.

  - Binary Reader supports TDE only for self-managed Oracle databases since RDS for Oracle does not support wallet password retrieval for TDE encryption keys.
- AWS DMS does not support connections to an Amazon RDS Oracle source using an Oracle Automatic Storage Management (ASM) proxy.
- AWS DMS does not support virtual columns.
- AWS DMS does not support the `ROWID` data type or materialized views based on a ROWID column.

  AWS DMS has partial support for Oracle Materialized Views. For full-loads, DMS can do a full-load copy of an Oracle Materialized View. DMS copies the Materialized View as a base table to the target system and ignores any ROWID columns in the Materialized View. For ongoing replication (CDC), DMS tries to replicate changes to the Materialized View data but the results might not be ideal. Specifically, if the Materialized View is completely refreshed, DMS replicates individual deletes for all the rows, followed by individual inserts for all the rows. That is a very resource intensive exercise and might perform poorly for materialized views with large numbers of rows. For ongoing replication where the materialized views do a fast refresh, DMS tries to process and replicate the fast refresh data changes. In either case, DMS skips any ROWID columns in the materialized view.
- AWS DMS does not 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. An example follows: `alter table yourtablename add supplemental log data (all) columns;`.
- An update for a row with a composite unique key that contains `null` can't be replicated at the target.
- AWS DMS does not support use of multiple Oracle TDE encryption keys on the same source endpoint. Each endpoint can have only one attribute for TDE encryption Key Name "`securityDbEncryptionName`", and one TDE password for this key.
- When replicating from Amazon RDS for Oracle, TDE is supported only with encrypted tablespace and using Oracle LogMiner.
- AWS DMS does not support multiple table rename operations in quick succession.
- When using Oracle 19.0 as source, AWS DMS does not support the following features:
  - Data-guard DML redirect
  - Partitioned hybrid tables
  - Schema-only Oracle accounts

- AWS DMS does not support migration of tables or views of type `BIN$` or `DR$`.
- Beginning with Oracle 18.x, AWS DMS does not support change data capture (CDC) from Oracle Express Edition (Oracle Database XE).
- When migrating data from a CHAR column, DMS truncates any trailing spaces.
- AWS DMS does not support replication from application containers.
- AWS DMS does not support performing Oracle Flashback Database and restore points, as these operations affect the consistency of Oracle Redo Log files.
- Prior to AWS DMS version 3.5.3, Direct-load `INSERT` procedure with the parallel execution option is not supported in the following cases:
  - Uncompressed tables with more than 255 columns
  - Row size exceeds 8K
  - Exadata HCC tables
  - Database running on Big Endian platform
- A source table with neither primary nor unique key requires ALL COLUMN supplemental logging to be enabled. It creates more redo log activities and may increase DMS CDC latency.
- AWS DMS does not migrate data from invisible columns in your source database. To include these columns in your migration scope, use the `ALTER TABLE` statement to make these columns visible.
- For all Oracle versions, AWS DMS does not replicate the result of `UPDATE` operations on `XMLTYPE` and LOB columns.
- AWS DMS does not support replication from tables with temporal validity constraints.
- If the Oracle source becomes unavailable during a full load task, AWS DMS might mark the task as completed after multiple reconnection attempts, even though the data migration remains incomplete. In this scenario, the target tables contain only the records migrated before the connection loss, potentially creating data inconsistencies between the source and target systems. To ensure data completeness, you must either restart the full load task entirely or reload the specific tables affected by the connection interruption.

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
- Using a self-signed certificate for Oracle SSL

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. If your CA certificate contains more than one PEM file (like Amazon RDS global or regional bundle), you must split it into separate files and add them into the Oracle wallet using following bash script. This script requires two parameter inputs: the path to the CA certificate and the path to the folder of the previously created Oracle wallet.

   ```
   #!/usr/bin/env bash
   ```

```
certnum=$(grep -c BEGIN <(cat $1))

cnt=0
temp_cert=""
while read line
do
if [ -n "$temp_cert" -a "$line" == "-----BEGIN CERTIFICATE-----" ]
then
cnt=$(expr $cnt + 1)
printf "\rImporting certificate # $cnt of $certnum"
orapki wallet add -wallet "$2" -trusted_cert -cert <(echo -n "${temp_cert}") -
auto_login_only 1>/dev/null 2>/dev/null
temp_cert=""
fi
temp_cert+="$line"$'\n'
done < <(cat $1)

cnt=$(expr $cnt + 1)
printf "\rImporting certificate # $cnt of $certnum"
orapki wallet add -wallet "$2" -trusted_cert -cert <(echo -n "${temp_cert}") -
auto_login_only 1>/dev/null 2>/dev/null
echo ""
```

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 steps following, assuming an Oracle wallet password of `oracle123`.

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 -p /u01/app/oracle/self_signed_cert
```

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

```
cd /u01/app/oracle/self_signed_cert
```

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 3650 -out self-rootCA.pem
```

Use input parameters like the following.

- `Country Name (2 letter code) [XX]`, for example: `AU`
- `State or Province Name (full name) []`, for example: `NSW`
- `Locality Name (e.g., city) [Default City]`, for example: `Sydney`
- `Organization Name (e.g., company) [Default Company Ltd]`, for example: `AmazonWebService`
- `Organizational Unit Name (e.g., section) []`, for example: `DBeng`
- `Common Name (e.g., your name or your server's hostname) []`, for example: `aws`
- `Email Address []`, for example: abcd.efgh@amazonwebservice.com

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

```
mkdir -p /u01/app/oracle/wallet
```

6. Create a new Oracle wallet.

```
orapki wallet create -wallet "/u01/app/oracle/wallet" -pwd oracle123 -
auto_login_local
```

7. Add the root certificate to the Oracle wallet.

```
orapki wallet add -wallet "/u01/app/oracle/wallet" -pwd oracle123 -trusted_cert
-cert /u01/app/oracle/self_signed_cert/self-rootCA.pem
```

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

```
orapki wallet display -wallet /u01/app/oracle/wallet -pwd oracle123
```

For example, this might display similar to the following.

```
Requested Certificates:
User Certificates:
Trusted Certificates:
Subject:        CN=aws,OU=DBeng,O= AmazonWebService,L=Sydney,ST=NSW,C=AU
```

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

```
orapki wallet add -wallet "/u01/app/oracle/wallet" -pwd oracle123
-dn "CN=aws" -keysize 2048 -sign_alg sha256
```

10. Run the following command.

```
openssl pkcs12 -in /u01/app/oracle/wallet/ewallet.p12 -nodes -out /u01/app/oracle/
wallet/nonoracle_wallet.pem
```

This has output like the following.

```
Enter Import Password:
MAC verified OK
Warning unsupported bag type: secretBag
```

11. Put 'dms' as the common name.

```
openssl req -new -key /u01/app/oracle/wallet/nonoracle_wallet.pem -out certdms.csr
```

Use input parameters like the following.

- `Country Name (2 letter code) [XX]`, for example: `AU`
- `State or Province Name (full name) []`, for example: `NSW`
- `Locality Name (e.g., city) [Default City]`, for example: `Sydney`

- `Organization Name (e.g., company) [Default Company Ltd]`, for example: `AmazonWebService`
- `Organizational Unit Name (e.g., section) []`, for example: `aws`
- `Common Name (e.g., your name or your server's hostname) []`, for example: `aws`
- `Email Address []`, for example: abcd.efgh@amazonwebservice.com

Make sure this is not same as step 4. You can do this, for example, by changing Organizational Unit Name to a different name as shown.

Enter the additional attributes following to be sent with your certificate request.

- `A challenge password []`, for example: `oracle123`
- `An optional company name []`, for example: `aws`

12. Get the certificate signature.

```
openssl req -noout -text -in certdms.csr | grep -i signature
```

The signature key for this post is `sha256WithRSAEncryption` .

13. Run the command following to generate the certificate (`.crt`) file.

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

This displays output like the following.

```
Signature ok
subject=/C=AU/ST=NSW/L=Sydney/O=awsweb/OU=DBeng/CN=aws
Getting CA Private Key
```

14. Add the certificate to the wallet.

```
orapki wallet add -wallet /u01/app/oracle/wallet -pwd oracle123 -user_cert -cert
 certdms.crt
```

15. View the wallet. It should have two entries. See the code following.

```
orapki wallet display -wallet /u01/app/oracle/wallet -pwd oracle123
```

16. Configure the `sqlnet.ora` file (`$ORACLE_HOME/network/admin/sqlnet.ora`).

```
WALLET_LOCATION =
   (SOURCE =
     (METHOD = FILE)
     (METHOD_DATA =
       (DIRECTORY = /u01/app/oracle/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 = /u01/app/oracle/wallet/)
    )
  )

SID_LIST_LISTENER =
 (SID_LIST =
  (SID_DESC =
   (GLOBAL_DBNAME = SID)
   (ORACLE_HOME = ORACLE_HOME)
   (SID_NAME = SID)
  )
 )
```

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

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

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

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

20. Restart the Oracle listener.

```
lsnrctl start
```

21. Show the Oracle listener status.

```
lsnrctl status
```

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

```
sqlplus -L ORACLE_USER@SID_ssl
```

23. Verify that you successfully connected using SSL.

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

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

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

```
cd /u01/app/oracle/self_signed_cert
```

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 self-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 ./
```

This has output like the following.

```
Trusted Certificates:
Subject:        CN=aws,OU=DBeng,O=AmazonWebService,L=Sydney,ST=NSW,C=AU
```

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.

Redo logs access method	TDE tablespace	TDE column
Oracle LogMiner	Yes	Yes
Binary Reader	Yes	Yes

AWS DMS supports Oracle TDE when using Binary Reader, on both the column level and the tablespace level. To use TDE encryption with AWS DMS, first 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 encryption key and password for TDE encryption

1. Run the following query to find the Oracle encryption wallet on the Oracle database host.

   ```
   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. Get the master key ID for either Non-CDB or CDB source as follows:

   a. For non-CDB source run the following query to retrieve Master encryption key ID:

   ```
   SQL>  select rownum, key_id, activation_time from v$encryption_keys;

   ROWNUM KEY_ID                                                      ACTIVATION_TIME
   ------ ------------------------------------------------------ ----------------
        1 AeKask0XZU+NvysflCYBEVwAAAAAAAAAAAAAAAAAAAAAAAAAAAAA   04-SEP-24
    10.20.56.605200 PM +00:00
        2 AV7WU9uhoU8rv8daE/HNnSwAAAAAAAAAAAAAAAAAAAAAAAAAAAAA   10-AUG-21
    07.52.03.966362 PM +00:00
   ```

```
     3 AckpoJ/f+k8xvzJ+gSuoVH4AAAAAAAAAAAAAAAAAAAAAAAAAAAAA   14-SEP-20
 09.26.29.048870 PM +00:00
```

Activation time is useful if you plan to start CDC from some point in the past. For example, using the above results, you can start CDC from some point between 10-AUG-21 07.52.03 PM and 14-SEP-20 09.26.29 PM using the Master Key ID in ROWNUM 2. When the task reaches the redo generated on or after 14-SEP-20 09.26.29 PM it fails, you must modify the source endpoint, provide the Master key ID in ROWNUM 3, and then resume the task.

b. For CDB source DMS requires CDB$ROOT Master encryption key. Connect to CDB$ROOT and execute the following query:

```
SQL> select rownum, key_id, activation_time from v$encryption_keys where con_id
 = 1;

ROWNUM KEY_ID                                                     ACTIVATION_TIME
------ ----------------------------------------------------
 -----------------------------------
     1 Aa2E/Vwb5U+zv5hCncS5ErMAAAAAAAAAAAAAAAAAAAAAAAAAAAAA 29-AUG-24
 12.51.19.699060 AM +00:00
```

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.AWGDC9glSk8Xv+3bVveiVSgAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ORACLE.SECURITY.DB.ENCRYPTION.AY1mRA8OXU9Qvzo3idU4OH4AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ORACLE.SECURITY.DB.ENCRYPTION.MASTERKEY
ORACLE.SECURITY.ID.ENCRYPTION.
ORACLE.SECURITY.KB.ENCRYPTION.
ORACLE.SECURITY.KM.ENCRYPTION.AY1mRA8OXU9Qvzo3idU4OH4AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Find the entry containing the master key ID that you found in step 2 (`AWGDC9glSk8Xv+3bVveiVSg`). This entry is the TDE encryption key name.

4. 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.AWGDC9glSk8Xv+3bVveiVSgAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
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.AWGDC9glSk8Xv+3bVveiVSgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 = AEMAASAASGYs0phWHfNt9J5mEMkkegGFiD4LLfQszDojgDzbfoYDEACv0x3pJC+UGD/
PdtE2jLIcBQcAeHgJChQGLA==
```

Enter the wallet password to see the result.

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

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

```
securityDbEncryptionName=ORACLE.SECURITY.DB.ENCRYPTION.AWGDC9glSk8Xv
+3bVveiVSgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

6. Provide the associated TDE password for this key on the console as part of the Oracle source's **Password** value. Use the following order to format the comma-separated password values, ended by the TDE password value.

```
Oracle_db_password,ASM_Password,AEMAASAASGYs0phWHfNt9J5mEMkkegGFiD4LLfQszDojgDzbfoYDEACv0x3
+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 password values in the following comma-separated order.

```
Oracle_db_password,,AEMAASAASGYs0phWHfNt9J5mEMkkegGFiD4LLfQszDojgDzbfoYDEACv0x3pJC
+UGD/PdtE2jLIcBQcAeHgJChQGLA==
```

If the TDE credentials you specify are incorrect, the AWS DMS migration task does not fail. However, the task also does not read or apply ongoing replication changes to the target database. After starting the task, monitor **Table statistics** on the console migration task page 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 fails. The error message contains the new TDE encryption key name. To specify new values and restart the task, use the preceding procedure.

Important

You can't manipulate a TDE wallet created in an Oracle Automatic Storage Management (ASM) location because OS level commands like `cp`, `mv`, `orapki`, and `mkstore` corrupt the wallet files stored in an ASM location. This restriction is specific to TDE wallet files stored in an ASM location only, but not for TDE wallet files stored in a local OS directory.

To manipulate a TDE wallet stored in ASM with OS level commands, create a local keystore and merge the ASM keystore into the local keystore as follows:

1. Create a local keystore.

```
ADMINISTER KEY MANAGEMENT create keystore file system wallet location
 identified by wallet password;
```

2. Merge the ASM keystore into the local keystore.

```
ADMINISTER KEY MANAGEMENT merge keystore ASM wallet location identified
 by wallet password into existing keystore file system wallet location
 identified by wallet password with backup;
```

Then, to list the encryption wallet entries and TDE password, run steps 3 and 4 against the local keystore.

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

Version	Basic	OLTP	HCC (from Oracle 11g R2 or newer)	Others
Oracle 11 or newer – Oracle LogMiner	Yes	Yes	Yes	Yes – Any compressi on method supported by Oracle LogMiner.
Oracle 11 or newer – Binary Reader	Yes	Yes	Yes – For more information, see the following note .	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

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

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 does not 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;
```

Storing REDO on Oracle ASM when using Oracle as a source for AWS DMS

For Oracle sources with high REDO generation, storing REDO on Oracle ASM can benefit performance, especially in a RAC configuration since you can configure DMS to distribute ASM REDO reads across all ASM nodes.

To utilize this configuration, use the `asmServer` connection attribute. For example, the following connection string distributes DMS REDO reads across 3 ASM nodes:

```
asmServer=(DESCRIPTION=(CONNECT_TIMEOUT=8)(ENABLE=BROKEN)(LOAD_BALANCE=ON)(FAILOVER=ON)
(ADDRESS_LIST=
(ADDRESS=(PROTOCOL=tcp)(HOST=asm_node1_ip_address)(PORT=asm_node1_port_number))
```

```
(ADDRESS=(PROTOCOL=tcp)(HOST=asm_node2_ip_address)(PORT=asm_node2_port_number))
(ADDRESS=(PROTOCOL=tcp)(HOST=asm_node3_ip_address)(PORT=asm_node3_port_number)))
(CONNECT_DATA=(SERVICE_NAME=+ASM)))
```

When using NFS to store Oracle REDO, it's important to make sure that applicable DNFS (direct NFS) client patches are applied, specifically any patch addressing Oracle bug 25224242. For additional information, review the following Oracle Publication regarding Direct NFS client related patches, [Recommended Patches for Direct NFS Client](https://).

Additionally, to improve NFS read performance, we recommended you increase the value of `rsize` and `wsize` in `fstab` for the the NFS volume, as shown in the following example.

```
NAS_name_here:/ora_DATA1_archive /u09/oradata/DATA1 nfs
 rw,bg,hard,nointr,tcp,nfsvers=3,_netdev,
timeo=600,rsize=262144,wsize=262144
```

Also, adjust the `tcp-max-xfer-size` value as follows:

```
vserver nfs modify -vserver vserver -tcp-max-xfer-size 262144
```

Endpoint settings when using Oracle as a source for AWS DMS

You can use endpoint settings to configure your Oracle source database similar to using extra connection attributes. You specify the settings when you create the source endpoint using the AWS DMS console, or by using the `create-endpoint` command in the AWS CLI, with the `--oracle-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following table shows the endpoint settings that you can use with Oracle as a source.

Name	Description
`AccessAlternateDirectly`	Set this attribute to false in order to use the Binary Reader to capture change data for an Amazon RDS for Oracle as the source. This tells the DMS instance to not access redo logs through any specified path prefix replacement using direct file access. For more information, see Configuring a CDC task to use Binary Reader with an RDS for Oracle source for AWS DMS. Default value: true

Name	Description
	Valid values: true/false Example: `--oracle-settings '{"AccessAlternate Directly": false}'`
`Additiona lArchived LogDestId`	Set this attribute with `ArchivedLogDestId` in a primary-Standby setup. This attribute is useful in a switchover when Oracle Data Guard database is used as a source. In this case, AWS DMS needs to know which destination to get archive redo logs from to read changes. This is because the previous primary is now a Standby instance after switchove r. Although AWS DMS supports the use of the Oracle `RESETLOGS` option to open the database, never use `RESETLOGS` unless necessary. For additional information about `RESETLOGS`, see RMAN Data Repair Concepts in the *Oracle® Database Backup and Recovery User's Guide*. Valid values : Archive destination Ids Example: `--oracle-settings '{"AdditionalArchi vedLogDestId": 2}'`

Name	Description
AddSupple mentalLogging	Set this attribute to set up table-level supplemental logging for the Oracle database. This attribute enables one of the following on all tables selected for a migration task, depending on table metadata: • PRIMARY KEY COLUMNS supplemental logging • UNIQUE KEY COLUMNS supplemental logging • ALL COLUMNS supplemental logging Default value: false Valid values: true/false Example: `--oracle-settings '{"AddSupplemental Logging": false}'` **Note** If you use this option, you still need to enable database-level supplemental logging as discussed previously.
AllowSele ctNestedTables	Set this attribute to true to enable replication of Oracle tables containin g columns that are nested tables or defined types. For more informati on, see Replicating nested tables using Oracle as a source for AWS DMS. Default value: false Valid values: true/false Example: `--oracle-settings '{"AllowSelectNest edTables": true}'`

Name	Description
`ArchivedLogDestId`	Specifies the ID of the destination for the archived redo logs. This value should be the same as a number in the dest_id column of the v$archived_log view. If you work with an additional redo log destination, we recommend that you use the `AdditionalArchivedLogDestId` attribute to specify the additional destination ID. Doing this improves performance by ensuring that the correct logs are accessed from the outset. Default value: 1 Valid values: Number Example: `--oracle-settings '{"ArchivedLogDestId": 1}'`
`ArchivedLogsOnly`	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. Default value: N Valid values: Y/N Example: `--oracle-settings '{"ArchivedLogsOnly": Y}'`

Name	Description
`asmUsePLSQLArray` (ECA Only)	Use this extra connection attribute (ECA) when capturing source changes with BinaryReader. This setting allows DMS to buffer 50 reads at ASM level per single read thread while controlling the number of threads using the `parallelASMReadThreads` attribute. When you set this attribute, AWS DMS binary reader uses an anonymous PL/SQL block to capture redo data and send it back to the replication instance as a large buffer. This reduces the number of round trips to the source. This can significantly improve source capture performance, but it does result in higher PGA memory consumption on the ASM Instance. Stability issues might arise if the memory target is not sufficient enough. You can use the following formula to estimate the total ASM instance PGA memory usage by a single DMS task: `number_of_redo_threads * parallelASMReadThreads * 7 MB` Default value: false Valid values: true/false ECA Example: `asmUsePLSQLArray=true;`
`ConvertTimestampWithZoneToUTC`	Set this attribute to `true` to convert the timestamp value of 'TIMESTAMP WITH TIME ZONE' and 'TIMESTAMP WITH LOCAL TIME ZONE' columns to UTC. By default the value of this attribute is 'false' and the data will be replicated using the source database timezone. Default value: false Valid values: true/false Example: `--oracle-settings '{"ConvertTimestampWithZoneToUTC": true}'`

Name	Description
`EnableHomogenousPartitionOps`	Set this attribute to `true` to enable replication of Oracle Partition and subPartition DDL Operations for Oracle *Homogenous* migration. Note that this feature and enhancement was introduced in AWS DMS version 3.4.7. Default value: false Valid values: true/false Example: `--oracle-settings '{"EnableHomogenousPartitionOps": true}'`
`EnableHomogenousTablespace`	Set this attribute to enable homogenous tablespace replication and create existing tables or indexes under the same tablespace on the target. Default value: false Valid values: true/false Example: `--oracle-settings '{"EnableHomogenousTablespace": true}'`

Name	Description
`EscapeCharacter`	Set this attribute to an escape character. This escape character allows you to make a single wildcard character behave as a normal character in table mapping expressions. For more information, see [Wildcards in table mapping](). Default value: Null Valid values: Any character other than a wildcard character Example: `--oracle-settings '{"EscapeCharacter": "#"}'` **Note** You can only use `escapeCharacter` for table names. It does not escape characters from schema names or column names.
`ExposeViews`	Use this attribute to pull data once from a view; you can't use it for ongoing replication. When you extract data from a view, the view is shown as a table on the target schema. Default value: false Valid values: true/false Example: `--oracle-settings '{"ExposeViews": true}'`

Name	Description
`ExtraArchivedLogDestIds`	Specifies the IDs of one more destinations for one or more archived redo logs. These IDs are the values of the dest_id column in the v$archived_log view. Use this setting with the ArchivedLogDestId extra connection attribute in a primary-to-single setup or a primary-to-multiple-standby setup. This setting is useful in a switchover when you use an Oracle Data Guard database as a source. In this case, AWS DMS needs information about what destination to get archive redo logs from to read changes. AWS DMS needs this because after the switchover the previous primary is a standby instance. Valid values: Archive destination Ids Example: `--oracle-settings '{"ExtraArchivedLogDestIds": 1}'`
`FailTasksOnLobTruncation`	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` . 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. Default value: false Valid values: Boolean Example: `--oracle-settings '{"FailTasksOnLobTruncation": true}'`
`filterTransactionsOfUser` (ECA Only)	Use this extra connection attribute (ECA) to allows DMS to ignore transactions from a specified user when replicating data from Oracle when using LogMiner. You can pass comma separated user name values, but they must be in all CAPITAL letters. ECA Example: `filterTransactionsOfUser=` *`USERNAME`*`;`

Name	Description
`NumberDataTypeScale`	Specifies the number scale. You can select a scale up to 38, or you can select -1 for FLOAT, or -2 for VARCHAR. By default, the NUMBER data type is converted to precision 38, scale 10. Default value: 10 Valid values: -2 to 38 (-2 for VARCHAR, -1 for FLOAT) Example: `--oracle-settings '{"NumberDataTypeScale": 12}'` **Note** Select a precision-scale combination, -1 (FLOAT) or -2 (VARCHAR). DMS supports any precision-scale combination supported by Oracle. If precision is 39 or above, select -2 (VARCHAR). The NumberDataTypeScale setting for the Oracle database is used for the NUMBER data type only (without the explicit precision and scale definition). You must note that loss of precision can happen when this setting is incorrectly configured.

Name	Description
OpenTrans actionWindow	Provides the timeframe in minutes to check for any open transactions for CDC only task. **Note** When you set `OpenTransactionWindow` to 1 or higher, DMS uses `SCN_TO_TIMESTAMP` to convert SCN values to timestamp values. Because of Oracle Database limitations, if you specify an SCN that is too old as the CDC start point, SCN_TO_TIMESTAMP will fail with an `ORA-08181` error, and you can't start CDC-only tasks. Default value: 0 Valid values: An integer from 0 to 240 Example: `openTransactionWindow=15;`
OraclePat hPrefix	Set this string attribute to the required value in order to use the Binary Reader to capture change data for an Amazon RDS for Oracle as the source. This value specifies the default Oracle root used to access the redo logs. For more information, see Configuring a CDC task to use Binary Reader with an RDS for Oracle source for AWS DMS. Default value: none Valid value: /rdsdbdata/db/ORCL_A/ Example: `--oracle-settings '{"OraclePathPrefix": " /rdsdbdata/db/ORCL_A/ "}'`

Name	Description
`ParallelA SMReadThreads`	Set this attribute to change the number of threads that DMS configures to perform change data capture (CDC) 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 `ReadAheadBlocks` attribute. For more information, see [Configuring a CDC task to use Binary Reader with an RDS for Oracle source for AWS DMS](). Default value: 2 Valid values: An integer from 2 to 8 Example: `--oracle-settings '{"ParallelASMRead Threads": 6;}'`
`ReadAheadBlocks`	Set this attribute to change the number of read-ahead blocks that DMS configures to perform CDC using Oracle Automatic Storage Managemen t (ASM) and non-ASM NAS storage. You can specify an integer value between 1000 (the default) and 2,000,000 (the maximum). Use this attribute together with the `ParallelASMReadThreads` attribute. For more information, see [Configuring a CDC task to use Binary Reader with an RDS for Oracle source for AWS DMS](). Default value: 1000 Valid values: An integer from 1000 to 2,000,000 Example: `--oracle-settings '{"ReadAheadBlocks": 150000}'`
`ReadTable SpaceName`	When set to `true`, this attribute supports tablespace replication. Default value: false Valid values: Boolean Example: `--oracle-settings '{"ReadTableSpaceName": true}'`

Name	Description
`ReplacePathPrefix`	Set this attribute to true in order to use the Binary Reader to capture change data for an Amazon RDS for Oracle as the source. This setting tells DMS instance to replace the default Oracle root with the specified `UsePathPrefix` setting to access the redo logs. For more information, see Configuring a CDC task to use Binary Reader with an RDS for Oracle source for AWS DMS. Default value: false Valid values: true/false Example: `--oracle-settings '{"ReplacePathPrefix": true}'`
`RetryInterval`	Specifies the number of seconds that the system waits before resending a query. Default value: 5 Valid values: Numbers starting from 1 Example: `--oracle-settings '{"RetryInterval": 6}'`
`SecurityDbEncryptionName`	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 Supported encryption methods for using Oracle as a source for AWS DMS. Default value: "" Valid values: String Example: `--oracle-settings '{"SecurityDbEncryptionName": "ORACLE.SECURITY.DB.ENCRYPTION.Adg8m2dhkU/0v/m5QUaaNJEAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"}'`

Name	Description
`SpatialSdo2GeoJsonFunctionName`	For Oracle version 12.1 or earlier sources migrating to PostgreSQL targets, use this attribute to convert SDO_GEOMETRY to GEOJSON format. By default, AWS DMS calls the `SDO2GEOJSON` custom function which must be present and accessible to the AWS DMS user. Or you can create your own custom function that mimics the operation of `SDOGEOJSON` and set `SpatialSdo2GeoJsonFunctionName` to call it instead. Default value: SDO2GEOJSON Valid values: String Example: `--oracle-settings '{"SpatialSdo2GeoJsonFunctionName": "myCustomSDO2GEOJSONFunction"}'`
`StandbyDelayTime`	Use this attribute to specify a time in minutes for the delay in standby sync. If the source is an Active Data Guard standby database, use this attribute to specify the time lag between primary and standby databases. 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. Default value:0 Valid values: Number Example: `--oracle-settings '{"StandbyDelayTime": 1}'` **Note:** When using DMS 3.4.6, 3.4.7 and higher, use of this connection setting is optional. In the latest version of DMS 3.4.6 and version 3.4.7, *`dms_user`* should have `select` permission on `V_$DATAGUARD_STATS`, allowing DMS to calculate standby delay time.

Name	Description
UseAlternateFolderForOnline	Set this attribute to true in order to use the Binary Reader to capture change data for an Amazon RDS for Oracle as the source. This tells the DMS instance to use any specified prefix replacement to access all online redo logs. For more information, see Configuring a CDC task to use Binary Reader with an RDS for Oracle source for AWS DMS. Default value: false Valid values: true/false Example: `--oracle-settings '{"UseAlternateFolderForOnline": true}'`
UseBfile	Set this attribute to Y in order to capture change data using the Binary Reader utility. Set `UseLogminerReader` to N 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 Using Oracle LogMiner or AWS DMS Binary Reader for CDC. Note: When setting this value as an Extra Connection Attribute (ECA), valid values are 'Y' and 'N'. When setting this value as an endpoint setting, valid values are `true` and `false`. Default value: N Valid values: Y/N (when setting this value as an ECA); true/false (when setting this value as as an endpoint setting). Example: `--oracle-settings '{"UseBfile": Y}'`

Name	Description
`UseLogminerReader`	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 useBfile=Y. For more information on this setting and using Oracle Automatic Storage Management (ASM), see Using Oracle LogMiner or AWS DMS Binary Reader for CDC. Note: When setting this value as an Extra Connection Attribute (ECA), valid values are 'Y' and 'N'. When setting this value as an endpoint setting, valid values are `true` and `false`. Default value: Y Valid values: Y/N (when setting this value as an ECA); true/false (when setting this value as as an endpoint setting). Example: `--oracle-settings '{"UseLogminerReader": Y}'`
`UsePathPrefix`	Set this string attribute to the required value in order to use the Binary Reader to capture change data for an Amazon RDS for Oracle as the source. This value specifies the path prefix used to replace the default Oracle root to access the redo logs. For more information, see Configuring a CDC task to use Binary Reader with an RDS for Oracle source for AWS DMS. Default value: none Valid value: /rdsdbdata/log/ Example: `--oracle-settings '{"UsePathPrefix": " /rdsdbdata/log/ "}'`

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.

Note

With the exception of the LONG and LONG RAW data types, when replicating from an Oracle source to an Oracle target (a *homogeneous replication*), all of the source and target data types will be identical. But the LONG data type will be mapped to CLOB and the LONG RAW data type will be mapped to BLOB.

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](#).

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 greater than 0, use NUMERIC. When scale is 0: • And precision is less than or equal to 2, use INT1. • And precision is greater than 2 and less than or equal to 4, use INT2. • And precision is greater than 4 and less than or equal to 9, use INT4. • And precision is greater than 9, use NUMERIC. • And precision is greater than or equal to scale, use NUMERIC.

Oracle data type	AWS DMS data type
	When scale is less than 0, use REAL8.
DATE	DATETIME
INTERVAL_YEAR TO MONTH	STRING (with interval year_to_month indication)
INTERVAL_DAY TO SECOND	STRING (with interval day_to_second indication)
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	• CLOB when the length is greater than 4,000 bytes • STRING when the length is 4,000 bytes or less
NCHAR	WSTRING
NVARCHAR2	• NCLOB when the length is greater than 4,000 bytes • WSTRING when the length is 4,000 bytes or less
RAW	BYTES
REAL	REAL8
BLOB	BLOB To use this data type with AWS DMS, you must enable the use of BLOB data types for a specific task. AWS DMS supports BLOB data types only in tables that include a primary key.

Oracle data type	AWS DMS data type
CLOB	CLOB To use this data type with AWS DMS, you must enable the use of CLOB data types for a specific task. During 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.
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, enable the use of LOBs for a specific task. During CDC or full load, AWS DMS supports LOB data types only in tables that have a primary key. Also, AWS DMS does not support full LOB mode for loading LONG columns. Instead, you can use limited LOB mode for migrating LONG columns to an Oracle target. In limited LOB mode, AWS DMS truncates any data to 64 KB that you set to LONG columns longer than 64 KB. For more information about LOB support in AWS DMS, see Setting LOB support for source databases in an AWS DMS task

<table>
<tr><th>Oracle data type</th><th>AWS DMS data type</th></tr>
<tr><td>LONG RAW</td><td>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, enable the use of LOBs for a specific task.

During CDC or full load, AWS DMS supports LOB data types only in tables that have a primary key.

Also, AWS DMS does not support full LOB mode for loading LONG RAW columns. Instead, you can use limited LOB mode for migrating LONG RAW columns to an Oracle target. In limited LOB mode, AWS DMS truncates any data to 64 KB that you set to LONG RAW columns longer than 64 KB. For more informati on about LOB support in AWS DMS, see Setting LOB support for source databases in an AWS DMS task</td></tr>
<tr><td>XMLTYPE</td><td>CLOB</td></tr>
<tr><td>SDO_GEOMETRY</td><td>BLOB (when an Oracle to Oracle migration)

CLOB (when an Oracle to PostgreSQL migration)</td></tr>
</table>

Oracle tables used as a source with columns of the following data types are not 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
- VARRAY

> **Note**
>
> Virtual columns are not supported.

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 Endpoint settings when using Oracle as a source for AWS DMS.
- AWS DMS supports Oracle Spatial Column migrations for Full LOB mode only. AWS DMS does not support Limited LOB or Inline LOB modes. For more information about LOB mode, see Setting LOB support for source databases in an AWS DMS task.
- Because AWS DMS only supports Full LOB mode for migrating Oracle Spatial Columns, the columns' table needs a primary key and a unique key. If the table does not have a primary key and a unique key, the table is skipped from migration.

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.

For information about versions of SQL Server that AWS DMS supports as a source, see Sources for AWS DMS.

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. For more information, see Permissions for SQL Server tasks.

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, you want to 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.

You can use CDC for ongoing migration from a SQL Server database. For information about configuring your source SQL server database for CDC, see Capturing data changes for ongoing replication from SQL Server.

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

Topics

Limitations on using SQL Server as a source for AWS DMS

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

- The identity property for a column isn't migrated to a target database column.
- The SQL Server endpoint doesn't support the use of tables with sparse columns.
- 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 UPDATETEXT 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.

- AWS DMS doesn't support server level audits on SQL Server 2008 or SQL Server 2008 R2 as sources. This is because of a known issue with SQL Server 2008 and 2008 R2. 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 and Geography 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 use inline lob mode.
- When using a Microsoft SQL Server source database in a replication task, the SQL Server Replication Publisher definitions aren't removed if you remove the task. A Microsoft SQL Server system administrator must delete those definitions from Microsoft SQL Server.
- Migrating data from schema-bound and non-schema-bound views is supported for full-load only tasks.
- 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';)`
- AWS DMS doesn't support change processing to set and unset column default values (using the `ALTER COLUMN SET DEFAULT` clause with `ALTER TABLE` statements).
- AWS DMS doesn't support change processing to set column nullability (using the `ALTER COLUMN [SET|DROP] NOT NULL` clause with `ALTER TABLE` statements).
- With SQL Server 2012 and SQL Server 2014, when using DMS replication with Availability Groups, the distribution database can't be placed in an availability group. SQL 2016 supports placing the distribution database into an availability group, except for distribution databases used in merge, bidirectional, or peer-to-peer replication topologies.
- For partitioned tables, AWS DMS doesn't support different data compression settings for each partition.
- When inserting a value into SQL Server spatial data types (GEOGRAPHY and GEOMETRY), you can either ignore the spatial reference system identifier (SRID) 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.
- The `readBackupOnly=true` endpoint setting (extra connection attribute) doesn't work on RDS for SQL Server source instances because of the way RDS performs backups.
- `EXCLUSIVE_AUTOMATIC_TRUNCATION` doesn’t work on Amazon RDS SQL Server source instances because RDS users don't have access to run the SQL Server stored procedure, `sp_repldone`.
- AWS DMS doesn't capture truncate commands.
- AWS DMS doesn't support replication from databases with accelerated database recovery (ADR) turned on.
- AWS DMS doesn't support capturing data definition language (DDL) and data manipulation language (DML) statements within a single transaction.
- AWS DMS doesn't support the replication of data-tier application packages (DACPAC).
- UPDATE statements that involve primary keys or unique indexes and update multiple data rows, can cause conflicts when you apply changes to the target database. This might happen, for example, when the target database applies updates as INSERT and DELETE statements instead of a single UPDATE statement. With the batch optimized apply mode, the table might be ignored. With the transactional apply mode, the UPDATE operation might result in constraint violations. To avoid this issue, reload the relevant table. Alternatively, locate the problematic records in the Apply Exceptions control table (`dmslogs.awsdms_apply_exceptions`) and edit them manually in the target database. For more information, see [Change processing tuning settings](#).
- AWS DMS doesn't support the replication of tables and schemas, where the name includes a special character from the following set.

  ```
  \\ -- \n \" \b \r ' \t ;
  ```
- Data masking isn't supported. AWS DMS migrates masked data without masking.

- AWS DMS replicates up to 32,767 tables with primary keys and up to 1,000 columns for each table. This is because AWS DMS creates a SQL Server replication article for each replicated table, and SQL Server replication articles have these limitations.
- When using Change Data Capture (CDC), you must define all columns that make up a unique index as `NOT NULL`. If this requirement is not met, SQL Server system error 22838 will result.
- You may lose events if SQL Server archives from the active transaction log to the backup log, or truncates them from the active transaction log.

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.
- AWS DMS doe snot support direct processing of transaction log backups at the file level from alternative shared folders.
- For Cloud SQL Server sources other than Amazon RDS for Microsoft SQL Server, AWS DMS supports ongoing replication (CDC) with the active transaction log only. You can't use the backup log with CDC. You may lose events if SQL server archives them from the active transaction log to the backup log, or truncates them from the active transaction log before DMS can read it.
- For Amazon RDS for Microsoft SQL Server sources, AWS DMS 3.5.2 and below supports ongoing replication (CDC) with the active transaction log only, because DMS can’t access the backup log with CDC. You may lose events if RDS for SQL Server archives them from the active transaction log to the backup log, or truncate them from the active transaction log before DMS can read it. This limitation does not apply to AWS DMS version 3.5.3 and above.
- AWS DMS does not support CDC for Amazon RDS Proxy for SQL Server as a source.
- If the SQL Server source becomes unavailable during a full load task, AWS DMS might mark the task as completed after multiple reconnection attempts, even though the data migration remains incomplete. In this scenario, the target tables contain only the records migrated before the connection loss, potentially creating data inconsistencies between the source and target systems. To ensure data completeness, you must either restart the full load task entirely or reload the specific tables affected by the connection interruption.

Permissions for SQL Server tasks

Topics

- Permissions for full load only tasks
- Permissions for tasks with ongoing replication

Permissions for full load only tasks

The following permissions are required to perform full load only tasks. Note that AWS DMS does not create the `dms_user` login. For information about creating a login for SQL Server, see Create a database user topic in *Microsoft's documentation*.

```
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;
                GRANT VIEW DEFINITION to dms_user;

                USE master;

                GRANT VIEW SERVER STATE TO dms_user;
```

Permissions for tasks with ongoing replication

Self-managed SQL Server instances can be configured for ongoing replication using DMS with or without using the `sysadmin` role. For SQL Server instances, where you can't grant the `sysadmin` role, ensure that the DMS user has the privileges described as follows.

Set up permissions for ongoing replication from a self-managed SQL Server database

1. Create a new SQL Server account with password authentication using SQL Server Management Studio (SSMS) or as described previously in Permissions for full load only tasks, for example, `self_managed_user`.
2. Run the following GRANT commands:

```
GRANT VIEW SERVER STATE TO self_managed_user;

USE msdb;
    GRANT SELECT ON msdb.dbo.backupset TO self_managed_user;
    GRANT SELECT ON msdb.dbo.backupmediafamily TO self_managed_user;
    GRANT SELECT ON msdb.dbo.backupfile TO self_managed_user;
```

```
USE db_name;
    CREATE USER self_managed_user FOR LOGIN self_managed_user;
    ALTER ROLE [db_owner] ADD MEMBER self_managed_user;
    GRANT VIEW DEFINITION to self_managed_user;
```

3. In addition to the preceding permissions, the user needs one of the following:

   - The user must be a member of the `sysadmin` fixed server role
   - Configurations and permissions as described in Setting up ongoing replication on a SQL Server in an availability group environment: Without sysadmin role or Setting up ongoing replication on a standalone SQL Server: Without sysadmin role, depending on your source configuration.

Set up permissions for ongoing replication from a cloud SQL Server database

A cloud-hosted SQL server instance is an instance running on Amazon RDS for Microsoft SQL Server, an Azure SQL Managed Instance, or any other managed cloud SQL Server instance supported by DMS.

Create a new SQL Server account with password authentication using SQL Server Management Studio (SSMS) or as described previously in Permissions for full load only tasks, for example, `rds_user`.

Run the following grant commands.

```
GRANT VIEW SERVER STATE TO rds_user;
```

For Amazon RDS for Microsoft SQL Server sources, DMS version 3.5.3 and above support reading from transaction log backups. To ensure that DMS is able to access the log backups, in addition to the above, either grant `master` user privileges, or the following privileges on an RDS SQL Server source:

```

USE msdb;
    GRANT EXEC ON msdb.dbo.rds_dms_tlog_download TO rds_user;
    GRANT EXEC ON msdb.dbo.rds_dms_tlog_read TO rds_user;
    GRANT EXEC ON msdb.dbo.rds_dms_tlog_list_current_lsn TO rds_user;
    GRANT EXEC ON msdb.dbo.rds_task_status TO rds_user;

USE db_name;
```

```
    CREATE USER rds_user FOR LOGIN rds_user;
    ALTER ROLE [db_owner] ADD MEMBER rds_user;
    GRANT VIEW DEFINITION to rds_user;
```

For Amazon Azure SQL Managed Instances grant the following privileges:

```
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;
```

Prerequisites for 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 a cloud database such as Amazon RDS or a Microsoft Azure SQL managed instance.

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, SQL Server Replication Publisher definitions for the source used in a DMS CDC task aren't removed when you remove the 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 transaction logging to be turned on in SQL Server. To turn on full transaction logging in SQL Server, either enable MS-REPLICATION or CHANGE DATA CAPTURE (CDC).
- SQL Server *tlog* entries won't be marked for re-use until the MS CDC capture job processes those changes.
- CDC operations aren't supported on memory-optimized tables. This limitation applies to SQL Server 2014 (when the feature was first introduced) and higher.
- AWS DMS change data capture requires a distribution database by default on Amazon EC2 or On-Prem SQL server as source. So, ensure that you have activated the distributor while configuring MS replication for tables with primary keys.

Supported compression methods for SQL Server

Note the following about support for SQL Server compression methods in AWS DMS:

- AWS DMS supports Row/Page compression in SQL Server version 2008 and later.
- AWS DMS doesn't support the Vardecimal storage format.
- AWS DMS doesn't support sparse columns and columnar structure compression.

Working with self-managed SQL Server AlwaysOn availability groups

SQL Server Always On availability groups provide high availability and disaster recovery as an enterprise-level alternative to database mirroring.

In AWS DMS, you can migrate changes from a single primary or secondary availability group replica.

Working with the primary availability group replica

To use the primary availability group as a source in AWS DMS, do the following:

1. Turn on the distribution option for all SQL Server instances in your availability replicas. For more information, see [Setting up ongoing replication on a self-managed SQL Server](#).
2. 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 your availability group listener.

When you start an AWS DMS task for the first time, it might take longer than usual to start. This slowness occurs because the creation of the table articles is being duplicated by the availability group server.

Working with a secondary availability group replica

To use a secondary availability group as a source in AWS DMS, do the following:

1. Use the same credentials for connecting to individual replicas as those used by the AWS DMS source endpoint user.
2. Ensure that your AWS DMS replication instance can resolve DNS names for all existing replicas, and connect to them. You can use the following SQL query to get DNS names for all of your replicas.

```
select ar.replica_server_name, ar.endpoint_url from sys.availability_replicas ar
JOIN sys.availability_databases_cluster adc
ON adc.group_id = ar.group_id AND adc.database_name = '<source_database_name>';
```

3. When you create the source endpoint, specify the DNS name of the availability group listener for the endpoint's **Server name** or for the endpoint secret's **Server address**. For more information about availability group listeners, see [What is an availability group listener?](#) in the SQL Server documentation.

   You can use either a public DNS server or an on-premises DNS server to resolve the availability group listener, the primary replica, and the secondary replicas. To use an on-premises DNS server, configure the Amazon Route 53 Resolver. For more information, see [Using your own on-premises name server](#).
4. Add the following extra connection attributes to your source endpoint.

Extra connection attribute	Value	Notes
`applicationIntent`	`ReadOnly`	Without this ODBC setting, the replication task is routed to the primary availability group replica. For more information, see [SQL Server Native Client Support for High](#)

Extra connection attribute	Value	Notes
		Availability, Disaster Recovery in the SQL Server documentation.
multiSubnetFailover	yes	For more information, see SQL Server Native Client Support for High Availability, Disaster Recovery in the SQL Server documentation.
alwaysOnSharedSynchedBackupIsEnabled	false	For more information, see Endpoint settings when using SQL Server as a source for AWS DMS.
activateSafeguard	false	For more information, see Limitations following.
setUpMsCdcForTables	false	For more information, see Limitations following.

5. Enable the distribution option on all replicas in your availability group. Add all nodes to the distributors list. For more information, see To set up distribution.

6. Run the following query on the primary read-write replica to enable publication of your database. You run this query only once for your database.

```
sp_replicationdboption @dbname = N'<source DB name>', @optname = N'publish', @value
 = N'true';
```

Limitations

Following are limitations for working with a secondary availability group replica:

- AWS DMS doesn't support Safeguard when using a read-only availability group replica as a source. For more information, see Endpoint settings when using SQL Server as a source for AWS DMS.

- AWS DMS doesn't support the `setUpMsCdcForTables` extra connection attribute when using a read-only availability group replica as a source. For more information, see [Endpoint settings when using SQL Server as a source for AWS DMS](#).
- AWS DMS can use a self-managed secondary availability group replica as a source database for ongoing replication (change data capture, or CDC) starting from version 3.4.7. Cloud SQL Server Multi-AZ read replicas are not supported. If you use previous versions of AWS DMS, make sure that you use the primary availability group replica as a source database for CDC.

Failover to other nodes

If you set the `ApplicationIntent` extra connection attribute for your endpoint to `ReadOnly`, your AWS DMS task connects to the read-only node with the highest read-only routing priority. It then fails over to other read-only nodes in your availability group when the highest priority read-only node is unavailable. If you don't set `ApplicationIntent`, your AWS DMS task only connects to the primary (read/write) node in your availability group.

Endpoint settings when using SQL Server as a source for AWS DMS

You can use endpoint settings to configure your SQL Server source database similar to using extra connection attributes. You specify the settings when you create the source endpoint using the AWS DMS console, or by using the `create-endpoint` command in the [AWS CLI](#), with the `--microsoft-sql-server-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following table shows the endpoint settings that you can use with SQL Server as a source.

Name	Description
`ActivateSafeguard`	This attribute turns Safeguard on or off. For information about Safeguard, see `SafeguardPolicy` following. Default value: `true` Valid values: {`false`, `true`} Example: `'{"ActivateSafeguard": true}'`
`AlwaysOnSharedSynchedBackupIsEnabled`	This attribute adjusts the behavior of AWS DMS when migrating from an SQL Server source database that

Name	Description
	is hosted as part of an Always On availability group cluster. 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 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 can't connect to any one of the nodes. If you need AWS DMS to poll all the nodes in the Always On cluster for transaction backups, set this attribute to `false`. Default value: `true` Valid values: `true` or `false` Example: `'{"AlwaysOnSharedSynchedBackupIsEnabled": false}'`
`"ApplicationIntent": "readonly"`	This ODBC driver attribute setting causes SQL Server to route your replication task to the highest priority read-only node. Without this setting, SQL Server routes your replication task to the primary read-write node.
`ConnectionTimeout`	Use this extra connection attribute (ECA) to set the endpoint connection timeout for the SQL Server instance, in seconds. The default value is 10 seconds. ECA Example: `ConnectionTimeout=30` .

Name	Description
`EnableNonSysadminWrapper`	Use this endpoint setting when you are setting up ongoing replication on a standalone SQL server without a sysadmin user. This parameter is supported on AWS DMS version 3.4.7 and higher. For information about setting up ongoing replication on a standalone SQL server, see Capturing data changes for ongoing replication from SQL Server. Default value: `false` Valid values: `true`, `false` Example: `'{"EnableNonSysadminWrapper": true}'`
`ExecuteTimeout`	Use this extra connection attribute (ECA) to set the client statement timeout for the SQL Server instance, in seconds. The default value is 60 seconds. Example: `'{"ExecuteTimeout": 100}'`
`FatalOnSimpleModel`	When set to `true`, this setting generates a fatal error when SQL Server database recovery model is set to `simple`. Default value: `false` Valid values: `true` or `false` Example: `'{"FatalOnSimpleModel": true}'`
`ForceLobLookup`	Forces LOB lookup on inline LOB. Default value: `false` Valid values: `true`, `false` Example: `'{"ForceLobLookup": false}'`

Name	Description
`"MultiSubnetFailover": "Yes"`	This ODBC driver attribute helps DMS to connect to the new primary in case of an Availability Group failover. This attribute is designed for situations when the connection is broken or the listener IP address is incorrect. In these situations, AWS DMS attempts to connect to all IP addresses associated with the Availabil ity Group listener.
`ReadBackupOnly`	Use of this attribute requires **sysadmin** privileges. When this attribute is set to `true`, during ongoing replicati on AWS DMS reads changes only from transaction log backups and doesn't read from the active transaction log file. Setting this parameter to `true` 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. Valid values: `true` or `true`. The default is `false`. Example: `'{"ReadBackupOnly": true}'` **Note** This parameter does not work on Amazon RDS SQL Server source instances because of the way RDS performs backups.

Name	Description
`SafeguardPolicy`	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 the unread changes. When this occurs, AWS DMS accesses the log backup to capture the missing changes. To minimize the need to access the log backup, AWS DMS prevents truncation using one of the following methods: 1. `RELY_ON_SQL_SERVER_REPLICATION_AGENT` (**Start transactions in the database)**: This is the default for AWS DMS. When you use this setting, AWS DMS requires that the SQL Server Log Reader Agent be running, so that AWS DMS can move transactions that are marked for replication from the active TLOG. Note that if the Log Reader Agent is not running, the active TLOG can become full, causing the source database to switch to read-only mode until you can resolve the issue. If you need to enable Microsoft Replication in your database for a purpose other than AWS DMS, then you must choose this setting. When you use this setting, AWS DMS minimizes log backup reads by creating a table called `awsdms_truncation_safeguard` and prevents TLOG truncation by mimicking an open transaction in the database. This keeps the database from truncating events and moving them to the backup log for five minutes (by default). Make sure that the table is not included in any maintenance plan, as it may cause the maintenance job to fail. You can safely delete the table if there are no tasks configured with the `Start Transactions` database option.

Name	Description
	2. `EXCLUSIVE_AUTOMATIC_TRUNCATION` **(Exclusively use `sp_repldone` with a single task)**: When you use this setting, AWS DMS has full control of the replication agent process that marks log entries as `ready for truncation` using `sp_repldone`. With this setting, AWS DMS doesn't use a dummy transaction as with the `RELY_ON_SQL_SERVER_REPLICATION_AGENT` (default) setting. You can only use this setting when MS Replication is not used for any other purpose other than AWS DMS on the source database. Also, when using this setting, only one AWS DMS task can access the database. If you need to run parallel AWS DMS tasks against the same database, use `RELY_ON_SQL_SERVER_REPLICATION_AGENT`. • This setting requires that the Log Reader Agent be stopped in the database. If the Log Reader Agent is running when the task starts, the AWS DMS task will force it to stop. Alternatively, you can stop the Log Reader Agent manually before starting the task. • When using this method with MS-CDC, you should stop and disable the **MS-CDC capture** and **MS-CDC cleanup** jobs. • You can't use this setting when the **Microsoft SQL Server Migration** job runs on a remote Distributor machine, because AWS DMS doesn't have access to the remote machine. • `EXCLUSIVE_AUTOMATIC_TRUNCATION` doesn't work on Amazon RDS for SQL Server source instances because Amazon RDS users don't have access to run the `sp_repldone` stored procedure. • If you set `SafeguardPolicy` to `EXCLUSIVE_AUTOMATIC_TRUNCATION` without using

Name	Description
	the sysadmin role, you must grant permissions on the `dbo.syscategories` and `dbo.sysjobs` objects to the `dmsuser` user. Default value: `RELY_ON_SQL_SERVER_REPLICATION_AGENT` Valid values: {`EXCLUSIVE_AUTOMATIC_TRUNCATION` , `RELY_ON_SQL_SERVER_REPLICATION_AGENT` } Example: `'{"SafeguardPolicy": "EXCLUSIVE_AUTOMATIC_TRUNCATION"}'`
`SetUpMsCdcForTables`	This attribute turns on MS-CDC for the source database and for tables in the task mapping that don't have MS-Replication enabled. Setting this value to `true` runs the `sp_cdc_enable_db` stored procedure on the source database, and runs the `sp_cdc_enable_table` stored procedure on each table in the task that doesn't have MS-Replication enabled in the source database. For more information about turning on distribution, see Setting up ongoing replication on a self-managed SQL Server. Valid values: {`true`, `false`} Example: `'{"SetUpMsCdcForTables": true}'`

Name	Description
`TlogAccessMode`	Indicates the mode used to fetch CDC data. Default value: `PreferTlog` Valid values: `BackupOnly` , `PreferBackup` , `PreferTlog` , `TlogOnly` Example: `'{"TlogAccessMode": "PreferTlog"}'`
`Use3rdPartyBackupDevice`	When this attribute is set to Y, AWS DMS processes third-party transaction log backups if they are created in native format.

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.

SQL Server data types	AWS DMS data types
BIGINT	INT8
BIT	BOOLEAN
DECIMAL	NUMERIC
INT	INT4
MONEY	NUMERIC

SQL Server data types	AWS DMS data types
NUMERIC (p,s)	NUMERIC
SMALLINT	INT2
SMALLMONEY	NUMERIC
TINYINT	UINT1
REAL	REAL4
FLOAT	REAL8
DATETIME	DATETIME
DATETIME2 (SQL Server 2008 and higher)	DATETIME
SMALLDATETIME	DATETIME
DATE	DATE
TIME	TIME
DATETIMEOFFSET	WSTRING
CHAR	STRING
VARCHAR	STRING

SQL Server data types	AWS DMS data types
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
NVARCHAR (max)	NCLOB NTEXT To use this data type with AWS DMS, you must enable the use of SupportLobs for a specific task. For more information about enabling Lob support, see Setting LOB support for source databases in an AWS DMS 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.

SQL Server data types	AWS DMS data types
BINARY	BYTES
VARBINARY	BYTES
VARBINARY (max)	BLOB IMAGE For SQL Server tables, AWS DMS updates LOB columns in the target even for UPDATE statements that don't change the value of the LOB column in SQL Server. To use this data type with AWS DMS, you must enable the use of BLOB data types for a specific task. AWS DMS supports BLOB data types only in tables that include a primary key.
TIMESTAMP	BYTES
UNIQUEIDENTIFIER	STRING
HIERARCHYID	Use HIERARCHYID when replicating to a SQL Server target endpoint. Use WSTRING (250) when replicating to all other target endpoints.

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

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

- CURSOR
- SQL_VARIANT
- TABLE

Note

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

Capturing data changes for ongoing replication from SQL Server

This topic describes how to set up CDC replication on a SQL Server source.

Topics

Capturing data changes for self-managed SQL Server on-premises or on Amazon EC2

To capture changes from a source Microsoft SQL Server database, make sure that the database is configured for full backups. Configure the database either in full recovery mode or bulk-logged mode.

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 sysadmin privileges to the AWS DMS endpoint user on the source SQL Server instance. Or you can follow the steps in this section to prepare the source and use a user that doesn't have sysadmin privileges for the AWS DMS endpoint.

MS-CDC

To capture changes for tables without primary keys. Enable MS-CDC at the database level and for all of the tables individually.

When setting up a SQL Server database for ongoing replication (CDC), you can do one of the following:

- Set up ongoing replication using the sysadmin role.

- Set up ongoing replication to not use the sysadmin role.

> **Note**
>
> You can use the following script to find all table without a primary or a unique key:
>
> ```
> USE [DBname]
> SELECT SCHEMA_NAME(schema_id) AS schema_name, name AS table_name
> FROM sys.tables
> WHERE OBJECTPROPERTY(object_id, 'TableHasPrimaryKey') = 0
>         AND  OBJECTPROPERTY(object_id, 'TableHasUniqueCnst') = 0
> ORDER BY schema_name, table_name;
> ```

Setting up ongoing replication on a self-managed SQL Server

This section contains information about setting up ongoing replication on a self-managed SQL server with or without using the sysadmin role.

Topics

Setting up ongoing replication on a self-managed SQL Server: Using sysadmin role

AWS DMS ongoing replication for SQL Server uses native SQL Server replication for tables with primary keys, and change data capture (CDC) for tables without primary keys.

Before setting up ongoing replication, see Prerequisites for using ongoing replication (CDC) from a SQL Server source.

For tables with primary keys, AWS DMS can generally configure the required artifacts on the source. However, for SQL Server source instances that are self-managed, make sure to first configure the SQL Server distribution manually. After you do so, AWS DMS source 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 isn't configured. You can use the following procedure to set up distribution.

To set up distribution

1. Connect to your SQL Server source database using the SQL Server Management Studio (SSMS) tool.
2. Open the context (right-click) menu for the **Replication** folder, and choose **Configure Distribution**. The Configure Distribution wizard appears.
3. Follow the wizard to enter the default values and create the distribution.

To set up CDC

AWS DMS version 3.4.7 and greater can set up MS CDC for your database and all of your tables automatically if you aren't using a read-only replica. To use this feature, set the `SetUpMsCdcForTables` ECA to true. For information about ECAs, see Endpoint settings.

For versions of AWS DMS earlier than 3.4.7, or for a read-only replica as a source, perform the following steps:

1. For tables without primary keys, set up MS-CDC for the database. To do so, use an account that has the sysadmin role assigned to it, and run the following command.

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

2. Next, set up MS-CDC for each of the source tables. For each table with unique keys but no primary key, run the following query to set up 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
```

3. For each table with no primary key or no unique keys, run the following query to set up MS-CDC.

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

For more information on setting up MS-CDC for specific tables, see the SQL Server documentation.

Setting up ongoing replication on a standalone SQL Server: Without sysadmin role

This section describes how to set up ongoing replication for a standalone SQL Server database source that doesn't require the user account to have sysadmin privileges.

Note

After running the steps in this section, the non-sysadmin DMS user will have permissions to do the following:

- Read changes from the online transactions log file
- Disk access to read changes from transactional log backup files
- Add or alter the publication which DMS uses
- Add articles to the publication

1. Set up Microsoft SQL Server for Replication as described in Capturing data changes for ongoing replication from SQL Server.
2. Enable MS-REPLICATION on the source database. This can either be done manually or by running the task once as a sysadmin user.
3. Create the awsdms schema on the source database using the following script:

```
use master
go
create schema awsdms
go
```

```
-- Create the table valued function [awsdms].[split_partition_list] on the Master
 database, as follows:
USE [master]
GO

set ansi_nulls on
go

set quoted_identifier on
go

if (object_id('[awsdms].[split_partition_list]','TF')) is not null

drop function [awsdms].[split_partition_list];

go

create function [awsdms].[split_partition_list]

(

@plist varchar(8000), --A delimited list of partitions

@dlm nvarchar(1) --Delimiting character

)

returns @partitionsTable table --Table holding the BIGINT values of the string
 fragments

(

pid bigint primary key

)

as

begin

declare @partition_id bigint;
```

```
declare @dlm_pos integer;

declare @dlm_len integer;

set @dlm_len = len(@dlm);

while (charindex(@dlm,@plist)>0)

begin

set @dlm_pos = charindex(@dlm,@plist);

set @partition_id = cast( ltrim(rtrim(substring(@plist,1,@dlm_pos-1))) as bigint);

insert into @partitionsTable (pid) values (@partition_id)

set @plist = substring(@plist,@dlm_pos+@dlm_len,len(@plist));

end

set @partition_id = cast (ltrim(rtrim(@plist)) as bigint);

insert into @partitionsTable (pid) values ( @partition_id );

return

end

GO
```

4. Create the `[awsdms].[rtm_dump_dblog]` procedure on the Master database using the following script:

```
use [MASTER]

go

if (object_id('[awsdms].[rtm_dump_dblog]','P')) is not null drop procedure
 [awsdms].[rtm_dump_dblog];
go


set ansi_nulls on
```

```
go

set quoted_identifier on
GO



CREATE procedure [awsdms].[rtm_dump_dblog]

(

@start_lsn varchar(32),

@seqno integer,

@filename varchar(260),

@partition_list varchar(8000), -- A comma delimited list: P1,P2,... Pn

@programmed_filtering integer,

@minPartition bigint,

@maxPartition bigint

)

as begin

declare @start_lsn_cmp varchar(32); -- Stands against the GT comparator

SET NOCOUNT ON -- – Disable "rows affected display"

set @start_lsn_cmp = @start_lsn;

if (@start_lsn_cmp) is null

set @start_lsn_cmp = '00000000:00000000:0000';

if (@partition_list is null)

begin

RAISERROR ('Null partition list waspassed',16,1);
```

```
return

end

if (@start_lsn) is not null

set @start_lsn = '0x'+@start_lsn;

if (@programmed_filtering=0)


SELECT

[Current LSN],

[operation],

[Context],

[Transaction ID],

[Transaction Name],

[Begin Time],

[End Time],

[Flag Bits],

[PartitionID],

[Page ID],

[Slot ID],

[RowLog Contents 0],

[Log Record],

[RowLog Contents 1]

FROM
```

```
fn_dump_dblog (

@start_lsn, NULL, N'DISK', @seqno, @filename,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default)

where [Current LSN] collate SQL_Latin1_General_CP1_CI_AS > @start_lsn_cmp collate
 SQL_Latin1_General_CP1_CI_AS

and

(

( [operation] in ('LOP_BEGIN_XACT','LOP_COMMIT_XACT','LOP_ABORT_XACT') )

or

( [operation] in ('LOP_INSERT_ROWS','LOP_DELETE_ROWS','LOP_MODIFY_ROW')

and

( ( [context] in ('LCX_HEAP','LCX_CLUSTERED','LCX_MARK_AS_GHOST') ) or ([context] =
 'LCX_TEXT_MIX' and (datalength([RowLog Contents 0]) in (0,1))))

and [PartitionID] in ( select * from master.awsdms.split_partition_list
 (@partition_list,','))
```

```
)

or

([operation] = 'LOP_HOBT_DDL')

)


else


SELECT

[Current LSN],

[operation],

[Context],

[Transaction ID],

[Transaction Name],

[Begin Time],

[End Time],

[Flag Bits],

[PartitionID],

[Page ID],

[Slot ID],

[RowLog Contents 0],

[Log Record],

[RowLog Contents 1] -- After Image

FROM
```

```
fn_dump_dblog (

@start_lsn, NULL, N'DISK', @seqno, @filename,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default,

default, default, default, default, default, default, default)

where [Current LSN] collate SQL_Latin1_General_CP1_CI_AS > @start_lsn_cmp collate
 SQL_Latin1_General_CP1_CI_AS

and

(

( [operation] in ('LOP_BEGIN_XACT','LOP_COMMIT_XACT','LOP_ABORT_XACT') )

or

( [operation] in ('LOP_INSERT_ROWS','LOP_DELETE_ROWS','LOP_MODIFY_ROW')

and

( ( [context] in ('LCX_HEAP','LCX_CLUSTERED','LCX_MARK_AS_GHOST') ) or ([context] =
 'LCX_TEXT_MIX' and (datalength([RowLog Contents 0]) in (0,1))))

and ([PartitionID] is not null) and ([PartitionID] >= @minPartition and
 [PartitionID]<=@maxPartition)

)
```

```
or

([operation] = 'LOP_HOBT_DDL')

)



SET NOCOUNT OFF -- Re-enable "rows affected display"

end

GO
```

5. Create the certificate on the Master database using the following script:

```
Use [master]
Go

CREATE CERTIFICATE [awsdms_rtm_dump_dblog_cert] ENCRYPTION BY PASSWORD =
 N'@5trongpassword'

WITH SUBJECT = N'Certificate for FN_DUMP_DBLOG Permissions';
```

6. Create the login from the certificate using the following script:

```
Use [master]
Go

CREATE LOGIN awsdms_rtm_dump_dblog_login FROM CERTIFICATE
 [awsdms_rtm_dump_dblog_cert];
```

7. Add the login to the sysadmin server role using the following script:

```
ALTER SERVER ROLE [sysadmin] ADD MEMBER [awsdms_rtm_dump_dblog_login];
```

8. Add the signature to [master].[awsdms].[rtm_dump_dblog] using the certificate, using the following script:

```
Use [master]
GO
ADD SIGNATURE
```

```
TO [master].[awsdms].[rtm_dump_dblog] BY CERTIFICATE [awsdms_rtm_dump_dblog_cert]
 WITH PASSWORD = '@5trongpassword';
```

Note

If you recreate the stored procedure, you need to add the signature again.

9. Create the [awsdms].[rtm_position_1st_timestamp] on the Master database using the following script:

```
use [master]
    if object_id('[awsdms].[rtm_position_1st_timestamp]','P') is not null
    DROP PROCEDURE [awsdms].[rtm_position_1st_timestamp];
    go
    create procedure [awsdms].[rtm_position_1st_timestamp]
    (
    @dbname                sysname,      -- Database name
    @seqno                 integer,      -- Backup set sequence/position number
 within file
    @filename              varchar(260), -- The backup filename
    @1stTimeStamp          varchar(40)   -- The timestamp to position by
    )
    as begin

    SET NOCOUNT ON       -- Disable "rows affected display"

    declare @firstMatching table
    (
    cLsn varchar(32),
    bTim datetime
    )

    declare @sql nvarchar(4000)
    declare @nl                       char(2)
    declare @tb                       char(2)
    declare @fnameVar                 nvarchar(254) = 'NULL'

    set @nl  = char(10); -- New line
    set @tb  = char(9)   -- Tab separator

    if (@filename is not null)
    set @fnameVar = ''''+@filename +''''
```

```
    set @sql='use ['+@dbname+'];'+@nl+
    'select top 1 [Current LSN],[Begin Time]'+@nl+
    'FROM fn_dump_dblog (NULL, NULL, NULL, '+ cast(@seqno as varchar(10))+','+
 @fnameVar+','+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default)'+@nl+
    'where operation=''LOP_BEGIN_XACT''' +@nl+
    'and [Begin Time]>= cast('+''''+@1stTimeStamp+''''+' as datetime)'+@nl

    --print @sql
    delete from  @firstMatching
    insert into @firstMatching  exec sp_executesql @sql    -- Get them all

    select top 1 cLsn as [matching LSN],convert(varchar,bTim,121) as [matching
 Timestamp] from @firstMatching;

    SET NOCOUNT OFF      -- Re-enable "rows affected display"

    end
    GO
```

10. Create the certificate on the Master database using the following script:

```
Use [master]
Go
CREATE CERTIFICATE [awsdms_rtm_position_1st_timestamp_cert]
ENCRYPTION BY PASSWORD = '@5trongpassword'
WITH SUBJECT = N'Certificate for FN_POSITION_1st_TIMESTAMP Permissions';
```

11. Create the login from the certificate using the following script:

```
Use [master]
Go
CREATE LOGIN awsdms_rtm_position_1st_timestamp_login FROM CERTIFICATE
 [awsdms_rtm_position_1st_timestamp_cert];
```

12. Add the login to the sysadmin role using the following script:

```
ALTER SERVER ROLE [sysadmin] ADD MEMBER [awsdms_rtm_position_1st_timestamp_login];
```

13. Add the signature to [master].[awsdms].[rtm_position_1st_timestamp] using the certificate, using the following script:

```
Use [master]
    GO
    ADD SIGNATURE
    TO [master].[awsdms].[rtm_position_1st_timestamp]
    BY CERTIFICATE [awsdms_rtm_position_1st_timestamp_cert]
    WITH PASSWORD = '@5trongpassword';
```

14. Grant the DMS user execute access to the new stored procedure using the following script:

```
use master
go
GRANT execute on [awsdms].[rtm_position_1st_timestamp] to dms_user;
```

15. Create a user with the following permissions and roles in each of the following databases:

> **Note**
>
> You should create the dmsnosysadmin user account with the same SID on each replica. The following SQL query can help verify the dmsnosysadmin account SID value on each replica. For more information about creating a user, see CREATE USER (Transact-SQL) in the Microsoft SQL server documentation. For more information about creating SQL user accounts for Azure SQL database, see Active geo-replication .

```
use master
go
grant select on sys.fn_dblog to [DMS_user]
grant view any definition to [DMS_user]
grant view server state to [DMS_user]--(should be granted to the login).
grant execute on sp_repldone to [DMS_user]
grant execute on sp_replincrementlsn to [DMS_user]
grant execute on sp_addpublication to [DMS_user]
grant execute on sp_addarticle to [DMS_user]
grant execute on sp_articlefilter to [DMS_user]
```

```
grant select on [awsdms].[split_partition_list] to [DMS_user]
grant execute on [awsdms].[rtm_dump_dblog] to [DMS_user]
```

```
use msdb
go
grant select on msdb.dbo.backupset to self_managed_user
grant select on msdb.dbo.backupmediafamily to self_managed_user
grant select on msdb.dbo.backupfile to self_managed_user
```

Run the following script on the source database:

```
use Source_DB
    Go
    EXEC sp_addrolemember N'db_owner', N'DMS_user'
```

16. Lastly, add an Extra Connection Attribute (ECA) to the source SQL Server endpoint:

```
enableNonSysadminWrapper=true;
```

Setting up ongoing replication on a SQL Server in an availability group environment: Without sysadmin role

This section describes how to set up ongoing replication for a SQL Server database source in an availability group environment that doesn't require the user account to have sysadmin privileges.

> **Note**
>
> After running the steps in this section, the non-sysadmin DMS user will have permissions to do the following:
>
> - Read changes from the online transactions log file
> - Disk access to read changes from transactional log backup files
> - Add or alter the publication which DMS uses
> - Add articles to the publication

To set up ongoing replication without using the sysadmin user in an Availability Group environment

1. Set up Microsoft SQL Server for Replication as described in [Capturing data changes for ongoing replication from SQL Server](#).
2. Enable MS-REPLICATION on the source database. This can either be done manually or by running the task once using a sysadmin user.

> **Note**
>
> You should either configure the MS-REPLICATION distributor as local or in a way that allows access to non-sysadmin users via the associated linked server.

3. If the **Exclusively use sp_repldone within a single task** endpoint option is enabled, stop the MS-REPLICATION Log Reader job.
4. Perform the following steps on each replica:

   1. Create the `[awsdms]`[awsdms] schema in the master database:

      ```
      CREATE SCHEMA [awsdms]
      ```

   2. Create the `[awsdms].[split_partition_list]` table valued function on the Master database:

      ```
      USE [master]
      GO

      SET ansi_nulls on
      GO

      SET quoted_identifier on
      GO

      IF (object_id('[awsdms].[split_partition_list]','TF')) is not null
        DROP FUNCTION [awsdms].[split_partition_list];
      GO

      CREATE FUNCTION [awsdms].[split_partition_list]
      (
        @plist varchar(8000),    --A delimited list of partitions
      ```

```
  @dlm nvarchar(1)    --Delimiting character
)
RETURNS @partitionsTable table --Table holding the BIGINT values of the string
 fragments
(
  pid bigint primary key
)
AS
BEGIN
  DECLARE @partition_id bigint;
  DECLARE @dlm_pos integer;
  DECLARE @dlm_len integer;
  SET @dlm_len = len(@dlm);
  WHILE (charindex(@dlm,@plist)>0)
  BEGIN
    SET @dlm_pos = charindex(@dlm,@plist);
    SET @partition_id = cast( ltrim(rtrim(substring(@plist,1,@dlm_pos-1))) as
 bigint);
    INSERT into @partitionsTable (pid) values (@partition_id)
    SET @plist = substring(@plist,@dlm_pos+@dlm_len,len(@plist));
  END
  SET @partition_id = cast (ltrim(rtrim(@plist)) as bigint);
  INSERT into @partitionsTable (pid) values (  @partition_id  );
  RETURN
END
GO
```

3. Create the `[awsdms].[rtm_dump_dblog]` procedure on the Master database:

```
USE [MASTER]
GO

IF (object_id('[awsdms].[rtm_dump_dblog]','P')) is not null
  DROP PROCEDURE [awsdms].[rtm_dump_dblog];
GO

SET ansi_nulls on
GO

SET quoted_identifier on
GO

CREATE PROCEDURE [awsdms].[rtm_dump_dblog]
(
```

```
  @start_lsn            varchar(32),
  @seqno                integer,
  @filename             varchar(260),
  @partition_list       varchar(8000), -- A comma delimited list: P1,P2,... Pn
  @programmed_filtering integer,
  @minPartition         bigint,
  @maxPartition         bigint
)
AS
BEGIN

  DECLARE @start_lsn_cmp varchar(32); -- Stands against the GT comparator

  SET NOCOUNT ON  -- Disable "rows affected display"

  SET @start_lsn_cmp = @start_lsn;
  IF (@start_lsn_cmp) is null
    SET @start_lsn_cmp = '00000000:00000000:0000';

  IF (@partition_list is null)
    BEGIN
      RAISERROR ('Null partition list was passed',16,1);
      return
      --set @partition_list = '0,';    -- A dummy which is never matched
    END

  IF (@start_lsn) is not null
    SET @start_lsn = '0x'+@start_lsn;

  IF (@programmed_filtering=0)
    SELECT
      [Current LSN],
      [operation],
      [Context],
      [Transaction ID],
      [Transaction Name],
      [Begin Time],
      [End Time],
      [Flag Bits],
      [PartitionID],
      [Page ID],
      [Slot ID],
      [RowLog Contents 0],
      [Log Record],
```

```
      [RowLog Contents 1] -- After Image
    FROM
      fn_dump_dblog (
        @start_lsn, NULL, N'DISK', @seqno, @filename,
        default, default, default, default, default, default, default,
        default, default, default, default, default, default, default,
        default, default, default, default, default, default, default,
        default, default, default, default, default, default, default,
        default, default, default, default, default, default, default,
        default, default, default, default, default, default, default,
        default, default, default, default, default, default, default,
        default, default, default, default, default, default, default,
        default, default, default, default, default, default, default)
    WHERE
      [Current LSN] collate SQL_Latin1_General_CP1_CI_AS > @start_lsn_cmp collate
SQL_Latin1_General_CP1_CI_AS -- This aims for implementing FN_DBLOG based on GT
comparator.
      AND
      (
        (  [operation] in ('LOP_BEGIN_XACT','LOP_COMMIT_XACT','LOP_ABORT_XACT') )
        OR
        (  [operation] in ('LOP_INSERT_ROWS','LOP_DELETE_ROWS','LOP_MODIFY_ROW')
          AND
          ( ( [context]   in ('LCX_HEAP','LCX_CLUSTERED','LCX_MARK_AS_GHOST') )
or ([context] = 'LCX_TEXT_MIX') )
          AND
          [PartitionID] in ( select * from master.awsdms.split_partition_list
(@partition_list,','))
        )
      OR
      ([operation] = 'LOP_HOBT_DDL')
    )
    ELSE
      SELECT
        [Current LSN],
        [operation],
        [Context],
        [Transaction ID],
        [Transaction Name],
        [Begin Time],
        [End Time],
        [Flag Bits],
        [PartitionID],
        [Page ID],
```

```
        [Slot ID],
        [RowLog Contents 0],
        [Log Record],
        [RowLog Contents 1] -- After Image
      FROM
        fn_dump_dblog (
          @start_lsn, NULL, N'DISK', @seqno, @filename,
          default, default, default, default, default, default, default,
          default, default, default, default, default, default, default,
          default, default, default, default, default, default, default,
          default, default, default, default, default, default, default,
          default, default, default, default, default, default, default,
          default, default, default, default, default, default, default,
          default, default, default, default, default, default, default,
          default, default, default, default, default, default, default,
          default, default, default, default, default, default, default)
      WHERE [Current LSN] collate SQL_Latin1_General_CP1_CI_AS > @start_lsn_cmp
 collate SQL_Latin1_General_CP1_CI_AS -- This aims for implementing FN_DBLOG
 based on GT comparator.
      AND
      (
        (  [operation] in ('LOP_BEGIN_XACT','LOP_COMMIT_XACT','LOP_ABORT_XACT') )
        OR
        (  [operation] in ('LOP_INSERT_ROWS','LOP_DELETE_ROWS','LOP_MODIFY_ROW')
          AND
          ( ( [context]   in ('LCX_HEAP','LCX_CLUSTERED','LCX_MARK_AS_GHOST') )
 or ([context] = 'LCX_TEXT_MIX') )
          AND ([PartitionID] is not null) and ([PartitionID] >= @minPartition and
 [PartitionID]<=@maxPartition)
        )
        OR
        ([operation] = 'LOP_HOBT_DDL')
      )
      SET NOCOUNT OFF -- Re-enable "rows affected display"
END
GO
```

4. Create a certificate on the Master Database:

```
USE [master]
GO
CREATE CERTIFICATE [awsdms_rtm_dump_dblog_cert]
  ENCRYPTION BY PASSWORD = N'@hardpassword1'
```

```
  WITH SUBJECT = N'Certificate for FN_DUMP_DBLOG Permissions'
```

5. Create a login from the certificate:

```
USE [master]
GO
CREATE LOGIN awsdms_rtm_dump_dblog_login FROM CERTIFICATE
  [awsdms_rtm_dump_dblog_cert];
```

6. Add the login to the sysadmin server role:

```
ALTER SERVER ROLE [sysadmin] ADD MEMBER [awsdms_rtm_dump_dblog_login];
```

7. Add the signature to the [master].[awsdms].[rtm_dump_dblog] procedure using the certificate:

```
USE [master]
GO

ADD SIGNATURE
  TO [master].[awsdms].[rtm_dump_dblog]
  BY CERTIFICATE [awsdms_rtm_dump_dblog_cert]
  WITH PASSWORD = '@hardpassword1';
```

> **Note**
>
> If you recreate the stored procedure, you need to add the signature again.

8. Create the `[awsdms].[rtm_position_1st_timestamp]` procedure on the Master database:

```
USE [master]
IF object_id('[awsdms].[rtm_position_1st_timestamp]','P') is not null
  DROP PROCEDURE [awsdms].[rtm_position_1st_timestamp];
GO
CREATE PROCEDURE [awsdms].[rtm_position_1st_timestamp]
(
  @dbname                sysname,      -- Database name
  @seqno                 integer,      -- Backup set sequence/position number
 within file
  @filename              varchar(260), -- The backup filename
```

```
  @1stTimeStamp          varchar(40)   -- The timestamp to position by
)
AS
BEGIN
  SET NOCOUNT ON       -- Disable "rows affected display"

  DECLARE @firstMatching table
  (
    cLsn varchar(32),
    bTim datetime
  )
  DECLARE @sql nvarchar(4000)
  DECLARE @nl                       char(2)
  DECLARE @tb                       char(2)
  DECLARE @fnameVar                 sysname = 'NULL'

  SET @nl  = char(10); -- New line
  SET @tb  = char(9)   -- Tab separator

  IF (@filename is not null)
    SET @fnameVar = ''''+@filename +''''
  SET @filename = ''''+@filename +''''
  SET @sql='use ['+@dbname+'];'+@nl+
    'SELECT TOP 1 [Current LSN],[Begin Time]'+@nl+
    'FROM fn_dump_dblog (NULL, NULL, NULL, '+ cast(@seqno as varchar(10))+','+
 @filename +','+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default,'+@nl+
    @tb+'default, default, default, default, default, default, default)'+@nl+
    'WHERE operation=''LOP_BEGIN_XACT''' +@nl+
    'AND [Begin Time]>= cast('+''''+@1stTimeStamp+''''+' as datetime)'+@nl

    --print @sql
    DELETE FROM @firstMatching
    INSERT INTO @firstMatching  exec sp_executesql @sql    -- Get them all
    SELECT TOP 1 cLsn as [matching LSN],convert(varchar,bTim,121) AS[matching
 Timestamp] FROM @firstMatching;
```

```
    SET NOCOUNT OFF      -- Re-enable "rows affected display"

END
GO
```

9. Create a certificate on the Master database:

```
USE [master]
GO
CREATE CERTIFICATE [awsdms_rtm_position_1st_timestamp_cert]
  ENCRYPTION BY PASSWORD = N'@hardpassword1'
  WITH SUBJECT = N'Certificate for FN_POSITION_1st_TIMESTAMP Permissions';
```

10. Create a login from the certificate:

```
USE [master]
GO
CREATE LOGIN awsdms_rtm_position_1st_timestamp_login FROM CERTIFICATE
  [awsdms_rtm_position_1st_timestamp_cert];
```

11. Add the login to the sysadmin server role:

```
ALTER SERVER ROLE [sysadmin] ADD MEMBER
 [awsdms_rtm_position_1st_timestamp_login];
```

12. Add the signature to the `[master].[awsdms].[rtm_position_1st_timestamp]` procedure using the certificate:

```
USE [master]
GO
ADD SIGNATURE
  TO [master].[awsdms].[rtm_position_1st_timestamp]
  BY CERTIFICATE [awsdms_rtm_position_1st_timestamp_cert]
  WITH PASSWORD = '@hardpassword1';
```

Note

If you recreate the stored procedure, you need to add the signature again.

13. Create a user with the following permissions/roles in each of the following databases:

Note

You should create the dmsnosysadmin user account with the same SID on each replica. The following SQL query can help verify the dmsnosysadmin account SID value on each replica. For more information about creating a user, see [CREATE USER (Transact-SQL)] in the [Microsoft SQL server documentation]. For more information about creating SQL user accounts for Azure SQL database, see [Active geo-replication].

```
SELECT @@servername servername, name, sid, create_date, modify_date
  FROM sys.server_principals
  WHERE name = 'dmsnosysadmin';
```

14. Grant permissions on the master database on each replica:

```
USE master
GO

GRANT select on sys.fn_dblog to dmsnosysadmin;
GRANT view any definition to dmsnosysadmin;
GRANT view server state to dmsnosysadmin -- (should be granted to the login).
GRANT execute on sp_repldone to dmsnosysadmin;
GRANT execute on sp_replincrementlsn to dmsnosysadmin;
GRANT execute on sp_addpublication to dmsnosysadmin;
GRANT execute on sp_addarticle to dmsnosysadmin;
GRANT execute on sp_articlefilter to dmsnosysadmin;
GRANT select on [awsdms].[split_partition_list] to dmsnosysadmin;
GRANT execute on [awsdms].[rtm_dump_dblog] to dmsnosysadmin;
GRANT execute on [awsdms].[rtm_position_1st_timestamp] to dmsnosysadmin;
```

15. Grant permissions on the msdb database on each replica:

```
USE msdb
GO
GRANT select on msdb.dbo.backupset TO self_managed_user
GRANT select on msdb.dbo.backupmediafamily TO self_managed_user
GRANT select on msdb.dbo.backupfile TO self_managed_user
```

16. Add the `db_owner` role to `dmsnosysadmin` on the source database. Because the database is synchronized, you can add the role on the primary replica only.

```
use <source DB>
GO
EXEC sp_addrolemember N'db_owner', N'dmsnosysadmin'
```

Setting up ongoing replication on a cloud SQL Server DB instance

This section describes how to set up CDC on a cloud-hosted SQL Server database instance. A cloud-hosted SQL server instance is an instance running on Amazon RDS for SQL Server, an Azure SQL Manged Instance, or any other managed cloud SQL Server instance. For information about limitations for ongoing replication for each database type, see Limitations on using SQL Server as a source for AWS DMS.

Before setting up ongoing replication, see Prerequisites for using ongoing replication (CDC) from a SQL Server source.

Unlike self-managed Microsoft 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 ongoing changes in a source SQL Server instance. Make sure to turn on MS-CDC for the Amazon RDS instance (using master user privileges) as in the following procedure.

To turn on MS-CDC for a cloud SQL Server DB instance

1. Run one of the following queries at the database level.

   For an RDS for SQL Server DB instance, use this query.

   ```
   exec msdb.dbo.rds_cdc_enable_db 'DB_name'
   ```

   For an Azure SQL managed DB instance, use this query.

   ```
   USE DB_name
   GO
   EXEC sys.sp_cdc_enable_db
   GO
   ```

2. For each table with a primary key, run the following query to turn on 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 turn on 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 turn on 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 RDS for SQL Server instances that are replicating using DMS version 3.5.3 and above, make sure that the retention period is set to the default value of 5 seconds. If you're upgrading or moving from DMS 3.5.2 and below to DMS 3.5.3 and above, change the polling interval value after the tasks are running on the new or upgraded instance. The following script sets the retention period to 5 seconds:

```
use dbname
EXEC sys.sp_cdc_change_job @job_type = 'capture' ,@pollinginterval = 5
exec sp_cdc_stop_job 'capture'
```

```
exec sp_cdc_start_job 'capture'
```

- The parameter `@pollinginterval` is measured in seconds with a recommended value set to 86399. This means that the transaction log retains changes for 86,399 seconds (one day) when `@pollinginterval = 86399`. The procedure `exec sp_cdc_start_job 'capture'` initiates the settings.

> **Note**
>
> With some versions of SQL Server, if the value of `pollinginterval` is set to more than 3599 seconds, the value resets to the default five seconds. When this happens, T-Log entries are purged before AWS DMS can read them. To determine which SQL Server versions are affected by this known issue, see this Microsoft KB article.

If you are using Amazon RDS with Multi-AZ, make sure that you also set your secondary to have the right values in case of failover.

```
exec rdsadmin..rds_set_configuration 'cdc_capture_pollinginterval' , <5 or
 preferred value>
```

To maintain the retention period when an AWS DMS replication task is stopped for more than one hour

> **Note**
>
> The following steps aren't needed for a RDS for SQL Server source replicating using DMS 3.5.3 and above.

1. Stop the job truncating the transaction logs by using the following command.

```
exec sp_cdc_stop_job 'capture'
```

2. Find your task on the AWS DMS console and resume the task.
3. Choose the **Monitoring** tab, and check the `CDCLatencySource` metric.

4. After the `CDCLatencySource` metric equals 0 (zero) and stays there, restart the job truncating the transaction logs using the following command.

```
exec sp_cdc_start_job 'capture'
```

Remember to start the job that truncates SQL Server transaction logs. Otherwise, storage on your SQL Server instance might fill up.

Recommended settings when using RDS for SQL Server as a source for AWS DMS

For AWS DMS 3.5.3 and above

> **Note**
>
> The initial release of the RDS for SQL Server log backup feature is enabled by default for endpoints that you created or modified after the release of DMS version 3.5.3. To use this feature for existing endpoints, modify the endpoint without making any changes.

AWS DMS version 3.5.3 introduces support for reading from log backups. DMS primarily relies on reading from the active transaction logs to replicate events. If a transaction is backed up before DMS can read it from the active log, the task accesses the RDS backups on-demand and reads from the subsequent backup logs until it catches up to the active transaction log. To ensure that DMS has access to log backups, set the RDS automated backup retention period to at least one day. For information about setting the automated backup retention period, see Backup retention period in the *Amazon RDS User Guide*.

A DMS task accessing log backups utilizes storage on the RDS instance. Note that the task only accesses the log backups needed for replication. Amazon RDS removes these downloaded backups in a couple of hours. This removal doesn't affect the Amazon RDS backups retained in Amazon S3, or Amazon RDS `RESTORE DATABASE` functionality. It is advisable to allocate additional storage on your RDS for SQL Server source if you intend to replicate using DMS. One way to estimate the amount of storage needed is to identify the backup from which DMS will start or resume replicating from, and add up the file sizes of all the subsequent backups using the RDS `tlog backup` metadata function. For more information about the `tlog backup` function, see Listing available transaction log backups in the *Amazon RDS User Guide*.

Alternately, you may choose to enable storage autoscaling and/ or trigger storage scaling based on the CloudWatch `FreeStorageSpace` metric for your Amazon RDS instance.

We strongly recommend that you don't start or resume from a point too far back in the transaction log backups, as it might lead to storage on your SQL Server instance filling up. In such cases, it is advisable to start a full load. Replicating from the transaction log backup is slower than reading from the active transaction logs. For more information, see Transaction log backup processing for RDS for SQL Server.

Note that accessing the log backups requires additional privileges. For more information, see as detailed in Set up permissions for ongoing replication from a cloud SQL Server database Make sure that you grant these privileges before the task starts replicating.

For AWS DMS 3.5.2 and below

When you work with Amazon RDS for SQL Server as a source, the MS-CDC capture job relies on the parameters `maxscans` and `maxtrans`. These parameters govern the maximum number of scans that the MS-CDC capture does on the transaction log and the number of transactions that are processed for each scan.

For databases, where a number of transactions is greater than `maxtrans*maxscans`, increasing the `polling_interval` value can cause an accumulation of active transaction log records. In turn, this accumulation can lead to an increase in the size of the transaction log.

Note that AWS DMS does not rely on the MS-CDC capture job. The MS-CDC capture job marks the transaction log entries as having been processed. This allows the transaction log backup job to remove the entries from the transaction log.

We recommend that you monitor the size of the transaction log and the success of the MS-CDC jobs. If the MS-CDC jobs fail, the transaction log could grow excessively and cause AWS DMS replication failures. You can monitor MS-CDC capture job errors using the `sys.dm_cdc_errors` dynamic management view in the source database. You can monitor the transaction log size using the `DBCC SQLPERF(LOGSPACE)` management command.

To address the transaction log increase that is caused by MS-CDC

1. Check the `Log Space Used %` for the database AWS DMS is replicating from and validate that it increases continuously.

   ```
   DBCC SQLPERF(LOGSPACE)
   ```

2. Identify what is blocking the transaction log backup process.

   ```
   Select log_reuse_wait, log_reuse_wait_desc, name from sys.databases where name =
    db_name();
   ```

   If the `log_reuse_wait_desc` value equals `REPLICATION`, the log backup retention is caused by the latency in MS-CDC.

3. Increase the number of events processed by the capture job by increasing the `maxtrans` and `maxscans` parameter values.

   ```
   EXEC sys.sp_cdc_change_job @job_type = 'capture' ,@maxtrans = 5000, @maxscans = 20
   exec sp_cdc_stop_job 'capture'
   exec sp_cdc_start_job 'capture'
   ```

To address this issue, set the values of `maxscans` and `maxtrans` so that `maxtrans*maxscans` is equal to the average number of events generated for tables that AWS DMS replicates from the source database for each day.

If you set these parameters higher than the recommended value, the capture jobs process all events in the transaction logs. If you set these parameters below the recommended value, MS-CDC latency increases and your transaction log grows.

Identifying appropriate values for `maxscans` and `maxtrans` can be difficult because changes in workload produce varying number of events. In this case, we recommend that you set up monitoring on MS-CDC latency. For more information, see [Monitor the process](#) in SQL Server documentation. Then configure `maxtrans` and `maxscans` dynamically based on the monitoring results.

If the AWS DMS task is unable to find the log sequence numbers (LSNs) needed to resume or continue the task, the task may fail and require a complete reload.

> **Note**
>
> When using AWS DMS to replicate data from an RDS for SQL Server source, you may encounter errors when trying to resume replication after a stop-start event of the Amazon RDS instance. This is due to the SQL Server Agent process restarting the capture job process when it restarts after the stop-start event. This bypasses the MS-CDC polling interval.

Because of this, on databases with transaction volumes lower than the MS-CDC capture job processing, this can cause data to be processed or marked as replicated and backed up before AWS DMS can resume from where it stopped, resulting in the following error:

```
[SOURCE_CAPTURE ]E: Failed to access LSN '0000dbd9:0006f9ad:0003' in
 the backup log sets since BACKUP/LOG-s are not available. [1020465]
 (sqlserver_endpoint_capture.c:764)
```

To mitigate this issue, set the `maxtrans` and `maxscans` values as recommended prior.

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.

Note

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

Using Microsoft Azure SQL Managed Instance as a source for AWS DMS

With AWS DMS, you can use Microsoft Azure SQL Managed Instance 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.

Using Microsoft Azure Database for PostgreSQL flexible server as a source for AWS DMS

With AWS DMS, you can use Microsoft Azure Database for PostgreSQL flexible server as a source in much the same way as you do PostgreSQL.

For information about versions of Microsoft Azure Database for PostgreSQL flexible server that AWS DMS supports as a source, see Sources for AWS DMS.

Setting up Microsoft Azure for PostgreSQL flexible server for logical replication and decoding

You can use logical replication and decoding features in Microsoft Azure Database for PostgreSQL flexible server during database migration.

For logical decoding, DMS uses either the `test_decoding` or `pglogical` plugin. If the `pglogical` plugin is available on a source PostgreSQL database, DMS creates a replication slot using `pglogical`, otherwise the `test_decoding` plugin is used.

To configure your Microsoft Azure for PostgreSQL flexible server as a source endpoint for DMS, perform the following steps:

1. Open the Server Parameters page on the portal.
2. Set the `wal_level` server parameter to `LOGICAL`.
3. If you want to use the `pglogical` extension, set the `shared_preload_libraries` and `azure.extensions` parameters to `pglogical`.
4. Set the `max_replication_slots` parameter to the maximum number of DMS tasks that you plan to run concurrently. In Microsoft Azure, the default value for this parameter is 10. This parameter's maximum value depends on the available memory of your PostgreSQL instance, allowing for between 2 and 8 replication slots per GB of memory.
5. Set the `max_wal_senders` parameter to a value greater than 1. The `max_wal_senders` parameter sets the number of concurrent tasks that can run. The default value is 10.
6. Set the `max_worker_processes` parameter value to at least 16. Otherwise, you may see errors such as the following:

   ```
   WARNING: out of background worker slots.
   ```
7. Save the changes. Restart the server to apply the changes.
8. Confirm that your PostgreSQL instance allows network traffic from your connecting resource.
9. Grant an existing user replication permissions, or create a new user with replication permissions, using the following commands.

   - Grant an existing user replication permissions using the following command:

     ```
     ALTER USER <existing_user> WITH REPLICATION;
     ```
   - Create a new user with replication permissions using the following command:

```
CREATE USER aws_dms_user PASSWORD 'aws_dms_user_password';
GRANT azure_pg_admin to aws_dms_user;
ALTER ROLE aws_dms_user REPLICATION LOGIN;
```

For more information about logical replication with PostgreSQL, see the following topics:

- Enabling change data capture (CDC) using logical replication
- Using native CDC start points to set up a CDC load of a PostgreSQL source
- Logical replication and logical decoding in Azure Database for PostgreSQL - Flexible Server in the Azure Database for PostgreSQL documentation.

Using Microsoft Azure Database for MySQL flexible server as a source for AWS DMS

With AWS DMS, you can use Microsoft Azure Database for MySQL flexible server as a source in much the same way as you do MySQL.

For information about versions of Microsoft Azure Database for MySQL flexible server that AWS DMS supports as a source, see Sources for AWS DMS.

For more information about using a customer-managed MySQL-compatible database with AWS DMS, see Using a self-managed MySQL-compatible database as a source for AWS DMS.

Limitations when using Azure MySQL as a source for AWS Database Migration Service

- The defaut value for the Azure MySQL flexible server system variable `sql_generate_invisible_primary_key` is `ON`, and the server automatically adds a generated invisible primary key (GIPK) to any table that is created without an explicit primary key. AWS DMS doesn't support ongoing replication for MySQL tables with GIPK constraints.

Using OCI MySQL Heatwave as a source for AWS DMS

With AWS DMS, you can use OCI MySQL Heatwave as a source in much the same way as you do MySQL. Using OCI MySQL Heatwave as a source requires a few additional configuration changes.

For information about versions of OCI MySQL Heatwave that AWS DMS supports as a source, see Sources for AWS DMS.

Setting up OCI MySQL Heatwave for logical replication

To configure your OCI MySQL Heatwave instance as a source endpoint for DMS, do the following:

1. Sign in to OCI Console, and open the main hamburger menu (≡) in the top left corner.
2. Choose **Databases**, **DB Systems**.
3. Open the **Configurations** menu.
4. Choose **Create configuration**.
5. Enter a configuration name, such as **`dms_configuration`**.
6. Choose the shape of your current OCI MySQL Heatwave instance. You can find the shape on the instance's **DB system configuration** properties tab under the **DB system configuration:Shape** section.
7. In the **User variables** section, choose the `binlog_row_value_options` system variable. Its default value is `PARTIAL_JSON`. Clear the value.
8. Choose the **Create** button.
9. Open your OCI MySQLHeatwave instance, and choose the **Edit** button.
10. In the **Configuration** section, choose the **Change configuration** button, and choose the shape configuration that you created in step 4.
11. Once the changes take effect, your instance is ready for logical replication.

Using Google Cloud for MySQL as a source for AWS DMS

With AWS DMS, you can use Google Cloud for MySQL as a source in much the same way as you do MySQL.

For information about versions of GCP MySQL that AWS DMS supports as a source, see Sources for AWS DMS.

For more information, see Using a MySQL-compatible database as a source for AWS DMS.

> **Note**
>
> Support for GCP MySQL 8.0 as a source is available in AWS DMS version 3.4.6.

AWS DMS doesn't support the SSL mode `verify-full` for GCP for MySQL instances.
The GCP MySQL security setting `Allow only SSL connections` isn't supported, because it requires both server and client certificate verification. AWS DMS only supports server certificate verification.
AWS DMS supports the default GCP CloudSQL for MySQL value of `CRC32` for the `binlog_checksum` database flag.

Using Google Cloud for PostgreSQL as a source for AWS DMS

With AWS DMS, you can use Google Cloud for PostgreSQL as a source in much the same way as you do self-managed PostgreSQL databases.

For information about versions of GCP PostgreSQL that AWS DMS supports as a source, see Sources for AWS DMS.

For more information, see Using a PostgreSQL database as an AWS DMS source.

Set up Google Cloud for PostgreSQL for logical replication and decoding

You can use logical replication and decoding features in Google Cloud SQL for PostgreSQL during database migration.

For logical decoding, DMS uses one of the following plugins:

- `test_decoding`
- `pglogical`

If the `pglogical` plugin is available on a source PostgreSQL database, DMS creates a replication slot using `pglogical`, otherwise the `test_decoding` plugin is used.

Note the following about using logical decoding with AWS DMS:

1. With Google Cloud SQL for PostgreSQL, enable logical decoding by setting the `cloudsql.logical_decoding` flag to on.
2. To enable `pglogical`, set the `cloudsql.enable_pglogical` flag to on, and restart the database.
3. To use logical decoding features, you create a PostgreSQL user with the `REPLICATION` attribute. When you are using the `pglogical` extension, the user must have the

`cloudsqlsuperuser` role. To create a user with the `cloudsqlsuperuser` role, do the following:

```
CREATE USER new_aws_dms_user WITH REPLICATION
IN ROLE cloudsqlsuperuser LOGIN PASSWORD 'new_aws_dms_user_password';
```

To set this attribue on an existing user, do the following:

```
ALTER USER existing_user WITH REPLICATION;
```

4. Set the `max_replication_slots` parameter to the maximum number of DMS tasks that you plan to run concurrently. In Google Cloud SQL, the default value for this parameter is 10. This parameter's maximum value depends on the available memory of your PostgreSQL instance, allowing for between 2 and 8 replication slots per GB of memory.

For more information about logical replication with PostgreSQL, see the following topics:

- Enabling change data capture (CDC) using logical replication
- Using native CDC start points to set up a CDC load of a PostgreSQL source
- Set up logical replication and decoding in the Cloud SQL for PostgreSQL documentation.

Using a PostgreSQL database as an AWS DMS source

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.

For information about versions of PostgreSQL that AWS DMS supports as a source, see Sources for AWS DMS.

AWS DMS supports PostgreSQL for these types of databases:

- On-premises databases
- Databases on an Amazon EC2 instance
- Databases on an Amazon RDS DB instance
- Databases on an DB instance based on Amazon Aurora PostgreSQL-Compatible Edition

- Databases on an DB instance based on Amazon Aurora PostgreSQL-Compatible Serverless Edition

> **Note**
>
> DMS supports Amazon Aurora PostgreSQL—Serverless V1 as a source for Full load only. But you can use Amazon Aurora PostgreSQL—Serverless V2 as a source for Full load, Full load + CDC, and CDC only tasks.

You can use Secure Socket Layers (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.

As an additional security requirement when using PostgreSQL as a source, the user account specified must be a registered user in the PostgreSQL database.

To configure a PostgreSQL database as an AWS DMS source endpoint, do the following:

- Create a PostgreSQL user with appropriate permissions to provide AWS DMS access to your PostgreSQL source database.

  > **Note**
  >
  > - If your PostgreSQL source database is self-managed, see Working with self-managed PostgreSQL databases as a source in AWS DMS for more information.
  > - If your PostgreSQL source database is managed by Amazon RDS, see Working with AWS-managed PostgreSQL databases as a DMS source for more information.

- Create a PostgreSQL source endpoint that conforms with your chosen PostgreSQL database configuration.
- Create a task or set of tasks to migrate your tables.

  To create a full-load-only task, no further endpoint configuration is needed.

  Before you create a task for change data capture (a CDC-only or full-load and CDC task), see Enabling CDC using a self-managed PostgreSQL database as a AWS DMS source or Enabling CDC with an AWS-managed PostgreSQL DB instance with AWS DMS.

Topics

Working with self-managed PostgreSQL databases as a source in AWS DMS

With a self-managed PostgreSQL database as a source, you can migrate data to either another PostgreSQL database, or one of the other target databases supported by AWS DMS. The database source can be an on-premises database or a self-managed engine running on an Amazon EC2 instance. You can use a DB instance for both full-load tasks and change data capture (CDC) tasks.

Prerequisites to using a self-managed PostgreSQL database as an AWS DMS source

Before migrating data from a self-managed PostgreSQL source database, do the following:

- Make sure that you use a PostgreSQL database that is version 9.4.x or higher.
- For full-load plus CDC tasks or CDC-only tasks, grant superuser permissions for the user account specified for the PostgreSQL source database. The user account needs superuser permissions to access replication-specific functions in the source. DMS user account needs SELECT permissions on all columns to migrate tables successfully. In the case of missing permissions on columns, DMS creates target table using regular DMS data type mappings which leads to metadata differences and task failures.

- Add the IP address of the AWS DMS replication server to the pg_hba.conf configuration file and enable replication and socket connections. An example follows.

```
# 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.

- If you're configuring a database as a source for logical replication using AWS DMS see [Enabling CDC using a self-managed PostgreSQL database as a AWS DMS source](#)

> **Note**
>
> Some AWS DMS transactions are idle for some time before the DMS engine uses them again. By using the parameter idle_in_transaction_session_timeout in PostgreSQL versions 9.6 and higher, you can cause idle transactions to time out and fail. Don't end idle transactions when you use AWS DMS.

Enabling CDC using a self-managed PostgreSQL database as a AWS DMS source

AWS DMS supports change data capture (CDC) using logical replication. To enable logical replication of a self-managed PostgreSQL source database, 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.

  Set the max_replication_slots value according to the number of tasks that you want to run. For example, to run five tasks you 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. Make sure to manually delete open slots. Note that DMS automatically drops replication slots when the task is deleted, if DMS created the slot.

- Set `max_wal_senders` to a value greater than 1.

  The `max_wal_senders` parameter sets the number of concurrent tasks that can run.
- The `wal_sender_timeout` parameter ends replication connections that are inactive longer than the specified number of milliseconds. The default for an on-premises PostgreSQL database is 60000 milliseconds (60 seconds). Setting the value to 0 (zero) disables the timeout mechanism, and is a valid setting for DMS.

  When setting `wal_sender_timeout` to a non-zero value, a DMS task with CDC requires a minimum of 10000 milliseconds (10 seconds), and fails if the value is less than 10000. Keep the value less than 5 minutes to avoid causing a delay during a Multi-AZ failover of a DMS replication instance.

Some parameters are static, and you can only set them at server start. Any changes to their entries in the configuration file (for a self-managed database) or DB parameter group (for an RDS for PostgreSQL database) are ignored until the server is restarted. For more information, see the PostgreSQL documentation.

For more information about enabling CDC, see Enabling change data capture (CDC) using logical replication.

Working with AWS-managed PostgreSQL databases as a DMS source

You can use an AWS-managed PostgreSQL DB instance as a source for AWS DMS. You can perform both full-load tasks and change data capture (CDC) tasks using an AWS-managed PostgreSQL source.

Prerequisites for using an AWS-managed PostgreSQL database as a DMS source

Before migrating data from an AWS-managed PostgreSQL source database, do the following:

- We recommend that you use an AWS user account with the minimum required permissions for the PostgreSQL DB instance as the user account for the PostgreSQL source endpoint for AWS DMS. Using the master account is not recommended. 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.

Make sure to create several objects from the master user account for the account that you use. For information about creating these, see [Migrating an Amazon RDS for PostgreSQL database without using the master user account]().

- If your source database is in a virtual private cloud (VPC), choose the VPC security group that provides access to the DB instance where the database resides. This is needed for the DMS replication instance to connect successfully to the source DB instance. When the database and DMS replication instance are in same VPC, add the appropriate security group to its own inbound rules.

> **Note**
>
> Some AWS DMS transactions are idle for some time before the DMS engine uses them again. By using the parameter `idle_in_transaction_session_timeout` in PostgreSQL versions 9.6 and higher, you can cause idle transactions to time out and fail. Don't end idle transactions when you use AWS DMS.

Enabling CDC with an AWS-managed PostgreSQL DB instance with AWS DMS

AWS DMS supports CDC on Amazon RDS PostgreSQL databases when the DB instance is configured to use logical replication. The following table summarizes the logical replication compatibility of each AWS-managed PostgreSQL version.

PostgreSQL version	AWS DMS full load support	AWS DMS CDC support
Aurora PostgreSQL version 2.1 with PostgreSQL 10.5 compatibility (or lower)	Yes	No
Aurora PostgreSQL version 2.2 with PostgreSQL 10.6 compatibility (or higher)	Yes	Yes
RDS for PostgreSQL with PostgreSQL 10.21 compatibi lity (or higher)	Yes	Yes

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, make sure to 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.
2. Set the `rds.logical_replication` parameter in your DB CLUSTER 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. The `wal_sender_timeout` parameter ends replication connections that are inactive longer than the specified number of milliseconds. The default for an AWS-managed PostgreSQL database is 30000 milliseconds (30 seconds). Setting the value to 0 (zero) disables the timeout mechanism, and is a valid setting for DMS.

   When setting `wal_sender_timeout` to a non-zero value, a DMS task with CDC requires a minimum of 10000 milliseconds (10 seconds), and fails if the value is between 0 and 10000. Keep the value less than 5 minutes to avoid causing a delay during a Multi-AZ failover of a DMS replication instance.
4. Ensure the value of the `max_worker_processes` parameter in your DB Cluster Parameter Group is equal to, or higher than the total combined values of `max_logical_replication_workers`, `autovacuum_max_workers`, and `max_parallel_workers`. A high number of background worker processes might impact application workloads on small instances. So, monitor performance of your database if you set `max_worker_processes` higher than the default value.
5. When using Aurora PostgreSQL as a source with CDC, set `synchronous_commit` to `ON`.

To use PostgreSQL MultiAZ DB Cluster Read Replica for CDC (ongoing replication)

1. Set the `rds.logical_replication` and `sync_replication_slots` parameters in your DB CLUSTER parameter group to 1. This static parameters require a reboot of the DB instance to take effect.

2. Run the following command to create the `awsdms_ddl_audit` table on Writer and to replace the `objects_schema` 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
);
```

3. Run the following command to create the `awsdms_intercept_ddl` function and to replace the `objects_schema` 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 $$
  declare _qry text;
BEGIN
  if (tg_tag='CREATE TABLE' or tg_tag='ALTER TABLE' or tg_tag='DROP TABLE' or
 tg_tag = 'CREATE TABLE AS') then
         SELECT current_query() into _qry;
         insert into objects_schema.awsdms_ddl_audit
         values
         (
         default,current_timestamp,current_user,cast(TXID_CURRENT()as
 varchar(16)),tg_tag,0,'',current_schema,_qry
         );
         delete from objects_schema.awsdms_ddl_audit;
end if;
END;
$$;
```

4. Run the following command to create the `awsdms_intercept_ddl` event trigger:

```
CREATE EVENT TRIGGER awsdms_intercept_ddl ON ddl_command_end EXECUTE PROCEDURE
 objects_schema.awsdms_intercept_ddl();
```

Ensure that all the users and roles that access these events have the necessary DDL permissions. For example:

```
grant all on public.awsdms_ddl_audit to public;
grant all on public.awsdms_ddl_audit_c_key_seq to public;
```

5. Create replication slot on Writer:

```
SELECT * FROM pg_create_logical_replication_slot('dms_read_replica_slot',
 'test_decoding', false, true);
```

6. Ensure the replication slot is available on Reader:

```
select * from pg_catalog.pg_replication_slots where slot_name =
 'dms_read_replica_slot';

slot_name            |plugin       |slot_type|datoid|database|temporary|
active|active_pid|xmin|catalog_xmin|restart_lsn|confirmed_flush_lsn|
wal_status|safe_wal_size|two_phase|inactive_since               |conflicting|
invalidation_reason|failover|synced|
---------------------+-------------+---------+------+--------+---------
+------+----------+----+------------+-----------+-------------------
+----------+-------------+---------+-----------------------------+-----------
+-------------------+--------+------+
dms_read_replica_slot|test_decoding|logical  |     5|postgres|false    |false |
        |    |3559        |0/180011B8 |0/180011F0         |reserved  |
 |true     |2025-02-10 15:45:04.083 +0100|false      |                   |false   |
false |
```

7. Create DMS source endpoint for Read Replica and set logical replication slot name via the Extra Connection Attribute:

```
slotName=dms_read_replica_slot;
```

8. Create and start the CDC/FL+CDC task.

> **Note**
>
> For CDC/FL+CDC migrations DMS considers task start time as a CDC start position. All older LSNs from replication slot are ignored.

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 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` endpoint setting to `false` 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.

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 *`OtherThanMaster`* account.
2. Log in to the PostgreSQL DB instance using the user account other than the master account, here the *`OtherThanMaster`* account.
3. Create the table `awsdms_ddl_audit` by running the following command, replacing *`objects_schema`* in the following code 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 $$
  declare _qry text;
BEGIN
  if (tg_tag='CREATE TABLE' or tg_tag='ALTER TABLE' or tg_tag='DROP TABLE' or
 tg_tag = 'CREATE TABLE AS') then
         SELECT current_query() into _qry;
         insert into objects_schema.awsdms_ddl_audit
         values
         (
         default,current_timestamp,current_user,cast(TXID_CURRENT()as
 varchar(16)),tg_tag,0,'',current_schema,_qry
         );
         delete from objects_schema.awsdms_ddl_audit;
end if;
END;
$$;
```

5. Log out of the *`OtherThanMaster`* 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();
```

7. Make sure that all users and roles that access these events have the necessary DDL permissions. For example:

```
grant all on public.awsdms_ddl_audit to public;
grant all on public.awsdms_ddl_audit_c_key_seq to public;
```

When you have completed the procedure preceding, you can create the AWS DMS source endpoint using the *OtherThanMaster* account.

Note

These events are triggered by `CREATE TABLE`, `ALTER TABLE`, and `DROP TABLE` statements.

Enabling change data capture (CDC) using logical replication

You can use PostgreSQL's native logical replication feature to enable change data capture (CDC) during database migration for PostgreSQL sources. You can use this feature with a self-managed PostgreSQL and also an Amazon RDS for PostgreSQL SQL DB instance. This approach reduces downtime and help ensure that the target database is in sync with the source PostgreSQL database.

AWS DMS supports CDC for PostgreSQL tables with primary keys. If a table does not have a primary key, the write-ahead logs (WAL) don't include a before image of the database row. In this case, DMS can't update the table. Here, you can use additional configuration settings and use table replica identity as a workaround. However, this approach can generate extra logs. We recommend that you use table replica identity as a workaround only after careful testing. For more information, see Additional configuration settings when using a PostgreSQL database as a DMS source.

Note

REPLICA IDENTITY FULL is supported with a logical decoding plugin, but isn't supported with a pglogical plugin. For more information, see pglogical documentation.

For full load and CDC and CDC only tasks, AWS DMS uses logical replication slots to retain WAL logs for replication until the logs are decoded. On restart (not resume) for a full load and CDC task or a CDC task, the replication slot gets recreated.

Note

For logical decoding, DMS uses either test_decoding or pglogical plugin. If the pglogical plugin is available on a source PostgreSQL database, DMS creates a replication slot using pglogical, otherwise a test_decoding plugin is used. For more information about the test_decoding plugin, see PostgreSQL Documentation.
If the database parameter `max_slot_wal_keep_size` is set to a non default value, and the `restart_lsn` of a replication slot falls behind the current LSN by more than this size, the DMS task fails due to removal of required WAL files.

Configuring the pglogical plugin

Implemented as a PostgreSQL extension, the pglogical plugin is a logical replication system and model for selective data replication. The following table identifies source PostgreSQL database versions that support the pglogical plugin.

PostgreSQL source	Supports pglogical
Self-managed PostgreSQL 9.4 or higher	Yes
Amazon RDS PostgreSQL 9.5 or lower	No
Amazon RDS PostgreSQL 9.6 or higher	Yes
Aurora PostgreSQL 1.x till 2.5.x	No
Aurora PostgreSQL 2.6.x or higher	Yes
Aurora PostgreSQL 3.3.x or higher	Yes

Before configuring pglogical for use with AWS DMS, first enable logical replication for change data capture (CDC) on your PostgreSQL source database.

- For information about enabling logical replication for CDC on *self-managed* PostgreSQL source databases, see [Enabling CDC using a self-managed PostgreSQL database as a AWS DMS source]
- For information about enabling logical replication for CDC on *AWS-managed* PostgreSQL source databases, see [Enabling CDC with an AWS-managed PostgreSQL DB instance with AWS DMS].

After logical replication is enabled on your PostgreSQL source database, use the following steps to configure pglogical for use with DMS.

To use the pglogical plugin for logical replication on a PostgreSQL source database with AWS DMS

1. Create a pglogical extension on your source PostgreSQL database:

   a. Set the correct parameter:

      - For self-managed PostgreSQL databases, set the database parameter `shared_preload_libraries= 'pglogical'`.
      - For PostgreSQL on Amazon RDS and Amazon Aurora PostgreSQL-Compatible Edition databases, set the parameter `shared_preload_libraries` to `pglogical` in the same RDS parameter group.

   b. Restart your PostgreSQL source database.

   c. On the PostgreSQL database, run the command, `create extension pglogical;`

2. Run the following command to verify that pglogical installed successfully:

   ```
   select * FROM pg_catalog.pg_extension
   ```

You can now create a AWS DMS task that performs change data capture for your PostgreSQL source database endpoint.

> **Note**
>
> If you don't enable pglogical on your PostgreSQL source database, AWS DMS uses the `test_decoding` plugin by default. When pglogical is enabled for logical decoding, AWS DMS uses pglogical by default. But you can set the extra connection attribute, `PluginName` to use the `test_decoding` plugin instead.

Using native CDC start points to set up a CDC load of a PostgreSQL source

To enable native CDC start points with PostgreSQL as a source, set 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 you use logical replication slots. For more information on setting the `slotName` extra connection attribute, see Endpoint settings and Extra Connection Attributes (ECAs) when using PostgreSQL as a DMS source.

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 does not have any active connections. If it does, resolve and close 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 console, AWS CLI or AWS DMS API. For example, using the CLI you might run the following `create-replication-task` command.

```
aws dms create-replication-task --replication-task-identifier postgresql-slot-name-
test
--source-endpoint-arn arn:aws:dms:us-
west-2:012345678901:endpoint:ABCD1EFGHIJK2LMNOPQRST3UV4
```

```
--target-endpoint-arn arn:aws:dms:us-
west-2:012345678901:endpoint:ZYX9WVUTSRQONM8LKJIHGF7ED6
--replication-instance-arn arn:aws:dms:us-
west-2:012345678901:rep:AAAAAAAAAAA5BB4CCC3DDDD2EE
--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:

- The `source-endpoint-arn` option is set to the new value that you created in step 2.
- The `replication-instance-arn` option is set to the same value as for the parent task from step 1.
- The `table-mappings` and `replication-task-settings` options are set to the same values as for the parent task from step 1.
- The `cdc-start-position` option 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.

To enable custom CDC start mode when creating a new CDC-only task using the AWS DMS console, do the following:

- In the **Task settings** section, for **CDC start mode for source transactions**, choose **Enable custom CDC start mode**.
- For **Custom CDC start point for source transactions**, choose **Specify a log sequence number**. Specify the system change number or choose **Specify a recovery checkpoint**, and provide a Recovery checkpoint.

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

When using native CDC start points with the pglogical plugin and you want to use a new replication slot, complete the setup steps following before creating a CDC task.

To use a new replication slot not previously created as part of another DMS task

1. Create a replication slot, as shown following:

```
SELECT * FROM pg_create_logical_replication_slot('replication_slot_name',
 'pglogical');
```

2. After the database creates the replication slot, get and note the **restart_lsn** and **confirmed_flush_lsn** values for the slot:

```
select * from pg_replication_slots where slot_name like 'replication_slot_name';
```

   Note that the Native CDC Start position for a CDC task created after the replication slot can't be older than the **confirmed_flush_lsn** value.

   For information about the **restart_lsn** and **confirmed_flush_lsn** values, see pg_replication_slots

3. Create a pglogical node.

```
SELECT pglogical.create_node(node_name := 'node_name', dsn := 'your_dsn_name');
```

4. Create two replication sets using the `pglogical.create_replication_set` function. The first replication set tracks updates and deletes for tables that have primary keys. The second replication set tracks only inserts, and has the same name as the first replication set, with the added prefix 'i'.

```
SELECT pglogical.create_replication_set('replication_slot_name', false, true, true,
 false);
SELECT pglogical.create_replication_set('ireplication_slot_name', true, false,
 false, true);
```

5. Add a table to the replication set.

```
SELECT pglogical.replication_set_add_table('replication_slot_name',
 'schemaname.tablename', true);
SELECT pglogical.replication_set_add_table('ireplication_slot_name',
 'schemaname.tablename', true);
```

6. Set the extra connection attribute (ECA) following when you create your source endpoint.

```
PluginName=PGLOGICAL;slotName=slot_name;
```

You can now create a CDC only task with a PostgreSQL native start point using the new replication slot. For more information about the pglogical plugin, see the [pglogical 3.7 documentation]

Migrating from PostgreSQL to PostgreSQL using AWS DMS

When you migrate from a database engine other than PostgreSQL to a PostgreSQL database, AWS DMS is almost always the best migration tool to use. But when you are migrating from a PostgreSQL database to a PostgreSQL database, PostgreSQL tools can be more effective.

Using PostgreSQL native tools to migrate data

We recommend that you use 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. To restore the data to a database with a different name, use the `pg_restore` command and the `-d` parameter.

If you are migrating data from a PostgreSQL source database running on EC2 to an Amazon RDS for PostgreSQL target, you can use the pglogical plugin.

For more information about importing a PostgreSQL database into Amazon RDS for PostgreSQL or Amazon Aurora PostgreSQL-Compatible Edition, 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, for example, from 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.

> **Note**
>
> When replicating partitioned tables from a PostgreSQL source to PostgreSQL target, you don’t need to mention the parent table as part of the selection criteria in the DMS task. Mentioning the parent table causes data to be duplicated in child tables on the target, possibly causing a PK violation. By selecting child tables alone in the table mapping selection criteria, the parent table is automatically populated.

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.

When performing data type migration, be aware of the following:

- In some cases, the PostgreSQL NUMERIC(p,s) data type does not specify any precision and scale. For DMS versions 3.4.2 and earlier, DMS uses a precision of 28 and a scale of 6 by default, NUMERIC(28,6). For example, the value 0.611111104488373 from the source is converted to 0.611111 on the PostgreSQL target.
- A table with an ARRAY data type must have a primary key. A table with an ARRAY data type missing a primary key gets suspended during full load.

The following table shows source PostgreSQL data types and whether they can be migrated successfully.

Data type	Migrates successfully	Partially migrates	does not migrate	Comments
INTEGER	X			
SMALLINT	X			

Data type	Migrates successfully	Partially migrates	does not migrate	Comments
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
VARCHAR(n)	X			
TEXT	X			
BYTEA	X			

Data type	Migrates successfully	Partially migrates	does not migrate	Comments
TIMESTAMP	X			Positive and negative infinity values are truncated to '9999-12-31 23:59:59' and '4713-01-01 00:00:00 BC' respectively.
TIMESTAMP WITH TIME ZONE		X		
DATE	X			
TIME	X			
TIME WITH TIME ZONE	X			
INTERVAL		X		
BOOLEAN	X			
ENUM			X	
CIDR	X			
INET			X	
MACADDR			X	
TSVECTOR			X	
TSQUERY			X	
XML		X		

Data type	Migrates successfully	Partially migrates	does not migrate	Comments
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			Requires Primary Key
COMPOSITE			X	
RANGE			X	
LINESTRING	X			PostGIS spatial data type
MULTIPOINT	X			PostGIS spatial data type

Data type	Migrates successfully	Partially migrates	does not migrate	Comments
MULTILINESTRING	X			PostGIS spatial data type
MULTIPOLYGON	X			PostGIS spatial data type
GEOMETRYCOLLECTION	X			PostGIS spatial data type

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 that the PostGIS plugin is enabled at the global level. Doing this ensures that AWS DMS creates the exact source spatial data columns for the PostgreSQL target DB 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

Migrating from Babelfish for Amazon Aurora PostgreSQL using AWS DMS

You can migrate Babelfish for Aurora PostgreSQL source tables to any supported target endpoints using AWS DMS.

When you create your AWS DMS source endpoint using the DMS console, API, or CLI commands, you set the source to **Amazon Aurora PostgreSQL**, and the database name to **babelfish_db**. In the **Endpoint Settings** section, make sure that the **DatabaseMode** is set to **Babelfish**, and **BabelfishDatabaseName** is set to the name of the source Babelfish T-SQL database. Instead of using the Babelfish TCP port **1433**, use the Aurora PostgreSQL TCP port **5432**.

You must create your tables before migrating data to make sure that DMS uses the correct data types and table metadata. If you don't create your tables on the target before running migration, DMS may create the tables with incorrect data types and permissions.

Adding transformation rules to your migration task

When you create a migration task for a Babelfish source, you need to include transformation rules that ensure that DMS uses the pre-created target tables.

If you set multi-database migration mode when you defined your Babelfish for PostgreSQL cluster, add a transformation rule that renames the schema name to the T-SQL schema. For example, if the T-SQL schema name is dbo, and your Babelfish for PostgreSQL schema name is mydb_dbo, rename the schema to dbo using a transformation rule. To find the PostgreSQL schema name, see Babelfish architecture in the *Amazon Aurora User Guide*.

If you use single-database mode, you don't need to use a transformation rule to rename database schemas. PostgreSQL schema names have a one-to-one mapping to the schema names in the T-SQL database.

The following transformation rule example shows how to rename the schema name from mydb_dbo back to dbo:

```
{
    "rules": [
        {
            "rule-type": "transformation",
            "rule-id": "566251737",
            "rule-name": "566251737",
            "rule-target": "schema",
            "object-locator": {
```

```
                "schema-name": "mydb_dbo"
            },
            "rule-action": "rename",
            "value": "dbo",
            "old-value": null
        },
        {
            "rule-type": "selection",
            "rule-id": "566111704",
            "rule-name": "566111704",
            "object-locator": {
                "schema-name": "mydb_dbo",
                "table-name": "%"
            },
            "rule-action": "include",
            "filters": []
        }
    ]
}
```

Limitations for using a PostgreSQL source endpoint with Babelfish tables

The following limitations apply when using a PostgreSQL source endpoint with Babelfish tables:

- DMS only supports migration from Babelfish version 16.2/15.6 and later, and DMS version 3.5.3 and later.
- DMS does not replicate Babelfish table definition changes to the target endpoint. A workaround for this limitation is to first apply the table definition changes on the target, and then change the table definition on the Babelfish source.
- When you create Babelfish tables with the BYTEA data type, DMS converts them to the `varbinary(max)` data type when migrating to SQL Server as a target.
- DMS does not support Full LOB mode for binary data types. Use limited LOB mode for binary data types instead.
- DMS does not support data validation for Babelfish as a source.
- For the **Target table preparation mode** task setting, use only the **Do nothing** or **Truncate** modes. Don't use the **Drop tables on target** mode. When using **Drop tables on target**, DMS may create the tables with incorrect data types.
- When using ongoing replication (CDC or Full load and CDC), set the `PluginName` extra connection attribute (ECA) to `TEST_DECODING`.

- DMS does not support replication (CDC or Full load and CDC) of partitioned table for Babelfish as a source.

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. Dropping a schema should be done with extreme caution. Never drop an operational schema, especially not a public one.

```
drop event trigger awsdms_intercept_ddl;
```

The event trigger does not belong to a specific schema.

```
drop function {AmazonRDSMigration}.awsdms_intercept_ddl()
drop table {AmazonRDSMigration}.awsdms_ddl_audit
drop schema {AmazonRDSMigration}
```

Additional configuration settings when using a PostgreSQL database as a DMS source

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 Endpoint settings and Extra Connection Attributes (ECAs) when using PostgreSQL as a DMS source.
- You can override connection string parameters. Choose this option to do either of the following:
  - Specify internal AWS DMS parameters. Such parameters are rarely required so aren't exposed in the user interface.
  - Specify pass-through (passthru) values for the specific database client. AWS DMS includes pass-through parameters in the connection sting passed to the database client.

- By using the table-level parameter `REPLICA IDENTITY` in PostgreSQL versions 9.4 and higher, you can control information written to write-ahead logs (WALs). In particular, it does so for WALs that identify rows that are updated or deleted. `REPLICA IDENTITY FULL` records the old values of all columns in the row. Use `REPLICA IDENTITY FULL` carefully for each table as `FULL` generates an extra number of WALs that might not be necessary. For more information, see ALTER TABLE-REPLICA IDENTITY

Read replica as a source for PostgreSQL

Use PostgreSQL read replicas as CDC sources in AWS DMS to reduce primary database load. This feature is available from PostgreSQL 16.x and requires AWS DMS version 3.6.1 or later. Using read replicas for CDC processing reduces the operational impact on your primary database.

> **Note**
>
> Amazon RDS PostgreSQL version 16.x has limitations for read replica logical replication in the Three-AZ (TAZ) configurations. For full read replica logical replication support in TAZ deployments, you must use PostgreSQL version 17.x or later.

Prerequisites

Before using a read replica as a CDC source for AWS DMS, you must enable logical replication on both the primary Database instance and its read replica to create logical decoding on a read replica. Perform the following actions:

- Enable logical replication on both, your primary database instance and its read replica along with any other required database parameters. For more information, see Working with AWS-managed PostgreSQL databases as a DMS source.
- For CDC-only tasks, create a replication slot on the primary (writer) instance. For more information, see Using native CDC start points to set up a CDC load of a PostgreSQL source. This action is necessary as read replicas do not support the creation of replication slot.
- For PostgreSQL version 16, the slot must be manually created on the read replica.
- For PostgreSQL version 17 and above, the replication slot must be created on the primary, and it is automatically synchronized to the read replica.
- When using Full Load + CDC or CDC-only tasks, AWS DMS can automatically manage logical replication slots on primary instances but not on read replicas. For PostgreSQL version 16

read replicas, you must manually drop and recreate replication slots before restarting a task (not resuming). Skipping this step can cause task failures or incorrect CDC starting positions. From PostgreSQL version 17 onwards, logical slot synchronization from the primary instance automates this process.

After completing the prerequisites, you can set up your AWS DMS source endpoint with replication `SlotName` of the read replica source in the endpoint settings and configure your AWS DMS task using native CDC start points. For more information see, Endpoint settings and Extra Connection Attributes (ECAs) when using PostgreSQL as a DMS source and Using native CDC start points to set up a CDC load of a PostgreSQL source.

Using the `MapBooleanAsBoolean` PostgreSQL endpoint setting

You can use PostgreSQL endpoint settings to map a boolean as a boolean from your PostgreSQL source to a Amazon Redshift target. By default, a BOOLEAN type is migrated as varchar(5). You can specify `MapBooleanAsBoolean` to let PostgreSQL to migrate the boolean type as boolean as shown in the example following.

```
--postgre-sql-settings '{"MapBooleanAsBoolean": true}'
```

Note that you must set this setting on both the source and target endpoints for it to take effect.

Since MySQL does not have a BOOLEAN type, use a transformation rule rather than this setting when migrating BOOLEAN data to MySQL.

Endpoint settings and Extra Connection Attributes (ECAs) when using PostgreSQL as a DMS source

You can use endpoint settings and extra connection attributes (ECAs) to configure your PostgreSQL source database. You specify endpoint settings when you create the source endpoint using the AWS DMS console, or by using the `create-endpoint` command in the AWS CLI, with the `--postgre-sql-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following table shows the endpoint settings and ECAs that you can use with PostgreSQL as a source.

Attribute name	Description
`CaptureDdls`	To capture DDL events, AWS DMS creates various artifacts in the PostgreSQL database when the task starts. You can later remove these artifacts as described in Removing AWS DMS artifacts from a PostgreSQL source database. If this value is set to false, you don't have to create tables or triggers on the source database. Streamed DDL events are captured. Default value: `true` Valid values: true/false Example: `--postgre-sql-settings '{"CaptureDdls": true}'`
`ConsumeMonotonicEvents`	Used to control how monolithic transactions with duplicate Log Sequence Numbers (LSNs) are replicated. When this parameter is `false`, events with duplicate LSNs are consumed and replicated on the target. When this parameter is `true`, only the first event is replicated while events with duplicate LSNs aren't consumed nor replicated on the target. Default value: `false` Valid values: false/true Example: `--postgre-sql-settings '{"ConsumeMonotonicEvents": true}'`
`DdlArtifactsSchema`	Sets the schema in which the operational DDL database artifacts are created. Default value: public

Attribute name	Description
	Valid values: String Example: `--postgre-sql-settings '{"DdlArtifactsSchema": " xyzddlschema "}'`
`DisableUnicodeSourceFilter`	Disables the Unicode source filter with PostgreSQL, for values passed into the Selection rule filter on Source Endpoint column values. By default DMS performs source filter comparisons using a Unicode string which can cause look ups to ignore the indexes in the text columns and slow down migrations. Unicode support should only be disabled when using a selection rule filter is on a text column in the Source database that is indexed. Default value: false Valid values: true/false Example: `--postgre-sql-settings '{"DisableUnicodeSourceFilter": " true"}'`
`ExecuteTimeout`	Sets the client statement timeout for the PostgreSQL instance, in seconds. The default value is 60 seconds. Example: `--postgre-sql-settings '{"ExecuteTimeout": 100}'`

Attribute name	Description
`FailTasksOnLobTruncation`	When set to `true`, this value causes a task to fail if the actual size of a LOB column is greater than the specified `LobMaxSize` . If task is set to Limited LOB mode and this option is set to true, the task fails instead of truncating the LOB data. Default value: false Valid values: Boolean Example: `--postgre-sql-settings '{"FailTasksOnLobTruncation": true}'`
`fetchCacheSize`	This extra connection attribute (ECA) sets the number of rows the cursor will fetch during full-load operation. Depending on the resources available in the replication instance, you can adjust the value higher or lower. Default value: `10000` Valid values: Number ECA Example: `fetchCacheSize=10000;`
`HeartbeatEnable`	The WAL heartbeat feature mimics a dummy transaction, so that idle logical replication slots don't hold onto old WAL logs that result in storage full situations on the source. This heartbeat keeps `restart_lsn` moving and prevents storage full scenarios. Default value: `false` Valid values: true/false Example: `--postgre-sql-settings '{"HeartbeatEnable": true}'`

Attribute name	Description
`HeartbeatFrequency`	Sets the WAL heartbeat frequency (in minutes). Default value: 5 Valid values: Number Example: `--postgre-sql-settings '{"HeartbeatFrequency": 1}'`
`HeartbeatSchema`	Sets the schema in which the heartbeat artifacts are created. Default value: `public` Valid values: String Example: `--postgre-sql-settings '{"HeartbeatSchema": "xyzheartbeatschema"}'`
`MapJsonbAsClob`	By default, AWS DMS maps JSONB to NCLOB. You can specify `MapJsonbAsClob` to let PostgreSQL migrate the JSONB type as CLOB. Example: `--postgre-sql-settings='{"MapJsonbAsClob": "true"}'`
`MapLongVarcharAs`	By default, AWS DMS maps VARCHAR to WSTRING. You can specify `MapLongVarcharAs` to let PostgreSQL migrate the VARCHAR(N) type (where N is greater than 16387) to the following types: • WSTRING • CLOB • NCLOB Example: `--postgre-sql-settings='{"MapLongVarcharAs": "CLOB"}'`

Attribute name	Description
`MapUnboundedNumericAsString`	This parameter treats columns with unbounded NUMERIC data types as STRING in order to successfully migrate without losing precision of the numeric value. Use this parameter only for replication from PostgreSQL source to PostgreSQL target, or databases with PostgreSQL compatibility. Default value: `false` Valid values: `false/true` Example: `--postgre-sql-settings '{"MapUnboundedNumericAsString": true}'` Using this parameter might result in some replication performance degradation because of transformation from numeric to string and back to numeric. This parameter is supported for use by DMS version 3.4.4 and higher **Note** Only use `MapUnboundedNumericAsString` in PostgreSQL source and target endpoints together. Use of `MapUnboundedNumericAsString` on source PostgreSQL endpoints restricts precision to 28 during CDC. Use of `MapUnboundedNumericAsString` on target endpoints, migrates data with Precision 28 Scale 6. Do not use `MapUnboundedNumericAsString` with non-PostgreSQL targets.

Attribute name	Description
PluginName	Specifies the plugin to use to create a replication slot. Valid values: pglogical , test_decoding Example: --postgre-sql-settings '{"Plugin Name": "test_decoding"}'
SlotName	Sets the name of a previously created logical replication slot for a CDC load of the PostgreSQL source instance. When used with the AWS DMS API CdcStartP osition 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 CdcStartP osition . If the specified slot does not exist or the task does not have a valid CdcStartPosition setting, DMS raises an error. For more information about setting the CdcStartP osition request parameter, see [Determining a CDC native start point](). For more information about using CdcStartPosition , see the documentation for the CreateReplicationTask , StartRepl icationTask , and ModifyReplicationTask API operations in the [AWS Database Migration Service API Reference](). Valid values: String Example: --postgre-sql-settings '{"SlotNa me": "abc1d2efghijk_34567890_z0y x98w7_6v54_32ut_1srq_1a2b34c5d67ef"}'

Attribute name	Description
`unboundedVarcharMaxSize`	This Extra Connection Attribute (ECA) defines the maximum size of a data column defined as type VarChar without a maximum length specifier. The default is 8000 bytes. The maximum value is 10485760 bytes.

Limitations on using a PostgreSQL database as a DMS source

The following limitations apply when using PostgreSQL as a source for AWS DMS:

- AWS DMS does not 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 does not have a primary key, AWS DMS ignores DELETE and UPDATE record operations for that table. As a workaround, see [Enabling change data capture (CDC) using logical replication](#).

  Note: We don't recommend migrating without a Primary Key/Unique Index, otherwise additional limitations apply such as "NO" Batch apply capability, Full LOB capability, Data Validation and inability to replicate to Redshift target efficiently.
- 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 does not support the **Start Process Changes from Timestamp** run option.
- AWS DMS does not replicate changes that result from partition or subpartition operations (`ADD`, `DROP`, or `TRUNCATE`).
- 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 does not support this type of replication.
- In most cases, AWS DMS supports change processing of CREATE, ALTER, and DROP DDL statements for tables. AWS DMS does not 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` data types in a PostgreSQL source are migrated to a SQL Server target as `bit` data type with inconsistent values. As a workaround, pre-create the table with a `VARCHAR(1)` data type for the column (or have AWS DMS create the table). Then have downstream processing treat an "F" as False and a "T" as True.
- AWS DMS does not support change processing of TRUNCATE operations.
- The OID LOB data type isn't migrated to the target.
- AWS DMS supports the PostGIS data type for only homogeneous migrations.
- 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 PostgreSQL contrib package. For more information about the test-decoding plugin, see the PostgreSQL documentation.
- AWS DMS does not support change processing to set and unset column default values (using the ALTER COLUMN SET DEFAULT clause on ALTER TABLE statements).
- AWS DMS does not support change processing to set column nullability (using the ALTER COLUMN [SET|DROP] NOT NULL clause on ALTER TABLE statements).
- When logical replication is enabled, the maximum number of changes kept in memory per transaction is 4 MB. After that, changes are spilled to disk. As a result `ReplicationSlotDiskUsage` increases, and `restart_lsn` doesn’t advance until the transaction is completed or stopped and the rollback finishes. Because it is a long transaction, it can take a long time to rollback. So, avoid long running transactions or many sub-transactions when logical replication is enabled. Instead, break the transaction into several smaller transactions.

  On Aurora PostgreSQL versions 13 and later, you can tune the `logical_decoding_work_mem` parameter to control when DMS spills change data to disk. For more information, see Spill files in Aurora PostgreSQL.

- A table with an ARRAY data type must have a primary key. A table with an ARRAY data type missing a primary key gets suspended during full load.
- AWS DMS does not support migrating table metadata related to table partitioning or table inheritance. When AWS DMS encounters a partitioned table or a table that uses inheritance, the following behavior is observed:
  - AWS DMS identifies and reports both parent and child tables involved in partitioning or inheritance on the source database.
  - **Table Creation on Target**: On the target database, AWS DMS creates the table as a standard (non-partitioned, non-inherited) table, preserving the structure and properties of the selected table(s) but not the partitioning or inheritance logic.
  - **Record Differentiation in Inherited Tables**: For tables that use inheritance, AWS DMS does not distinguish records belonging to child tables when populating the parent table. As a result, it does not utilize SQL queries with syntax such as: `SELECT * FROM ONLY parent_table_name`.
- To replicate partitioned tables from a PostgreSQL source to a PostgreSQL target, first manually create the parent and child tables on the target. Then define a separate task to replicate to those tables. In such a case, set the task configuration to **Truncate before loading**.
- 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 0.611111104488373 from the source is converted to 0.611111 on the PostgreSQL target.
- AWS DMS supports Aurora PostgreSQL Serverless V1 as a source for full load tasks only. AWS DMS supports Aurora PostgreSQL Serverless V2 as a source for full load, full load and CDC, and CDC-only tasks.
- AWS DMS does not support replication of a table with a unique index created with a coalesce function.
- If primary key definition on source and target does not match, results of replication may be unpredictable.
- When using the Parallel Load feature, table segmentation according to partitions or sub-partitions isn't supported. For more information about Parallel Load, see Using parallel load for selected tables, views, and collections
- AWS DMS does not support Deferred Constraints.
- AWS DMS version 3.4.7 supports PostgreSQL 14.x as a source with these limitations:
  - AWS DMS does not support change processing of two phase commits.

  - AWS DMS does not support logical replication to stream long in-progress transactions.
- AWS DMS does not support CDC for Amazon RDS Proxy for PostgreSQL as a source.
- When using source filters that don't contain a Primary Key column, `DELETE` operations won't be captured.
- If your source database is also a target for another third–party replication system, DDL changes might not migrate during CDC. Because that situation can prevent the `awsdms_intercept_ddl` event trigger from firing. To work around the situation, modify that trigger on your source database as follows:

```
alter event trigger awsdms_intercept_ddl enable always;
```

- AWS DMS does not support replication of changes made to primary key definitions in the source database. If the primary key structure is altered during an active replication task, subsequent changes to affected tables are not replicated to the target.
- In DDL replication as part of a script, the maximum total number of DDL commands per script is 8192 and maximum total number of lines per script is 8192 lines.
- AWS DMS does not support Materialized Views.
- For full load and CDC tasks using a read replica as the source, AWS DMS cannot create replication slots on read replicas.

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.

PostgreSQL data types	DMS data types
INTEGER	INT4
SMALLINT	INT2

PostgreSQL data types	DMS data types
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
CITEXT	NCLOB
BYTEA	BLOB
TIMESTAMP	DATETIME

PostgreSQL data types	DMS data types
TIMESTAMP WITH TIME ZONE	DATETIME
DATE	DATE
TIME	TIME
TIME WITH TIME ZONE	TIME
INTERVAL	STRING (128)—1 YEAR, 2 MONTHS, 3 DAYS, 4 HOURS, 5 MINUTES, 6 SECONDS
BOOLEAN	CHAR (5) false or true
ENUM	STRING (64)
CIDR	STRING (50)
INET	STRING (50)
MACADDR	STRING (18)
BIT (n)	STRING (n)
BIT VARYING (n)	STRING (n)
UUID	STRING
TSVECTOR	CLOB
TSQUERY	CLOB
XML	CLOB
POINT	STRING (255) "(x,y)"
LINE	STRING (255) "(x,y,z)"
LSEG	STRING (255) "((x1,y1),(x2,y2))"
BOX	STRING (255) "((x1,y1),(x2,y2))"

PostgreSQL data types	DMS data types
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)

Working with LOB source data types for PostgreSQL

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`. Use this length 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.

You do this 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.

For information about versions of MySQL that AWS DMS supports as a source, see Sources for AWS DMS.

You can use SSL to encrypt connections between your MySQL-compatible endpoint and the replication instance. For more information on using SSL with a MySQL-compatible endpoint, see Using SSL with AWS Database Migration Service.

In the following sections, the term "self-managed" applies to any database that is installed either on-premises or on Amazon EC2. The term "AWS-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
- Using any MySQL-compatible database as a source for AWS DMS
- Using a self-managed MySQL-compatible database as a source for AWS DMS
- Using an AWS-managed MySQL-compatible database as a source for AWS DMS
- Limitations on using a MySQL database as a source for AWS DMS
- Support for XA transactions
- Endpoint settings when using MySQL as a source for AWS DMS
- Source data types for MySQL

> **Note**
>
> When configuring AWS Database Migration Service (AWS DMS) mapping rules, it is important to avoid using wildcards (%) for database or schema names. Instead, you

must explicitly specify only the user-created databases that need to be migrated. Using a wildcard character includes all databases in the migration process, including system databases which are not required on the target instance. Since the MySQL Amazon RDS master user lacks the necessary permissions to import data into target system databases, attempting to migrate these system databases fails.

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, we recommend that you use a homogeneous data migrations migration project. homogeneous data migrations uses native database tools to provide an improved data migration performance and accuracy when compared to AWS DMS.

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 AWS-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 CDC tasks only. In other words, full-load-only tasks don't require this privilege.

  > **Note**
  >
  > For MariaDB version 10.5.2+, you can use BINLOG MONITOR – it is a replacement for REPLICATION CLIENT.

- **REPLICATION SLAVE** – This privilege is required for 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.

Grant the following privileges if you use MySQL-specific premigration assessments:

```
grant select on mysql.user to <dms_user>;
grant select on mysql.db to <dms_user>;
grant select on mysql.tables_priv to <dms_user>;
grant select on mysql.role_edges to <dms_user>  #only for MySQL version 8.0.11 and
 higher
grant select on performance_schema.replication_connection_status to <dms_user>;
 #Required for primary instance validation - MySQL version 5.7 and higher only
```

If you're using an RDS source and plan to run MySQL-specific premigration assessments, add the following permission:

```
grant select on mysql.rds_configuration to <dms_user>;  #Required for binary log
 retention check
```

If parameter `BatchEnable` is `true` it is required to grant:

```
grant create temporary tables on `<schema>`.* to <dms_user>;
```

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

To use CDC, make sure to enable binary logging. To enable binary logging, the following parameters must be configured in MySQL's `my.ini` (Windows) or `my.cnf` (UNIX) file.

Parameter	Value
server_id	Set this parameter to a value of 1 or greater.
log-bin	Set the path to the binary log file, such as log-bin=E:\MySql_Logs\BinLog . Don't include the file extension.
binlog_format	Set this parameter to ROW. We recommend this setting during replication because in certain cases when binlog_format is set to STATEMENT , it can cause inconsistency when replicating data to the target. The database engine also writes similar inconsistent data to the target when binlog_format is set to MIXED, because the database engine automatically switches to STATEMENT -based logging which can result in writing inconsistent data on the target database.
expire_logs_days	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.
binlog_checksum	Set this parameter to NONE for DMS version 3.4.7 or prior.
binlog_row_image	Set this parameter to FULL.
log_slave_updates	Set this parameter to TRUE if you are using a MySQL or MariaDB read-replica as a source.

If you are using a MySQL or MariaDB read-replica as a source for a DMS migration task using **Migrate existing data and replicate ongoing changes** mode, there is a possibility of data loss. DMS won't write a transaction during either full load or CDC under the following conditions:

- The transaction had been committed to the primary instance before the DMS task started.
- The transaction hadn't been committed to the replica until after the DMS task started, due to lag between the primary instance and the replica.

The longer the lag between the primary instance and the replica, the greater potential there is for data loss.

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
`ndb_log_bin`	Set this parameter to `ON`. This value ensures that changes in clustered tables are logged to the binary log.
`ndb_log_update_as_write`	Set this parameter to `OFF`. This value prevents writing UPDATE statements as INSERT statements in the binary log.
`ndb_log_updated_only`	Set this parameter to `OFF`. This value ensures that the binary log contains the entire row and not just the changed columns.

Using an AWS-managed MySQL-compatible database as a source for AWS DMS

You can use the following AWS-managed MySQL-compatible databases as sources for AWS DMS:

- MySQL Community Edition
- MariaDB Community Edition
- Amazon Aurora MySQL-Compatible Edition

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

- To enable binary logs for RDS for MySQL and for RDS for MariaDB, enable automatic backups at the instance level. To enable binary logs for an Aurora MySQL cluster, change the variable `binlog_format` in the parameter group.

  For more information about setting up automatic backups, see Working with automated backups in the *Amazon RDS User Guide*.

  For more information about setting up binary logging for an Amazon RDS for MySQL database, see Setting the binary logging format in the *Amazon RDS User Guide*.

  For more information about setting up binary logging for an Aurora MySQL cluster, see How do I turn on binary logging for my Amazon Aurora MySQL cluster?.

- If you plan to use CDC, turn on binary logging. For more information on setting up binary logging for an Amazon RDS for MySQL database, see [Setting the binary logging format](#) in the *Amazon RDS User Guide*.
- Ensure that the binary logs are available to AWS DMS. Because AWS-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"`.

> **Note**
>
> On MySQL or MariaDB, `binlog_format` is a dynamic parameter, so you don't have to reboot to make the new value take effect. However, the new value will only apply to new sessions. If you switch the `binlog_format` to `ROW` for replication purposes, your database can still create subsequent binary logs using the `MIXED` format, if those sessions started before you changed the value. This may prevent AWS DMS from properly capturing all changes on the source database. When you change the `binlog_format` setting on a MariaDB or MySQL database, be sure to restart the database to close all existing sessions, or restart any application performing DML (Data Manipulation Language) operations. Forcing your database to restart all sessions after changing the `binlog_format` parameter to `ROW` will ensure that your database writes all subsequent source database changes using the correct format, so that AWS DMS can properly capture those changes.

- Set the `binlog_row_image` parameter to `"Full"`.
- Set the `binlog_checksum` parameter to `"NONE"` for DMS version 3.4.7 or prior. 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, and ensure the `log_slave_updates` parameter is set to TRUE.

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, 5.7, or 8.0 to enable CDC. CDC is supported for self-managed MySQL 5.5 sources.
- For CDC, `CREATE TABLE`, `ADD COLUMN`, and `DROP COLUMN` changing the column data type, and `renaming a column` are supported. However, `DROP TABLE`, `RENAME TABLE`, and updates made to other attributes, such as column default value, column nullability, character set and so on, are not supported.
- For partitioned tables on the source, when you set **Target table preparation mode** to **Drop tables on target**, AWS DMS creates a simple table without any partitions on the MySQL target. To migrate partitioned tables to a partitioned table on the target, precreate 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 Microsoft Windows or OS X using HFS+.
- You can use Aurora MySQL-Compatible Edition Serverless v1 for full load, but you can't use it for CDC. This is because you can't enable the prerequisites for MySQL. For more information, see [Parameter groups and Aurora Serverless v1](#).

  Aurora MySQL-Compatible Edition Serverless v2 supports CDC.
- 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 does not 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 replicas as a source for AWS DMS unless your DMS migration task mode is **Migrate existing data**—full load only.

- If the MySQL-compatible source is stopped during full load, the AWS DMS task does not 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 does not 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 replica instance.
- AWS DMS currently does not support views migration for MariaDB.
- AWS DMS does not support DDL changes for partitioned tables for MySQL. To skip table suspension for partition DDL changes during CDC, set `skipTableSuspensionForPartitionDdl` to `true`.
- AWS DMS only supports XA transactions in version 3.5.0 and higher. Previous versions do not support XA transactions. AWS DMS does not support XA transactions in MariaDB version 10.6 or higher For more information, see [the section called "Support for XA transactions"] following.
- AWS DMS does not use GTIDs for replication, even if the source data contains them.
- AWS DMS does not support Aurora MySQL enhanced binary log.
- AWS DMS does not support binary log transaction compression.
- AWS DMS does not propagate ON DELETE CASCADE and ON UPDATE CASCADE events for MySQL databases using the InnoDB storage engine. For these events, MySQL does not generate binlog events to reflect the cascaded operations on the child tables. Consequently, AWS DMS can't replicate the corresponding changes to the child tables. For more information, see [Indexes, Foreign Keys, or Cascade Updates or Deletes Not Migrated].
- AWS DMS does not capture changes to computed (`VIRTUAL` and `GENERATED ALWAYS`) columns. To work around this limitation, do the following:
  - Pre-create the target table in the target database, and create the AWS DMS task with the `DO_NOTHING` or `TRUNCATE_BEFORE_LOAD` full-load task setting.

  - Add a transformation rule to remove the computed column from the task scope. For information about transformation rules, see [Transformation rules and actions](#).
- Due to internal MySQL limitation, AWS DMS can process BINLOGs no larger than 4GB size. BINLOGs larger than 4GB may result in DMS task failures or other unpredictable behavior. You must reduce the size of transactions to avoid BINLOGs larger than 4GB.
- AWS DMS does not support back-ticks (`` ` ``) or single quotes (`'`) in schema, table, and column names.
- AWS DMS does not migrate data from invisible columns in your source database. To include these columns in your migration scope, use the ALTER TABLE statement to make these columns visible.

Support for XA transactions

An Extended Architecture (XA) transaction is a transaction that can be used to group a series of operations from multiple transactional resources into a single, reliable global transaction. An XA transaction uses a two-phase commit protocol. In general, capturing changes while there are open XA transactions might lead to loss of data. If your database does not use XA transactions, you can ignore this permission and the configuration `IgnoreOpenXaTransactionsCheck` by using the deafult value TRUE. To start replicating from a source that has XA transactions, do the following:

- Ensure that the AWS DMS endpoint user has the following permission:

```
grant XA_RECOVER_ADMIN on *.* to 'userName'@'%';
```

- Set the endpoint setting `IgnoreOpenXaTransactionsCheck` to `false`.

> **Note**
>
> AWS DMS doesn’t support XA transactions on MariaDB Source DB version 10.6 or higher.

Endpoint settings when using MySQL as a source for AWS DMS

You can use endpoint settings to configure your MySQL source database similar to using extra connection attributes. You specify the settings when you create the source endpoint using the AWS

DMS console, or by using the `create-endpoint` command in the [AWS CLI](https://docs.aws.amazon.com/cli/latest/reference/dms/index.html), with the `--my-sql-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following table shows the endpoint settings that you can use with MySQL as a source.

Name	Description
`ConnectionTimeout`	Use this extra connection attribute (ECA) to set the endpoint connection timeout for the MySQL instance, in seconds. The default value is 10 seconds. ECA Example: `ConnectionTimeout=30` .
`EventsPollInterval`	Specifies how often to check the binary log for new changes/events when the database is idle. Default value: 5 Valid values: 1–60 Example: `--my-sql-settings '{"EventsPollInterval": 5}'` In the example, AWS DMS checks for changes in the binary logs every five seconds.
`ExecuteTimeout`	For AWS DMS versions 3.4.7 and higher, sets the client statement timeout for a MySQL source endpoint, in seconds. Default value: 60 Example: `--my-sql-settings '{"ExecuteTimeout": 1500}'`
`ServerTimezone`	Specifies the time zone for the source MySQL database. Example: `--my-sql-settings '{"ServerTimezone": " US/Pacific "}'`
`AfterConnectScript`	Specifies a script to run immediately after AWS DMS connects to the endpoint. The migration task continues

Name	Description
	running regardless if the SQL statement succeeds or fails. Valid values: One or more valid SQL statements, set off by a semicolon. Example: `--my-sql-settings '{"AfterConnectScript": "ALTER SESSION SET CURRENT_SCHEMA=system"}'`
`CleanSourceMetadataOnMismatch`	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. Default value: `false` Example: `--my-sql-settings '{"CleanSourceMetadataOnMismatch": false}'`
`skipTableSuspensionForPartitionDdl`	AWS DMS does not support DDL changes for partitioned tables for MySQL. For AWS DMS versions 3.4.6 and higher, setting this to `true` skips table suspension for partition DDL changes during CDC. AWS DMS ignores partitioned-table-related DDL, and continues to process further binary log changes. Default value: `false` Example: `--my-sql-settings '{"skipTableSuspensionForPartitionDdl": true}'`

Name	Description
`IgnoreOpenXaTransactionsCheck`	For AWS DMS versions 3.5.0 and higher, specifies whether tasks should ignore open XA transactions while starting. Set this to `false` if your source has XA transactions. Default value: `true` Example: `--my-sql-settings '{"IgnoreOpenXaTransactionsCheck": false}'`

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.

MySQL data types	AWS DMS data types
INT	INT4
BIGINT	INT8
MEDIUMINT	INT4
TINYINT	INT1
SMALLINT	INT2
UNSIGNED TINYINT	UINT1
UNSIGNED SMALLINT	UINT2
UNSIGNED MEDIUMINT	UINT4

MySQL data types	AWS DMS data types
UNSIGNED INT	UINT4
UNSIGNED BIGINT	UINT8
DECIMAL(10)	NUMERIC (10,0)
BINARY	BYTES(1)
BIT	BOOLEAN
BIT(64)	BYTES(8)
BLOB	BYTES(65535)
LONGBLOB	BLOB
MEDIUMBLOB	BLOB
TINYBLOB	BYTES(255)
DATE	DATE
DATETIME	DATETIME DATETIME without a parenthetical value is replicated without milliseconds. DATETIME with a parenthetical value of 1 to 5 (such as `DATETIME(5)`) is replicated with milliseco nds. When replicating a DATETIME column, the time remains the same on the target. It is not converted to UTC.
TIME	STRING

MySQL data types	AWS DMS data types
TIMESTAMP	DATETIME When replicating a TIMESTAMP column, the time is converted to UTC on the target.
YEAR	INT2
DOUBLE	REAL8
FLOAT	REAL(DOUBLE) If the FLOAT values are not in the range following, use a transformation to map FLOAT to STRING. For more information about transformations, see Transformation rules and actions. The supported FLOAT range is -1.79E+308 to -2.23E-308, 0, and 2.23E-308 to 1.79E+308
VARCHAR (45)	WSTRING (45)
VARCHAR (2000)	WSTRING (2000)
VARCHAR (4000)	WSTRING (4000)
VARBINARY (4000)	BYTES (4000)
VARBINARY (2000)	BYTES (2000)
CHAR	WSTRING
TEXT	WSTRING
LONGTEXT	NCLOB
MEDIUMTEXT	NCLOB
TINYTEXT	WSTRING(255)

MySQL data types	AWS DMS data types
GEOMETRY	BLOB
POINT	BLOB
LINESTRING	BLOB
POLYGON	BLOB
MULTIPOINT	BLOB
MULTILINESTRING	BLOB
MULTIPOLYGON	BLOB
GEOMETRYCOLLECTION	BLOB
ENUM	WSTRING (*length*) Here, *length* is the length of the longest value in the ENUM.
SET	WSTRING (*length*) Here, *length* is the total length of all values in the SET, including commas.
JSON	CLOB

Note

In some cases, you might specify the DATETIME and TIMESTAMP data types with a "zero" value (that is, 0000-00-00). If so, 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.

Using an SAP ASE database as a source for AWS DMS

You can migrate data from an SAP Adaptive Server Enterprise (ASE) database—formerly known as Sybase—using AWS DMS. With an SAP ASE database as a source, you can migrate data to any of the other supported AWS DMS target databases.

For information about versions of SAP ASE that AWS DMS supports as a source, see Sources for AWS DMS.

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

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. For more information, see Sybase Infocenter Archive.
- Disable `RepAgent` on the SAP ASE database. For more information, see Stop and disable the RepAgent thread in the primary database.
- To replicate to SAP ASE version 15.7 on an Windows EC2 instance configured for non-Latin characters (for example, Chinese), install SAP ASE 15.7 SP121 on the target computer.

> **Note**
>
> For ongoing change data capture (CDC) replication, DMS runs `dbcc logtransfer` and `dbcc log` to read data from the transaction log.

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:

- You can run only one AWS DMS task with ongoing replication or CDC for each SAP ASE database. You can run multiple full-load-only tasks in parallel.
- 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';
```

- Zero values located at the end of binary data type strings are truncated when replicated to the target database. For example, `0x0000000000000000000000000100000100000000` in the source table becomes `0x00000000000000000000000001000001` in the target table.
- 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 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.
- AWS DMS does not support clusters or using MSA (Multi-Site Availability)/Warm Standby as a source.
- When `AR_H_TIMESTAMP` transformation header expression is used in mapping rules, the milliseconds won't be captured for an added column.
- Running Merge operations during CDC will result in a non-recoverable error. To bring the target back in sync, run a full load.
- Rollback trigger events are not supported for tables that use a data row locking scheme.

- AWS DMS can't resume a replication task after dropping a table within the task scope from a source SAP database. If the DMS replication task was stopped and performed any DML operation (INSERT,UPDATE,DELETE) followed by dropping the table, you must restart the replication task.

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, you need to grant permissions. 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 Endpoint settings when using SAP ASE as a source for AWS DMS.

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).

Endpoint settings when using SAP ASE as a source for AWS DMS

You can use endpoint settings to configure your SAP ASE source database similar to using extra connection attributes. You specify the settings when you create the source endpoint using the AWS DMS console, or by using the `create-endpoint` command in the [AWS CLI](https://docs.aws.amazon.com/cli/latest/reference/dms/index.html), with the `--sybase-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following table shows the endpoint settings that you can use with SAP ASE as a source.

Name	Description
`Charset`	Set this attribute to the SAP ASE name that correspon ds to the international character set. Some character sets require the endpoint setting `Provider` to be set to `Adaptive Server Enterprise 16.0.02.00 Full`. Default value: `iso_1` Example: `--sybase-settings '{"Charset": "utf8"}'` Valid values: • `acsii_8` • `big5hk` • `cp437` • `cp850` • `cp852` • `cp852` • `cp855` • `cp857` • `cp858` • `cp860` • `cp864` • `cp866`

Name	Description
	• `cp869` • `cp874` • `cp932` • `cp936` • `cp950` • `cp1250` • `cp1251` • `cp1252` • `cp1253` • `cp1254` • `cp1255` • `cp1256` • `cp1257` • `cp1258` • `deckanji` • `euccns` • `eucgb` • `eucjis` • `eucksc` • `gb18030` • `greek8` • `iso_1` • `iso88592` • `iso88595` • `iso88596` • `iso88597` • `iso88598` • `iso88599` • `iso15`

Name	Description
	• kz1048 • koi8 • roman8 • iso88599 • sjis • tis620 • turkish8 • utf8 For any further questions about supported character sets in a SAP ASE database, see Adaptive Server Enterprise: Supported character sets.
EnableReplication	Set this attribute if you want to enable sp_setrep table on tables from the database end and not through AWS DMS. Default value: true Valid values: true or false Example: --sybase-settings '{"EnableReplication": false}'

Name	Description
`EncryptPassword`	Set this attribute if you have enabled `"net password encryption reqd"` at the source database. Default value: `0` Valid values: `0`, 1, or 2 Example: `--sybase-settings '{"EncryptPassword": 1}'` For more information on these parameter values, see [Adaptive Server Enterprise: Using the EncryptPassword Connection string property]().
`Provider`	Set this attribute if you want to use Transport Layer Security (TLS) 1.2 for versions of ASE 15.7 and higher. Note that AWS requires TLS version 1.2 or later, and recommends version 1.3. Default value: `Adaptive Server Enterprise` Valid values: `Adaptive Server Enterprise 16.03.06`, `Adaptive Server Enterprise 16.0.02.00 Full` Example: `--sybase-settings '{"Provider": "Adaptive Server Enterprise 16.03.06"}'`

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 section for your target endpoint.

For additional information about AWS DMS data types, see Data types for AWS Database Migration Service.

SAP ASE data types	AWS DMS data types
BIGINT	INT8
UNSIGNED BIGINT	UINT8
INT	INT4
UNSIGNED INT	UINT4
SMALLINT	INT2
UNSIGNED SMALLINT	UINT2
TINYINT	UINT1
DECIMAL	NUMERIC
NUMERIC	NUMERIC
FLOAT	REAL8
DOUBLE	REAL8
REAL	REAL4
MONEY	NUMERIC
SMALLMONEY	NUMERIC
DATETIME	DATETIME
BIGDATETIME	DATETIME(6)
SMALLDATETIME	DATETIME
DATE	DATE
TIME	TIME

SAP ASE data types	AWS DMS data types
BIGTIME	TIME
CHAR	STRING
UNICHAR	WSTRING
NCHAR	WSTRING
VARCHAR	STRING
UNIVARCHAR	WSTRING
NVARCHAR	WSTRING
BINARY	BYTES
VARBINARY	BYTES
BIT	BOOLEAN
TEXT	CLOB
UNITEXT	NCLOB
IMAGE	BLOB

Using MongoDB as a source for AWS DMS

For information about versions of MongoDB that AWS DMS supports as a source, see Sources for AWS DMS.

Note the following about MongoDB version support:

- Versions of AWS DMS 3.4.5 and later support MongoDB versions 4.2 and 4.4.
- Versions of AWS DMS 3.4.5 and later and versions of MongoDB 4.2 and later support distributed transactions. For more information on MongoDB distributed transactions, see Transactions in the MongoDB documentation.
- Versions of AWS DMS 3.5.0 and later don't support versions of MongoDB prior to 3.6.

- Versions of AWS DMS 3.5.1 and later support MongoDB version 5.0.
- Versions of AWS DMS 3.5.2 and later support MongoDB version 6.0.
- Versions of AWS DMS 3.5.4 and later support MongoDB version 7.0 and 8.0.

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.
- A *database* in MongoDB is a set of collections, and is roughly equivalent to a schema in a relational database.
- 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, *Document mode* or *Table mode*. You specify which migration mode to use when you create the MongoDB endpoint or by setting the **Metadata mode** parameter from the AWS DMS console. Optionally, you can create a second column named `_id` that acts as the primary key by selecting the check mark button for **_id as a separate column** in the endpoint configuration panel.

Your 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 CDC, set this parameter to *true*.

When using CDC with sources that produce [multi-document transactions](), the `ExtractDocId` parameter **must be** set to *true*. If this parameter is not enabled, the AWS DMS task will fail when it encounters a multi-document transaction.

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.

If the target endpoint is Amazon DocumentDB, run the migration in **Document mode**.

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

5a94815f4 0bd44d1b02bdfe0	1	2	3
5a94815f4 0bd44d1b02bdfe1	4	5	6

Nested values are flattened into a column containing dot-separated key names. The column is named the concatenation of the flattened field names separated by periods. For example, AWS DMS migrates a JSON document with a field of nested values such as `{"a" : {"b" : {"c": 1}}}` into a column named `a.b.c.`

To create the target columns, AWS DMS scans a specified number of MongoDB documents and creates a set of all the fields and their types. AWS DMS then uses this set to create the columns of the target table. If you create or modify your MongoDB source endpoint using the console, you can specify the number of documents to scan. The default value is 1000 documents. If you use the AWS CLI, you can use the extra connection attribute `docsToInvestigate`.

In table mode, AWS DMS manages documents and collections like this:

- When you add a document to an existing collection, the document is replicated. If there are fields that don't exist in the target, those fields aren't replicated.
- When you update a document, the updated document is replicated. If there are fields that don't exist in the target, those fields aren't replicated.
- Deleting a document is fully supported.
- Adding a new collection doesn't result in a new table on the target when done during a CDC task.
- In the Change Data Capture(CDC) phase, AWS DMS doesn't support renaming a collection.

Topics

- Endpoint configuration settings when using MongoDB as a source for AWS DMS
- Source data types for MongoDB

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" } ]
  }
)
```

For a MongoDB 3.x source, 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"]
})
```

For a MongoDB 4.x source, the following code creates a user with minimal privileges.

```
{ resource: { db: "", collection: "" }, actions: [ "find", "changeStream" ] }
```

For example, create the following role in the "admin" database.

```
use admin
db.createRole(
{
role: "changestreamrole",
```

```
privileges: [
{ resource: { db: "", collection: "" }, actions: [ "find","changeStream" ] }
],
roles: []
}
)
```

And once the role is created, create a user in the database to be migrated.

```
 use test
> db.createUser(
{
user: "dms-user12345",
pwd: "password",
roles: [ { role: "changestreamrole", db: "admin" }, "read"]
})
```

Configuring a MongoDB replica set for CDC

To use ongoing replication or 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.

> **Note**
>
> AWS DMS uses the operations log (oplog) to capture changes during ongoing replication. If MongoDB flushes out the records from the oplog before AWS DMS reads them, your tasks fail. We recommend sizing the oplog to retain changes for at least 24 hours.

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** – For backward compatibility
- **SCRAM-SHA-1** – The default when using MongoDB version 3.x and 4.0

If an authentication method isn't specified, AWS DMS uses the default method for the version of the MongoDB source.

Segmenting MongoDB collections and migrating in parallel

To improve performance of a migration task, MongoDB source endpoints support two options for parallel full load in table mapping.

In other words, you can migrate a collection in parallel by using either autosegmentation or range segmentation with table mapping for a parallel full load in JSON settings. With autosegmentation, you can specify the criteria for AWS DMS to automatically segment your source for migration in each thread. With range segmentation, you can tell AWS DMS the specific range of each segment for DMS to migrate in each thread. For more information on these settings, see Table and collection settings rules and operations.

Migrating a MongoDB database in parallel using autosegmentation ranges

You can migrate your documents in parallel by specifying the criteria for AWS DMS to automatically partition (segment) your data for each thread. In particular, you specify the number of documents to migrate per thread. Using this approach, AWS DMS attempts to optimize segment boundaries for maximum performance per thread.

You can specify the segmentation criteria using the table-settings options following in table mapping.

Table-settings option	Description
`"type"`	(Required) Set to `"partitions-auto"` for MongoDB as a source.
`"number-of-partitions"`	(Optional) Total number of partitions (segments) used for migration. The default is 16.
`"collection-count-from-metadata"`	(Optional) If this option is set to `true`, AWS DMS uses an estimated collection count for determining the number of partitions. If this option is set to `false`, AWS DMS uses the actual collection count. The default is `true`.
`"max-records-skip-per-page"`	(Optional) The number of records to skip at once when determining the boundaries for each partition. AWS DMS uses a paginated skip approach to determine the minimum boundary for a partition. The default is 10,000. Setting a relatively large value can result in cursor timeouts and task failures. Setting a relatively low value results in more operations per page and a slower full load.

Table-settings option	Description
`"batch-size"`	(Optional) Limits the number of documents returned in one batch. Each batch requires a round trip to the server. If the batch size is zero (0), the cursor uses the server-defined maximum batch size. The default is 0.

The example following shows a table mapping for autosegmentation.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "admin",
                "table-name": "departments"
            },
            "rule-action": "include",
            "filters": []
        },
        {
            "rule-type": "table-settings",
            "rule-id": "2",
            "rule-name": "2",
            "object-locator": {
                "schema-name": "admin",
                "table-name": "departments"
            },
            "parallel-load": {
                "type": "partitions-auto",
                "number-of-partitions": 5,
                "collection-count-from-metadata": "true",
                "max-records-skip-per-page": 1000000,
                "batch-size": 50000
            }
        }
    ]
}
```

Autosegmentation has the limitation following. The migration for each segment fetches the collection count and the minimum `_id` for the collection separately. It then uses a paginated skip to calculate the minimum boundary for that segment.

Therefore, ensure that the minimum `_id` value for each collection remains constant until all the segment boundaries in the collection are calculated. If you change the minimum `_id` value for a collection during its segment boundary calculation, it can cause data loss or duplicate row errors.

Migrating a MongoDB database in parallel using range segmentation

You can migrate your documents in parallel by specifying the ranges for each segment in a thread. Using this approach, you tell AWS DMS the specific documents to migrate in each thread according to your choice of document ranges per thread.

The image following shows a MongoDB collection that has seven items, and `_id` as the primary key.

Key	Value	Type
(1) ObjectId("5f805c74873173399a278d78")	{ 3 fields }	Object
_id	ObjectId("5f805c74873173399a278d78")	ObjectId
num	1	Int32
name	a	String
(2) ObjectId("5f805c97873173399a278d79")	{ 3 fields }	Object
_id	ObjectId("5f805c97873173399a278d79")	ObjectId
num	2	Int32
name	b	String
(3) ObjectId("5f805cb0873173399a278d7a")	{ 3 fields }	Object
_id	ObjectId("5f805cb0873173399a278d7a")	ObjectId
num	3	Int32
name	c	String
(4) ObjectId("5f805cbb873173399a278d7b")	{ 3 fields }	Object
_id	ObjectId("5f805cbb873173399a278d7b")	ObjectId
num	4	Int32
name	d	String
(5) ObjectId("5f805cc5873173399a278d7c")	{ 3 fields }	Object
_id	ObjectId("5f805cc5873173399a278d7c")	ObjectId
num	5	Int32
name	e	String
(6) ObjectId("5f805cd0873173399a278d7d")	{ 3 fields }	Object
_id	ObjectId("5f805cd0873173399a278d7d")	ObjectId
num	6	Int32
name	f	String
(7) ObjectId("5f805cdd873173399a278d7e")	{ 3 fields }	Object
_id	ObjectId("5f805cdd873173399a278d7e")	ObjectId
num	7	Int32
name	g	String

To split the collection into three specific segments for AWS DMS to migrate in parallel, you can add table mapping rules to your migration task. This approach is shown in the following JSON example.

```
{ // Task table mappings:
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "testdatabase",
        "table-name": "testtable"
      },
      "rule-action": "include"
```

```
    }, // "selection" :"rule-type"
    {
      "rule-type": "table-settings",
      "rule-id": "2",
      "rule-name": "2",
      "object-locator": {
        "schema-name": "testdatabase",
        "table-name": "testtable"
      },
      "parallel-load": {
        "type": "ranges",
        "columns": [
           "_id",
           "num"
        ],
        "boundaries": [
          // First segment selects documents with _id less-than-or-equal-to
 5f805c97873173399a278d79
          // and num less-than-or-equal-to 2.
          [
             "5f805c97873173399a278d79",
             "2"
          ],
          // Second segment selects documents with _id > 5f805c97873173399a278d79 and
          // _id less-than-or-equal-to 5f805cc5873173399a278d7c and
          // num > 2 and num less-than-or-equal-to 5.
          [
             "5f805cc5873173399a278d7c",
             "5"
          ]
          // Third segment is implied and selects documents with _id >
 5f805cc5873173399a278d7c.
        ] // :"boundaries"
      } // :"parallel-load"
    } // "table-settings" :"rule-type"
  ] // :"rules"
} // :Task table mappings
```

That table mapping definition splits the source collection into three segments and migrates in parallel. The following are the segmentation boundaries.

```
Data with _id less-than-or-equal-to "5f805c97873173399a278d79" and num less-than-or-
equal-to 2 (2 records)
Data with _id > "5f805c97873173399a278d79" and num > 2 and _id  less-than-or-equal-to
 "5f805cc5873173399a278d7c" and num less-than-or-equal-to 5 (3 records)
Data with _id > "5f805cc5873173399a278d7c" and num > 5 (2 records)
```

After the migration task is complete, you can verify from the task logs that the tables loaded in parallel, as shown in the following example. You can also verify the MongoDB `find` clause used to unload each segment from the source table.

```
[TASK_MANAGER    ] I:  Start loading segment #1 of 3 of table
 'testdatabase'.'testtable' (Id = 1) by subtask 1. Start load timestamp
 0005B191D638FE86  (replicationtask_util.c:752)

[SOURCE_UNLOAD   ] I:  Range Segmentation filter for Segment #0 is initialized.
 (mongodb_unload.c:157)

[SOURCE_UNLOAD   ] I:  Range Segmentation filter for Segment #0 is: { "_id" :
 { "$lte" : { "$oid" : "5f805c97873173399a278d79" } }, "num" : { "$lte" :
 { "$numberInt" : "2" } } }  (mongodb_unload.c:328)

[SOURCE_UNLOAD   ] I: Unload finished for segment #1 of segmented table
 'testdatabase'.'testtable' (Id = 1). 2 rows sent.

[TASK_MANAGER    ] I: Start loading segment #1 of 3 of table
 'testdatabase'.'testtable' (Id = 1) by subtask 1. Start load timestamp
 0005B191D638FE86 (replicationtask_util.c:752)

[SOURCE_UNLOAD   ] I: Range Segmentation filter for Segment #0 is initialized.
 (mongodb_unload.c:157)

[SOURCE_UNLOAD   ] I: Range Segmentation filter for Segment #0 is: { "_id" : { "$lte" :
 { "$oid" : "5f805c97873173399a278d79" } }, "num" : { "$lte" : { "$numberInt" :
 "2" } } } (mongodb_unload.c:328)

[SOURCE_UNLOAD   ] I: Unload finished for segment #1 of segmented table
 'testdatabase'.'testtable' (Id = 1). 2 rows sent.

[TARGET_LOAD     ] I: Load finished for segment #1 of segmented table
 'testdatabase'.'testtable' (Id = 1). 1 rows received. 0 rows skipped. Volume
 transfered 480.
```

```
[TASK_MANAGER    ] I: Load finished for segment #1 of table
 'testdatabase'.'testtable' (Id = 1) by subtask 1. 2 records transferred.
```

Currently, AWS DMS supports the following MongoDB data types as a segment key column:

- Double
- String
- ObjectId
- 32 bit integer
- 64 bit integer

Migrating multiple databases when using MongoDB as a source for AWS DMS

AWS DMS versions 3.4.5 and higher support migrating multiple databases in a single task for all supported MongoDB versions. If you want to migrate multiple databases, take these steps:

1. When you create the MongoDB source endpoint, do one of the following:
   - On the DMS console's **Create endpoint** page, make sure that **Database name** is empty under **Endpoint configuration**.
   - Using the AWS CLI `CreateEndpoint` command, assign an empty string value to the `DatabaseName` parameter in `MongoDBSettings`.
2. For each database that you want to migrate from a MongoDB source, specify the database name as a schema name in the table mapping for the task. You can do this using either the guided input in the console or directly in JSON. For more information on the guided input, see Specifying table selection and transformations rules from the console. For more information on the JSON, see Selection rules and actions.

For example, you might specify the JSON following to migrate three MongoDB databases.

Example Migrate all tables in a schema

The JSON following migrates all tables from the `Customers`, `Orders`, and `Suppliers` databases in your source endpoint to your target endpoint.

```
{
```

```
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "Customers",
                "table-name": "%"
            },
            "rule-action": "include",
            "filters": []
        },
        {
            "rule-type": "selection",
            "rule-id": "2",
            "rule-name": "2",
            "object-locator": {
                "schema-name": "Orders",
                "table-name": "%"
            },
            "rule-action": "include",
            "filters": []
        },
        {
            "rule-type": "selection",
            "rule-id": "3",
            "rule-name": "3",
            "object-locator": {
                "schema-name": "Inventory",
                "table-name": "%"
            },
            "rule-action": "include",
            "filters": []
        }
    ]
}
```

Limitations when using MongoDB as a source for AWS DMS

The following are limitations when using MongoDB as a source for AWS DMS:

- In table mode, the documents in a collection must be consistent in the data type that they use for the value in the same field. For example, if a document in a collection includes `'{ a:`

`{ b:value ... }'`, all documents in the collection that reference the *value* of the `a.b` field must use the same data type for *value*, wherever it appears in the collection.

- 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, make sure to specify the prefixed column. For example, you specify `${oid__id}` and not `${_id}`, or `${array__addresses}` and not `${_addresses}`.
- Collection names and key names can't include the dollar symbol ($).
- AWS DMS doesn't support collections containing the same field with different case (upper, lower) in table mode with a RDBMS target. For example, AWS DMS does not support having two collections named `Field1` and `field1`.
- Table mode and document mode have the limitations described preceding.
- Migrating in parallel using autosegmentation has the limitations described preceding.
- Source filters aren't supported for MongoDB.
- AWS DMS doesn't support documents where the nesting level is greater than 97.
- AWS DMS requires UTF-8 encoded source data when migrating to non-DocumentDB targets. For sources with non-UTF-8 characters, convert them to UTF-8 before migration or migrate to Amazon DocumentDB instead.
- AWS DMS doesn't support the following features of MongoDB version 5.0:
  - Live resharding
  - Client-Side Field Level Encryption (CSFLE)
  - Timeseries collection migration

    > **Note**
    >
    > A timeseries collection migrated in the full-load phase will be converted to a normal collection in Amazon DocumentDB, because DocumentDB doesn't support timeseries collections.

Endpoint configuration settings when using MongoDB as a source for AWS DMS

When you set up your MongoDB source endpoint, you can specify multiple endpoint configuration settings using the AWS DMS console.

The following table describes the configuration settings available when using MongoDB databases as an AWS DMS source.

Setting (attribute)	Valid values	Default value and description
Authentication mode	`"none"` `"password"`	The value `"password"` prompts for a user name and password. When `"none"` is specified, user name and password parameters aren't used.
Authentication source	A valid MongoDB database name.	The name of the MongoDB database that you want to use to validate your credentials for authentication. The default value is `"admin"`.
Authentication mechanism	`"default"` `"mongodb_cr"` `"scram_sha_1"`	The authentication mechanism. The value `"default"` is `"scram_sha_1"` . This setting isn't used when `authType` is set to `"no"`.
Metadata mode	Document and table	Chooses document mode or table mode.
Number of documents to scan (`docsToInvestigate`)	A positive integer greater than `0`.	Use this option in table mode only to define the target table definition.
_id as a separate column	Check mark in box	Optional check mark box that creates a second column named `_id` that acts as the primary key.
`ExtractDocID`	`true`	`false` – Use this attribute when `NestingLevel` is set to `"none"`.

Setting (attribute)	Valid values	Default value and description
	`false`	When using CDC with sources that produce multi-document transactions, the `ExtractDocId` parameter **must be** set to `true`. If this parameter is not enabled, the AWS DMS task will fail when it encounters a multi-document transaction.
`socketTimeoutMS`	An integer greater than or equal to 0. Extra Connection Attribute (ECA) only.	This setting is in units of milliseconds and configures the connection timeout for MongoDB clients. If the value is less than or equal to zero, then the MongoDB client default is used.
`UseUpdateLookUp`	`true` `false`	When true, during CDC update events, AWS DMS copies over the entire updated document to the target. When set to false, AWS DMS uses the MongoDB update command to only update modified fields in the document on the target.
`ReplicateShardCollections`	`true` `false`	When true, AWS DMS replicates data to shard collections. AWS DMS only uses this setting if the target endpoint is a DocumentDB elastic cluster. When this setting is true, note the following: • You must set `TargetTablePrepMode` to `nothing`. • AWS DMS automatically sets `useUpdateLookup` to `false`.

Setting (attribute)	Valid values	Default value and description
useTransactionVerification	true false	When false, disables the verification between the change stream and oplogs. **Note** You can miss operations if discrepancies between change streams and oplog entries occur, as the default DMS behavior is to fail the task in such scenarios. Default: true.
useOplog	true false	When true, enables DMS task to read directly from 'oplog' rather than using the change stream. Default: false.

If you choose **Document** as **Metadata mode**, different options are available.

If the target endpoint is DocumentDB, make sure to run the migration in **Document mode** Also, modify your source endpoint and select the option **_id as separate column**. This is a mandatory prerequisite if your source MongoDB workload involves transactions.

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.

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 DocumentDB (with MongoDB compatibility) as a source for AWS DMS

For information about versions of Amazon DocumentDB (with MongoDB compatibility) that AWS DMS supports as a source, see Sources for AWS DMS.

Using Amazon DocumentDB as a source, you can migrate data from one Amazon DocumentDB cluster to another Amazon DocumentDB cluster. You can also migrate data from an Amazon DocumentDB cluster to one of the other target endpoints supported by AWS DMS.

If you are new to Amazon DocumentDB, be aware of the following important concepts for Amazon DocumentDB databases:

- A record in Amazon DocumentDB is a *document*, 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 Amazon DocumentDB is a group of documents, and is roughly equivalent to a relational database table.
- A *database* in Amazon DocumentDB is a set of collections, and is roughly equivalent to a schema in a relational database.

AWS DMS supports two migration modes when using Amazon DocumentDB as a source, document mode and table mode. You specify the migration mode when you create the Amazon DocumentDB source endpoint in the AWS DMS console, using either the **Metadata mode** option or the extra connection attribute `nestingLevel`. Following, you can find an explanation how the choice of migration mode affects the resulting format of the target data.

Document mode

In *document mode,* the JSON document is migrated as is. That means the document data is consolidated into one of two items. When you use a relational database as a target, the data is a single column named `_doc` in a target table. When you use a nonrelational database as a target, the data is a single JSON document. Document mode is the default mode, which we recommend when migrating to an Amazon DocumentDB target.

For example, consider the following documents in a Amazon DocumentDB 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 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 the target.

When using CDC with sources that produce [multi-document transactions](), the `ExtractDocId` parameter **must be** set to `true`. If this parameter is not enabled, the AWS DMS task will fail when it encounters a multi-document transaction.

> **Note**
>
> If you add a new collection to the source database, AWS DMS creates a new target table for the collection and replicates any documents.

Table mode

In *table mode,* AWS DMS transforms each top-level field in a Amazon DocumentDB 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 Amazon DocumentDB 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
5a94815f4 0bd44d1b02bdfe0	1	2	3
5a94815f4 0bd44d1b02bdfe1	4	5	6

Nested values are flattened into a column containing dot-separated key names. The column is named using 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 Amazon DocumentDB 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 Amazon DocumentDB source endpoint using the console, you can specify the number of documents to scan. The default value is 1,000 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.
- In the Change Data Capture(CDC) phase, AWS DMS doesn't support renaming a collection.

Topics

Setting permissions to use Amazon DocumentDB as a source

When using Amazon DocumentDB source for an AWS DMS migration, you can create a user account with root privileges. Or you can create a user with permissions only for the database to be migrated.

The following code creates a user as the root account.

```
use admin
db.createUser(
  {
    user: "root",
    pwd: "password",
    roles: [ { role: "root", db: "admin" } ]
  })
```

For Amazon DocumentDB 3.6, the code following creates a user with minimal privileges on the database to be migrated.

```
use db_name
db.createUser(
    {
        user: "dms-user",
        pwd: "password",
        roles: [{ role: "read", db: "db_name" }]
    }
)
```

For Amazon DocumentDB 4.0 and higher, AWS DMS uses a deployment-wide change stream. Here, the code following creates a user with minimal privileges.

```
db.createUser(
{
    user: "dms-user",
    pwd: "password",
    roles: [ { role: "readAnyDatabase", db: "admin" }]
})
```

Configuring CDC for an Amazon DocumentDB cluster

To use ongoing replication or CDC with Amazon DocumentDB, AWS DMS requires access to the Amazon DocumentDB cluster's change streams. For a description of the time-ordered sequence of update events in your cluster's collections and databases, see Using change streams in the *Amazon DocumentDB Developer Guide*.

Authenticate to your Amazon DocumentDB cluster using the MongoDB shell. Then run the following command to enable change streams.

```
db.adminCommand({modifyChangeStreams: 1,
    database: "DB_NAME",
    collection: "",
    enable: true});
```

This approach enables the change stream for all collections in your database. After change streams are enabled, you can create a migration task that migrates existing data and at the same time replicates ongoing changes. AWS DMS continues to capture and apply changes even after the bulk data is loaded. Eventually, the source and target databases synchronize, minimizing downtime for a migration.

> **Note**
>
> AWS DMS uses the operations log (oplog) to capture changes during ongoing replication. If Amazon DocumentDB flushes out the records from the oplog before AWS DMS reads them, your tasks will fail. We recommend sizing the oplog to retain changes for at least 24 hours.

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 the `rds-combined-ca-bundle.pem` file, you can import the public key that it contains into AWS DMS. The following steps describe how to do so.

To import your public key using the AWS DMS console

1. Sign in to the AWS Management Console and choose AWS DMS.
2. In the navigation pane, choose **Certificates**.

3. Choose **Import certificate**. The **Import new CA certificate** page appears.
4. In the **Certificate configuration** section, do one of the following:
   - For **Certificate identifier**, enter a unique name for the certificate, such as `docdb-cert`.
   - Choose **Choose file**, navigate to the location where you saved the `rds-combined-ca-bundle.pem` file, and select it.
5. Choose **Add new CA certificate**.

The AWS CLI following example uses the AWS DMS `import-certificate` command to import the public key `rds-combined-ca-bundle.pem` file.

```
aws dms import-certificate \
    --certificate-identifier docdb-cert \
    --certificate-pem file://./rds-combined-ca-bundle.pem
```

Creating an Amazon DocumentDB source endpoint

You can create an Amazon DocumentDB source endpoint using either the console or AWS CLI. Use the procedure following with the console.

To configure an Amazon DocumentDB source endpoint using the AWS DMS console

1. Sign in to the AWS Management Console and choose AWS DMS.
2. Choose **Endpoints** from the navigation pane, then choose **Create Endpoint**.
3. For **Endpoint identifier**, provide a name that helps you easily identify it, such as `docdb-source`.
4. For **Source engine**, choose **Amazon DocumentDB (with MongoDB compatibility)**.
5. For **Server name**, enter the name of the server where your Amazon DocumentDB database endpoint resides. For example, you might enter the public DNS name of your Amazon EC2 instance, such as `democluster.cluster-cjf6q8nxfefi.us-east-2.docdb.amazonaws.com`.
6. For **Port**, enter 27017.
7. For **SSL mode**, choose **verify-full**. If you have disabled SSL on your Amazon DocumentDB cluster, you can skip this step.

8. For **CA certificate**, choose the Amazon DocumentDB certificate, `rds-combined-ca-bundle.pem`. For instructions on adding this certificate, see [Connecting to Amazon DocumentDB using TLS](#).

9. For **Database name**, enter the name of the database to be migrated.

Use the following procedure with the CLI.

To configure an Amazon DocumentDB source endpoint using the AWS CLI

- Run the following AWS DMS `create-endpoint` command to configure an Amazon DocumentDB source endpoint, replacing placeholders with your own values.

```
aws dms create-endpoint \
           --endpoint-identifier a_memorable_name \
           --endpoint-type source \
           --engine-name docdb \
           --username value \
           --password value \
           --server-name servername_where_database_endpoint_resides \
           --port 27017 \
           --database-name name_of_endpoint_database
```

Segmenting Amazon DocumentDB collections and migrating in parallel

To improve performance of a migration task, Amazon DocumentDB source endpoints support two options of the parallel full load feature in table mapping. In other words, you can migrate a collection in parallel by using either the autosegmentation or the range segmentation options of table mapping for a parallel full load in JSON settings. The auto-segmenting options allow you to specify the criteria for AWS DMS to automatically segment your source for migration in each thread. The range segmentation options allow you to tell AWS DMS the specific range of each segment for DMS to migrate in each thread. For more information on these settings, see [Table and collection settings rules and operations](#).

Migrating an Amazon DocumentDB database in parallel using autosegmentation ranges

You can migrate your documents in parallel by specifying the criteria for AWS DMS to automatically partition (segment) your data for each thread, especially the number of documents to migrate per thread. Using this approach, AWS DMS attempts to optimize segment boundaries for maximum performance per thread.

You can specify the segmentation criteria using the table-settings options following in table-mapping:

Table-settings option	Description
`"type"`	(Required) Set to `"partitions-auto"` for Amazon DocumentDB as a source.
`"number-of-partitions"`	(Optional) Total number of partitions (segments) used for migration. The default is 16.
`"collection-count-from-metadata"`	(Optional) If set to `true`, AWS DMS uses an estimated collection count for determining the number of partitions. If set to `false`, AWS DMS uses the actual collection count. The default is `true`.
`"max-records-skip-per-page"`	(Optional) The number of records to skip at once when determining the boundaries for each partition. AWS DMS uses a paginated skip approach to determine the minimum boundary for a partition. The default is 10000. Setting a relatively large value might result in curser timeouts and task failures. Setting a relatively low value results in more operations per page and a slower full load.
`"batch-size"`	(Optional) Limits the number of documents returned in one batch. Each batch requires a round trip to the server. If the batch size is zero (0), the cursor uses the server-defined maximum batch size. The default is 0.

The example following shows a table mapping for autosegmentation.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "admin",
                "table-name": "departments"
            },
            "rule-action": "include",
            "filters": []
        },
        {
            "rule-type": "table-settings",
            "rule-id": "2",
            "rule-name": "2",
            "object-locator": {
                "schema-name": "admin",
                "table-name": "departments"
            },
            "parallel-load": {
                "type": "partitions-auto",
                "number-of-partitions": 5,
                "collection-count-from-metadata": "true",
                "max-records-skip-per-page": 1000000,
                "batch-size": 50000
            }
        }
    ]
}
```

Auto-segmentation has the limitation following. The migration for each segment fetches the collection count and the minimum `_id` for the collection separately. It then uses a paginated skip to calculate the minimum boundary for that segment. Therefore, ensure that the minimum `_id` value for each collection remains constant until all the segment boundaries in the collection are calculated. If you change the minimum `_id` value for a collection during its segment boundary calculation, this might cause data loss or duplicate row errors.

Migrating an Amazon DocumentDB database in parallel using specific segment ranges

The following example shows an Amazon DocumentDB collection that has seven items, and `_id` as the primary key.

Key	Value	Type
(1) ObjectId("5f805c74873173399a278d78")	{ 3 fields }	Object
_id	ObjectId("5f805c74873173399a278d78")	ObjectId
num	1	Int32
name	a	String
(2) ObjectId("5f805c97873173399a278d79")	{ 3 fields }	Object
_id	ObjectId("5f805c97873173399a278d79")	ObjectId
num	2	Int32
name	b	String
(3) ObjectId("5f805cb0873173399a278d7a")	{ 3 fields }	Object
_id	ObjectId("5f805cb0873173399a278d7a")	ObjectId
num	3	Int32
name	c	String
(4) ObjectId("5f805cbb873173399a278d7b")	{ 3 fields }	Object
_id	ObjectId("5f805cbb873173399a278d7b")	ObjectId
num	4	Int32
name	d	String
(5) ObjectId("5f805cc5873173399a278d7c")	{ 3 fields }	Object
_id	ObjectId("5f805cc5873173399a278d7c")	ObjectId
num	5	Int32
name	e	String
(6) ObjectId("5f805cd0873173399a278d7d")	{ 3 fields }	Object
_id	ObjectId("5f805cd0873173399a278d7d")	ObjectId
num	6	Int32
name	f	String
(7) ObjectId("5f805cdd873173399a278d7e")	{ 3 fields }	Object
_id	ObjectId("5f805cdd873173399a278d7e")	ObjectId
num	7	Int32
name	g	String

To split the collection into three segments and migrate in parallel, you can add table mapping rules to your migration task as shown in the following JSON example.

```
{ // Task table mappings:
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
```

```
        "schema-name": "testdatabase",
        "table-name": "testtable"
      },
      "rule-action": "include"
    }, // "selection" :"rule-type"
    {
      "rule-type": "table-settings",
      "rule-id": "2",
      "rule-name": "2",
      "object-locator": {
        "schema-name": "testdatabase",
        "table-name": "testtable"
      },
      "parallel-load": {
        "type": "ranges",
        "columns": [
           "_id",
           "num"
        ],
        "boundaries": [
          // First segment selects documents with _id less-than-or-equal-to
 5f805c97873173399a278d79
          // and num less-than-or-equal-to 2.
          [
             "5f805c97873173399a278d79",
             "2"
          ],
          // Second segment selects documents with _id > 5f805c97873173399a278d79 and
          // _id less-than-or-equal-to 5f805cc5873173399a278d7c and
          // num > 2 and num less-than-or-equal-to 5.
          [
             "5f805cc5873173399a278d7c",
             "5"
          ]
          // Third segment is implied and selects documents with _id >
 5f805cc5873173399a278d7c.
        ] // :"boundaries"
      } // :"parallel-load"
    } // "table-settings" :"rule-type"
  ] // :"rules"
} // :Task table mappings
```

That table mapping definition splits the source collection into three segments and migrates in parallel. The following are the segmentation boundaries.

```
Data with _id less-than-or-equal-to "5f805c97873173399a278d79" and num less-than-or-
equal-to 2 (2 records)
Data with _id less-than-or-equal-to "5f805cc5873173399a278d7c" and num less-than-or-
equal-to 5 and not in (_id less-than-or-equal-to  "5f805c97873173399a278d79" and num
 less-than-or-equal-to 2) (3 records)
Data not in (_id less-than-or-equal-to "5f805cc5873173399a278d7c" and num less-than-or-
equal-to 5) (2 records)
```

After the migration task is complete, you can verify from the task logs that the tables loaded in parallel, as shown in the following example. You can also verify the Amazon DocumentDB `find` clause used to unload each segment from the source table.

```
[TASK_MANAGER    ] I:  Start loading segment #1 of 3 of table
 'testdatabase'.'testtable' (Id = 1) by subtask 1. Start load timestamp
 0005B191D638FE86  (replicationtask_util.c:752)

[SOURCE_UNLOAD   ] I:  Range Segmentation filter for Segment #0 is initialized.
 (mongodb_unload.c:157)

[SOURCE_UNLOAD   ] I:  Range Segmentation filter for Segment #0 is: { "_id" :
 { "$lte" : { "$oid" : "5f805c97873173399a278d79" } }, "num" : { "$lte" :
 { "$numberInt" : "2" } } }  (mongodb_unload.c:328)

[SOURCE_UNLOAD   ] I: Unload finished for segment #1 of segmented table
 'testdatabase'.'testtable' (Id = 1). 2 rows sent.

[TASK_MANAGER    ] I: Start loading segment #1 of 3 of table
 'testdatabase'.'testtable' (Id = 1) by subtask 1. Start load timestamp
 0005B191D638FE86 (replicationtask_util.c:752)

[SOURCE_UNLOAD   ] I: Range Segmentation filter for Segment #0 is initialized.
 (mongodb_unload.c:157)

[SOURCE_UNLOAD   ] I: Range Segmentation filter for Segment #0 is: { "_id" : { "$lte" :
 { "$oid" : "5f805c97873173399a278d79" } }, "num" : { "$lte" : { "$numberInt" :
 "2" } } } (mongodb_unload.c:328)
```

```
[SOURCE_UNLOAD   ] I: Unload finished for segment #1 of segmented table
 'testdatabase'.'testtable' (Id = 1). 2 rows sent.

[TARGET_LOAD     ] I: Load finished for segment #1 of segmented table
 'testdatabase'.'testtable' (Id = 1). 1 rows received. 0 rows skipped. Volume
 transfered 480.

[TASK_MANAGER    ] I: Load finished for segment #1 of table
 'testdatabase'.'testtable' (Id = 1) by subtask 1. 2 records transferred.
```

Currently, AWS DMS supports the following Amazon DocumentDB data types as a segment key column:

- Double
- String
- ObjectId
- 32 bit integer
- 64 bit integer

Migrating multiple databases when using Amazon DocumentDB as a source for AWS DMS

AWS DMS versions 3.4.5 and higher support migrating multiple databases in a single task only for Amazon DocumentDB versions 4.0 and higher. If you want to migrate multiple databases, do the following:

1. When you create the Amazon DocumentDB source endpoint:
   - In the AWS Management Console for AWS DMS, leave **Database name** empty under **Endpoint configuration** on the **Create endpoint** page.
   - In the AWS Command Line Interface (AWS CLI), assign an empty string value to the **DatabaseName** parameter in **DocumentDBSettings** that you specify for the **CreateEndpoint** action.
2. For each database that you want to migrate from this Amazon DocumentDB source endpoint, specify the name of each database as the name of a schema in the table-mapping for the task using either the guided input in the console or directly in JSON. For more information on the

guided input, see the description of the Specifying table selection and transformations rules from the console. For more information on the JSON, see Selection rules and actions.

For example, you might specify the JSON following to migrate three Amazon DocumentDB databases.

Example Migrate all tables in a schema

The JSON following migrates all tables from the `Customers`, `Orders`, and `Suppliers` databases in your source enpoint to your target endpoint.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "Customers",
                "table-name": "%"
            },
            "object-locator": {
                "schema-name": "Orders",
                "table-name": "%"
            },
            "object-locator": {
                "schema-name": "Inventory",
                "table-name": "%"
            },
            "rule-action": "include"
        }
    ]
}
```

Limitations when using Amazon DocumentDB as a source for AWS DMS

The following are limitations when using Amazon DocumentDB 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, make sure to specify the prefixed column. For example, specify `${oid__id}` and not `${_id}`, or `${array__addresses}` and not `${_addresses}`.

- Collection names and key names can't include the dollar symbol ($).
- Table mode and document mode have the limitations discussed preceding.
- Migrating in parallel using autosegmentation has the limitations described preceding.
- An Amazon DocumentDB (MongoDB compatible) source doesn’t support using a specific timestamp as a start position for change data capture (CDC). An ongoing replication task starts capturing changes regardless of the timestamp.
- AWS DMS doesn't support documents where the nesting level is greater than 97 for AWS DMS versions lower than 3.5.2.
- Source filters aren't supported for DocumentDB.
- AWS DMS doesn’t support CDC (change data capture) replication for DocumentDB as a source in elastic cluster mode.

Using endpoint settings with Amazon DocumentDB as a source

You can use endpoint settings to configure your Amazon DocumentDB source database similar to using extra connection attributes. You specify the settings when you create the source endpoint using the AWS DMS console, or by using the `create-endpoint` command in the [AWS CLI](), with the `--doc-db-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following table shows the endpoint settings that you can use with Amazon DocumentDB as a source.

Attribute name	Valid values	Default value and description
`NestingLevel`	`"none"` `"one"`	`"none"` – Specify `"none"` to use document mode. Specify `"one"` to use table mode.
`ExtractDocID`	`true` `false`	`false` – Use this attribute when `NestingLevel` is set to `"none"`.

Attribute name	Valid values	Default value and description
		When using CDC with sources that produce multi-document transactions, the `ExtractDocId` parameter **must be** set to `true`. If this parameter is not enabled, the AWS DMS task will fail when it encounters a multi-document transaction.
`DocsToInvestigate`	A positive integer greater than `0`.	`1000` – Use this attribute when `NestingLevel` is set to `"one"`.
`ReplicateShardCollections`	`true` `false`	When true, AWS DMS replicates data to shard collections. AWS DMS only uses this setting if the target endpoint is a DocumentDB elastic cluster. When this setting is true, note the following: • You must set `TargetTablePrepMode` to `nothing`. • AWS DMS automatically sets `useUpdateLookup` to `false`.

Source data types for Amazon DocumentDB

In the following table, you can find the Amazon DocumentDB source data types that are supported when using AWS DMS. You can also find the default mapping from AWS DMS data types in this table. For more information about 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.

Amazon DocumentDB data types	AWS DMS data types
Boolean	Bool

Amazon DocumentDB data types	AWS DMS data types
Binary	BLOB
Date	Date
Timestamp	Date
Int	INT4
Long	INT8
Double	REAL8
String (UTF-8)	CLOB
Array	CLOB
OID	String

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 can be in the following formats:

- Comma-separated value (.csv)
- Parquet (DMS version 3.5.3 and later). For information about using Parquet-format files, see Using Parquet-format files in Amazon S3 as a source for AWS DMS.

For source data files in comma-separated value (.csv) format, name them using the following naming convention. 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 `amzn-s3-demo-bucket`, at the following Amazon S3 path.

```
s3://amzn-s3-demo-bucket/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://amzn-s3-demo-bucket/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 [Endpoint settings for Amazon S3 as a source for AWS DMS](#).

You can specify a bucket owner and prevent sniping by using the `ExpectedBucketOwner` Amazon S3 endpoint setting, as shown following. Then, when you make a request to test a connection or perform a migration, S3 checks the account ID of the bucket owner against the specified parameter.

```
--s3-settings='{"ExpectedBucketOwner": "AWS_Account_ID"}'
```

Topics

- Source data types for Amazon S3
- Using Parquet-format files in Amazon S3 as a source for AWS DMS

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": [
                {
                    "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, its 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.
  - `ColumnLength` – the number of bytes in this column. Maximum column length is limited to2147483647 Bytes (2,047 MegaBytes) since an S3 source doesn't support FULL LOB mode. `ColumnLength` is valid for the following data types:
    - BYTE

  - STRING
- `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`).
- `ColumnDateFormat` – the input date format for a column with DATE, TIME, and DATETIME types, and used to parse a data string into a date object. Possible values include:

```
- YYYY-MM-dd HH:mm:ss
- YYYY-MM-dd HH:mm:ss.F
- YYYY/MM/dd HH:mm:ss
- YYYY/MM/dd HH:mm:ss.F
- MM/dd/YYYY HH:mm:ss
- MM/dd/YYYY HH:mm:ss.F
- YYYYMMdd HH:mm:ss
- YYYYMMdd HH:mm:ss.F
```

- `TableColumnsTotal` – the total number of columns. This number must match the number of elements in the `TableColumns` array.

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 DMS versions prior to 3.4.2, 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.

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"
    }
...
```

For a column of the DATETIME type with data that contains fractional seconds, specify the scale. *Scale* is the number of digits for the fractional seconds, and can range from 0 to 9. You use the `ColumnScale` element for this, as shown following.

```
...
{
      "ColumnName": "HireDate",
      "ColumnType": "DATETIME",
      "ColumnScale": "3"
}
...
```

If you don't specify otherwise, AWS DMS assumes `ColumnScale` is zero and truncates the fractional seconds.

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, 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://amzn-s3-demo-bucket/changedata
```

If you set `cdcPath` to *`changedata`* and `bucketFolder` to *`myFolder`*, then AWS DMS reads the CDC files at the following path.

```
s3://amzn-s3-demo-bucket/myFolder/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. For AWS DMS version 3.4.7 and higher, DMS must access the source bucket through a VPC endpoint or a public route. For information about VPC endpoints, see Configuring VPC endpoints for AWS DMS.

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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket*/*"
            ]
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-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 if versioning is enabled on the Amazon S3 bucket.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket*/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket*"
            ]
```

```
        }
    ]
}
```

Limitations when using Amazon S3 as a source for AWS DMS

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

- Don't enable versioning for S3. If you need S3 versioning, use lifecycle policies to actively delete old versions. Otherwise, you might encounter endpoint test connection failures because of an S3 `list-object` call timeout. To create a lifecycle policy for an S3 bucket, see [Managing your storage lifecycle](). To delete a version of an S3 object, see [Deleting object versions from a versioning-enabled bucket]().
- A VPC-enabled (gateway VPC) S3 bucket is supported in versions 3.4.7 and higher.
- MySQL converts the `time` datatype to `string`. To see `time` data type values in MySQL, define the column in the target table as `string`, and set the task's **Target table preparation mode** setting to **Truncate**.
- AWS DMS uses the `BYTE` data type internally for data in both `BYTE` and `BYTES` data types.
- S3 source endpoints do not support the DMS table reload feature.
- AWS DMS doesn't support Full LOB mode with Amazon S3 as a Source.

The following limitations apply when using Parquet-format files in Amazon S3 as a source:

- Dates in `MMYYYYDD`, or `DDMMYYYY` are not supported for the S3 Parquet Source date-partitioning feature.

Endpoint settings for Amazon S3 as a source for AWS DMS

You can use endpoint settings to configure your Amazon S3 source database similar to using extra connection attributes. You specify the settings when you create the source endpoint using the AWS DMS console, or by using the `create-endpoint` command in the [AWS CLI](), with the `--s3-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

> **Note**
>
> AWS DMS defaults to a secure connection to the Amazon S3 endpoint without requiring to specify SSL mode or certificate.

The following table shows the endpoint settings that you can use with Amazon S3 as a source.

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 `s3://amzn-s3-demo-bucket /bucketFolder /schemaName /tableName /` and `s3://amzn-s3-demo-bucket /bucketFolder /` respectively. If this attribute isn't specified, then the path used is `schemaName /tableName /`. `'{"BucketFolder": " sourceData "}'`
`BucketName`	The name of the S3 bucket. `'{"BucketName": " amzn-s3-demo-bucket "}'`
`CdcPath`	The location of CDC files. This attribute is required if a task captures change data; otherwise, it's optional. If `CdcPath` is present, then AWS DMS reads CDC files from this path and replicates the data changes to the target endpoint. For more information, see Using CDC with Amazon S3 as a source for AWS DMS. `'{"CdcPath": " changeData "}'`
`CsvDelimiter`	The delimiter used to separate columns in the source files. The default is a comma. An example follows. `'{"CsvDelimiter": ","}'`
`CsvNullValue`	A user-defined string that AWS DMS treats as null when reading from the source. The default is an empty string. If you do not set this parameter, AWS DMS treats an empty string as a null value. If you set

Option	Description
	this parameter to a string such as "\N", AWS DMS treats this string as the null value, and treats empty strings as an empty string value.
`CsvRowDelimiter`	The delimiter used to separate rows in the source files. The default is a newline (\n). `'{"CsvRowDelimiter": "\n"}'`
`DataFormat`	Set this value to `Parquet` to read data in Parquet format. `'{"DataFormat": "Parquet"}'`
`IgnoreHeaderRows`	When this value is set to 1, AWS DMS ignores the first row header in a .csv file. A value of 1 enables the feature, a value of 0 disables the feature. The default is 0. `'{"IgnoreHeaderRows": 1}'`
`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` or `n`, 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` `'{"Rfc4180": false}'`

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.

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.

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.
- DATE
- TIME
- DATETIME – For more information and an example, see the DATETIME type example in Defining external tables for Amazon S3 as a source for AWS DMS.
- INT1
- INT2
- INT4
- INT8
- NUMERIC – Requires `ColumnPrecision` and `ColumnScale`. AWS DMS supports the following maximum values:
  - **ColumnPrecision: 38**
  - **ColumnScale: 31**

  For more information and an example, see the NUMERIC type example in Defining external tables for Amazon S3 as a source for AWS DMS.
- REAL4
- REAL8
- STRING – Requires `ColumnLength`. For more information, see Defining external tables for Amazon S3 as a source for AWS DMS.
- UINT1
- UINT2
- UINT4
- UINT8
- BLOB

- CLOB
- BOOLEAN

Using Parquet-format files in Amazon S3 as a source for AWS DMS

In AWS DMS version 3.5.3 and later, you can use Parquet-format files in an S3 bucket as a source for both Full-Load or CDC replication.

DMS only supports Parquet format files as a source that DMS generates by migrating data to an S3 target endpoint. File names must be in the supported format, or DMS won't include them in the migration.

For source data files in Parquet format, they must be in the following folder and naming convention.

```
schema/table1/LOAD00001.parquet
schema/table2/LOAD00002.parquet
schema/table2/LOAD00003.parquet
```

For source data files for CDC data in Parquet format, name and store them using the following folder and naming convention.

```
schema/table/20230405-094615814.parquet
schema/table/20230405-094615853.parquet
schema/table/20230405-094615922.parquet
```

To access files in Parquet format, set the following endpoint settings:

- Set `DataFormat` to `Parquet`.
- Do not set the `cdcPath` setting. Make sure that you create your Parquet-format files in the specified schema/ table folders.

For more information about settings for S3 endpoints, see S3Settings in the *AWS Database Migration Service API Reference*.

Supported datatypes for Parquet-format files

AWS DMS supports the following source and target data types when migrating data from Parquet-format files. Ensure that your target table has columns of the correct data types before migrating.

Source data type	Target data type
BYTE	BINARY
DATE	DATE32
TIME	TIME32
DATETIME	TIMESTAMP
INT1	INT8
INT2	INT16
INT4	INT32
INT8	INT64
NUMERIC	DECIMAL
REAL4	FLOAT
REAL8	DOUBLE
STRING	STRING
UINT1	UINT8
UINT2	UINT16
UINT4	UINT32
UINT8	UINT
WSTRING	STRING
BLOB	BINARY
NCLOB	STRING
CLOB	STRING

Source data type	Target data type
BOOLEAN	BOOL

Using IBM Db2 for Linux, Unix, Windows, and Amazon RDS database (Db2 LUW) as a source for AWS DMS

You can migrate data from an IBM Db2 for Linux, Unix, Windows, and Amazon RDS (Db2 LUW) database to any supported target database using AWS Database Migration Service (AWS DMS).

For information about versions of Db2 on Linux, Unix, Windows, and RDS that AWS DMS supports as a source, see Sources for AWS DMS.

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.

When AWS DMS reads data from an IBM Db2 source database, it uses the default isolation level CURSOR STABILITY (CS) for Db2 version 9.7 and above. For more information, see IBM Db2 for Linux, UNIX and Windows documentation.

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`.

  If your database is recoverable, then AWS DMS can access the Db2 `ARCHIVE LOG` if needed.
- Ensure that the DB2 transaction logs are available, with a sufficient retention period to be processed by AWS DMS.
- DB2 requires `SYSADM` or `DBADM` authorization to extract transaction log records. Grant the user account the following permissions:
  - `SYSADM` or `DBADM`
  - `DATAACCESS`

> **Note**
>
> For full-load only tasks, the DMS user account needs DATAACCESS permission.

- When using IBM DB2 for LUW version 9.7 as a source, set the extra connection attribute (ECA), `CurrentLsn` as follows:

  `CurrentLsn=LSN` where *LSN* specifies a log sequence number (LSN) where you want the replication to start. Or, `CurrentLsn=scan`.
- When using Amazon RDS for Db2 LUW as a source, ensure that the archive logs are available to AWS DMS. Because AWS-managed Db2 databases purge the archive 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:

```
db2 "call rdsadmin.set_archive_log_retention( ?, 'TESTDB', '24')"
```

  For more information about Amazon RDS for Db2 LUW procedures, see the [Amazon RDS for Db2 stored procedure reference](#) in the *Amazon Relational Database Service User Guide*.
- Grant the following privileges if you use DB2 specific premigration assessments:

```
GRANT CONNECT ON DATABASE TO USER <DMS_USER>;
GRANT SELECT ON SYSIBM.SYSDUMMY1 TO USER <DMS_USER>;
GRANT SELECT ON SYSIBMADM.ENV_INST_INFO TO USER <DMS_USER>;
GRANT SELECT ON SYSIBMADM.DBCFG TO USER <DMS_USER>;
GRANT SELECT ON SYSCAT.SCHEMATA TO USER <DMS_USER>;
GRANT SELECT ON SYSCAT.COLUMNS TO USER <DMS_USER>;
GRANT SELECT ON SYSCAT.TABLES TO USER <DMS_USER>;
GRANT EXECUTE ON FUNCTION SYSPROC.AUTH_LIST_AUTHORITIES_FOR_AUTHID TO <DMS_USER>;
GRANT EXECUTE ON PACKAGE NULLID.SYSSH200 TO USER <DMS_USER>;
```

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. For example, you can create a Full Load migration task with any one of the nodes in the cluster, then create separate tasks from each node.

AWS DMS doesn't support the `BOOLEAN` data type in your source Db2 LUW database.

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.
- 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 for columns with data types like VARCHAR and VARGRAPHIC. To work around this limitation, move the table to a table space with a higher page size. For more information, see [What can I do if I want to change the pagesize of DB2 tablespaces](#).
- For ongoing replication, DMS doesn't support migrating data loaded at the page level by the DB2 LOAD utility. Instead, use the IMPORT utility which uses SQL inserts. For more information, see [differences between the import and load utilities](#).
- While a replication task is running, DMS captures CREATE TABLE DDLs only if the tables were created with the DATA CAPTURE CHANGE attribute.
- DMS has the following limitations when using the Db2 Database Partition Feature (DPF):
  - DMS can't coordinate transactions across Db2 nodes in a DPF environment. This is due to constraints within the IBM DB2READLOG API interface. In DPF, transactions may span multiple Db2 nodes, depending upon how DB2 partitions the data. As a result, your DMS solution must capture transactions from each Db2 node independently.

- DMS can capture local transactions from each Db2 node in the DPF cluster by setting `connectNode` to `1` on multiple DMS source endpoints. This configuration corresponds to logical node numbers defined in the DB2 server configuration file `db2nodes.cfg`.
- Local transactions on individual Db2 nodes may be parts of a larger, global transaction. DMS applies each local transaction independently on the target, without coordination with transactions on other Db2 nodes. This independent processing can lead to complications, especially when rows are moved between partitions.
- When DMS replicates from multiple Db2 nodes, there is no assurance of the correct order of operations on the target, because DMS applies operations independently for each Db2 node. You must ensure that capturing local transactions independently from each Db2 node works for your specific use case.
- When migrating from a DPF environment, we recommend first running a Full Load task without cached events, and then running CDC-only tasks. We recommend running one task per Db2 node, starting from the Full Load start timestamp or LRI (log record identifier) you set using the `StartFromContext` endpoint extra connection attribute. For information about determining your replication start point, see [Finding the LSN or LRI value for replication start] in the *IBM Support documentation*.

- For ongoing replication (CDC), if you plan to start replication from a specific timestamp, you must set the `StartFromContext` extra connection attribute to the required timestamp.
- Currently, DMS doesn't support the Db2 pureScale Feature, an extension of DB2 LUW that you can use to scale your database solution.
- The `DATA CAPTURE CHANGES` table option is a crucial prerequisite for DB2 data replication processes. Neglecting to enable this option when creating tables can cause missing data, especially for CDC (Change Data Capture) only replication tasks initiated from an earlier starting point. AWS DMS will enable this attribute by default when restarting a CDC or FULL+CDC task. However, any changes made in the source database before the task restart may be missed.

```
ALTER TABLE TABLE_SCHEMA.TABLE_NAME DATA CAPTURE CHANGES INCLUDE LONGVAR COLUMNS;
```

Endpoint settings when using Db2 LUW as a source for AWS DMS

You can specify the settings when you create the source endpoint using the AWS DMS console, or by using the `create-endpoint` command in the AWS CLI, with the

```
--ibm-db2-settings '{"EndpointSetting1": "value1","EndpointSetting2": "value2"}'
```

JSON syntax.

The following table shows the endpoint settings that you can use with Db2 LUW as a source.

Setting Name	Description
`CurrentLsn`	For ongoing replication (CDC), use `CurrentLsn` to specify a log sequence number (LSN) where you want the replication to start.
`MaxKBytesPerRead`	Maximum number of bytes per read, as a NUMBER value. The default is 64 KB.
`SetDataCaptureChanges`	Enables ongoing replication (CDC) as a BOOLEAN value. The default is true.

Extra Connection Attributes (ECAs) when using Db2 LUW as a source for AWS DMS

You can specify the Extra Connection Attributes (ECAs) when you create the source endpoint using the AWS DMS console, or by using the `create-endpoint` command in the AWS CLI, with the

```
--extra-connection-attributes 'ECAname1=value1;ECAname2=value2;'
```

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

Attribute Name	Description
`ConnectionTimeout`	Use this ECA to set the endpoint connection timeout for the Db2 LUW endpoint, in seconds. The default value is 10 seconds. Example: `ConnectionTimeout=30;`
`executeTimeout`	Extra connection attribute which sets the statement (query) timeout for the DB2 LUW endpoint, in seconds. The default value is 60 seconds.

Attribute Name	Description
	Example: `executeTimeout=120;`

Attribute Name	Description
`StartFromContext`	For ongoing replication (CDC), use `StartFromContext` to specify a log's lower limit from where to start the replication. `StartFromContext` accepts different forms of values. Valid values include: • `timestamp` (UTC). Example: `StartFromContext=timestamp:2026-02-23T13:00:00;` • `NOW` For IBM DB2 LUW version 10.5 and higher, NOW combined with `'{"CurrentLsn": "scan"}'` endpoint setting, starts the task from the latest Log Sequence Offset (LSO). Example: `StartFromContext=NOW;` • A specific Log Record Identifier (LRI). Example: `StartFromContext=0100000000000022CC000000000004FB13;` To determine the LRI/LSN range of a log file, run the `db2flsn` command as shown in the example following. `db2flsn -db SAMPLE -lrirange 2` The output from that example is similar to the following . `S0000002.LOG: has LRI range 0000000000000000100000000000022540000000000004F9A6 to`

Attribute Name	Description
	`` 0000000000000001000000000000022CC000000000004 FB13 ``` In that output, the log file is S0000002.LOG and the **StartFromContext** LRI value is the 34 bytes at the end of the range. ``` 0100000000000022CC000000000004FB13 ```

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.

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

Db2 LUW data types	AWS DMS data types
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
VARGRAPHIC (n)	WSTRING, for varying-length graphic strings with a length greater than 0 and less than or equal to16,352 double byte chars
LONG VARGRAPHIC (n)	CLOB, for varying-length graphic strings with a length greater than 0 and less than or equal to16,352 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

Db2 LUW data types	AWS DMS data types
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

Using IBM Db2 for z/OS databases as a source for AWS DMS

You can migrate data from an IBM for z/OS database to any supported target database using AWS Database Migration Service (AWS DMS).

For information about versions of Db2 for z/OS that AWS DMS supports as a source, see Sources for AWS DMS.

Prerequisites when using Db2 for z/OS as a source for AWS DMS

To use an IBM Db2 for z/OS database as a source in AWS DMS, grant the following privileges to the Db2 for z/OS user specified in the source endpoint connection settings.

```
GRANT SELECT ON SYSIBM.SYSTABLES TO Db2USER;
GRANT SELECT ON SYSIBM.SYSTABLESPACE TO Db2USER;
GRANT SELECT ON SYSIBM.SYSTABLEPART TO Db2USER;
GRANT SELECT ON SYSIBM.SYSCOLUMNS TO Db2USER;
GRANT SELECT ON SYSIBM.SYSDATABASE TO Db2USER;
GRANT SELECT ON SYSIBM.SYSDUMMY1 TO Db2USER
```

Also grant SELECT ON *user defined* source tables.

An AWS DMS IBM Db2 for z/OS source endpoint relies on the IBM Data Server Driver for ODBC to access data. The database server must have a valid IBM ODBC Connect license for DMS to connect to this endpoint.

Limitations when using Db2 for z/OS as a source for AWS DMS

The following limitations apply when using an IBM Db2 for z/OS database as a source for AWS DMS:

- Only Full Load replication tasks are supported. Change data capture (CDC) isn't supported.
- Parallel load isn't supported.
- Data validation of views are not supported.
- Schema, table, and columns names must be specified in UPPER case in table mappings for Column/table level transformations and row level selection filters.

Source data types for IBM Db2 for z/OS

Data migrations that use Db2 for z/OS as a source for AWS DMS support most Db2 for z/OS data types. The following table shows the Db2 for z/OS source data types that are supported when using AWS DMS, and the default mapping from AWS DMS data types.

For more information about Db2 for z/OS data types, see the IBM Db2 for z/OS 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.

Db2 for z/OS data types	AWS DMS data types
INTEGER	INT4
SMALLINT	INT2
BIGINT	INT8
DECIMAL (p,s)	NUMERIC (p,s)

Db2 for z/OS data types	AWS DMS data types
	If a decimal point is set to a comma (,) in the DB2 configuration, configure Replicate to support the DB2 setting.
FLOAT	REAL8
DOUBLE	REAL8
REAL	REAL4
DECFLOAT (p)	If precision is 16, then REAL8; if precision is 34, then STRING
GRAPHIC (n)	If n>=127 then WSTRING, for fixed-length graphic strings of double byte chars with a length greater than 0 and less than or equal to 127
VARGRAPHIC (n)	WSTRING, for varying-length graphic strings with a length greater than 0 and less than or equal to16,352 double byte chars
LONG VARGRAPHIC (n)	CLOB, for varying-length graphic strings with a length greater than 0 and less than or equal to16,352 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

Db2 for z/OS data types	AWS DMS data types
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
BINARY	BYTES
VARBINARY	BYTES
ROWID	BYTES. For more information about working with ROWID, see following.
TIMESTAMP WITH TIME ZONE	Not supported.

ROWID columns are migrated by default when the target table prep mode for the task is set to DROP_AND_CREATE (the default). Data validation ignores these columns because the rows are meaningless outside the specific database and table. To turn off migration of these columns, you can do one of the following preparatory steps:

- Precreate the target table without these columns. Then, set the target table prep mode of the task to either DO_NOTHING or TRUNCATE_BEFORE_LOAD. You can use AWS Schema Conversion Tool (AWS SCT) to precreate the target table without the columns.
- Add a table mapping rule to a task that filters out these columns so that they're ignored. For more information, see Transformation rules and actions.

EBCDIC collations in PostgreSQL for AWS Mainframe Modernization service

AWS Mainframe Modernization program helps you modernize your mainframe applications to AWS managed runtime environments. It provides tools and resources that help you plan and implement your migration and modernization projects. For more information about mainframe modernization and migration, see Mainframe Modernization with AWS.

Some IBM Db2 for z/OS data sets are encoded in the Extended Binary Coded Decimal Interchange (EBCDIC) character set. This is a character set that was developed before ASCII (American Standard Code for Information Interchange) became commonly used. A *code page* maps each character of text to the characters in a character set. A traditional code page contains the mapping information between a code point and a character ID. A *character ID* is an 8-byte character data string. A *code point* is an 8-bit binary number that represents a character. Code points are usually shown as hexadecimal representations of their binary values.

If you currently use either the Micro Focus or BluAge component of the Mainframe Modernization service, you must tell AWS DMS to *shift* (translate) certain code points. You can use AWS DMS task settings to perform the shifts. The following example shows how to use the AWS DMS `CharacterSetSettings` operation to map the shifts in a DMS task setting.

```
"CharacterSetSettings": {
        "CharacterSetSupport": null,
        "CharacterReplacements": [
{"SourceCharacterCodePoint": "0000","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "00B8","TargetCharacterCodePoint": "0160"}
,{"SourceCharacterCodePoint": "00BC","TargetCharacterCodePoint": "0161"}
,{"SourceCharacterCodePoint": "00BD","TargetCharacterCodePoint": "017D"}
,{"SourceCharacterCodePoint": "00BE","TargetCharacterCodePoint": "017E"}
,{"SourceCharacterCodePoint": "00A8","TargetCharacterCodePoint": "0152"}
,{"SourceCharacterCodePoint": "00B4","TargetCharacterCodePoint": "0153"}
```

```
,{"SourceCharacterCodePoint": "00A6","TargetCharacterCodePoint": "0178"}
            }
        ]
    }
```

Some EBCDIC collations already exist for PostgreSQL that understand the shifting that's needed. Several different code pages are supported. The sections following provide JSON samples of what you must shift for all the supported code pages. You can simply copy-and-past the necessary JSON that you need in your DMS task.

Micro Focus specific EBCDIC collations

For Micro Focus, shift a subset of characters as needed for the following collations.

```
da-DK-cp1142m-x-icu
de-DE-cp1141m-x-icu
en-GB-cp1146m-x-icu
en-US-cp1140m-x-icu
es-ES-cp1145m-x-icu
fi-FI-cp1143m-x-icu
fr-FR-cp1147m-x-icu
it-IT-cp1144m-x-icu
nl-BE-cp1148m-x-icu
```

Example Micro Focus data shifts per collation:

en_us_cp1140m

Code Shift:

```
0000    0180
00A6    0160
00B8    0161
00BC    017D
00BD    017E
00BE    0152
00A8    0153
00B4    0178
```

Corresponding input mapping for an AWS DMS task:

```
 {"SourceCharacterCodePoint": "0000","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "00A6","TargetCharacterCodePoint": "0160"}
,{"SourceCharacterCodePoint": "00B8","TargetCharacterCodePoint": "0161"}
,{"SourceCharacterCodePoint": "00BC","TargetCharacterCodePoint": "017D"}
,{"SourceCharacterCodePoint": "00BD","TargetCharacterCodePoint": "017E"}
,{"SourceCharacterCodePoint": "00BE","TargetCharacterCodePoint": "0152"}
,{"SourceCharacterCodePoint": "00A8","TargetCharacterCodePoint": "0153"}
,{"SourceCharacterCodePoint": "00B4","TargetCharacterCodePoint": "0178"}
```

en_us_cp1141m

Code Shift:

```
0000    0180
00B8    0160
00BC    0161
00BD    017D
00BE    017E
00A8    0152
00B4    0153
00A6    0178
```

Corresponding input mapping for an AWS DMS task:

```
 {"SourceCharacterCodePoint": "0000","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "00B8","TargetCharacterCodePoint": "0160"}
,{"SourceCharacterCodePoint": "00BC","TargetCharacterCodePoint": "0161"}
,{"SourceCharacterCodePoint": "00BD","TargetCharacterCodePoint": "017D"}
,{"SourceCharacterCodePoint": "00BE","TargetCharacterCodePoint": "017E"}
,{"SourceCharacterCodePoint": "00A8","TargetCharacterCodePoint": "0152"}
,{"SourceCharacterCodePoint": "00B4","TargetCharacterCodePoint": "0153"}
,{"SourceCharacterCodePoint": "00A6","TargetCharacterCodePoint": "0178"}
```

en_us_cp1142m

Code Shift:

```
0000    0180
00A6    0160
```

```
00B8    0161
00BC    017D
00BD    017E
00BE    0152
00A8    0153
00B4    0178
```

Corresponding input mapping for an AWS DMS task:

```
 {"SourceCharacterCodePoint": "0000","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "00A6","TargetCharacterCodePoint": "0160"}
,{"SourceCharacterCodePoint": "00B8","TargetCharacterCodePoint": "0161"}
,{"SourceCharacterCodePoint": "00BC","TargetCharacterCodePoint": "017D"}
,{"SourceCharacterCodePoint": "00BD","TargetCharacterCodePoint": "017E"}
,{"SourceCharacterCodePoint": "00BE","TargetCharacterCodePoint": "0152"}
,{"SourceCharacterCodePoint": "00A8","TargetCharacterCodePoint": "0153"}
,{"SourceCharacterCodePoint": "00B4","TargetCharacterCodePoint": "0178"}
```

en_us_cp1143m

Code Shift:

```
0000    0180
00B8    0160
00BC    0161
00BD    017D
00BE    017E
00A8    0152
00B4    0153
00A6    0178
```

Corresponding input mapping for an AWS DMS task:

```
 {"SourceCharacterCodePoint": "0000","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "00B8","TargetCharacterCodePoint": "0160"}
,{"SourceCharacterCodePoint": "00BC","TargetCharacterCodePoint": "0161"}
,{"SourceCharacterCodePoint": "00BD","TargetCharacterCodePoint": "017D"}
,{"SourceCharacterCodePoint": "00BE","TargetCharacterCodePoint": "017E"}
,{"SourceCharacterCodePoint": "00A8","TargetCharacterCodePoint": "0152"}
```

```
,{"SourceCharacterCodePoint": "00B4","TargetCharacterCodePoint": "0153"}
,{"SourceCharacterCodePoint": "00A6","TargetCharacterCodePoint": "0178"}
```

en_us_cp1144m

Code Shift:

```
0000    0180
00B8    0160
00BC    0161
00BD    017D
00BE    017E
00A8    0152
00B4    0153
00A6    0178
```

Corresponding input mapping for an AWS DMS task:

```
 {"SourceCharacterCodePoint": "0000","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "00B8","TargetCharacterCodePoint": "0160"}
,{"SourceCharacterCodePoint": "00BC","TargetCharacterCodePoint": "0161"}
,{"SourceCharacterCodePoint": "00BD","TargetCharacterCodePoint": "017D"}
,{"SourceCharacterCodePoint": "00BE","TargetCharacterCodePoint": "017E"}
,{"SourceCharacterCodePoint": "00A8","TargetCharacterCodePoint": "0152"}
,{"SourceCharacterCodePoint": "00B4","TargetCharacterCodePoint": "0153"}
,{"SourceCharacterCodePoint": "00A6","TargetCharacterCodePoint": "0178"}
```

en_us_cp1145m

Code Shift:

```
0000    0180
00A6    0160
00B8    0161
00A8    017D
00BC    017E
00BD    0152
00BE    0153
00B4    0178
```

Corresponding input mapping for an AWS DMS task:

```
 {"SourceCharacterCodePoint": "0000","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "00A6","TargetCharacterCodePoint": "0160"}
,{"SourceCharacterCodePoint": "00B8","TargetCharacterCodePoint": "0161"}
,{"SourceCharacterCodePoint": "00A8","TargetCharacterCodePoint": "017D"}
,{"SourceCharacterCodePoint": "00BC","TargetCharacterCodePoint": "017E"}
,{"SourceCharacterCodePoint": "00BD","TargetCharacterCodePoint": "0152"}
,{"SourceCharacterCodePoint": "00BE","TargetCharacterCodePoint": "0153"}
,{"SourceCharacterCodePoint": "00B4","TargetCharacterCodePoint": "0178"}
```

en_us_cp1146m

Code Shift:

```
0000    0180
00A6    0160
00B8    0161
00BC    017D
00BD    017E
00BE    0152
00A8    0153
00B4    0178
```

Corresponding input mapping for an AWS DMS task:

```
 {"SourceCharacterCodePoint": "0000","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "00A6","TargetCharacterCodePoint": "0160"}
,{"SourceCharacterCodePoint": "00B8","TargetCharacterCodePoint": "0161"}
,{"SourceCharacterCodePoint": "00BC","TargetCharacterCodePoint": "017D"}
,{"SourceCharacterCodePoint": "00BD","TargetCharacterCodePoint": "017E"}
,{"SourceCharacterCodePoint": "00BE","TargetCharacterCodePoint": "0152"}
,{"SourceCharacterCodePoint": "00A8","TargetCharacterCodePoint": "0153"}
,{"SourceCharacterCodePoint": "00B4","TargetCharacterCodePoint": "0178"}
```

en_us_cp1147m

Code Shift:

```
0000    0180
00B8    0160
00A8    0161
00BC    017D
00BD    017E
00BE    0152
00B4    0153
00A6    0178
```

Corresponding input mapping for an AWS DMS task:

```
 {"SourceCharacterCodePoint": "0000","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "00B8","TargetCharacterCodePoint": "0160"}
,{"SourceCharacterCodePoint": "00A8","TargetCharacterCodePoint": "0161"}
,{"SourceCharacterCodePoint": "00BC","TargetCharacterCodePoint": "017D"}
,{"SourceCharacterCodePoint": "00BD","TargetCharacterCodePoint": "017E"}
,{"SourceCharacterCodePoint": "00BE","TargetCharacterCodePoint": "0152"}
,{"SourceCharacterCodePoint": "00B4","TargetCharacterCodePoint": "0153"}
,{"SourceCharacterCodePoint": "00A6","TargetCharacterCodePoint": "0178"}
```

en_us_cp1148m

Code Shift:

```
0000    0180
00A6    0160
00B8    0161
00BC    017D
00BD    017E
00BE    0152
00A8    0153
00B4    0178
```

Corresponding input mapping for an AWS DMS task:

```
 {"SourceCharacterCodePoint": "0000","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "00A6","TargetCharacterCodePoint": "0160"}
```

```
,{"SourceCharacterCodePoint": "00B8","TargetCharacterCodePoint": "0161"}
,{"SourceCharacterCodePoint": "00BC","TargetCharacterCodePoint": "017D"}
,{"SourceCharacterCodePoint": "00BD","TargetCharacterCodePoint": "017E"}
,{"SourceCharacterCodePoint": "00BE","TargetCharacterCodePoint": "0152"}
,{"SourceCharacterCodePoint": "00A8","TargetCharacterCodePoint": "0153"}
,{"SourceCharacterCodePoint": "00B4","TargetCharacterCodePoint": "0178"}
```

BluAge specific EBCDIC collations

For BluAge, shift all of the following *low values* and *high values* as needed. These collations should only be used to support the Mainframe Migration BluAge service.

```
da-DK-cp1142b-x-icu
 da-DK-cp277b-x-icu
 de-DE-cp1141b-x-icu
 de-DE-cp273b-x-icu
 en-GB-cp1146b-x-icu
 en-GB-cp285b-x-icu
 en-US-cp037b-x-icu
 en-US-cp1140b-x-icu
 es-ES-cp1145b-x-icu
 es-ES-cp284b-x-icu
 fi-FI-cp1143b-x-icu
 fi-FI-cp278b-x-icu
 fr-FR-cp1147b-x-icu
 fr-FR-cp297b-x-icu
 it-IT-cp1144b-x-icu
 it-IT-cp280b-x-icu
 nl-BE-cp1148b-x-icu
 nl-BE-cp500b-x-icu
```

Example BluAge Data Shifts:

da-DK-cp277b and **da-DK-cp1142b**

Code Shift:

```
0180    0180
0001    0181
0002    0182
0003    0183
```

```
009C    0184
0009    0185
0086    0186
007F    0187
0097    0188
008D    0189
008E    018A
000B    018B
000C    018C
000D    018D
000E    018E
000F    018F
0010    0190
0011    0191
0012    0192
0013    0193
009D    0194
0085    0195
0008    0196
0087    0197
0018    0198
0019    0199
0092    019A
008F    019B
001C    019C
001D    019D
001E    019E
001F    019F
0080    01A0
0081    01A1
0082    01A2
0083    01A3
0084    01A4
000A    01A5
0017    01A6
001B    01A7
0088    01A8
0089    01A9
008A    01AA
008B    01AB
008C    01AC
0005    01AD
0006    01AE
0007    01AF
```

```
0090    01B0
0091    01B1
0016    01B2
0093    01B3
0094    01B4
0095    01B5
0096    01B6
0004    01B7
0098    01B8
0099    01B9
009A    01BA
009B    01BB
0014    01BC
0015    01BD
009E    01BE
001A    01BF
009F    027F
```

Corresponding input mapping for an AWS DMS task:

```
 {"SourceCharacterCodePoint": "0180","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "0001","TargetCharacterCodePoint": "0181"}
,{"SourceCharacterCodePoint": "0002","TargetCharacterCodePoint": "0182"}
,{"SourceCharacterCodePoint": "0003","TargetCharacterCodePoint": "0183"}
,{"SourceCharacterCodePoint": "009C","TargetCharacterCodePoint": "0184"}
,{"SourceCharacterCodePoint": "0009","TargetCharacterCodePoint": "0185"}
,{"SourceCharacterCodePoint": "0086","TargetCharacterCodePoint": "0186"}
,{"SourceCharacterCodePoint": "007F","TargetCharacterCodePoint": "0187"}
,{"SourceCharacterCodePoint": "0097","TargetCharacterCodePoint": "0188"}
,{"SourceCharacterCodePoint": "008D","TargetCharacterCodePoint": "0189"}
,{"SourceCharacterCodePoint": "008E","TargetCharacterCodePoint": "018A"}
,{"SourceCharacterCodePoint": "000B","TargetCharacterCodePoint": "018B"}
,{"SourceCharacterCodePoint": "000C","TargetCharacterCodePoint": "018C"}
,{"SourceCharacterCodePoint": "000D","TargetCharacterCodePoint": "018D"}
,{"SourceCharacterCodePoint": "000E","TargetCharacterCodePoint": "018E"}
,{"SourceCharacterCodePoint": "000F","TargetCharacterCodePoint": "018F"}
,{"SourceCharacterCodePoint": "0010","TargetCharacterCodePoint": "0190"}
,{"SourceCharacterCodePoint": "0011","TargetCharacterCodePoint": "0191"}
,{"SourceCharacterCodePoint": "0012","TargetCharacterCodePoint": "0192"}
,{"SourceCharacterCodePoint": "0013","TargetCharacterCodePoint": "0193"}
,{"SourceCharacterCodePoint": "009D","TargetCharacterCodePoint": "0194"}
,{"SourceCharacterCodePoint": "0085","TargetCharacterCodePoint": "0195"}
```

```
,{"SourceCharacterCodePoint": "0008","TargetCharacterCodePoint": "0196"}
,{"SourceCharacterCodePoint": "0087","TargetCharacterCodePoint": "0197"}
,{"SourceCharacterCodePoint": "0018","TargetCharacterCodePoint": "0198"}
,{"SourceCharacterCodePoint": "0019","TargetCharacterCodePoint": "0199"}
,{"SourceCharacterCodePoint": "0092","TargetCharacterCodePoint": "019A"}
,{"SourceCharacterCodePoint": "008F","TargetCharacterCodePoint": "019B"}
,{"SourceCharacterCodePoint": "001C","TargetCharacterCodePoint": "019C"}
,{"SourceCharacterCodePoint": "001D","TargetCharacterCodePoint": "019D"}
,{"SourceCharacterCodePoint": "001E","TargetCharacterCodePoint": "019E"}
,{"SourceCharacterCodePoint": "001F","TargetCharacterCodePoint": "019F"}
,{"SourceCharacterCodePoint": "0080","TargetCharacterCodePoint": "01A0"}
,{"SourceCharacterCodePoint": "0081","TargetCharacterCodePoint": "01A1"}
,{"SourceCharacterCodePoint": "0082","TargetCharacterCodePoint": "01A2"}
,{"SourceCharacterCodePoint": "0083","TargetCharacterCodePoint": "01A3"}
,{"SourceCharacterCodePoint": "0084","TargetCharacterCodePoint": "01A4"}
,{"SourceCharacterCodePoint": "000A","TargetCharacterCodePoint": "01A5"}
,{"SourceCharacterCodePoint": "0017","TargetCharacterCodePoint": "01A6"}
,{"SourceCharacterCodePoint": "001B","TargetCharacterCodePoint": "01A7"}
,{"SourceCharacterCodePoint": "0088","TargetCharacterCodePoint": "01A8"}
,{"SourceCharacterCodePoint": "0089","TargetCharacterCodePoint": "01A9"}
,{"SourceCharacterCodePoint": "008A","TargetCharacterCodePoint": "01AA"}
,{"SourceCharacterCodePoint": "008B","TargetCharacterCodePoint": "01AB"}
,{"SourceCharacterCodePoint": "008C","TargetCharacterCodePoint": "01AC"}
,{"SourceCharacterCodePoint": "0005","TargetCharacterCodePoint": "01AD"}
,{"SourceCharacterCodePoint": "0006","TargetCharacterCodePoint": "01AE"}
,{"SourceCharacterCodePoint": "0007","TargetCharacterCodePoint": "01AF"}
,{"SourceCharacterCodePoint": "0090","TargetCharacterCodePoint": "01B0"}
,{"SourceCharacterCodePoint": "0091","TargetCharacterCodePoint": "01B1"}
,{"SourceCharacterCodePoint": "0016","TargetCharacterCodePoint": "01B2"}
,{"SourceCharacterCodePoint": "0093","TargetCharacterCodePoint": "01B3"}
,{"SourceCharacterCodePoint": "0094","TargetCharacterCodePoint": "01B4"}
,{"SourceCharacterCodePoint": "0095","TargetCharacterCodePoint": "01B5"}
,{"SourceCharacterCodePoint": "0096","TargetCharacterCodePoint": "01B6"}
,{"SourceCharacterCodePoint": "0004","TargetCharacterCodePoint": "01B7"}
,{"SourceCharacterCodePoint": "0098","TargetCharacterCodePoint": "01B8"}
,{"SourceCharacterCodePoint": "0099","TargetCharacterCodePoint": "01B9"}
,{"SourceCharacterCodePoint": "009A","TargetCharacterCodePoint": "01BA"}
,{"SourceCharacterCodePoint": "009B","TargetCharacterCodePoint": "01BB"}
,{"SourceCharacterCodePoint": "0014","TargetCharacterCodePoint": "01BC"}
,{"SourceCharacterCodePoint": "0015","TargetCharacterCodePoint": "01BD"}
,{"SourceCharacterCodePoint": "009E","TargetCharacterCodePoint": "01BE"}
,{"SourceCharacterCodePoint": "001A","TargetCharacterCodePoint": "01BF"}
,{"SourceCharacterCodePoint": "009F","TargetCharacterCodePoint": "027F"}
```

```
```

de-DE-273b and de-DE-1141b

Code Shift:

```
0180    0180
0001    0181
0002    0182
0003    0183
009C    0184
0009    0185
0086    0186
007F    0187
0097    0188
008D    0189
008E    018A
000B    018B
000C    018C
000D    018D
000E    018E
000F    018F
0010    0190
0011    0191
0012    0192
0013    0193
009D    0194
0085    0195
0008    0196
0087    0197
0018    0198
0019    0199
0092    019A
008F    019B
001C    019C
001D    019D
001E    019E
001F    019F
0080    01A0
0081    01A1
0082    01A2
0083    01A3
0084    01A4
000A    01A5
```

```
0017    01A6
001B    01A7
0088    01A8
0089    01A9
008A    01AA
008B    01AB
008C    01AC
0005    01AD
0006    01AE
0007    01AF
0090    01B0
0091    01B1
0016    01B2
0093    01B3
0094    01B4
0095    01B5
0096    01B6
0004    01B7
0098    01B8
0099    01B9
009A    01BA
009B    01BB
0014    01BC
0015    01BD
009E    01BE
001A    01BF
009F    027F
```

Corresponding input mapping for an AWS DMS task:

```
 {"SourceCharacterCodePoint": "0180","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "0001","TargetCharacterCodePoint": "0181"}
,{"SourceCharacterCodePoint": "0002","TargetCharacterCodePoint": "0182"}
,{"SourceCharacterCodePoint": "0003","TargetCharacterCodePoint": "0183"}
,{"SourceCharacterCodePoint": "009C","TargetCharacterCodePoint": "0184"}
,{"SourceCharacterCodePoint": "0009","TargetCharacterCodePoint": "0185"}
,{"SourceCharacterCodePoint": "0086","TargetCharacterCodePoint": "0186"}
,{"SourceCharacterCodePoint": "007F","TargetCharacterCodePoint": "0187"}
,{"SourceCharacterCodePoint": "0097","TargetCharacterCodePoint": "0188"}
,{"SourceCharacterCodePoint": "008D","TargetCharacterCodePoint": "0189"}
,{"SourceCharacterCodePoint": "008E","TargetCharacterCodePoint": "018A"}
,{"SourceCharacterCodePoint": "000B","TargetCharacterCodePoint": "018B"}
```

```
,{"SourceCharacterCodePoint": "000C","TargetCharacterCodePoint": "018C"}
,{"SourceCharacterCodePoint": "000D","TargetCharacterCodePoint": "018D"}
,{"SourceCharacterCodePoint": "000E","TargetCharacterCodePoint": "018E"}
,{"SourceCharacterCodePoint": "000F","TargetCharacterCodePoint": "018F"}
,{"SourceCharacterCodePoint": "0010","TargetCharacterCodePoint": "0190"}
,{"SourceCharacterCodePoint": "0011","TargetCharacterCodePoint": "0191"}
,{"SourceCharacterCodePoint": "0012","TargetCharacterCodePoint": "0192"}
,{"SourceCharacterCodePoint": "0013","TargetCharacterCodePoint": "0193"}
,{"SourceCharacterCodePoint": "009D","TargetCharacterCodePoint": "0194"}
,{"SourceCharacterCodePoint": "0085","TargetCharacterCodePoint": "0195"}
,{"SourceCharacterCodePoint": "0008","TargetCharacterCodePoint": "0196"}
,{"SourceCharacterCodePoint": "0087","TargetCharacterCodePoint": "0197"}
,{"SourceCharacterCodePoint": "0018","TargetCharacterCodePoint": "0198"}
,{"SourceCharacterCodePoint": "0019","TargetCharacterCodePoint": "0199"}
,{"SourceCharacterCodePoint": "0092","TargetCharacterCodePoint": "019A"}
,{"SourceCharacterCodePoint": "008F","TargetCharacterCodePoint": "019B"}
,{"SourceCharacterCodePoint": "001C","TargetCharacterCodePoint": "019C"}
,{"SourceCharacterCodePoint": "001D","TargetCharacterCodePoint": "019D"}
,{"SourceCharacterCodePoint": "001E","TargetCharacterCodePoint": "019E"}
,{"SourceCharacterCodePoint": "001F","TargetCharacterCodePoint": "019F"}
,{"SourceCharacterCodePoint": "0080","TargetCharacterCodePoint": "01A0"}
,{"SourceCharacterCodePoint": "0081","TargetCharacterCodePoint": "01A1"}
,{"SourceCharacterCodePoint": "0082","TargetCharacterCodePoint": "01A2"}
,{"SourceCharacterCodePoint": "0083","TargetCharacterCodePoint": "01A3"}
,{"SourceCharacterCodePoint": "0084","TargetCharacterCodePoint": "01A4"}
,{"SourceCharacterCodePoint": "000A","TargetCharacterCodePoint": "01A5"}
,{"SourceCharacterCodePoint": "0017","TargetCharacterCodePoint": "01A6"}
,{"SourceCharacterCodePoint": "001B","TargetCharacterCodePoint": "01A7"}
,{"SourceCharacterCodePoint": "0088","TargetCharacterCodePoint": "01A8"}
,{"SourceCharacterCodePoint": "0089","TargetCharacterCodePoint": "01A9"}
,{"SourceCharacterCodePoint": "008A","TargetCharacterCodePoint": "01AA"}
,{"SourceCharacterCodePoint": "008B","TargetCharacterCodePoint": "01AB"}
,{"SourceCharacterCodePoint": "008C","TargetCharacterCodePoint": "01AC"}
,{"SourceCharacterCodePoint": "0005","TargetCharacterCodePoint": "01AD"}
,{"SourceCharacterCodePoint": "0006","TargetCharacterCodePoint": "01AE"}
,{"SourceCharacterCodePoint": "0007","TargetCharacterCodePoint": "01AF"}
,{"SourceCharacterCodePoint": "0090","TargetCharacterCodePoint": "01B0"}
,{"SourceCharacterCodePoint": "0091","TargetCharacterCodePoint": "01B1"}
,{"SourceCharacterCodePoint": "0016","TargetCharacterCodePoint": "01B2"}
,{"SourceCharacterCodePoint": "0093","TargetCharacterCodePoint": "01B3"}
,{"SourceCharacterCodePoint": "0094","TargetCharacterCodePoint": "01B4"}
,{"SourceCharacterCodePoint": "0095","TargetCharacterCodePoint": "01B5"}
,{"SourceCharacterCodePoint": "0096","TargetCharacterCodePoint": "01B6"}
,{"SourceCharacterCodePoint": "0004","TargetCharacterCodePoint": "01B7"}
```

```
,{"SourceCharacterCodePoint": "0098","TargetCharacterCodePoint": "01B8"}
,{"SourceCharacterCodePoint": "0099","TargetCharacterCodePoint": "01B9"}
,{"SourceCharacterCodePoint": "009A","TargetCharacterCodePoint": "01BA"}
,{"SourceCharacterCodePoint": "009B","TargetCharacterCodePoint": "01BB"}
,{"SourceCharacterCodePoint": "0014","TargetCharacterCodePoint": "01BC"}
,{"SourceCharacterCodePoint": "0015","TargetCharacterCodePoint": "01BD"}
,{"SourceCharacterCodePoint": "009E","TargetCharacterCodePoint": "01BE"}
,{"SourceCharacterCodePoint": "001A","TargetCharacterCodePoint": "01BF"}
,{"SourceCharacterCodePoint": "009F","TargetCharacterCodePoint": "027F"}
```

en-GB-285b and **en-GB-1146b**

Code Shift:

```
0180    0180
0001    0181
0002    0182
0003    0183
009C    0184
0009    0185
0086    0186
007F    0187
0097    0188
008D    0189
008E    018A
000B    018B
000C    018C
000D    018D
000E    018E
000F    018F
0010    0190
0011    0191
0012    0192
0013    0193
009D    0194
0085    0195
0008    0196
0087    0197
0018    0198
0019    0199
0092    019A
008F    019B
```

```
001C    019C
001D    019D
001E    019E
001F    019F
0080    01A0
0081    01A1
0082    01A2
0083    01A3
0084    01A4
000A    01A5
0017    01A6
001B    01A7
0088    01A8
0089    01A9
008A    01AA
008B    01AB
008C    01AC
0005    01AD
0006    01AE
0007    01AF
0090    01B0
0091    01B1
0016    01B2
0093    01B3
0094    01B4
0095    01B5
0096    01B6
0004    01B7
0098    01B8
0099    01B9
009A    01BA
009B    01BB
0014    01BC
0015    01BD
009E    01BE
001A    01BF
009F    027F
```

Corresponding input mapping for an AWS DMS task:

```
{"SourceCharacterCodePoint": "0180","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "0001","TargetCharacterCodePoint": "0181"}
```

```
,{"SourceCharacterCodePoint": "0002","TargetCharacterCodePoint": "0182"}
,{"SourceCharacterCodePoint": "0003","TargetCharacterCodePoint": "0183"}
,{"SourceCharacterCodePoint": "009C","TargetCharacterCodePoint": "0184"}
,{"SourceCharacterCodePoint": "0009","TargetCharacterCodePoint": "0185"}
,{"SourceCharacterCodePoint": "0086","TargetCharacterCodePoint": "0186"}
,{"SourceCharacterCodePoint": "007F","TargetCharacterCodePoint": "0187"}
,{"SourceCharacterCodePoint": "0097","TargetCharacterCodePoint": "0188"}
,{"SourceCharacterCodePoint": "008D","TargetCharacterCodePoint": "0189"}
,{"SourceCharacterCodePoint": "008E","TargetCharacterCodePoint": "018A"}
,{"SourceCharacterCodePoint": "000B","TargetCharacterCodePoint": "018B"}
,{"SourceCharacterCodePoint": "000C","TargetCharacterCodePoint": "018C"}
,{"SourceCharacterCodePoint": "000D","TargetCharacterCodePoint": "018D"}
,{"SourceCharacterCodePoint": "000E","TargetCharacterCodePoint": "018E"}
,{"SourceCharacterCodePoint": "000F","TargetCharacterCodePoint": "018F"}
,{"SourceCharacterCodePoint": "0010","TargetCharacterCodePoint": "0190"}
,{"SourceCharacterCodePoint": "0011","TargetCharacterCodePoint": "0191"}
,{"SourceCharacterCodePoint": "0012","TargetCharacterCodePoint": "0192"}
,{"SourceCharacterCodePoint": "0013","TargetCharacterCodePoint": "0193"}
,{"SourceCharacterCodePoint": "009D","TargetCharacterCodePoint": "0194"}
,{"SourceCharacterCodePoint": "0085","TargetCharacterCodePoint": "0195"}
,{"SourceCharacterCodePoint": "0008","TargetCharacterCodePoint": "0196"}
,{"SourceCharacterCodePoint": "0087","TargetCharacterCodePoint": "0197"}
,{"SourceCharacterCodePoint": "0018","TargetCharacterCodePoint": "0198"}
,{"SourceCharacterCodePoint": "0019","TargetCharacterCodePoint": "0199"}
,{"SourceCharacterCodePoint": "0092","TargetCharacterCodePoint": "019A"}
,{"SourceCharacterCodePoint": "008F","TargetCharacterCodePoint": "019B"}
,{"SourceCharacterCodePoint": "001C","TargetCharacterCodePoint": "019C"}
,{"SourceCharacterCodePoint": "001D","TargetCharacterCodePoint": "019D"}
,{"SourceCharacterCodePoint": "001E","TargetCharacterCodePoint": "019E"}
,{"SourceCharacterCodePoint": "001F","TargetCharacterCodePoint": "019F"}
,{"SourceCharacterCodePoint": "0080","TargetCharacterCodePoint": "01A0"}
,{"SourceCharacterCodePoint": "0081","TargetCharacterCodePoint": "01A1"}
,{"SourceCharacterCodePoint": "0082","TargetCharacterCodePoint": "01A2"}
,{"SourceCharacterCodePoint": "0083","TargetCharacterCodePoint": "01A3"}
,{"SourceCharacterCodePoint": "0084","TargetCharacterCodePoint": "01A4"}
,{"SourceCharacterCodePoint": "000A","TargetCharacterCodePoint": "01A5"}
,{"SourceCharacterCodePoint": "0017","TargetCharacterCodePoint": "01A6"}
,{"SourceCharacterCodePoint": "001B","TargetCharacterCodePoint": "01A7"}
,{"SourceCharacterCodePoint": "0088","TargetCharacterCodePoint": "01A8"}
,{"SourceCharacterCodePoint": "0089","TargetCharacterCodePoint": "01A9"}
,{"SourceCharacterCodePoint": "008A","TargetCharacterCodePoint": "01AA"}
,{"SourceCharacterCodePoint": "008B","TargetCharacterCodePoint": "01AB"}
,{"SourceCharacterCodePoint": "008C","TargetCharacterCodePoint": "01AC"}
,{"SourceCharacterCodePoint": "0005","TargetCharacterCodePoint": "01AD"}
```

```
,{"SourceCharacterCodePoint": "0006","TargetCharacterCodePoint": "01AE"}
,{"SourceCharacterCodePoint": "0007","TargetCharacterCodePoint": "01AF"}
,{"SourceCharacterCodePoint": "0090","TargetCharacterCodePoint": "01B0"}
,{"SourceCharacterCodePoint": "0091","TargetCharacterCodePoint": "01B1"}
,{"SourceCharacterCodePoint": "0016","TargetCharacterCodePoint": "01B2"}
,{"SourceCharacterCodePoint": "0093","TargetCharacterCodePoint": "01B3"}
,{"SourceCharacterCodePoint": "0094","TargetCharacterCodePoint": "01B4"}
,{"SourceCharacterCodePoint": "0095","TargetCharacterCodePoint": "01B5"}
,{"SourceCharacterCodePoint": "0096","TargetCharacterCodePoint": "01B6"}
,{"SourceCharacterCodePoint": "0004","TargetCharacterCodePoint": "01B7"}
,{"SourceCharacterCodePoint": "0098","TargetCharacterCodePoint": "01B8"}
,{"SourceCharacterCodePoint": "0099","TargetCharacterCodePoint": "01B9"}
,{"SourceCharacterCodePoint": "009A","TargetCharacterCodePoint": "01BA"}
,{"SourceCharacterCodePoint": "009B","TargetCharacterCodePoint": "01BB"}
,{"SourceCharacterCodePoint": "0014","TargetCharacterCodePoint": "01BC"}
,{"SourceCharacterCodePoint": "0015","TargetCharacterCodePoint": "01BD"}
,{"SourceCharacterCodePoint": "009E","TargetCharacterCodePoint": "01BE"}
,{"SourceCharacterCodePoint": "001A","TargetCharacterCodePoint": "01BF"}
,{"SourceCharacterCodePoint": "009F","TargetCharacterCodePoint": "027F"}
```


en-us-037b and **en-us-1140b**

Code Shift:

```
0180    0180
0001    0181
0002    0182
0003    0183
009C    0184
0009    0185
0086    0186
007F    0187
0097    0188
008D    0189
008E    018A
000B    018B
000C    018C
000D    018D
000E    018E
000F    018F
0010    0190
0011    0191
```

```
0012    0192
0013    0193
009D    0194
0085    0195
0008    0196
0087    0197
0018    0198
0019    0199
0092    019A
008F    019B
001C    019C
001D    019D
001E    019E
001F    019F
0080    01A0
0081    01A1
0082    01A2
0083    01A3
0084    01A4
000A    01A5
0017    01A6
001B    01A7
0088    01A8
0089    01A9
008A    01AA
008B    01AB
008C    01AC
0005    01AD
0006    01AE
0007    01AF
0090    01B0
0091    01B1
0016    01B2
0093    01B3
0094    01B4
0095    01B5
0096    01B6
0004    01B7
0098    01B8
0099    01B9
009A    01BA
009B    01BB
0014    01BC
0015    01BD
```

```
009E    01BE
001A    01BF
009F    027F
```

Corresponding input mapping for an AWS DMS task:

```
{"SourceCharacterCodePoint": "0180","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "0001","TargetCharacterCodePoint": "0181"}
,{"SourceCharacterCodePoint": "0002","TargetCharacterCodePoint": "0182"}
,{"SourceCharacterCodePoint": "0003","TargetCharacterCodePoint": "0183"}
,{"SourceCharacterCodePoint": "009C","TargetCharacterCodePoint": "0184"}
,{"SourceCharacterCodePoint": "0009","TargetCharacterCodePoint": "0185"}
,{"SourceCharacterCodePoint": "0086","TargetCharacterCodePoint": "0186"}
,{"SourceCharacterCodePoint": "007F","TargetCharacterCodePoint": "0187"}
,{"SourceCharacterCodePoint": "0097","TargetCharacterCodePoint": "0188"}
,{"SourceCharacterCodePoint": "008D","TargetCharacterCodePoint": "0189"}
,{"SourceCharacterCodePoint": "008E","TargetCharacterCodePoint": "018A"}
,{"SourceCharacterCodePoint": "000B","TargetCharacterCodePoint": "018B"}
,{"SourceCharacterCodePoint": "000C","TargetCharacterCodePoint": "018C"}
,{"SourceCharacterCodePoint": "000D","TargetCharacterCodePoint": "018D"}
,{"SourceCharacterCodePoint": "000E","TargetCharacterCodePoint": "018E"}
,{"SourceCharacterCodePoint": "000F","TargetCharacterCodePoint": "018F"}
,{"SourceCharacterCodePoint": "0010","TargetCharacterCodePoint": "0190"}
,{"SourceCharacterCodePoint": "0011","TargetCharacterCodePoint": "0191"}
,{"SourceCharacterCodePoint": "0012","TargetCharacterCodePoint": "0192"}
,{"SourceCharacterCodePoint": "0013","TargetCharacterCodePoint": "0193"}
,{"SourceCharacterCodePoint": "009D","TargetCharacterCodePoint": "0194"}
,{"SourceCharacterCodePoint": "0085","TargetCharacterCodePoint": "0195"}
,{"SourceCharacterCodePoint": "0008","TargetCharacterCodePoint": "0196"}
,{"SourceCharacterCodePoint": "0087","TargetCharacterCodePoint": "0197"}
,{"SourceCharacterCodePoint": "0018","TargetCharacterCodePoint": "0198"}
,{"SourceCharacterCodePoint": "0019","TargetCharacterCodePoint": "0199"}
,{"SourceCharacterCodePoint": "0092","TargetCharacterCodePoint": "019A"}
,{"SourceCharacterCodePoint": "008F","TargetCharacterCodePoint": "019B"}
,{"SourceCharacterCodePoint": "001C","TargetCharacterCodePoint": "019C"}
,{"SourceCharacterCodePoint": "001D","TargetCharacterCodePoint": "019D"}
,{"SourceCharacterCodePoint": "001E","TargetCharacterCodePoint": "019E"}
,{"SourceCharacterCodePoint": "001F","TargetCharacterCodePoint": "019F"}
,{"SourceCharacterCodePoint": "0080","TargetCharacterCodePoint": "01A0"}
,{"SourceCharacterCodePoint": "0081","TargetCharacterCodePoint": "01A1"}
,{"SourceCharacterCodePoint": "0082","TargetCharacterCodePoint": "01A2"}
,{"SourceCharacterCodePoint": "0083","TargetCharacterCodePoint": "01A3"}
```

```
,{"SourceCharacterCodePoint": "0084","TargetCharacterCodePoint": "01A4"}
,{"SourceCharacterCodePoint": "000A","TargetCharacterCodePoint": "01A5"}
,{"SourceCharacterCodePoint": "0017","TargetCharacterCodePoint": "01A6"}
,{"SourceCharacterCodePoint": "001B","TargetCharacterCodePoint": "01A7"}
,{"SourceCharacterCodePoint": "0088","TargetCharacterCodePoint": "01A8"}
,{"SourceCharacterCodePoint": "0089","TargetCharacterCodePoint": "01A9"}
,{"SourceCharacterCodePoint": "008A","TargetCharacterCodePoint": "01AA"}
,{"SourceCharacterCodePoint": "008B","TargetCharacterCodePoint": "01AB"}
,{"SourceCharacterCodePoint": "008C","TargetCharacterCodePoint": "01AC"}
,{"SourceCharacterCodePoint": "0005","TargetCharacterCodePoint": "01AD"}
,{"SourceCharacterCodePoint": "0006","TargetCharacterCodePoint": "01AE"}
,{"SourceCharacterCodePoint": "0007","TargetCharacterCodePoint": "01AF"}
,{"SourceCharacterCodePoint": "0090","TargetCharacterCodePoint": "01B0"}
,{"SourceCharacterCodePoint": "0091","TargetCharacterCodePoint": "01B1"}
,{"SourceCharacterCodePoint": "0016","TargetCharacterCodePoint": "01B2"}
,{"SourceCharacterCodePoint": "0093","TargetCharacterCodePoint": "01B3"}
,{"SourceCharacterCodePoint": "0094","TargetCharacterCodePoint": "01B4"}
,{"SourceCharacterCodePoint": "0095","TargetCharacterCodePoint": "01B5"}
,{"SourceCharacterCodePoint": "0096","TargetCharacterCodePoint": "01B6"}
,{"SourceCharacterCodePoint": "0004","TargetCharacterCodePoint": "01B7"}
,{"SourceCharacterCodePoint": "0098","TargetCharacterCodePoint": "01B8"}
,{"SourceCharacterCodePoint": "0099","TargetCharacterCodePoint": "01B9"}
,{"SourceCharacterCodePoint": "009A","TargetCharacterCodePoint": "01BA"}
,{"SourceCharacterCodePoint": "009B","TargetCharacterCodePoint": "01BB"}
,{"SourceCharacterCodePoint": "0014","TargetCharacterCodePoint": "01BC"}
,{"SourceCharacterCodePoint": "0015","TargetCharacterCodePoint": "01BD"}
,{"SourceCharacterCodePoint": "009E","TargetCharacterCodePoint": "01BE"}
,{"SourceCharacterCodePoint": "001A","TargetCharacterCodePoint": "01BF"}
,{"SourceCharacterCodePoint": "009F","TargetCharacterCodePoint": "027F"}
```

es-ES-284b and **es-ES-1145b**

Code Shift:

```
0180    0180
0001    0181
0002    0182
0003    0183
009C    0184
0009    0185
0086    0186
007F    0187
```

```
0097    0188
008D    0189
008E    018A
000B    018B
000C    018C
000D    018D
000E    018E
000F    018F
0010    0190
0011    0191
0012    0192
0013    0193
009D    0194
0085    0195
0008    0196
0087    0197
0018    0198
0019    0199
0092    019A
008F    019B
001C    019C
001D    019D
001E    019E
001F    019F
0080    01A0
0081    01A1
0082    01A2
0083    01A3
0084    01A4
000A    01A5
0017    01A6
001B    01A7
0088    01A8
0089    01A9
008A    01AA
008B    01AB
008C    01AC
0005    01AD
0006    01AE
0007    01AF
0090    01B0
0091    01B1
0016    01B2
0093    01B3
```

```
0094    01B4
0095    01B5
0096    01B6
0004    01B7
0098    01B8
0099    01B9
009A    01BA
009B    01BB
0014    01BC
0015    01BD
009E    01BE
001A    01BF
009F    027F
```

Corresponding input mapping for an AWS DMS task:

```
 {"SourceCharacterCodePoint": "0180","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "0001","TargetCharacterCodePoint": "0181"}
,{"SourceCharacterCodePoint": "0002","TargetCharacterCodePoint": "0182"}
,{"SourceCharacterCodePoint": "0003","TargetCharacterCodePoint": "0183"}
,{"SourceCharacterCodePoint": "009C","TargetCharacterCodePoint": "0184"}
,{"SourceCharacterCodePoint": "0009","TargetCharacterCodePoint": "0185"}
,{"SourceCharacterCodePoint": "0086","TargetCharacterCodePoint": "0186"}
,{"SourceCharacterCodePoint": "007F","TargetCharacterCodePoint": "0187"}
,{"SourceCharacterCodePoint": "0097","TargetCharacterCodePoint": "0188"}
,{"SourceCharacterCodePoint": "008D","TargetCharacterCodePoint": "0189"}
,{"SourceCharacterCodePoint": "008E","TargetCharacterCodePoint": "018A"}
,{"SourceCharacterCodePoint": "000B","TargetCharacterCodePoint": "018B"}
,{"SourceCharacterCodePoint": "000C","TargetCharacterCodePoint": "018C"}
,{"SourceCharacterCodePoint": "000D","TargetCharacterCodePoint": "018D"}
,{"SourceCharacterCodePoint": "000E","TargetCharacterCodePoint": "018E"}
,{"SourceCharacterCodePoint": "000F","TargetCharacterCodePoint": "018F"}
,{"SourceCharacterCodePoint": "0010","TargetCharacterCodePoint": "0190"}
,{"SourceCharacterCodePoint": "0011","TargetCharacterCodePoint": "0191"}
,{"SourceCharacterCodePoint": "0012","TargetCharacterCodePoint": "0192"}
,{"SourceCharacterCodePoint": "0013","TargetCharacterCodePoint": "0193"}
,{"SourceCharacterCodePoint": "009D","TargetCharacterCodePoint": "0194"}
,{"SourceCharacterCodePoint": "0085","TargetCharacterCodePoint": "0195"}
,{"SourceCharacterCodePoint": "0008","TargetCharacterCodePoint": "0196"}
,{"SourceCharacterCodePoint": "0087","TargetCharacterCodePoint": "0197"}
,{"SourceCharacterCodePoint": "0018","TargetCharacterCodePoint": "0198"}
,{"SourceCharacterCodePoint": "0019","TargetCharacterCodePoint": "0199"}
```

```
,{"SourceCharacterCodePoint": "0092","TargetCharacterCodePoint": "019A"}
,{"SourceCharacterCodePoint": "008F","TargetCharacterCodePoint": "019B"}
,{"SourceCharacterCodePoint": "001C","TargetCharacterCodePoint": "019C"}
,{"SourceCharacterCodePoint": "001D","TargetCharacterCodePoint": "019D"}
,{"SourceCharacterCodePoint": "001E","TargetCharacterCodePoint": "019E"}
,{"SourceCharacterCodePoint": "001F","TargetCharacterCodePoint": "019F"}
,{"SourceCharacterCodePoint": "0080","TargetCharacterCodePoint": "01A0"}
,{"SourceCharacterCodePoint": "0081","TargetCharacterCodePoint": "01A1"}
,{"SourceCharacterCodePoint": "0082","TargetCharacterCodePoint": "01A2"}
,{"SourceCharacterCodePoint": "0083","TargetCharacterCodePoint": "01A3"}
,{"SourceCharacterCodePoint": "0084","TargetCharacterCodePoint": "01A4"}
,{"SourceCharacterCodePoint": "000A","TargetCharacterCodePoint": "01A5"}
,{"SourceCharacterCodePoint": "0017","TargetCharacterCodePoint": "01A6"}
,{"SourceCharacterCodePoint": "001B","TargetCharacterCodePoint": "01A7"}
,{"SourceCharacterCodePoint": "0088","TargetCharacterCodePoint": "01A8"}
,{"SourceCharacterCodePoint": "0089","TargetCharacterCodePoint": "01A9"}
,{"SourceCharacterCodePoint": "008A","TargetCharacterCodePoint": "01AA"}
,{"SourceCharacterCodePoint": "008B","TargetCharacterCodePoint": "01AB"}
,{"SourceCharacterCodePoint": "008C","TargetCharacterCodePoint": "01AC"}
,{"SourceCharacterCodePoint": "0005","TargetCharacterCodePoint": "01AD"}
,{"SourceCharacterCodePoint": "0006","TargetCharacterCodePoint": "01AE"}
,{"SourceCharacterCodePoint": "0007","TargetCharacterCodePoint": "01AF"}
,{"SourceCharacterCodePoint": "0090","TargetCharacterCodePoint": "01B0"}
,{"SourceCharacterCodePoint": "0091","TargetCharacterCodePoint": "01B1"}
,{"SourceCharacterCodePoint": "0016","TargetCharacterCodePoint": "01B2"}
,{"SourceCharacterCodePoint": "0093","TargetCharacterCodePoint": "01B3"}
,{"SourceCharacterCodePoint": "0094","TargetCharacterCodePoint": "01B4"}
,{"SourceCharacterCodePoint": "0095","TargetCharacterCodePoint": "01B5"}
,{"SourceCharacterCodePoint": "0096","TargetCharacterCodePoint": "01B6"}
,{"SourceCharacterCodePoint": "0004","TargetCharacterCodePoint": "01B7"}
,{"SourceCharacterCodePoint": "0098","TargetCharacterCodePoint": "01B8"}
,{"SourceCharacterCodePoint": "0099","TargetCharacterCodePoint": "01B9"}
,{"SourceCharacterCodePoint": "009A","TargetCharacterCodePoint": "01BA"}
,{"SourceCharacterCodePoint": "009B","TargetCharacterCodePoint": "01BB"}
,{"SourceCharacterCodePoint": "0014","TargetCharacterCodePoint": "01BC"}
,{"SourceCharacterCodePoint": "0015","TargetCharacterCodePoint": "01BD"}
,{"SourceCharacterCodePoint": "009E","TargetCharacterCodePoint": "01BE"}
,{"SourceCharacterCodePoint": "001A","TargetCharacterCodePoint": "01BF"}
,{"SourceCharacterCodePoint": "009F","TargetCharacterCodePoint": "027F"}
```

fi_FI-278b and **fi-FI-1143b**

Code Shift:

```
0180    0180
0001    0181
0002    0182
0003    0183
009C    0184
0009    0185
0086    0186
007F    0187
0097    0188
008D    0189
008E    018A
000B    018B
000C    018C
000D    018D
000E    018E
000F    018F
0010    0190
0011    0191
0012    0192
0013    0193
009D    0194
0085    0195
0008    0196
0087    0197
0018    0198
0019    0199
0092    019A
008F    019B
001C    019C
001D    019D
001E    019E
001F    019F
0080    01A0
0081    01A1
0082    01A2
0083    01A3
0084    01A4
000A    01A5
0017    01A6
001B    01A7
0088    01A8
0089    01A9
```

```
008A    01AA
008B    01AB
008C    01AC
0005    01AD
0006    01AE
0007    01AF
0090    01B0
0091    01B1
0016    01B2
0093    01B3
0094    01B4
0095    01B5
0096    01B6
0004    01B7
0098    01B8
0099    01B9
009A    01BA
009B    01BB
0014    01BC
0015    01BD
009E    01BE
001A    01BF
009F    027F
```

Corresponding input mapping for an AWS DMS task:

```
 {"SourceCharacterCodePoint": "0180","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "0001","TargetCharacterCodePoint": "0181"}
,{"SourceCharacterCodePoint": "0002","TargetCharacterCodePoint": "0182"}
,{"SourceCharacterCodePoint": "0003","TargetCharacterCodePoint": "0183"}
,{"SourceCharacterCodePoint": "009C","TargetCharacterCodePoint": "0184"}
,{"SourceCharacterCodePoint": "0009","TargetCharacterCodePoint": "0185"}
,{"SourceCharacterCodePoint": "0086","TargetCharacterCodePoint": "0186"}
,{"SourceCharacterCodePoint": "007F","TargetCharacterCodePoint": "0187"}
,{"SourceCharacterCodePoint": "0097","TargetCharacterCodePoint": "0188"}
,{"SourceCharacterCodePoint": "008D","TargetCharacterCodePoint": "0189"}
,{"SourceCharacterCodePoint": "008E","TargetCharacterCodePoint": "018A"}
,{"SourceCharacterCodePoint": "000B","TargetCharacterCodePoint": "018B"}
,{"SourceCharacterCodePoint": "000C","TargetCharacterCodePoint": "018C"}
,{"SourceCharacterCodePoint": "000D","TargetCharacterCodePoint": "018D"}
,{"SourceCharacterCodePoint": "000E","TargetCharacterCodePoint": "018E"}
,{"SourceCharacterCodePoint": "000F","TargetCharacterCodePoint": "018F"}
```

```
,{"SourceCharacterCodePoint": "0010","TargetCharacterCodePoint": "0190"}
,{"SourceCharacterCodePoint": "0011","TargetCharacterCodePoint": "0191"}
,{"SourceCharacterCodePoint": "0012","TargetCharacterCodePoint": "0192"}
,{"SourceCharacterCodePoint": "0013","TargetCharacterCodePoint": "0193"}
,{"SourceCharacterCodePoint": "009D","TargetCharacterCodePoint": "0194"}
,{"SourceCharacterCodePoint": "0085","TargetCharacterCodePoint": "0195"}
,{"SourceCharacterCodePoint": "0008","TargetCharacterCodePoint": "0196"}
,{"SourceCharacterCodePoint": "0087","TargetCharacterCodePoint": "0197"}
,{"SourceCharacterCodePoint": "0018","TargetCharacterCodePoint": "0198"}
,{"SourceCharacterCodePoint": "0019","TargetCharacterCodePoint": "0199"}
,{"SourceCharacterCodePoint": "0092","TargetCharacterCodePoint": "019A"}
,{"SourceCharacterCodePoint": "008F","TargetCharacterCodePoint": "019B"}
,{"SourceCharacterCodePoint": "001C","TargetCharacterCodePoint": "019C"}
,{"SourceCharacterCodePoint": "001D","TargetCharacterCodePoint": "019D"}
,{"SourceCharacterCodePoint": "001E","TargetCharacterCodePoint": "019E"}
,{"SourceCharacterCodePoint": "001F","TargetCharacterCodePoint": "019F"}
,{"SourceCharacterCodePoint": "0080","TargetCharacterCodePoint": "01A0"}
,{"SourceCharacterCodePoint": "0081","TargetCharacterCodePoint": "01A1"}
,{"SourceCharacterCodePoint": "0082","TargetCharacterCodePoint": "01A2"}
,{"SourceCharacterCodePoint": "0083","TargetCharacterCodePoint": "01A3"}
,{"SourceCharacterCodePoint": "0084","TargetCharacterCodePoint": "01A4"}
,{"SourceCharacterCodePoint": "000A","TargetCharacterCodePoint": "01A5"}
,{"SourceCharacterCodePoint": "0017","TargetCharacterCodePoint": "01A6"}
,{"SourceCharacterCodePoint": "001B","TargetCharacterCodePoint": "01A7"}
,{"SourceCharacterCodePoint": "0088","TargetCharacterCodePoint": "01A8"}
,{"SourceCharacterCodePoint": "0089","TargetCharacterCodePoint": "01A9"}
,{"SourceCharacterCodePoint": "008A","TargetCharacterCodePoint": "01AA"}
,{"SourceCharacterCodePoint": "008B","TargetCharacterCodePoint": "01AB"}
,{"SourceCharacterCodePoint": "008C","TargetCharacterCodePoint": "01AC"}
,{"SourceCharacterCodePoint": "0005","TargetCharacterCodePoint": "01AD"}
,{"SourceCharacterCodePoint": "0006","TargetCharacterCodePoint": "01AE"}
,{"SourceCharacterCodePoint": "0007","TargetCharacterCodePoint": "01AF"}
,{"SourceCharacterCodePoint": "0090","TargetCharacterCodePoint": "01B0"}
,{"SourceCharacterCodePoint": "0091","TargetCharacterCodePoint": "01B1"}
,{"SourceCharacterCodePoint": "0016","TargetCharacterCodePoint": "01B2"}
,{"SourceCharacterCodePoint": "0093","TargetCharacterCodePoint": "01B3"}
,{"SourceCharacterCodePoint": "0094","TargetCharacterCodePoint": "01B4"}
,{"SourceCharacterCodePoint": "0095","TargetCharacterCodePoint": "01B5"}
,{"SourceCharacterCodePoint": "0096","TargetCharacterCodePoint": "01B6"}
,{"SourceCharacterCodePoint": "0004","TargetCharacterCodePoint": "01B7"}
,{"SourceCharacterCodePoint": "0098","TargetCharacterCodePoint": "01B8"}
,{"SourceCharacterCodePoint": "0099","TargetCharacterCodePoint": "01B9"}
,{"SourceCharacterCodePoint": "009A","TargetCharacterCodePoint": "01BA"}
,{"SourceCharacterCodePoint": "009B","TargetCharacterCodePoint": "01BB"}
```

```
,{"SourceCharacterCodePoint": "0014","TargetCharacterCodePoint": "01BC"}
,{"SourceCharacterCodePoint": "0015","TargetCharacterCodePoint": "01BD"}
,{"SourceCharacterCodePoint": "009E","TargetCharacterCodePoint": "01BE"}
,{"SourceCharacterCodePoint": "001A","TargetCharacterCodePoint": "01BF"}
,{"SourceCharacterCodePoint": "009F","TargetCharacterCodePoint": "027F"}
```

fr-FR-297b and **fr-FR-1147b**

Code Shift:

```
0180    0180
0001    0181
0002    0182
0003    0183
009C    0184
0009    0185
0086    0186
007F    0187
0097    0188
008D    0189
008E    018A
000B    018B
000C    018C
000D    018D
000E    018E
000F    018F
0010    0190
0011    0191
0012    0192
0013    0193
009D    0194
0085    0195
0008    0196
0087    0197
0018    0198
0019    0199
0092    019A
008F    019B
001C    019C
001D    019D
001E    019E
001F    019F
0080    01A0
```

```
0081    01A1
0082    01A2
0083    01A3
0084    01A4
000A    01A5
0017    01A6
001B    01A7
0088    01A8
0089    01A9
008A    01AA
008B    01AB
008C    01AC
0005    01AD
0006    01AE
0007    01AF
0090    01B0
0091    01B1
0016    01B2
0093    01B3
0094    01B4
0095    01B5
0096    01B6
0004    01B7
0098    01B8
0099    01B9
009A    01BA
009B    01BB
0014    01BC
0015    01BD
009E    01BE
001A    01BF
009F    027F
```

Corresponding input mapping for an AWS DMS task:

```
{"SourceCharacterCodePoint": "0180","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "0001","TargetCharacterCodePoint": "0181"}
,{"SourceCharacterCodePoint": "0002","TargetCharacterCodePoint": "0182"}
,{"SourceCharacterCodePoint": "0003","TargetCharacterCodePoint": "0183"}
,{"SourceCharacterCodePoint": "009C","TargetCharacterCodePoint": "0184"}
,{"SourceCharacterCodePoint": "0009","TargetCharacterCodePoint": "0185"}
,{"SourceCharacterCodePoint": "0086","TargetCharacterCodePoint": "0186"}
```

```
,{"SourceCharacterCodePoint": "007F","TargetCharacterCodePoint": "0187"}
,{"SourceCharacterCodePoint": "0097","TargetCharacterCodePoint": "0188"}
,{"SourceCharacterCodePoint": "008D","TargetCharacterCodePoint": "0189"}
,{"SourceCharacterCodePoint": "008E","TargetCharacterCodePoint": "018A"}
,{"SourceCharacterCodePoint": "000B","TargetCharacterCodePoint": "018B"}
,{"SourceCharacterCodePoint": "000C","TargetCharacterCodePoint": "018C"}
,{"SourceCharacterCodePoint": "000D","TargetCharacterCodePoint": "018D"}
,{"SourceCharacterCodePoint": "000E","TargetCharacterCodePoint": "018E"}
,{"SourceCharacterCodePoint": "000F","TargetCharacterCodePoint": "018F"}
,{"SourceCharacterCodePoint": "0010","TargetCharacterCodePoint": "0190"}
,{"SourceCharacterCodePoint": "0011","TargetCharacterCodePoint": "0191"}
,{"SourceCharacterCodePoint": "0012","TargetCharacterCodePoint": "0192"}
,{"SourceCharacterCodePoint": "0013","TargetCharacterCodePoint": "0193"}
,{"SourceCharacterCodePoint": "009D","TargetCharacterCodePoint": "0194"}
,{"SourceCharacterCodePoint": "0085","TargetCharacterCodePoint": "0195"}
,{"SourceCharacterCodePoint": "0008","TargetCharacterCodePoint": "0196"}
,{"SourceCharacterCodePoint": "0087","TargetCharacterCodePoint": "0197"}
,{"SourceCharacterCodePoint": "0018","TargetCharacterCodePoint": "0198"}
,{"SourceCharacterCodePoint": "0019","TargetCharacterCodePoint": "0199"}
,{"SourceCharacterCodePoint": "0092","TargetCharacterCodePoint": "019A"}
,{"SourceCharacterCodePoint": "008F","TargetCharacterCodePoint": "019B"}
,{"SourceCharacterCodePoint": "001C","TargetCharacterCodePoint": "019C"}
,{"SourceCharacterCodePoint": "001D","TargetCharacterCodePoint": "019D"}
,{"SourceCharacterCodePoint": "001E","TargetCharacterCodePoint": "019E"}
,{"SourceCharacterCodePoint": "001F","TargetCharacterCodePoint": "019F"}
,{"SourceCharacterCodePoint": "0080","TargetCharacterCodePoint": "01A0"}
,{"SourceCharacterCodePoint": "0081","TargetCharacterCodePoint": "01A1"}
,{"SourceCharacterCodePoint": "0082","TargetCharacterCodePoint": "01A2"}
,{"SourceCharacterCodePoint": "0083","TargetCharacterCodePoint": "01A3"}
,{"SourceCharacterCodePoint": "0084","TargetCharacterCodePoint": "01A4"}
,{"SourceCharacterCodePoint": "000A","TargetCharacterCodePoint": "01A5"}
,{"SourceCharacterCodePoint": "0017","TargetCharacterCodePoint": "01A6"}
,{"SourceCharacterCodePoint": "001B","TargetCharacterCodePoint": "01A7"}
,{"SourceCharacterCodePoint": "0088","TargetCharacterCodePoint": "01A8"}
,{"SourceCharacterCodePoint": "0089","TargetCharacterCodePoint": "01A9"}
,{"SourceCharacterCodePoint": "008A","TargetCharacterCodePoint": "01AA"}
,{"SourceCharacterCodePoint": "008B","TargetCharacterCodePoint": "01AB"}
,{"SourceCharacterCodePoint": "008C","TargetCharacterCodePoint": "01AC"}
,{"SourceCharacterCodePoint": "0005","TargetCharacterCodePoint": "01AD"}
,{"SourceCharacterCodePoint": "0006","TargetCharacterCodePoint": "01AE"}
,{"SourceCharacterCodePoint": "0007","TargetCharacterCodePoint": "01AF"}
,{"SourceCharacterCodePoint": "0090","TargetCharacterCodePoint": "01B0"}
,{"SourceCharacterCodePoint": "0091","TargetCharacterCodePoint": "01B1"}
,{"SourceCharacterCodePoint": "0016","TargetCharacterCodePoint": "01B2"}
```

```
,{"SourceCharacterCodePoint": "0093","TargetCharacterCodePoint": "01B3"}
,{"SourceCharacterCodePoint": "0094","TargetCharacterCodePoint": "01B4"}
,{"SourceCharacterCodePoint": "0095","TargetCharacterCodePoint": "01B5"}
,{"SourceCharacterCodePoint": "0096","TargetCharacterCodePoint": "01B6"}
,{"SourceCharacterCodePoint": "0004","TargetCharacterCodePoint": "01B7"}
,{"SourceCharacterCodePoint": "0098","TargetCharacterCodePoint": "01B8"}
,{"SourceCharacterCodePoint": "0099","TargetCharacterCodePoint": "01B9"}
,{"SourceCharacterCodePoint": "009A","TargetCharacterCodePoint": "01BA"}
,{"SourceCharacterCodePoint": "009B","TargetCharacterCodePoint": "01BB"}
,{"SourceCharacterCodePoint": "0014","TargetCharacterCodePoint": "01BC"}
,{"SourceCharacterCodePoint": "0015","TargetCharacterCodePoint": "01BD"}
,{"SourceCharacterCodePoint": "009E","TargetCharacterCodePoint": "01BE"}
,{"SourceCharacterCodePoint": "001A","TargetCharacterCodePoint": "01BF"}
,{"SourceCharacterCodePoint": "009F","TargetCharacterCodePoint": "027F"}
```

it-IT-280b and **it-IT-1144b**

Code Shift:

```
0180    0180
0001    0181
0002    0182
0003    0183
009C    0184
0009    0185
0086    0186
007F    0187
0097    0188
008D    0189
008E    018A
000B    018B
000C    018C
000D    018D
000E    018E
000F    018F
0010    0190
0011    0191
0012    0192
0013    0193
009D    0194
0085    0195
0008    0196
```

```
0087    0197
0018    0198
0019    0199
0092    019A
008F    019B
001C    019C
001D    019D
001E    019E
001F    019F
0080    01A0
0081    01A1
0082    01A2
0083    01A3
0084    01A4
000A    01A5
0017    01A6
001B    01A7
0088    01A8
0089    01A9
008A    01AA
008B    01AB
008C    01AC
0005    01AD
0006    01AE
0007    01AF
0090    01B0
0091    01B1
0016    01B2
0093    01B3
0094    01B4
0095    01B5
0096    01B6
0004    01B7
0098    01B8
0099    01B9
009A    01BA
009B    01BB
0014    01BC
0015    01BD
009E    01BE
001A    01BF
009F    027F
```

Corresponding input mapping for an AWS DMS task:

```
 {"SourceCharacterCodePoint": "0180","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "0001","TargetCharacterCodePoint": "0181"}
,{"SourceCharacterCodePoint": "0002","TargetCharacterCodePoint": "0182"}
,{"SourceCharacterCodePoint": "0003","TargetCharacterCodePoint": "0183"}
,{"SourceCharacterCodePoint": "009C","TargetCharacterCodePoint": "0184"}
,{"SourceCharacterCodePoint": "0009","TargetCharacterCodePoint": "0185"}
,{"SourceCharacterCodePoint": "0086","TargetCharacterCodePoint": "0186"}
,{"SourceCharacterCodePoint": "007F","TargetCharacterCodePoint": "0187"}
,{"SourceCharacterCodePoint": "0097","TargetCharacterCodePoint": "0188"}
,{"SourceCharacterCodePoint": "008D","TargetCharacterCodePoint": "0189"}
,{"SourceCharacterCodePoint": "008E","TargetCharacterCodePoint": "018A"}
,{"SourceCharacterCodePoint": "000B","TargetCharacterCodePoint": "018B"}
,{"SourceCharacterCodePoint": "000C","TargetCharacterCodePoint": "018C"}
,{"SourceCharacterCodePoint": "000D","TargetCharacterCodePoint": "018D"}
,{"SourceCharacterCodePoint": "000E","TargetCharacterCodePoint": "018E"}
,{"SourceCharacterCodePoint": "000F","TargetCharacterCodePoint": "018F"}
,{"SourceCharacterCodePoint": "0010","TargetCharacterCodePoint": "0190"}
,{"SourceCharacterCodePoint": "0011","TargetCharacterCodePoint": "0191"}
,{"SourceCharacterCodePoint": "0012","TargetCharacterCodePoint": "0192"}
,{"SourceCharacterCodePoint": "0013","TargetCharacterCodePoint": "0193"}
,{"SourceCharacterCodePoint": "009D","TargetCharacterCodePoint": "0194"}
,{"SourceCharacterCodePoint": "0085","TargetCharacterCodePoint": "0195"}
,{"SourceCharacterCodePoint": "0008","TargetCharacterCodePoint": "0196"}
,{"SourceCharacterCodePoint": "0087","TargetCharacterCodePoint": "0197"}
,{"SourceCharacterCodePoint": "0018","TargetCharacterCodePoint": "0198"}
,{"SourceCharacterCodePoint": "0019","TargetCharacterCodePoint": "0199"}
,{"SourceCharacterCodePoint": "0092","TargetCharacterCodePoint": "019A"}
,{"SourceCharacterCodePoint": "008F","TargetCharacterCodePoint": "019B"}
,{"SourceCharacterCodePoint": "001C","TargetCharacterCodePoint": "019C"}
,{"SourceCharacterCodePoint": "001D","TargetCharacterCodePoint": "019D"}
,{"SourceCharacterCodePoint": "001E","TargetCharacterCodePoint": "019E"}
,{"SourceCharacterCodePoint": "001F","TargetCharacterCodePoint": "019F"}
,{"SourceCharacterCodePoint": "0080","TargetCharacterCodePoint": "01A0"}
,{"SourceCharacterCodePoint": "0081","TargetCharacterCodePoint": "01A1"}
,{"SourceCharacterCodePoint": "0082","TargetCharacterCodePoint": "01A2"}
,{"SourceCharacterCodePoint": "0083","TargetCharacterCodePoint": "01A3"}
,{"SourceCharacterCodePoint": "0084","TargetCharacterCodePoint": "01A4"}
,{"SourceCharacterCodePoint": "000A","TargetCharacterCodePoint": "01A5"}
,{"SourceCharacterCodePoint": "0017","TargetCharacterCodePoint": "01A6"}
,{"SourceCharacterCodePoint": "001B","TargetCharacterCodePoint": "01A7"}
,{"SourceCharacterCodePoint": "0088","TargetCharacterCodePoint": "01A8"}
,{"SourceCharacterCodePoint": "0089","TargetCharacterCodePoint": "01A9"}
```

```
,{"SourceCharacterCodePoint": "008A","TargetCharacterCodePoint": "01AA"}
,{"SourceCharacterCodePoint": "008B","TargetCharacterCodePoint": "01AB"}
,{"SourceCharacterCodePoint": "008C","TargetCharacterCodePoint": "01AC"}
,{"SourceCharacterCodePoint": "0005","TargetCharacterCodePoint": "01AD"}
,{"SourceCharacterCodePoint": "0006","TargetCharacterCodePoint": "01AE"}
,{"SourceCharacterCodePoint": "0007","TargetCharacterCodePoint": "01AF"}
,{"SourceCharacterCodePoint": "0090","TargetCharacterCodePoint": "01B0"}
,{"SourceCharacterCodePoint": "0091","TargetCharacterCodePoint": "01B1"}
,{"SourceCharacterCodePoint": "0016","TargetCharacterCodePoint": "01B2"}
,{"SourceCharacterCodePoint": "0093","TargetCharacterCodePoint": "01B3"}
,{"SourceCharacterCodePoint": "0094","TargetCharacterCodePoint": "01B4"}
,{"SourceCharacterCodePoint": "0095","TargetCharacterCodePoint": "01B5"}
,{"SourceCharacterCodePoint": "0096","TargetCharacterCodePoint": "01B6"}
,{"SourceCharacterCodePoint": "0004","TargetCharacterCodePoint": "01B7"}
,{"SourceCharacterCodePoint": "0098","TargetCharacterCodePoint": "01B8"}
,{"SourceCharacterCodePoint": "0099","TargetCharacterCodePoint": "01B9"}
,{"SourceCharacterCodePoint": "009A","TargetCharacterCodePoint": "01BA"}
,{"SourceCharacterCodePoint": "009B","TargetCharacterCodePoint": "01BB"}
,{"SourceCharacterCodePoint": "0014","TargetCharacterCodePoint": "01BC"}
,{"SourceCharacterCodePoint": "0015","TargetCharacterCodePoint": "01BD"}
,{"SourceCharacterCodePoint": "009E","TargetCharacterCodePoint": "01BE"}
,{"SourceCharacterCodePoint": "001A","TargetCharacterCodePoint": "01BF"}
,{"SourceCharacterCodePoint": "009F","TargetCharacterCodePoint": "027F"}
```

nl-BE-500b and **nl-BE-1148b**

Code Shift:

```
0180    0180
0001    0181
0002    0182
0003    0183
009C    0184
0009    0185
0086    0186
007F    0187
0097    0188
008D    0189
008E    018A
000B    018B
000C    018C
000D    018D
```

```
000E    018E
000F    018F
0010    0190
0011    0191
0012    0192
0013    0193
009D    0194
0085    0195
0008    0196
0087    0197
0018    0198
0019    0199
0092    019A
008F    019B
001C    019C
001D    019D
001E    019E
001F    019F
0080    01A0
0081    01A1
0082    01A2
0083    01A3
0084    01A4
000A    01A5
0017    01A6
001B    01A7
0088    01A8
0089    01A9
008A    01AA
008B    01AB
008C    01AC
0005    01AD
0006    01AE
0007    01AF
0090    01B0
0091    01B1
0016    01B2
0093    01B3
0094    01B4
0095    01B5
0096    01B6
0004    01B7
0098    01B8
0099    01B9
```

```
009A    01BA
009B    01BB
0014    01BC
0015    01BD
009E    01BE
001A    01BF
009F    027F
```

Corresponding input mapping for an AWS DMS task:

```
 {"SourceCharacterCodePoint": "0180","TargetCharacterCodePoint": "0180"}
,{"SourceCharacterCodePoint": "0001","TargetCharacterCodePoint": "0181"}
,{"SourceCharacterCodePoint": "0002","TargetCharacterCodePoint": "0182"}
,{"SourceCharacterCodePoint": "0003","TargetCharacterCodePoint": "0183"}
,{"SourceCharacterCodePoint": "009C","TargetCharacterCodePoint": "0184"}
,{"SourceCharacterCodePoint": "0009","TargetCharacterCodePoint": "0185"}
,{"SourceCharacterCodePoint": "0086","TargetCharacterCodePoint": "0186"}
,{"SourceCharacterCodePoint": "007F","TargetCharacterCodePoint": "0187"}
,{"SourceCharacterCodePoint": "0097","TargetCharacterCodePoint": "0188"}
,{"SourceCharacterCodePoint": "008D","TargetCharacterCodePoint": "0189"}
,{"SourceCharacterCodePoint": "008E","TargetCharacterCodePoint": "018A"}
,{"SourceCharacterCodePoint": "000B","TargetCharacterCodePoint": "018B"}
,{"SourceCharacterCodePoint": "000C","TargetCharacterCodePoint": "018C"}
,{"SourceCharacterCodePoint": "000D","TargetCharacterCodePoint": "018D"}
,{"SourceCharacterCodePoint": "000E","TargetCharacterCodePoint": "018E"}
,{"SourceCharacterCodePoint": "000F","TargetCharacterCodePoint": "018F"}
,{"SourceCharacterCodePoint": "0010","TargetCharacterCodePoint": "0190"}
,{"SourceCharacterCodePoint": "0011","TargetCharacterCodePoint": "0191"}
,{"SourceCharacterCodePoint": "0012","TargetCharacterCodePoint": "0192"}
,{"SourceCharacterCodePoint": "0013","TargetCharacterCodePoint": "0193"}
,{"SourceCharacterCodePoint": "009D","TargetCharacterCodePoint": "0194"}
,{"SourceCharacterCodePoint": "0085","TargetCharacterCodePoint": "0195"}
,{"SourceCharacterCodePoint": "0008","TargetCharacterCodePoint": "0196"}
,{"SourceCharacterCodePoint": "0087","TargetCharacterCodePoint": "0197"}
,{"SourceCharacterCodePoint": "0018","TargetCharacterCodePoint": "0198"}
,{"SourceCharacterCodePoint": "0019","TargetCharacterCodePoint": "0199"}
,{"SourceCharacterCodePoint": "0092","TargetCharacterCodePoint": "019A"}
,{"SourceCharacterCodePoint": "008F","TargetCharacterCodePoint": "019B"}
,{"SourceCharacterCodePoint": "001C","TargetCharacterCodePoint": "019C"}
,{"SourceCharacterCodePoint": "001D","TargetCharacterCodePoint": "019D"}
,{"SourceCharacterCodePoint": "001E","TargetCharacterCodePoint": "019E"}
,{"SourceCharacterCodePoint": "001F","TargetCharacterCodePoint": "019F"}
```

```
,{"SourceCharacterCodePoint": "0080","TargetCharacterCodePoint": "01A0"}
,{"SourceCharacterCodePoint": "0081","TargetCharacterCodePoint": "01A1"}
,{"SourceCharacterCodePoint": "0082","TargetCharacterCodePoint": "01A2"}
,{"SourceCharacterCodePoint": "0083","TargetCharacterCodePoint": "01A3"}
,{"SourceCharacterCodePoint": "0084","TargetCharacterCodePoint": "01A4"}
,{"SourceCharacterCodePoint": "000A","TargetCharacterCodePoint": "01A5"}
,{"SourceCharacterCodePoint": "0017","TargetCharacterCodePoint": "01A6"}
,{"SourceCharacterCodePoint": "001B","TargetCharacterCodePoint": "01A7"}
,{"SourceCharacterCodePoint": "0088","TargetCharacterCodePoint": "01A8"}
,{"SourceCharacterCodePoint": "0089","TargetCharacterCodePoint": "01A9"}
,{"SourceCharacterCodePoint": "008A","TargetCharacterCodePoint": "01AA"}
,{"SourceCharacterCodePoint": "008B","TargetCharacterCodePoint": "01AB"}
,{"SourceCharacterCodePoint": "008C","TargetCharacterCodePoint": "01AC"}
,{"SourceCharacterCodePoint": "0005","TargetCharacterCodePoint": "01AD"}
,{"SourceCharacterCodePoint": "0006","TargetCharacterCodePoint": "01AE"}
,{"SourceCharacterCodePoint": "0007","TargetCharacterCodePoint": "01AF"}
,{"SourceCharacterCodePoint": "0090","TargetCharacterCodePoint": "01B0"}
,{"SourceCharacterCodePoint": "0091","TargetCharacterCodePoint": "01B1"}
,{"SourceCharacterCodePoint": "0016","TargetCharacterCodePoint": "01B2"}
,{"SourceCharacterCodePoint": "0093","TargetCharacterCodePoint": "01B3"}
,{"SourceCharacterCodePoint": "0094","TargetCharacterCodePoint": "01B4"}
,{"SourceCharacterCodePoint": "0095","TargetCharacterCodePoint": "01B5"}
,{"SourceCharacterCodePoint": "0096","TargetCharacterCodePoint": "01B6"}
,{"SourceCharacterCodePoint": "0004","TargetCharacterCodePoint": "01B7"}
,{"SourceCharacterCodePoint": "0098","TargetCharacterCodePoint": "01B8"}
,{"SourceCharacterCodePoint": "0099","TargetCharacterCodePoint": "01B9"}
,{"SourceCharacterCodePoint": "009A","TargetCharacterCodePoint": "01BA"}
,{"SourceCharacterCodePoint": "009B","TargetCharacterCodePoint": "01BB"}
,{"SourceCharacterCodePoint": "0014","TargetCharacterCodePoint": "01BC"}
,{"SourceCharacterCodePoint": "0015","TargetCharacterCodePoint": "01BD"}
,{"SourceCharacterCodePoint": "009E","TargetCharacterCodePoint": "01BE"}
,{"SourceCharacterCodePoint": "001A","TargetCharacterCodePoint": "01BF"}
,{"SourceCharacterCodePoint": "009F","TargetCharacterCodePoint": "027F"}
```

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.

Note

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

- Amazon DynamoDB
- Amazon OpenSearch Service
- Amazon Kinesis Data Streams

Amazon Aurora PostgreSQL Limitless is available as a target for AWS Database Migration Service (AWS DMS). For more information see Using a PostgreSQL database as a target for AWS Database Migration Service.

Topics

- Using Amazon RDS for Db2 and IBM Db2 LUW as a target for AWS DMS

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. 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.

For information about versions of Oracle that AWS DMS supports as a target, see Targets for AWS DMS.

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 `RDSMASTER` and you want to migrate from the user `PERFDATA1` to `PERFDATA2`. 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": "PERFDATA1"
},
"value": "PERFDATA2"
}
```

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 tablespaces, 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.

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 Endpoint settings when using Oracle as a source for AWS DMS

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

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, you can create multiple replication tasks. You can also migrate data to different schemas on a target. To do this, you need to use schema transformation rules on the AWS DMS table mappings.
- 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.
- You can express the total number of columns per table supported in Batch optimized apply mode, using the following formula:

  ```
  2 * columns_in_original_table + columns_in_primary_key <= 999
  ```

  For example, if the original table has 25 columns and its Primary Key consists of 5 columns, then the total number of columns is 55. If a table exceeds the supported number of columns, then all of the changes are applied in one-by-one mode.
- AWS DMS doesn't support Autonomous DB on Oracle Cloud Infrastructure (OCI).
- In transactional apply mode, an Oracle target can process DML statements up to 32 KB in size. While this limit is sufficient for many use cases, DML statements exceeding 32 KB will fail with the error: "ORA-01460: unimplemented or unreasonable conversion requested." To resolve this issue, you must enable the batch apply feature by setting the `BatchApplyEnabled` task setting to `true`. Batch apply reduces the overall statement size, allowing you to bypass the 32 KB limitation. For more information, see the section called “Target metadata task settings”.
- AWS DMS Direct path full load for LOB tables may fail with error ORA-39777 due to special handling requirements for LOB data. This error occurs during the direct path load process and can disrupt migration tasks involving LOB columns. To resolve, disable the `useDirectPathFullLoad` setting on the target endpoint and retry the load operation.

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
- DELETE ANY TABLE

For the following requirements, 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 (which is the default for full load), `GRANT LOCK ANY TABLE to` *`dms_user`*`;`.
- If schema is different when using "DROP and CREATE" table prep mode, `GRANT CREATE ANY INDEX to` *`dms_user`*`;`.
- 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.
- To validate LOB columns with the validation feature, grant EXECUTE privelege on `SYS.DBMS_CRYPTO` to the DMS user.

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
- SELECT on SYS.DBA_REGISTRY

If any of the required privileges cannot be granted to V$xxx, then grant them to V_$xxx.

Premigration assessments

To use the premigration assessments listed in Oracle assessments with Oracle as a Target, you must add the following permissions to the user account specified in the Oracle database target endpoint:

```
GRANT SELECT ON V_$INSTANCE TO dms_user;
GRANT EXECUTE ON SYS.DBMS_XMLGEN TO dms_user;
```

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.

Endpoint settings when using Oracle as a target for AWS DMS

You can use endpoint settings to configure your Oracle target database similar to using extra connection attributes. You specify the settings when you create the target endpoint using the AWS DMS console, or by using the `create-endpoint` command in the AWS CLI, with the `--oracle-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following table shows the endpoint settings that you can use with 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: `true` Valid values: `true/false` Example: `--oracle-settings '{"UseDirectPathFullLoad": false}'`

Name	Description
`DirectPathParallelLoad`	When set to `true`, this attribute specifies a parallel load when `UseDirectPathFullLoad` is set to Y. 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 Table and collection settings rules and operations. 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 Enabling Constraints After a Parallel Direct Path Load. If constraints or indexes are enabled, setting this attribute to `true` has no effect. Default value: `false` Valid values: `true/false` Example: `--oracle-settings '{"DirectPathParallelLoad": true}'`
`DirectPathNoLog`	When set to `true`, 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 Direct-Load INSERT. This attribute also only applies when you set `UseDirectPathFullLoad` to Y. Default value: `false` Valid values: `true/false` Example: `--oracle-settings '{"DirectPathNoLog": true}'`

Name	Description
`CharLengthSemantics`	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 CHAR. Otherwise, the character column length is in bytes. Default value: Not set to CHAR Valid values: CHAR Example: `--oracle-settings '{"CharLengthSemantics": "CHAR"}'`
`AlwaysReplaceEmptyString`	AWS DMS adds an extra space to replicate an empty string when migrating to an Oracle target. In general, Oracle doesn't have a notation for an empty string. When you insert an empty string on varchar2, you load empty strings as NULL. If you want to insert the data as NULL on Oracle, set this attribute to FALSE. Default value: `true` Valid values: `true/false` Example: `--oracle-settings '{"AlwaysReplaceEmptyString": false}'`

Target data types for Oracle

A target Oracle database used with AWS DMS supports most Oracle data types. The following table shows the Oracle target data types that are supported when using AWS DMS and the default mapping from AWS DMS data types. For more information about how to view the data type that is mapped from the source, see the section for the source you are using.

AWS DMS data type	Oracle data type
BOOLEAN	NUMBER (1)
BYTES	RAW (length)
DATE	DATETIME
TIME	TIMESTAMP (0)
DATETIME	TIMESTAMP (scale)
INT1	NUMBER (3)
INT2	NUMBER (5)
INT4	NUMBER (10)
INT8	NUMBER (19)
NUMERIC	NUMBER (p,s)
REAL4	FLOAT
REAL8	FLOAT
STRING	With date indication: DATE With time indication: TIMESTAMP With timestamp indication: TIMESTAMP With timestamp_with_timezone indication: TIMESTAMP WITH TIMEZONE With timestamp_with_local_timezone indication: TIMESTAMP WITH LOCAL TIMEZONE With interval_year_to_month indication: INTERVAL YEAR TO MONTH With interval_day_to_second indication: INTERVAL DAY TO SECOND

AWS DMS data type	Oracle data type
	If length > 4000: CLOB In all other cases: VARCHAR2 (length)
UINT1	NUMBER (3)
UINT2	NUMBER (5)
UINT4	NUMBER (10)
UINT8	NUMBER (19)
WSTRING	If length > 2000: NCLOB In all other cases: NVARCHAR2 (length)
BLOB	BLOB To use this data type with AWS DMS, you must enable the use of BLOBs for a specific task. BLOB data types are supported only in tables that include a primary key
CLOB	CLOB To use this data type with AWS DMS, you must enable the use of CLOBs for a specific task. During change data capture (CDC), CLOB data types are supported only in tables that include a primary key. STRING 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.

AWS DMS data type	Oracle data type
NCLOB	NCLOB To use this data type with AWS DMS, you must enable the use of NCLOBs for a specific task. During CDC, NCLOB data types are supported only in tables that include a primary key. WSTRING 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.
XMLTYPE	The XMLTYPE target data type is only relevant in Oracle-to-Oracle replication tasks. When the source database is Oracle, the source data types are replicated as-is to the Oracle target. For example, an XMLTYPE data type on the source is created as an XMLTYPE data type on the target.

Using a Microsoft SQL Server database as a target for AWS Database Migration Service

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

For information about versions of SQL Server that AWS DMS supports as a target, see Targets for AWS DMS.

AWS DMS supports the on-premises and Amazon RDS editions of Enterprise, Standard, Workgroup, and Developer.

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

Topics

- Limitations on using SQL Server as a target for AWS Database Migration Service
- Security requirements when using SQL Server as a target for AWS Database Migration Service
- Endpoint settings when using SQL Server as a target for AWS DMS
- Target data types for Microsoft SQL Server

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 BCP loading by setting the extra connection attribute (ECA) `'useBCPFullLoad=false'` on the endpoint. For information about setting ECAs on endpoints, see Creating source and target endpoints. 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, do the following:

  - 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.
  - To avoid having to change downstream processing, add a transformation rule to the task to change the "F" values to "0" and "T" values to 1, and store them as the SQL server bit datatype.
- AWS DMS doesn't support change processing to set column nullability (using the `ALTER COLUMN [SET|DROP] NOT NULL` clause with `ALTER TABLE` statements).
- Windows Authentication isn't supported.

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.

Endpoint settings when using SQL Server as a target for AWS DMS

You can use endpoint settings to configure your SQL Server target database similar to using extra connection attributes. You specify the settings when you create the target endpoint using the AWS DMS console, or by using the `create-endpoint` command in the [AWS CLI](), with the `--microsoft-sql-server-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following table shows the endpoint settings that you can use with SQL Server as a target.

Name	Description
`ControlTablesFileGroup`	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. Default value: n/a Valid values: String Example: `--microsoft-sql-server-settings '{"ControlTablesFileGroup": "filegroup1"}'` The following is an example of a command for creating a filegroup.

Name	Description
	`` 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 ```
`ConnectionTimeout`	Use this extra connection attribute (ECA) to set the endpoint connection timeout for the SQL Server instance, in seconds. The default value is 10 seconds. ECA Example: `ConnectionTimeout=30` .
`ExecuteTimeout`	Use this extra connection attribute (ECA) to set the client statement timeout for the SQL Server instance, in seconds. The default value is 60 seconds. Example: `'{"ExecuteTimeout": 100}'`
`UseBCPFullLoad`	Use this to attribute to transfer data for full-load operations using BCP. When the target table contains an identity column that does not exist in the source table, you must disable the **use BCP for loading table** option. Default value: true Valid values: true/false Example: `--microsoft-sql-server-settings '{"UseBCPFullLoad": false}'`

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.

AWS DMS data type	SQL Server data type
BOOLEAN	TINYINT
BYTES	VARBINARY(length)
DATE	For SQL Server 2008 and higher, use DATE. For earlier versions, if the scale is 3 or less use DATETIME. In all other cases, use VARCHAR (37).
TIME	For SQL Server 2008 and higher, use DATETIME2 (%d). For earlier versions, if the scale is 3 or less use DATETIME. In all other cases, use VARCHAR (37).
DATETIME	For SQL Server 2008 and higher, 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:

AWS DMS data type	SQL Server data type
	• For SQL Server 2008 and higher, use DATETIME2. • For earlier versions, if the scale is 3 or less use DATETIME. In all other cases, use VARCHAR (37). If the column is not a date or time column, use VARCHAR (length).
UINT1	TINYINT
UINT2	SMALLINT
UINT4	INT
UINT8	BIGINT
WSTRING	NVARCHAR (length)
BLOB	VARBINARY(max) IMAGE To use this data type with AWS DMS, you must enable the use of BLOBs for a specific task. AWS DMS supports BLOB data types only in tables that include a primary key.
CLOB	VARCHAR(max) To use this data type with AWS DMS, you must enable the use of CLOBs for a specific task. During 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.

For information about versions of PostgreSQL that AWS DMS supports as a target, see Targets for AWS DMS.

Note

- Amazon Aurora Serverless is available as a target for Amazon Aurora with PostgreSQL compatibility. For more information about Amazon Aurora Serverless, see Using Amazon Aurora Serverless v2 in the *Amazon Aurora User Guide*.
- Aurora Serverless DB clusters are accessible only from an Amazon VPC and can't use a public IP address. So, if you intend to have a replication instance in a different region than Aurora PostgreSQL Serverless, you must configure vpc peering. Otherwise, check the availability of Aurora PostgreSQL Serverless regions, and decide to use one of those regions for both Aurora PostgreSQL Serverless and your replication instance.
- Babelfish capability is built into Amazon Aurora and doesn't have an additional cost. For more information, see Using Babelfish for Aurora PostgreSQL as a target for AWS Database Migration Service.

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

The following limitations apply when using a PostgreSQL database as a target for AWS DMS:

- For heterogeneous migrations, the JSON data type is converted to the Native CLOB data type internally.
- 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.

- AWS DMS doesn't support replication to a table with a unique index created with coalesce function.
- If your tables use sequences, then update the value of NEXTVAL for each sequence in the target database after you stop the replication from the source database. AWS DMS copies data from your source database, but doesn't migrate sequences to the target during the ongoing replication.

Limitations on using Amazon Aurora PostgreSQL Limitless as a target for AWS Database Migration Service

The following limitations apply when using Amazon Aurora PostgreSQL Limitless as a target for AWS DMS:

- AWS DMS Data Validation does not support Amazon Aurora PostgreSQL Limitless.
- AWS DMS migrates source tables as Standard tables, which are not distributed. After migration, you can convert these Standard tables to Limitless tables by following the official conversion 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;
```

Endpoint settings and Extra Connection Attributes (ECAs) when using PostgreSQL as a target for AWS DMS

You can use endpoint settings and Extra Connection Attributes (ECAs) to configure your PostgreSQL target database.

You specify the settings when you create the target endpoint using the AWS DMS console, or by using the `create-endpoint` command in the AWS CLI, with the `--postgre-sql-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

You specify ECAs using the `ExtraConnectionAttributes` parameter for your endpoint.

The following table shows the endpoint settings that you can use with PostgreSQL as a target.

Name	Description
`ConnectionTimeout`	Use this extra connection attribute (ECA) to set the endpoint connection timeout for the PostgreSQL instance, in seconds. The default value is 10 seconds. ECA Example: `ConnectionTimeout=30` .
`MaxFileSize`	Specifies the maximum size (in KB) of any .csv file used to transfer data to PostgreSQL. Default value: 32,768 KB (32 MB) Valid values: 1–1,048,576 KB (up to 1.1 GB) Example: `--postgre-sql-settings '{"MaxFileSize": 512}'`
`ExecuteTimeout`	Sets the client statement timeout for the PostgreSQL instance, in seconds. The default value is 60 seconds.

Name	Description
	Example: `--postgre-sql-settings '{"ExecuteTimeout": 100}'`
`AfterConnectScript= SET session_replication_role = replica`	This attribute has AWS DMS bypass foreign keys and user triggers to reduce the time it takes to bulk load data.

Name	Description
`MapUnboundedNumericAsString`	This parameter treats columns with unbounded NUMERIC data types as STRING in order to successfully migrate without losing precision of the numeric value. Use this parameter only for replication from PostgreSQL source to PostgreSQL target, or databases with PostgreSQL compatibility. Default value: false Valid values: false/true Example: `--postgre-sql-settings '{"MapUnboundedNumericAsString": "true"}` Using this parameter might result in some replication performance degradation because of transformation from numeric to string and back to numeric. This parameter is supported for use by DMS version 3.4.4 and higher **Note** Only use `MapUnboundedNumericAsString` in PostgreSQL source and target endpoints together. Use of `MapUnboundedNumericAsString` on source PostgreSQL endpoints restricts precision to 28 during CDC. Use of `MapUnboundedNumericAsString` on target endpoints, migrates data with Precision 28 Scale 6. Do not use `MapUnboundedNumericAsString` with non-PostgreSQL targets.

Name	Description
`loadUsingCSV`	Use this Extra Connection Attribute (ECA) to transfer data for full-load operations using \COPY command. Default value: `true` Valid values: true/false ECA Example: `loadUsingCSV=true;` **Note:** Setting this ECA to false might result in some replication performance degradation because of INSERTs being executed directly.
`DatabaseMode`	Use this attribute to change the default behaviour of the replication’s handling of Postgresql compatible endpoints that require some additional configuration, such as Babelfish endpoints. Default value: `DEFAULT` Valid values: `DEFAULT`, `BABELFISH` Example: `DatabaseMode=default;`
`BabelfishDatabaseName`	Use this attribute to specify the name of the target Babelfish T-SQL database being migrated to. This is required if `DatabaseMode` is set to `Babelfish`. This is not the reserved `babelfish_db` database. Example: `BabelfishDatabaseName=TargetDb;`

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.

For additional information about AWS DMS data types, see Data types for AWS Database Migration Service.

AWS DMS data type	PostgreSQL data type
BOOLEAN	BOOLEAN
BLOB	BYTEA
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
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

AWS DMS data type	PostgreSQL data type
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 Babelfish for Aurora PostgreSQL as a target for AWS Database Migration Service

You can migrate SQL Server source tables to a Babelfish for Amazon Aurora PostgreSQL target using AWS Database Migration Service. With Babelfish, Aurora PostgreSQL understands T-SQL, Microsoft SQL Server's proprietary SQL dialect, and supports the same communications protocol. So, applications written for SQL Server can now work with Aurora with fewer code changes. Babelfish capability is built into Amazon Aurora and doesn't have an additional cost. You can activate Babelfish on your Amazon Aurora cluster from the Amazon RDS console.

When you create your AWS DMS target endpoint using the AWS DMS console, API, or CLI commands, specify the target engine as **Amazon Aurora PostgreSQL**, and name the database, **babelfish_db**. In the **Endpoint Settings** section, add settings to set `DatabaseMode` to `Babelfish` and `BabelfishDatabaseName` to the name of the target Babelfish T-SQL database.

Adding transformation rules to your migration task

When you define a migration task for a Babelfish target, you need to include transformation rules that ensure DMS uses the pre-created T-SQL Babelfish tables in the target database.

First, add a transformation rule to your migration task that makes all table names lowercase. Babelfish stores as lowercase in the PostgreSQL `pg_class` catalog the names of tables that you create using T-SQL. However, when you have SQL Server tables with mixed-case names, DMS creates the tables using PostgreSQL native data types instead of the T-SQL compatible data types. For that reason, be sure to add a transformation rule that makes all table names lowercase. Note that column names should not be transformed to lowercase.

Next, if you used the multidatabase migration mode when you defined your cluster, add a transformation rule that renames the original SQL Server schema. Make sure to rename the SQL Server schema name to include the name of the T-SQL database. For example, if the original SQL Server schema name is dbo, and your T-SQL database name is mydb, rename the schema to mydb_dbo using a transformation rule.

> **Note**
>
> When using Babelfish for Aurora PostgreSQL 16 or later, the default migration mode is "mutidatabase". When running DMS migration tasks, ensure to review the migration mode parameter and update the transformation rules if needed.

If you use single database mode, you don't need a transformation rule to rename schema names. Schema names have a one-to-one mapping with the target T-SQL database in Babelfish.

The following sample transformation rule makes all table names lowercase, and renames the original SQL Server schema name from `dbo` to `mydb_dbo`.

```
{
   "rules": [
   {
      "rule-type": "transformation",
      "rule-id": "566251737",
      "rule-name": "566251737",
      "rule-target": "schema",
      "object-locator": {
         "schema-name": "dbo"
```

```
      },
      "rule-action": "rename",
      "value": "mydb_dbo",
      "old-value": null
   },
   {
      "rule-type": "transformation",
      "rule-id": "566139410",
      "rule-name": "566139410",
      "rule-target": "table",
      "object-locator": {
         "schema-name": "%",
         "table-name": "%"
      },
      "rule-action": "convert-lowercase",
      "value": null,
      "old-value": null
   },
   {
      "rule-type": "selection",
      "rule-id": "566111704",
      "rule-name": "566111704",
      "object-locator": {
         "schema-name": "dbo",
         "table-name": "%"
      },
      "rule-action": "include",
      "filters": []
   }
]
}
```

Limitations to using a PostgreSQL target endpoint with Babelfish tables

The following limitations apply when using a PostgreSQL target endpoint with Babelfish tables:

- For **Target table preparation** mode, use only the **Do nothing** or **Truncate** modes. Don't use the **Drop tables on target** mode. In that mode, DMS creates the tables as PostgreSQL tables that T-SQL might not recognize.
- AWS DMS doesn't support the sql_variant data type.

- Babelfish under Postgres endpoint does not support HEIRARCHYID, GEOMETRY (prior to 3.5.4) and GEOGRAPHY (prior to 3.5.4) data types. To migrate these data types, you can add transformation rules to convert the data type to wstring(250).
- Babelfish only supports migrating BINARY, VARBINARY, and IMAGE data types using the BYTEA data type. For earlier versions of Aurora PostgreSQL, you can use DMS to migrate these tables to a Babelfish target endpoint. You don't have to specify a length for the BYTEA data type, as shown in the following example.

  ```
  [Picture] [VARBINARY](max) NULL
  ```

  Change the preceding T-SQL data type to the T-SQL supported BYTEA data type.

  ```
  [Picture] BYTEA NULL
  ```

- For earlier versions of Aurora PostgreSQL Babelfish, if you create a migration task for ongoing replication from SQL Server to Babelfish using the PostgreSQL target endpoint, you need to assign the SERIAL data type to any tables that use IDENTITY columns. Starting with Aurora PostgreSQL (version 15.3/14.8 and higher) and Babelfish (version 3.2.0 and higher), the identity column is supported, and it is no longer required to assign the SERIAL data type. For more information, see SERIAL Usage in the Sequences and Identity section of the *SQL Server to Aurora PostgreSQL Migration Playbook*. Then, when you create the table in Babelfish, change the column definition from the following.

  ```
      [IDCol] [INT] IDENTITY(1,1) NOT NULL PRIMARY KEY
  ```

  Change the preceding into the following.

  ```
      [IDCol] SERIAL PRIMARY KEY
  ```

  Babelfish-compatible Aurora PostgreSQL creates a sequence using the default configuration and adds a NOT NULL constraint to the column. The newly created sequence behaves like a regular sequence (incremented by 1) and has no composite SERIAL option.

- After migrating data with tables that use IDENTITY columns or the SERIAL data type, reset the PostgreSQL-based sequence object based on the maximum value for the column. After performing a full load of the tables, use the following T-SQL query to generate statements to seed the associated sequence object.

```
DECLARE @schema_prefix NVARCHAR(200) = ''

IF current_setting('babelfishpg_tsql.migration_mode') = 'multi-db'
        SET @schema_prefix = db_name() + '_'

SELECT 'SELECT setval(pg_get_serial_sequence(''' + @schema_prefix +
 schema_name(tables.schema_id) + '.' + tables.name + ''', ''' + columns.name + ''')
               ,(select max(' + columns.name + ') from ' +
 schema_name(tables.schema_id) + '.' + tables.name + '));'
FROM sys.tables tables
JOIN sys.columns columns ON tables.object_id = columns.object_id
WHERE columns.is_identity = 1

UNION ALL

SELECT 'SELECT setval(pg_get_serial_sequence(''' + @schema_prefix + table_schema +
 '.' + table_name + ''',
''' + column_name + '''),(select max(' + column_name + ') from ' + table_schema + '.'
 + table_name + '));'
FROM information_schema.columns
WHERE column_default LIKE 'nextval(%';
```

The query generates a series of SELECT statements that you execute in order to update the maximum IDENTITY and SERIAL values.

- For Babelfish versions prior to 3.2, **Full LOB mode** might result in a table error. If that happens, create a separate task for the tables that failed to load. Then use **Limited LOB mode** to specify the appropriate value for the **Maximum LOB size (KB)**. Another option is to set the SQL Server Endpoint Connection Attribute setting `ForceFullLob=True`.
- For Babelfish versions prior to 3.2, performing data validation with Babelfish tables that don't use integer based primary keys generates a message that a suitable unique key can't be found. Starting with Aurora PostgreSQL (version 15.3/14.8 and higher) and Babelfish (version 3.2.0 and higher), data validation for non-integer primary keys is supported.
- Because of precision differences in the number of decimal places for seconds, DMS reports data validation failures for Babelfish tables that use `DATETIME` data types. To suppress those failures, you can add the following validation rule type for `DATETIME` data types.

```
{
         "rule-type": "validation",
         "rule-id": "3",
```

```
            "rule-name": "3",
            "rule-target": "column",
            "object-locator": {
                "schema-name": "dbo",
                "table-name": "%",
                "column-name": "%",
                "data-type": "datetime"
            },
            "rule-action": "override-validation-function",
            "source-function": "case when ${column-name} is NULL then NULL else 0 end",
            "target-function": "case when ${column-name} is NULL then NULL else 0 end"
        }
```

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 AWS-managed service such as Amazon RDS, Amazon Aurora, or Amazon S3. Or 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.

For information about versions of MySQL that AWS DMS supports as a target, see Targets for AWS DMS.

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.

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

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, CREATE TEMPORARY
 TABLES  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** section of the AWS DMS console for your 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.
- Grant the following privileges if you use MySQL-specific premigration assessments.

```
grant select on mysql.user to <dms_user>;
grant select on mysql.db to <dms_user>;
grant select on mysql.tables_priv to <dms_user>;
grant select on mysql.role_edges to <dms_user>  #only for MySQL version 8.0.11 and
 higher
```

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 loading data to a MySQL-compatible target in a full load task, AWS DMS doesn't report errors caused by constraints in the task logs, which can cause duplicate key errors or mismatches with the number of records. This is caused by the way MySQL handles local data with the `LOAD DATA` command. Be sure to do the following during the full load phase:
  - Disable constraints
  - Use AWS DMS validation to make sure the data is consistent.
- 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 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 Aurora Serverless v2 in the *Amazon Aurora User Guide*.
- AWS DMS does not support using a reader endpoint for Aurora or Amazon RDS, unless the instances are in writable mode, that is, the `read_only` and `innodb_read_only` parameters are set to `0` or `OFF`. For more information about using Amazon RDS and Aurora as targets, see the following:
  - Determining which DB instance you are connected to
  - Updating read replicas with MySQL
- When replicating TIME datatype, fractional part of time value is not replicated.
- When replicating TIME datatype with Extra Connection Attribute `loadUsingCSV=false`, the time value is capped to range `[00:00:00, 23:59:59]`.

Endpoint settings when using a MySQL-compatible database as a target for AWS DMS

You can use endpoint settings to configure your MySQL-compatible target database similar to using extra connection attributes. You specify the settings when you create the target endpoint using the AWS DMS console, or by using the `create-endpoint` command in the AWS CLI, with the `--my-sql-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following table shows the endpoint settings that you can use with MySQL as a target.

Name	Description
`ConnectionTimeout`	Use this extra connection attribute (ECA) to set the endpoint connection timeout for the MySQL instance, in

Name	Description
	seconds. The default value is 10 seconds. ECA Example: `ConnectionTimeout=30` .
`TargetDbType`	Specifies where to migrate source tables on the target, either to a single database or multiple databases. If you specify `SPECIFIC_DATABASE` , you need to specify the database name, either when using the AWS CLI or the AWS Management Console. Default value: `MULTIPLE_DATABASES` Valid values: {`SPECIFIC_DATABASE` , `MULTIPLE_DATABASES` } Example: `--my-sql-settings '{"TargetDbType": "MULTIPLE_DATABASES"}'`
`ParallelLoadThreads`	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. Default value: 1 Valid values: 1–5 Example: `--my-sql-settings '{"ParallelLoadThreads": 1}'`

Name	Description
`AfterConnectScript`	Specifies a script to run immediately after AWS DMS connects to the endpoint. For example, you can specify that the MySQL-com patible 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. Example: `--my-sql-settings '{"AfterConnectScript": "SET character_set_connection='latin1'"}'`
`MaxFileSize`	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 `--my-sql-settings '{"MaxFileSize": 512}'`

You can also use extra connection attributes to configure your MySQL-compatible target database.

The following table shows the extra connection attributes that you can use with MySQL as a target.

Name	Description
`Initstmt=SET FOREIGN_KEY_CHECKS=0;`	Disables foreign key checks. Example: `--extra-connection-attributes "Initstmt=SET FOREIGN_KEY_CHECKS=0;"`
`Initstmt=SET time_zone`	Specifies the time zone for the target MySQL-com patible database.

Name	Description
	Default value: UTC Valid values: The time zone names available in the target MySQL database. Example: `--extra-connection-attributes "Initstmt=SET time_zone= US/Pacific ;"`

Alternatively, you can use the `AfterConnectScript` parameter of the `--my-sql-settings` command to disable foreign key checks and specify the time zone for your database.

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.

AWS DMS data types	MySQL data types
BOOLEAN	BOOLEAN
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)"

AWS DMS data types	MySQL data types
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.

AWS DMS data types	MySQL data types
BLOB	If the length is from 1 through 65,535, then use BLOB. If the length is from 65,536 through 2,147,483,647, then use LONGBLOB. If the length is 0, then use LONGBLOB (full LOB support).
NCLOB	If the length is from 1 through 65,535, then use TEXT. If the length is from 65,536 through 2,147,483,647, then use LONGTEXT with ucs2 for CHARACTER SET. If the length is 0, then use LONGTEXT (full LOB support) with ucs2 for CHARACTER SET.
CLOB	If the length is from 1 through 65,535, then use TEXT. If the length is from 65,536 through 2147483647, then use LONGTEXT. If the length is 0, then use LONGTEXT (full LOB support).

Using an Amazon Redshift database as a target for AWS Database Migration Service

You can migrate data to Amazon Redshift databases using AWS Database Migration Service. Amazon Redshift is a fully managed, petabyte-scale data warehouse service in the cloud. With an Amazon Redshift database as a target, you can migrate data from all of the other supported source databases.

You can use Amazon Redshift Serverless as a target for AWS DMS. For more information, see Using AWS DMS with Amazon Redshift Serverless as a Target following.

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 AWS DMS.

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

- Privileges required for using Redshift as a target
- Limitations on using Amazon Redshift as a target for AWS Database Migration Service
- Configuring an Amazon Redshift database as a target for AWS Database Migration Service
- Using enhanced VPC routing with Amazon Redshift as a target for AWS Database Migration Service
- Creating and using AWS KMS keys to encrypt Amazon Redshift target data
- Endpoint settings when using Amazon Redshift as a target for AWS DMS
- Using a data encryption key, and an Amazon S3 bucket as intermediate storage
- Multithreaded task settings for Amazon Redshift
- Target data types for Amazon Redshift
- Using AWS DMS with Amazon Redshift Serverless as a Target

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 AWS DMS.
- AWS DMS converts BLOBs, CLOBs, and NCLOBs to a VARCHAR on the target Amazon Redshift instance. Amazon Redshift does not 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 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.
- When Row Level Security is enabled on the tables in Redshift, you must grant appropriate permissions to all your DMS users.

Privileges required for using Redshift as a target

Use the GRANT command to define access privileges for a user or user group. Privileges include access options such as being able to read data in tables and views, write data, and create tables. For more information about using GRANT with Amazon Redshift, see GRANT in the *Amazon Redshift Database Developer Guide*.

The following is the syntax to give specific privileges for a table, database, schema, function, procedure, or language-level privileges on Amazon Redshift tables and views.

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | REFERENCES } [,...] | ALL
 [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...] | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO { username [ WITH GRANT OPTION ] | GROUP group_name | PUBLIC } [, ...]

GRANT { { CREATE | TEMPORARY | TEMP } [,...] | ALL [ PRIVILEGES ] }
    ON DATABASE db_name [, ...]
    TO { username [ WITH GRANT OPTION ] | GROUP group_name | PUBLIC } [, ...]

GRANT { { CREATE | USAGE } [,...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO { username [ WITH GRANT OPTION ] | GROUP group_name | PUBLIC } [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON { FUNCTION function_name ( [ [ argname ] argtype [, ...] ] ) [, ...] | ALL
 FUNCTIONS IN SCHEMA schema_name [, ...] }
    TO { username [ WITH GRANT OPTION ] | GROUP group_name | PUBLIC } [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON { PROCEDURE procedure_name ( [ [ argname ] argtype [, ...] ] ) [, ...] | ALL
 PROCEDURES IN SCHEMA schema_name [, ...] }
    TO { username [ WITH GRANT OPTION ] | GROUP group_name | PUBLIC } [, ...]
```

```
GRANT USAGE
    ON LANGUAGE language_name [, ...]
    TO { username [ WITH GRANT OPTION ] | GROUP group_name | PUBLIC } [, ...]
```

The following is the syntax for column-level privileges on Amazon Redshift tables and views.

```
GRANT { { SELECT | UPDATE } ( column_name [, ...] ) [, ...] | ALL [ PRIVILEGES ]
 ( column_name [,...] ) }
     ON { [ TABLE ] table_name [, ...] }
     TO { username | GROUP group_name | PUBLIC } [, ...]
```

The following is the syntax for the ASSUMEROLE privilege granted to users and groups with a specified role.

```
GRANT ASSUMEROLE
    ON { 'iam_role' [, ...] | ALL }
    TO { username | GROUP group_name | PUBLIC } [, ...]
    FOR { ALL | COPY | UNLOAD } [, ...]
```

Limitations on using Amazon Redshift as a target for AWS Database Migration Service

The following limitations apply when using an Amazon Redshift database as a target:

- Don’t enable versioning for the S3 bucket you use as intermediate storage for your Amazon Redshift target. If you need S3 versioning, use lifecycle policies to actively delete old versions. Otherwise, you might encounter endpoint test connection failures because of an S3 `list-object` call timeout. To create a lifecycle policy for an S3 bucket, see Managing your storage lifecycle. To delete a version of an S3 object, see Deleting object versions from a versioning-enabled bucket.
- 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 does not 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.
- You may experience performance issues if your source system performs UPDATE operations on the primary key of a source table. These performance issues occur when applying changes to the target. This is because UPDATE (and DELETE) operations depend on the primary key value to identify the target row. If you update the primary key of a source table, your task log will contain messages like the following:

```
Update on table 1 changes PK to a PK that was previously updated in the same bulk
 update.
```

- DMS does not support custom DNS names when configuring an endpoint for a Redshift cluster, and you need to use the Amazon provided DNS name. Since the Amazon Redshift cluster must be in the same AWS account and Region as the replication instance, validation fails if you use a custom DNS endpoint.
- Amazon Redshift has a default 4-hour idle session timeout. When there isn't any activity within the DMS replication task, Redshift disconnects the session after 4 hours. Errors can result from DMS being unable to connect and potentially needing to restart. As a workaround, set a SESSION TIMEOUT limit greater than 4 hours for the DMS replication user. Or, see the description of ALTER USER in the *Amazon Redshift Database Developer Guide*.
- When AWS DMS replicates source table data without a primary or unique key, CDC latency might be high resulting in an unacceptable level of performance.
- Truncating partitions is not supported during CDC replication from Oracle source to Redshift target.
- Duplicate records might appear in target tables because Amazon Redshift does not enforce primary keys and AWS DMS may replay CDC when a task is resumed. To prevent duplicates, use the `ApplyErrorInsertPolicy=INSERT_RECORD` setting. For more information, see Error handling task settings. Alternatively, you can implement application-level duplicate detection and post-migration cleanup procedures.

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

Using enhanced VPC routing with 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 Using a data encryption key, and an Amazon S3 bucket as intermediate storage.

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

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 AWS-managed policy.
- A KMS 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 AWS-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](#).

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 following example.

    JSON

    ```
    {
        "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 AWS KMS 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 database similar to using extra connection attributes. You specify the settings when you create the target endpoint using the AWS DMS console, or by using the `create-endpoint` command in the AWS CLI, with the `--redshift-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following table shows the endpoint settings that you can use with Amazon Redshift as a target.

Name	Description
`MaxFileSize`	Specifies the maximum size (in KB) of any .csv file used to transfer data to Amazon Redshift. Default value: 32768 KB (32 MB) Valid values: 1–1048576 Example: `--redshift-settings '{"MaxFileSize": 512}'`
`FileTransferUploadStreams`	Specifies the number of threads used to upload a single file. Default value: 10 Valid values: 1–64 Example: `--redshift-settings '{"FileTransferUploadStreams": 20}'`
`Acceptanydate`	Specifies if any date format is accepted, including invalid dates formats such as 0000-00-00. Boolean value. Default value: false

Name	Description
	Valid values: true | false (Boolean) Example: `--redshift-settings '{"Accept anydate": true}'`
`Dateformat`	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 `Dateformat` parameter. The `auto` argument recognizes several formats that are not supported when using a `Dateformat` string. The `auto` keyword is case-sensitive. Default value: empty Valid values: "*`dateformat_string`*" or `auto` Example:`--redshift-settings '{"Dateformat": "auto"}'`
`Timeformat`	Specifies the time format. This is a string input and is empty by default. The `auto` argument recognizes several formats that aren't supported when using a `Timeformat` string. If your date and time values use formats different from each other, use the `auto` argument with the `Timeformat` parameter. Default value: 10 Valid values: "*`Timeformat_string`*" | "auto" | "epochsecs" | "epochmillisecs" Example:`--redshift-settings '{"Timeformat": "auto"}'`

Name	Description
`Emptyasnull`	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. Default value: false Valid values: true | false Example: `--redshift-settings '{"Emptyasnull": true}'`
`TruncateColumns`	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. Default value: false Valid values: true | false Example: `--redshift-settings '{"TruncateColumns": true}'`
`RemoveQuotes`	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 Amazon Redshift Database Developer Guide. Default value: false Valid values: true | false Example: `--redshift-settings '{"RemoveQuotes": true}'`

Name	Description
`TrimBlanks`	Removes the trailing white-space characters from a VARCHAR string. This parameter applies only to columns with a VARCHAR data type. Default value: false Valid values: true | false Example: `--redshift-settings '{"TrimBlanks": true}'`
`EncryptionMode`	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. **Note** You can also use the CLI `modify-endpoint` command to change the value of the `EncryptionMode` setting for an existing endpoint from `SSE_KMS` to `SSE_S3`. But you can't change the `EncryptionMode` value from `SSE_S3` to `SSE_KMS`. Default value: `SSE_S3` Valid values: `SSE_S3` or `SSE_KMS` Example: `--redshift-settings '{"EncryptionMode": "SSE_S3"}'`

Name	Description
`ServerSideEncryptionKmsKeyId`	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 AWS KMS keys created for your account. When you create the key, you must associate specific policies and roles with it. For more information, see [Creating and using AWS KMS keys to encrypt Amazon Redshift target data](). Example: `--redshift-settings '{"ServerSideEncryptionKmsKeyId":"arn:aws:kms:us-east-1:111122223333:key/11a1a1a1-aaaa-9999-abab-2bbbbbb222a2"}'`
`EnableParallelBatchInMemoryCSVFiles`	The `EnableParallelBatchInMemoryCSVFiles` setting improves performance of larger multithreaded full load tasks by having DMS write to disk instead of memory. The default value is `false`.
`CompressCsvFiles`	Use this attribute to compress data sent to a Amazon Redshift target during migration. The default value is `true`, and compression is enabled by default.

Name	Description
`CaseSensitiveNames`	Specifies whether to enable case-sensitive handling of identifiers (such as table and column names) when working with Redshift target endpoints. This setting is particularly important when working with Zero-ETL Redshift databases, where the parameter `enable_case_sensitive_identifier` is set to `true` by default. When `CaseSensitiveNames` is set to true, DMS preserves the case sensitivity of identifiers during migration. Default value: false Valid values: true | false (Boolean) Example: `--redshift-settings '{"CaseSensitiveNames": true}'`

Using a data encryption key, and an Amazon S3 bucket as intermediate storage

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.
- Map a boolean as a boolean from a PostgreSQL source. By default, a BOOLEAN type is migrated as varchar(1). You can specify `MapBooleanAsBoolean` to let your Redshift target migrate the boolean type as boolean, as shown in the example following.

```
--redshift-settings '{"MapBooleanAsBoolean": true}'
```

  Note that you must set this setting on both the source and target endpoints for it to take effect.

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-2debbef226d1"}'
```

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-2debbef226d1`. 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 User 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 AWS KMS data encryption key use this default S3 bucket (see KMS key settings for data encryption).

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. If your service access role is based on the `AmazonDMSRedshiftS3Role` policy, this value must have a prefix of `dms-`, for example, `dms-my-bucket-name`.
- `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 AWS-managed policy in Creating and using AWS KMS keys to encrypt Amazon Redshift target data.

> **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"}'
```

Multithreaded task settings for Amazon Redshift

You can improve performance of full load and change data capture (CDC) tasks for an Amazon Redshift target endpoint by using multithreaded task settings. They enable you to specify the number of concurrent threads and the number of records to store in a buffer.

Multithreaded full load task settings for Amazon Redshift

To promote full load performance, you can use the following `ParallelLoad*` task settings:

- `ParallelLoadThreads` – Specifies the number of concurrent threads that DMS uses during a full load to push data records to an Amazon Redshift target endpoint. The default value is zero (0) and the maximum value is 32. For more information, see Full-load task settings.

  You can use the `enableParallelBatchInMemoryCSVFiles` attribute set to `false` when using the `ParallelLoadThreads` task setting. The attribute improves performance of larger multithreaded full load tasks by having DMS write to disk instead of memory. The default value is `true`.
- `ParallelLoadBufferSize` – Specifies the maximum data record requests while using parallel load threads with Redshift target. The default value is 100 and the maximum value is 1,000. We recommend you use this option when ParallelLoadThreads > 1 (greater than one).

> **Note**
>
> Support for the use of `ParallelLoad*` task settings during FULL LOAD to Amazon Redshift target endpoints is available in AWS DMS versions 3.4.5 and higher.
> The `ReplaceInvalidChars` Redshift endpoint setting is not supported for use during change data capture (CDC) or during a parallel load enabled FULL LOAD migration task. It is supported for FULL LOAD migration when parallel load isn't enabled. For more information see RedshiftSettings in the *AWS Database Migration Service API Reference*

Multithreaded CDC task settings for Amazon Redshift

To promote CDC performance, you can use the following `ParallelApply*` task settings:

- `ParallelApplyThreads` – Specifies the number of concurrent threads that AWS DMS uses during a CDC load to push data records to a Amazon Redshift target endpoint. The default value is zero (0) and the maximum value is 32. The minimum recommended value equals the number of slices in your cluster.

- `ParallelApplyBufferSize` – Specifies the maximum data record requests while using parallel apply threads with Redshift target. The default value is 100 and the maximum value is 1,000. We recommend to use this option when ParallelApplyThreads > 1 (greater than one).

  To obtain the most benefit for Redshift as a target, we recommend that the value of `ParallelApplyBufferSize` be at least two times (double or more) the number of `ParallelApplyThreads`.

> **Note**
>
> Support for the use of `ParallelApply*` task settings during CDC to Amazon Redshift target endpoints is available in AWS DMS versions 3.4.3 and higher.

The level of parallelism applied depends on the correlation between the total *batch size* and the *maximum file size* used to transfer data. When using multithreaded CDC task settings with a Redshift target, benefits are gained when batch size is large in relation to the maximum file size. For example, you can use the following combination of endpoint and task settings to tune for optimal performance.

```
// Redshift endpoint setting

        MaxFileSize=250000;

// Task settings

        BatchApplyEnabled=true;
        BatchSplitSize =8000;
        BatchApplyTimeoutMax =1800;
        BatchApplyTimeoutMin =1800;
        ParallelApplyThreads=32;
        ParallelApplyBufferSize=100;
```

Using the settings in the previous example, a customer with a heavy transactional workload benefits by their batch buffer, containing 8000 records, getting filled in 1800 seconds, utilizing 32 parallel threads with a 250 MB maximum file size.

For more information, see Change processing tuning settings.

Note

DMS queries that run during ongoing replication to a Redshift cluster can share the same WLM (workload management) queue with other application queries that are running. So, consider properly configuring WLM properties to influence performance during ongoing replication to a Redshift target. For example, if other parallel ETL queries are running, DMS runs slower and performance gains are lost.

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.

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, depending on Redshift target column type, then one of the following: TIMESTAMP (s) TIMESTAMPTZ (s) — If source timestamp contains a zone offset (such as in SQL Server or Oracle) it converts to UTC on insert/update.

AWS DMS data types	Amazon Redshift data types
	If it does not contain an offset, then time is considered in UTC already. 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)

AWS DMS data types	Amazon Redshift data types
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 does not support VARCHARs larger than 64 KB.
NCLOB	NVARCHAR (maximum LOB size) The maximum LOB size cannot exceed 63 KB. Amazon Redshift does not support VARCHARs larger than 64 KB.
CLOB	VARCHAR (maximum LOB size) The maximum LOB size cannot exceed 63 KB. Amazon Redshift does not support VARCHARs larger than 64 KB.

Using AWS DMS with Amazon Redshift Serverless as a Target

AWS DMS supports using Amazon Redshift Serverless as a target endpoint. For information about using Amazon Redshift Serverless, see Amazon Redshift Serverless in the Amazon Redshift Management Guide.

This topic describes how to use a Amazon Redshift Serverless endpoint with AWS DMS.

> **Note**
>
> When creating an Amazon Redshift Serverless endpoint, for the **DatabaseName** field of your RedshiftSettings endpoint configuration, use either the name of the Amazon Redshift data warehouse or the name of the workgroup endpoint. For the **ServerName**

field, use the value for Endpoint displayed in the **Workgroup** page for the serverless cluster (for example, `default-workgroup.093291321484.us-east-1.redshift-serverless.amazonaws.com`). For information about creating an endpoint, see Creating source and target endpoints. For information about the workgroup endpoint, see Connecting to Amazon Redshift Serverless .

Trust Policy with Amazon Redshift Serverless as a target

When using Amazon Redshift Serverless as a target endpoint, you must add the following highlighted section to your trust policy. This trust policy is attached to the `dms-access-for-endpoint` role.

For more information about using a trust policy with AWS DMS, see Creating the IAM roles to use with AWS DMS.

Limitations when using Amazon Redshift Serverless as a target

Using Redshift Serverless as a target has the following limitations:

- AWS DMS only supports Amazon Redshift Serverless as an endpoint in regions that support Amazon Redshift Serverless. For information about which regions support Amazon Redshift Serverless, see **Redshift Serverless API** in the Amazon Redshift endpoints and quotas topic in the AWS General Reference.
- When using Enhanced VPC Routing, make sure that you create an Amazon S3 endpoint in the same VPC as your Redshift Serverless or Redshift Provisioned cluster. For more information, see Using enhanced VPC routing with Amazon Redshift as a target for AWS Database Migration Service.
- AWS DMS does not support Enhanced Throughput for Amazon Redshift Serverless as a target. For more information, see Enhanced Throughput for Full-Load Oracle to Amazon Redshift and Amazon S3 Migrations.
- AWS DMS does not support connections to Amazon Redshift Redshift Serverless when the SSL mode is set to `verify-full`. For connections requiring SSL verification to Amazon Redshift Serverless targets, use alternative SSL modes such as `require` or `verify-ca`.

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.

For information about versions of SAP ASE that AWS DMS supports as a target, see Targets for AWS DMS.

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.

Limitations when using a SAP ASE database as a target for AWS DMS

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

- AWS DMS doesn't support tables that include fields with the following data types. Replicated columns with these data types show as null.
  - User-defined type (UDT)

Endpoint settings when using SAP ASE as a target for AWS DMS

You can use endpoint settings to configure your SAP ASE target database similar to using extra connection attributes. You specify the settings when you create the target endpoint using the AWS DMS console, or by using the `create-endpoint` command in the AWS CLI, with the `--sybase-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following table shows the endpoint settings that you can use with SAP ASE as a target.

Name	Description
`Driver`	Set this attribute if you want to use TLS for versions of ASE 15.7 and higher. Default value: `Adaptive Server Enterprise` Example: `driver=Adaptive Server Enterprise 16.03.06;` Valid values: `Adaptive Server Enterprise 16.03.06`
`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.

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

AWS DMS data types	SAP ASE data types
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

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. You can control the size of the files that AWS DMS creates by using the [MaxFileSize](#) endpoint setting.

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, shown with full-load files.

```
database_schema_name/table_name/LOAD00000001.csv
database_schema_name/table_name/LOAD00000002.csv
...
database_schema_name/table_name/LOAD00000009.csv
database_schema_name/table_name/LOAD0000000A.csv
database_schema_name/table_name/LOAD0000000B.csv
...database_schema_name/table_name/LOAD0000000F.csv
database_schema_name/table_name/LOAD00000010.csv
...
```

You can specify the column delimiter, row delimiter, and other parameters using the extra connection attributes. For more information on the extra connection attributes, see [Endpoint settings when using Amazon S3 as a target for AWS DMS](#) at the end of this section.

To prevent spoofing, AWS DMS validates bucket ownership before performing operations. By default, when the `ExpectedBucketOwner` Amazon S3 endpoint setting is not specified, AWS DMS uses the AWS account ID that owns the AWS DMS service role as the expected bucket owner.

To migrate data to an S3 bucket owned by a different AWS account, you must explicitly specify the actual bucket owner in the `ExpectedBucketOwner` Amazon S3 endpoint setting, as shown following. Otherwise, the cross-account replication task will fail.

```
--s3-settings '{"ExpectedBucketOwner": "AWS_Account_ID"}'
```

When you use AWS DMS to replicate data changes using a CDC task, the first column of the .csv or .parquet output file indicates how the row 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 `INSERT`ed 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 `UPDATE`d 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 `UPDATE`d 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 `DELETE`d in the source table. The rest of the line reflects how the row in the EMPLOYEE table appeared before it was deleted.

Note that by default for CDC, AWS DMS stores the row changes for each database table without regard to transaction order. If you want to store the row changes in CDC files according to transaction order, you need to use S3 endpoint settings to specify this and the folder path where you want the CDC transaction files to be stored on the S3 target. For more information, see Capturing data changes (CDC) including transaction order on the S3 target.

To control the frequency of writes to an Amazon S3 target during a data replication task, you can configure the `cdcMaxBatchInterval` and `cdcMinFileSize` extra connection attributes. This can result in better performance when analyzing the data without any additional overhead operations. For more information, see Endpoint settings when using Amazon S3 as a target for AWS DMS

Topics

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*.
- For AWS DMS version 3.4.7 and higher, DMS must access the source bucket through a VPC endpoint or a public route. For information about VPC endpoints, see Configuring VPC endpoints for AWS DMS.

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.

JSON

```
{
    "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"
            ]
        }
    ]
}
```

For prerequisites for using validation with S3 as a target, see S3 target validation prerequisites.

Limitations to using Amazon S3 as a target

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

- Don't enable versioning for S3. If you need S3 versioning, use lifecycle policies to actively delete old versions. Otherwise, you might encounter endpoint test connection failures because of an S3 `list-object` call timeout. To create a lifecycle policy for an S3 bucket, see Managing your storage lifecycle. To delete a version of an S3 object, see Deleting object versions from a versioning-enabled bucket.
- A VPC-enabled (gateway VPC) S3 bucket is supported in versions 3.4.7 and higher.
- 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. Note that when a column is added, dropped,

or renamed on the source database, no ALTER statement is recorded in the target S3 bucket, and AWS DMS does not alter previously created records to match the new structure. After the change, AWS DMS creates any new records using the new table structure.

> **Note**
>
> A truncate DDL operation removes all files and corresponding table folders from an S3 bucket. You can use task settings to disable that behavior and configure the way DMS handles DDL behavior during change data capture (CDC). For more information, see [Task settings for change processing DDL handling](#).

- Full LOB mode is not supported.
- Changes to the source table structure during full load are not supported. Changes to 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.
- `BatchApply` is not supported for an S3 endpoint. Using Batch Apply (for example, the `BatchApplyEnabled` target metadata task setting) for an S3 target might result in loss of data.
- You can't use `DatePartitionEnabled` or `addColumnName` together with `PreserveTransactions` or `CdcPath`.
- AWS DMS doesn't support renaming multiple source tables to the same target folder using transformation rules.
- If there is intensive writing to the source table during the full load phase, DMS may write duplicate records to the S3 bucket or cached changes.
- If you configure the task with a `TargetTablePrepMode` of `DO_NOTHING`, DMS may write duplicate records to the S3 bucket if the task stops and resumes abruptly during the full load phase.
- If you configure the target endpoint with a `PreserveTransactions` setting of `true`, reloading a table doesn't clear previously generated CDC files. For more information, see [Capturing data changes (CDC) including transaction order on the S3 target](#).

For limitations for using validation with S3 as a target, see [Limitations for using S3 target validation](#).

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 Endpoint settings when using Amazon S3 as a target for AWS DMS. 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 Using data encryption, parquet files, and CDC on your Amazon S3 target.
- Extra connection attributes that you provide as a semicolon-separated list when you create the endpoint. For more information, see Endpoint settings when using Amazon S3 as a target for AWS DMS.

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 User Guide*. For more information about table mapping using JSON, see [Specifying table selection and transformations rules using JSON].

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:

Parameter	Possible values	Description
		• A single table in a single schema • A single table in some or all schemas • Some or all tables in a single schema • Some or all tables in some or all schemas

Parameter	Possible values	Description
`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 [Object tagging](#) in the *Amazon Simple Storage Service User Guide*. 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. **Note** **Important** If you insert a dynamic value for the `key` parameter , you can generate tags with duplicate names for an S3 object, depending on how you use it. In this case, only one

Parameter	Possible values	Description
		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"
              }
           ]
        }
    ]
}
```

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-*your-KMS-key-ARN* is the Amazon Resource Name (ARN) for your KMS key and it is required your IAM role has access permissions, see Using data encryption, parquet files, and CDC on your Amazon S3 target.

- 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 Endpoint settings when using Amazon S3 as a target for AWS DMS.

To encrypt Amazon S3 target objects using a 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.
- A KMS key 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.

   JSON

```
{
    "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 a KMS key 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.

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.

    JSON

    ```
    {
        "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-S3-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-S3-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 KMS key has been created.

You have now created a new KMS key with a specified alias (for example, `DMS-S3-endpoint-encryption-key`). This key enables AWS DMS to encrypt Amazon S3 target objects.

Using date-based folder partitioning

AWS DMS supports S3 folder partitions based on a transaction commit date when you use Amazon S3 as your target endpoint. Using date-based folder partitioning, you can write data from a single source table to a time-hierarchy folder structure in an S3 bucket. By partitioning folders when creating an S3 target endpoint, you can do the following:

- Better manage your S3 objects
- Limit the size of each S3 folder
- Optimize data lake queries or other subsequent operations

You can enable date-based folder partitioning when you create an S3 target endpoint. You can enable it when you either migrate existing data and replicate ongoing changes (full load + CDC), or replicate data changes only (CDC only). When you migrate existing data and replicate ongoing changes, only ongoing changes will be partitioned. Use the following target endpoint settings:

- `DatePartitionEnabled` – Specifies partitioning based on dates. Set this Boolean option to `true` to partition S3 bucket folders based on transaction commit dates.

  You can't use this setting with `PreserveTransactions` or `CdcPath`.

  The default value is `false`.
- `DatePartitionSequence` – Identifies the sequence of the date format to use during folder partitioning. Set this ENUM option to `YYYYMMDD`, `YYYYMMDDHH`, `YYYYMM`, `MMYYYYDD`, or `DDMMYYYY`. The default value is `YYYYMMDD`. Use this setting when `DatePartitionEnabled` is set to `true.`
- `DatePartitionDelimiter` – Specifies a date separation delimiter to use during folder partitioning. Set this ENUM option to `SLASH`, `DASH`, `UNDERSCORE`, or `NONE`. The default value is `SLASH`. Use this setting when `DatePartitionEnabled` is set to `true`.
- `DatePartitionTimezone` – When creating an S3 target endpoint, set `DatePartitionTimezone` to convert the current UTC time into a specified time zone. The conversion occurs when a date partition folder is created and a CDC filename is generated. The time zone format is Area/Location. Use this parameter when `DatePartitionedEnabled` is set to `true`, as shown in the following example:

```
s3-settings='{"DatePartitionEnabled": true, "DatePartitionSequence": "YYYYMMDDHH",
 "DatePartitionDelimiter": "SLASH", "DatePartitionTimezone":"Asia/Seoul",
 "BucketName": "dms-nattarat-test"}'
```

The following example shows how to enable date-based folder partitioning, with default values for the data partition sequence and the delimiter. It uses the `--s3-settings '{json-settings}'` option of the AWS CLI.`create-endpoint` command.

```
  --s3-settings '{"DatePartitionEnabled": true,"DatePartitionSequence":
"YYYYMMDD","DatePartitionDelimiter": "SLASH"}'
```

Parallel load of partitioned sources when using Amazon S3 as a target for AWS DMS

You can configure a parallel full load of partitioned data sources to Amazon S3 targets. This approach improves the load times for migrating partitioned data from supported source database engines to the S3 target. To improve the load times of partitioned source data, you create S3 target subfolders mapped to the partitions of every table in the source database. These partition-bound subfolders allow AWS DMS to run parallel processes to populate each subfolder on the target.

To configure a parallel full load of an S3 target, S3 supports three `parallel-load` rule types for the `table-settings` rule of table mapping:

- `partitions-auto`
- `partitions-list`
- `ranges`

For more information on these parallel-load rule types, see Table and collection settings rules and operations.

For the `partitions-auto` and `partitions-list` rule types, AWS DMS uses each partition name from the source endpoint to identify the target subfolder structure, as follows.

```
bucket_name/bucket_folder/database_schema_name/table_name/partition_name/
LOADseq_num.csv
```

Here, the subfolder path where data is migrated and stored on the S3 target includes an additional *`partition_name`* subfolder that corresponds to a source partition with the same name. This *`partition_name`* subfolder then stores one or more `LOAD`*`seq_num`*`.csv` files containing data migrated from the specified source partition. Here, *seq_num* is the sequence number postfix on the .csv file name, such as `00000001` in the .csv file with the name, `LOAD00000001.csv`.

However, some database engines, such as MongoDB and DocumentDB, don't have the concept of partitions. For these database engines, AWS DMS adds the running source segment index as a prefix to the target .csv file name, as follows.

```
.../database_schema_name/table_name/SEGMENT1_LOAD00000001.csv
.../database_schema_name/table_name/SEGMENT1_LOAD00000002.csv
...
.../database_schema_name/table_name/SEGMENT2_LOAD00000009.csv
.../database_schema_name/table_name/SEGMENT3_LOAD0000000A.csv
```

Here, the files `SEGMENT1_LOAD00000001.csv` and `SEGMENT1_LOAD00000002.csv` are named with the same running source segment index prefix, `SEGMENT1`. They're named as so because the migrated source data for these two .csv files is associated with the same running source segment index. On the other hand, the migrated data stored in each of the target `SEGMENT2_LOAD00000009.csv` and `SEGMENT3_LOAD0000000A.csv` files is associated with different running source segment indexes. Each file has its file name prefixed with the name of its running segment index, `SEGMENT2` and `SEGMENT3`.

For the `ranges` parallel-load type, you define the column names and column values using the `columns` and `boundaries` settings of the `table-settings` rules. With these rules, you can specify partitions corresponding to segment names, as follows.

```
"parallel-load": {
    "type": "ranges",
    "columns": [
         "region",
         "sale"
    ],
    "boundaries": [
          [
               "NORTH",
               "1000"
          ],
          [
               "WEST",
               "3000"
          ]
    ],
```

```
    "segment-names": [
          "custom_segment1",
          "custom_segment2",
          "custom_segment3"
    ]
}
```

Here, the `segment-names` setting defines names for three partitions to migrate data in parallel on the S3 target. The migrated data is parallel-loaded and stored in .csv files under the partition subfolders in order, as follows.

```
.../database_schema_name/table_name/custom_segment1/LOAD[00000001...].csv
.../database_schema_name/table_name/custom_segment2/LOAD[00000001...].csv
.../database_schema_name/table_name/custom_segment3/LOAD[00000001...].csv
```

Here, AWS DMS stores a series of .csv files in each of the three partition subfolders. The series of .csv files in each partition subfolder is named incrementally starting from `LOAD00000001.csv` until all the data is migrated.

In some cases, you might not explicitly name partition subfolders for a `ranges` parallel-load type using the `segment-names` setting. In these case, AWS DMS applies the default of creating each series of .csv files under its *`table_name`* subfolder. Here, AWS DMS prefixes the file names of each series of .csv files with the name of the running source segment index, as follows.

```
.../database_schema_name/table_name/SEGMENT1_LOAD[00000001...].csv
.../database_schema_name/table_name/SEGMENT2_LOAD[00000001...].csv
.../database_schema_name/table_name/SEGMENT3_LOAD[00000001...].csv
...
.../database_schema_name/table_name/SEGMENTZ_LOAD[00000001...].csv
```

Endpoint settings when using Amazon S3 as a target for AWS DMS

You can use endpoint settings to configure your Amazon S3 target database similar to using extra connection attributes. You specify the settings when you create the target endpoint using the AWS DMS console, or by using the `create-endpoint` command in the [AWS CLI](https://docs.aws.amazon.com/cli/latest/reference/dms/index.html), with the `--s3-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

Note

DMS writes changes to Parquet files based on the commit order from the source database, but when migrating multiple tables, the original transaction order is not preserved due to table-level partitioning. To maintain transaction sequence information, configure the `TimestampColumnName` endpoint setting to include the source commit timestamp for each row, which you can then use in downstream processing to reconstruct the original transaction sequence. Unlike CSV format, which offers the `PreserveTransactions` setting, Parquet files handle transactions differently due to their columnar storage structure, but this approach enables accurate tracking of source commit times, supports post-migration transaction order reconstruction, and allows efficient data processing while maintaining data consistency.

The following table shows the endpoint settings that you can use with Amazon S3 as a target.

Option	Description
`CsvNullValue`	An optional parameter that specifies how AWS DMS treats null values. While handling the null value, you can use this parameter to pass a user-defined string as null when writing to the target. For example, when target columns are nullable, you can use this option to different iate between the empty string value and the null value. Default value: `""` Valid values: any valid string Example: `--s3-settings '{"CsvNullValue": "NULL"}'` If the source database column value is null, in S3 CSV file, the column value is `NULL` instead of "" string.
`AddColumnName`	An optional parameter that when set to `true` or `y` you can use to add column name information to the .csv output file. You can't use this parameter with `PreserveTransactions` or `CdcPath`.

Option	Description
	Default value: `false` Valid values: `true`, `false`, `y`, `n` Example: `--s3-settings '{"AddColumnName": true}'`
`AddTrailingPaddingCharacter`	Use the S3 target endpoint setting `AddTrailingPaddingCharacter` to add padding on string data. The default value is `false`. Type: Boolean Example: `--s3-settings '{"AddTrailingPaddingCharacter": true}'`
`BucketFolder`	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 *`BucketFolder`* /*`schema_name`* /*`table_name`* /. If this parameter isn't specified, then the path used is *`schema_name`* /*`table_name`* /. Example: `--s3-settings '{"BucketFolder": "testFolder"}'`
`BucketName`	The name of the S3 bucket where S3 target objects are created as .csv or .parquet files. Example: `--s3-settings '{"BucketName": "buckettest"}'`

Option	Description
`CannedAclForObjects`	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 Canned ACL in the *Amazon S3 Developer Guide*. Default value: NONE Valid values for this attribute are: NONE; PRIVATE; PUBLIC_READ; PUBLIC_READ_WRITE; AUTHENTICATED_READ; AWS_EXEC_READ; BUCKET_OWNER_READ; BUCKET_OWNER_FULL_CONTROL. Example: `--s3-settings '{"CannedAclForObjects": "PUBLIC_READ"}'`

Option	Description
`CdcInsertsOnly`	An optional parameter during a change data capture (CDC) load to write only INSERT operations to the comma-separated value (.csv) or columnar storage (.parquet) output files. By default (the `false` setting), the first field in a .csv or .parquet record contains the letter I (INSERT), U (UPDATE), or D (DELETE). This letter indicates whether the row was inserted, updated, or deleted at the source database for a CDC load to the target. If `cdcInsertsOnly` is set to `true` or `y`, only INSERTs from the source database are migrated to the .csv or .parquet file. For .csv format only, how these INSERTS are recorded depends on the value of `IncludeOpForFullLoad` . If `IncludeOpForFullLoad` is set to `true`, the first field of every CDC record is set to I to indicate the INSERT operation at the source. If `IncludeOpForFullLoad` is set to `false`, 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 [Indicating source DB operations in migrated S3 data](). Default value: `false` Valid values: `true`, `false`, `y`, `n` Example: `--s3-settings '{"CdcInsertsOnly": true}'`

Option	Description
`CdcInsertsAndUpdates`	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 `false`, but when `cdcInsertsAndUpdates` is set to `true` or `y`, INSERTs and UPDATEs from the source database are migrated to the .csv or .parquet file. For .csv file format only, how these INSERTs and UPDATEs are recorded depends on the value of the `includeOpForFullLoad` parameter. If `includeOpForFullLoad` is set to `true`, the first field of every CDC record is set to either `I` or `U` to indicate INSERT and UPDATE operations at the source. But if `includeOpForFullLoad` is set to `false`, CDC records are written without an indication of INSERT or UPDATE operations at the source. For more information about how these parameters work together, see [Indicating source DB operations in migrated S3 data](#). **Note** `CdcInsertsOnly` and `cdcInsertsAndUpdates` can't both be set to true for the same endpoint. Set either `cdcInsertsOnly` or `cdcInsertsAndUpdates` to `true` for the same endpoint, but not both. Default value: `false` Valid values: `true`, `false`, `y`, `n` Example: `--s3-settings '{"CdcInsertsAndUpdates": true}'`

Option	Description
`CdcPath`	Specifies the folder path of CDC files. For an S3 source, this setting is required if a task captures change data; otherwise, it's optional. If `CdcPath` is set, DMS reads CDC files from this path and replicates the data changes to the target endpoint. For an S3 target if you set `PreserveTransactions` to true, DMS verifies that you have set this parameter to a folder path on your S3 target where DMS can save the transaction order for the CDC load. DMS creates this CDC folder path in either your S3 target working directory or the S3 target location specified by `BucketFolder` and `BucketName`. You can't use this parameter with `DatePartitionEnabled` or `AddColumnName`. Type: String For example, if you specify `CdcPath` as `MyChangedData`, and you specify `BucketName` as `MyTargetBucket` but do not specify `BucketFolder`, DMS creates the following CDC folder path: `MyTargetBucket/MyChangedData`. If you specify the same `CdcPath`, and you specify `BucketName` as `MyTargetBucket` and `BucketFolder` as `MyTargetData`, DMS creates the following CDC folder path: `MyTargetBucket/MyTargetData/MyChangedData`. **Note** This setting is supported in AWS DMS versions 3.4.2 and higher. When capturing data changes in transaction order, DMS always stores the row changes in .csv files regardless of the value of the DataFormat S3 setting on the target.

Option	Description
`CdcMaxBatchInterval`	Maximum interval length condition, defined in seconds, to output a file to Amazon S3. Default Value: 60 seconds When `CdcMaxBatchInterval` is specified and `CdcMinFileSize` is specified, the file write is triggered by whichever parameter condition is met first. **Note** Starting with AWS DMS version 3.5.3, when using PostgreSQL or Aurora PostgreSQL as the source and Amazon S3 with Parquet as the target, the frequency of `confirmed_flush_lsn` updates depends on the amount of data the target endpoint is configured to retain in memory. AWS DMS sends the `confirmed_flush_lsn` back to the source only after the data in memory is written to Amazon S3. If you configure the `CdcMaxBatchInterval` parameter to a higher value, you may observe increased replication slot usage on the source database.
`CdcMinFileSize`	Minimum file size condition as defined in kilobytes to output a file to Amazon S3. Default Value: 32000 KB When `CdcMinFileSize` is specified and `CdcMaxBatchInterval` is specified, the file write is triggered by whichever parameter condition is met first.

Option	Description
`PreserveTransactions`	If set to `true`, DMS saves the transaction order for change data capture (CDC) on the Amazon S3 target specified by `CdcPath`. You can't use this parameter with `DatePartitionEnabled` or `AddColumnName` . Type: Boolean When capturing data changes in transaction order, DMS always stores the row changes in .csv files regardless of the value of the DataFormat S3 setting on the target. **Note** This setting is supported in AWS DMS versions 3.4.2 and higher.

Option	Description
`IncludeOpForFullLoad`	An optional parameter during a full load to write the INSERT operations to the comma-separated value (.csv) output files only. For full load, records can only be inserted. By default (the `false` 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 `IncludeOpForFullLoad` is set to `true` or `y`, the INSERT is recorded as an I annotation in the first field of the .csv file. **Note** This parameter works together with `CdcInsertsOnly` or `CdcInsertsAndUpdates` for output to .csv files only. For more information about how these parameters work together, see Indicating source DB operations in migrated S3 data. Default value: `false` Valid values: `true`, `false`, `y`, `n` Example: `--s3-settings '{"IncludeOpForFullLoad": true}'`
`CompressionType`	An optional parameter when set to `GZIP` uses GZIP to compress the target .csv files. When this parameter is set to the default, it leaves the files uncompressed. Default value: `NONE` Valid values: `GZIP` or `NONE` Example: `--s3-settings '{"CompressionType": "GZIP"}'`

Option	Description
`CsvDelimiter`	The delimiter used to separate columns in .csv source files. The default is a comma (,). Example: `--s3-settings '{"CsvDelimiter": ","}'`
`CsvRowDelimiter`	The delimiter used to separate rows in the .csv source files. The default is a newline (\n). Example: `--s3-settings '{"CsvRowDelimiter": "\n"}'`
`MaxFileSize`	A value that specifies the maximum size (in KB) of any .csv file to be created while migrating to an S3 target during full load. Default value: 1,048,576 KB (1 GB) Valid values: 1–1,048,576 Example: `--s3-settings '{"MaxFileSize": 512}'`
`Rfc4180`	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 `true` or `y` using Amazon S3 as a target, if the data has quotation marks, commas, 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. Default value: `true` Valid values: `true`, `false`, `y`, `n` Example: `--s3-settings '{"Rfc4180": false}'`

Option	Description
`EncryptionMode`	The server-side encryption mode that you want to encrypt your .csv or .parquet object files copied to S3. 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. **Note** 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`. Default value: `SSE_S3` Valid values: `SSE_S3` or `SSE_KMS` Example: `--s3-settings '{"EncryptionMode": SSE_S3}'`
`ServerSideEncryptionKmsKeyId`	If you set `EncryptionMode` to `SSE_KMS`, set this parameter to the Amazon Resource Name (ARN) for the 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 KMS key. For more information, see Creating AWS KMS keys to encrypt Amazon S3 target objects. Example: `--s3-settings '{"ServerSideEncryptionKmsKeyId":"arn:aws:kms:us-east-1:111122223333:key/11a1a1a1-aaaa-9999-abab-2bbbbbb222a2"}'`

Option	Description
`DataFormat`	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 informati on about .parquet files, see https://parquet.apache.org/. Default value: `csv` Valid values: `csv` or `parquet` Example: `--s3-settings '{"DataFormat": "parquet"}'`
`EncodingType`	The Parquet encoding type. The encoding type options include the following: • `rle-dictionary` – This dictionary encoding uses a combination of bit-packing and run-length encoding to more efficiently store repeating values. • `plain` – No encoding. • `plain-dictionary` – 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. Default value: `rle-dictionary` Valid values: `rle-dictionary` , `plain`, or `plain-dictionary` Example: `--s3-settings '{"EncodingType": "plain-dictionary"}'`

Option	Description
`DictPageSizeLimit`	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. Default value: 1,024,000 (1 MB) Valid values: Any valid integer value Example: `--s3-settings '{"DictPageSizeLimit": 2,048,000}'`
`RowGroupLength`	The number of rows in one row group of a .parquet file. Default value: 10,024 (10 KB) Valid values: Any valid integer Example: `--s3-settings '{"RowGroupLength": 20,048}'`
`DataPageSize`	The maximum allowed size, in bytes, for a data page in a .parquet file. Default value: 1,024,000 (1 MB) Valid values: Any valid integer Example: `--s3-settings '{"DataPageSize": 2,048,000}'`
`ParquetVersion`	The version of the .parquet file format. Default value: `PARQUET_1_0` Valid values: `PARQUET_1_0` or `PARQUET_2_0` Example: `--s3-settings '{"ParquetVersion": "PARQUET_2_0"}'`

Option	Description
`EnableStatistics`	Set to `true` or `y` to enable statistics about .parquet file pages and row groups. Default value: `true` Valid values: `true`, `false`, `y`, `n` Example: `--s3-settings '{"EnableStatistics": false}'`
`TimestampColumnName`	An optional parameter to include a timestamp column in the S3 target endpoint data. AWS DMS includes an additional `STRING` column in the .csv or .parquet object files of your migrated data when you set `TimestampColumnName` to a non blank value. 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. For a CDC load, each row of the timestamp column contains the timestamp for the commit of that row in the source database. 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. When the `AddColumnName` parameter is set to `true`, DMS also includes the name for the timestamp column that you set as the non blank value of `TimestampColumnName`. Example: `--s3-settings '{"TimestampColumnName": "TIMESTAMP"}'`

Option	Description
`UseTaskStartTimeForFullLoadTimestamp`	When set to `true`, this parameter uses the task start time as the timestamp column value instead of the time data is written to target. For full load, when `UseTaskStartTimeForFullLoadTimestamp` is set to `true`, each row of the timestamp column contains the task start time. For CDC loads, each row of the timestamp column contains the transaction commit time. When `UseTaskStartTimeForFullLoadTimestamp` is set to `false`, the full load timestamp in the timestamp column increments with the time data arrives at the target. Default value: `false` Valid values: `true`, `false` Example: `--s3-settings '{"UseTaskStartTimeForFullLoadTimestamp": true}'` `UseTaskStartTimeForFullLoadTimestamp: true` helps make the S3 target `TimestampColumnName` for a full load sortable with `TimestampColumnName` for a CDC load.

Option	Description
`ParquetTimestampInMillisecond`	An optional parameter that specifies the precision of any `TIMESTAMP` column values written to an S3 object file in .parquet format. When this attribute is set to `true` or `y`, AWS DMS writes all `TIMESTAMP` columns in a .parquet formatted file with millisecond precision. Otherwise, DMS writes them with microsecond precision. Currently, Amazon Athena and AWS Glue can handle only millisecond precision for `TIMESTAMP` 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. **Note** • AWS DMS writes any `TIMESTAMP` column values written to an S3 file in .csv format with microsecond precision. • The setting of this attribute has no effect on the string format of the timestamp column value inserted by setting the `TimestampColumnName` attribute. Default value: `false` Valid values: `true`, `false`, `y`, `n` Example: `--s3-settings '{"ParquetTimestampInMillisecond": true}'`

Option	Description
`GlueCatalogGeneration`	To generate an AWS Glue Data Catalog, set this endpoint setting to `true`. Default value: `false` Valid values: `true`, `false`, Example: `--s3-settings '{"GlueCatalogGeneration": true}'` **Note:** Don't use `GlueCatalogGeneration` with `PreserveTransactions` and `CdcPath`.

Using AWS Glue Data Catalog with an Amazon S3 target for AWS DMS

AWS Glue is a service that provides simple ways to categorize data, and consists of a metadata repository known as AWS Glue Data Catalog. You can integrate AWS Glue Data Catalog with your Amazon S3 target endpoint and query Amazon S3 data through other AWS services such as Amazon Athena. Amazon Redshift works with AWS Glue but AWS DMS doesn't support that as a pre-built option.

To generate the data catalog, set the `GlueCatalogGeneration` endpoint setting to `true`, as shown in the following AWS CLI 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", "DataFormat": "parquet", "GlueCatalogGeneration":
 true}'
```

For a Full load replication task that includes `csv` type data, set `IncludeOpForFullLoad` to `true`.

Don't use `GlueCatalogGeneration` with `PreserveTransactions` and `CdcPath`. The AWS Glue crawler can't reconcile the different schemas of files stored under the specified `CdcPath`.

For Amazon Athena to index your Amazon S3 data, and for you to query your data using standard SQL queries through Amazon Athena, the IAM role attached to the endpoint must have the following policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload"
            ],
            "Resource": [
                "arn:aws:s3:::bucket123",
                "arn:aws:s3:::bucket123/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:GetDatabase",
                "glue:CreateTable",
                "glue:DeleteTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition"
            ],
            "Resource": [
                "arn:aws:glue:*:111122223333:catalog",
```

```
                "arn:aws:glue:*:111122223333:database/*",
                "arn:aws:glue:*:111122223333:table/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:StartQueryExecution",
                "athena:GetQueryExecution",
                "athena:CreateWorkGroup"
            ],
            "Resource": "arn:aws:athena:*:111122223333:workgroup/
glue_catalog_generation_for_task_*"
        }
    ]
}
```

References

- For more information about AWS Glue, see Concepts in the *AWS Glue Developer Guide* .
- For more information about AWS Glue Data Catalog see Components in the *AWS Glue Developer Guide* .

Using data encryption, parquet files, and CDC on your Amazon S3 target

You can use S3 target endpoint settings to configure the following:

- A custom KMS key to encrypt your S3 target objects.
- Parquet files as the storage format for S3 target objects.
- Change data capture (CDC) including transaction order on the S3 target.
- Integrate AWS Glue Data Catalog with your Amazon S3 target endpoint and query Amazon S3 data through other services such as Amazon Athena.

AWS KMS key settings for data encryption

The following examples show configuring a custom 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 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", "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.

Capturing data changes (CDC) including transaction order on the S3 target

By default when AWS DMS runs a CDC task, it stores all the row changes logged in your source database (or databases) in one or more files for each table. Each set of files containing changes for the same table reside in a single target directory associated with that table. AWS DMS creates as many target directories as database tables migrated to the Amazon S3 target endpoint. The files are stored on the S3 target in these directories without regard to transaction order. For more information on the file naming conventions, data contents, and format, see Using Amazon S3 as a target for AWS Database Migration Service.

To capture source database changes in a manner that also captures the transaction order, you can specify S3 endpoint settings that direct AWS DMS to store the row changes for *all* database tables in one or more .csv files created depending on transaction size. These .csv *transaction files* contain all row changes listed sequentially in transaction order for all tables involved in each transaction. These transaction files reside together in a single *transaction directory* that you also specify on the S3 target. In each transaction file, the transaction operation and the identity of the database and source table for each row change is stored as part of the row data as follows.

```
operation,table_name,database_schema_name,field_value,...
```

Here, `operation` is the transaction operation on the changed row, `table_name` is the name of the database table where the row is changed, `database_schema_name` is the name of the database schema where the table resides, and `field_value` is the first of one or more field values that specify the data for the row.

The example following of a transaction file shows changed rows for one or more transactions that involve two tables.

```
I,Names_03cdcad11a,rdsTempsdb,13,Daniel
U,Names_03cdcad11a,rdsTempsdb,23,Kathy
D,Names_03cdcad11a,rdsTempsdb,13,Cathy
I,Names_6d152ce62d,rdsTempsdb,15,Jane
I,Names_6d152ce62d,rdsTempsdb,24,Chris
I,Names_03cdcad11a,rdsTempsdb,16,Mike
```

Here, the transaction operation on each row is indicated by `I` (insert), `U` (update), or `D` (delete) in the first column. The table name is the second column value (for example, `Names_03cdcad11a`). The name of the database schema is the value of the third column (for example, `rdsTempsdb`). And the remaining columns are populated with your own row data (for example, `13,Daniel`).

In addition, AWS DMS names the transaction files it creates on the Amazon S3 target using a time stamp according to the following naming convention.

```
CDC_TXN-timestamp.csv
```

Here, *timestamp* is the time when the transaction file was created, as in the following example.

```
CDC_TXN-20201117153046033.csv
```

This time stamp in the file name ensures that the transaction files are created and listed in transaction order when you list them in their transaction directory.

> **Note**
>
> When capturing data changes in transaction order, AWS DMS always stores the row changes in .csv files regardless of the value of the `DataFormat` S3 setting on the target.

To control the frequency of writes to an Amazon S3 target during a data replication task, you can configure the `CdcMaxBatchInterval` and `CdcMinFileSize` settings. This can result in better performance when analyzing the data without any additional overhead operations. For more information, see Endpoint settings when using Amazon S3 as a target for AWS DMS

To tell AWS DMS to store all row changes in transaction order

1. Set the `PreserveTransactions` S3 setting on the target to `true`.
2. Set the `CdcPath` S3 setting on the target to a relative folder path where you want AWS DMS to store the .csv transaction files.

   AWS DMS creates this path either under the default S3 target bucket and working directory or under the bucket and bucket folder that you specify using the `BucketName` and `BucketFolder` S3 settings on the target.

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. How AWS DMS creates and sets this first field depends on the migration task type and settings of `includeOpForFullLoad`, `cdcInsertsOnly`, and `cdcInsertsAndUpdates`.

For a full load when `includeOpForFullLoad` is `true`, AWS 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), AWS 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.

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.

With these parameter settings		**DMS sets target records as follows for .csv and .parquet output**	
includeOpForFullLoad	**cdcInsertsOnly**	**For full load**	**For CDC load**
`true`	`true`	Added first field value set to `I`	Added first field value set to `I`
`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**	
includeOpForFullLoad	**cdcInsertsAndUpdates**	**For full load**	**For CDC load**
`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`

Target data types for S3 Parquet

The following table shows the Parquet 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.

AWS DMS data type	S3 parquet data type
`BYTES`	`BINARY`
`DATE`	`DATE32`
`TIME`	`TIME32`

AWS DMS data type	S3 parquet data type
DATETIME	TIMESTAMP
INT1	INT8
INT2	INT16
INT4	INT32
INT8	INT64
NUMERIC	DECIMAL
REAL4	FLOAT
REAL8	DOUBLE
STRING	STRING
UINT1	UINT8
UINT2	UINT16
UINT4	UINT32
UINT8	UINT64
WSTRING	STRING
BLOB	BINARY
NCLOB	STRING
CLOB	STRING
BOOLEAN	BOOL

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.

> **Note**
>
> The **SSL Mode** option on the AWS DMS console or API doesn't apply to some data streaming and NoSQL services like Kinesis and DynamoDB. They are secure by default, so AWS DMS shows the SSL mode setting is equal to none (**SSL Mode=None**). You don't need to provide any additional configuration for your endpoint to make use of SSL. For example, when using DynamoDB as a target endpoint, it is secure by default. All API calls to DynamoDB use SSL, so there is no need for an additional SSL option in the AWS DMS endpoint. You can securely put data and retrieve data through SSL endpoints using the HTTPS protocol, which AWS DMS uses by default when connecting to a DynamoDB database.

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.

> **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.

- `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 and collection settings rules and operations](#).

> **Note**
>
> 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.
> - Standard AWS DMS usage fees apply to resources you use.

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 grant access to the DynamoDB tables that are being migrated into. The minimum set of access permissions is shown in the following IAM policy.

JSON

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

JSON

```
{
    "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-2:111122223333:table/name1",
                "arn:aws:dynamodb:us-west-2:111122223333:table/OtherName*",
                "arn:aws:dynamodb:us-west-2:111122223333:table/
awsdms_apply_exceptions",
                "arn:aws:dynamodb:us-west-2:111122223333: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.
- AWS DMS doesn't support the task setting `TargetTablePrepMode=TRUNCATE_BEFORE_LOAD` for DynamoDB as a target.
- AWS DMS doesn't support the task setting `TaskRecoveryTableEnabled` for DynamoDB as a target.
- `BatchApply` is not supported for a DynamoDB endpoint.
- AWS DMS cannot migrate attributes whose names match reserved words in DynamoDB. For more information, see Reserved words in DynamoDB in the *Amazon DynamoDB Developer Guide*.

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	income
Daniel	Sheridan	Dan	101 Main St Cambridge, MA	800-867-5309	100 Secret St, Unknownville, MA	123-456-7890	12345678

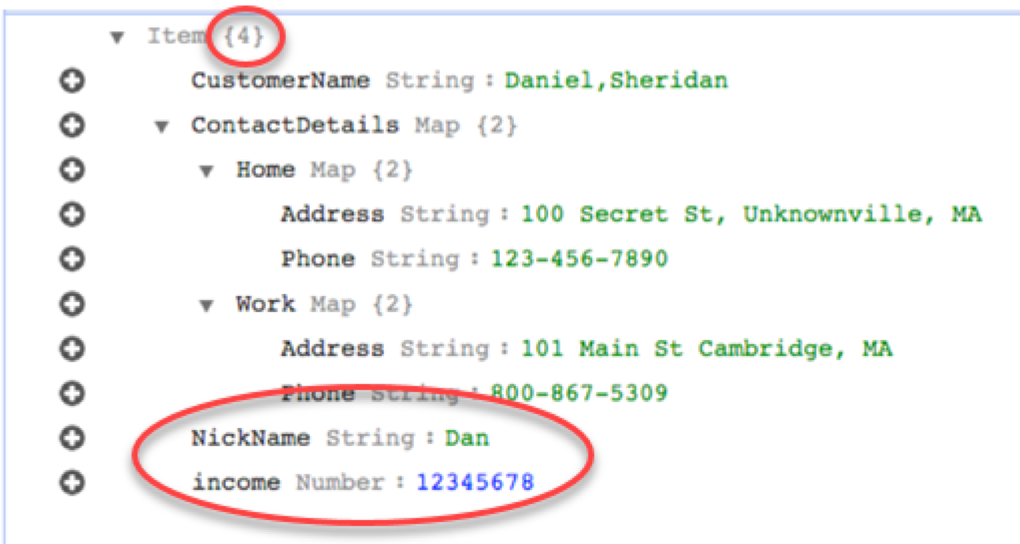
To migrate this information to DynamoDB, you create rules to map the data into a DynamoDB table item. Note the columns listed for the `exclude-columns` parameter. These columns are not directly mapped over to the target. Instead, attribute mapping is used to combine the data into new items, such as where *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"
```

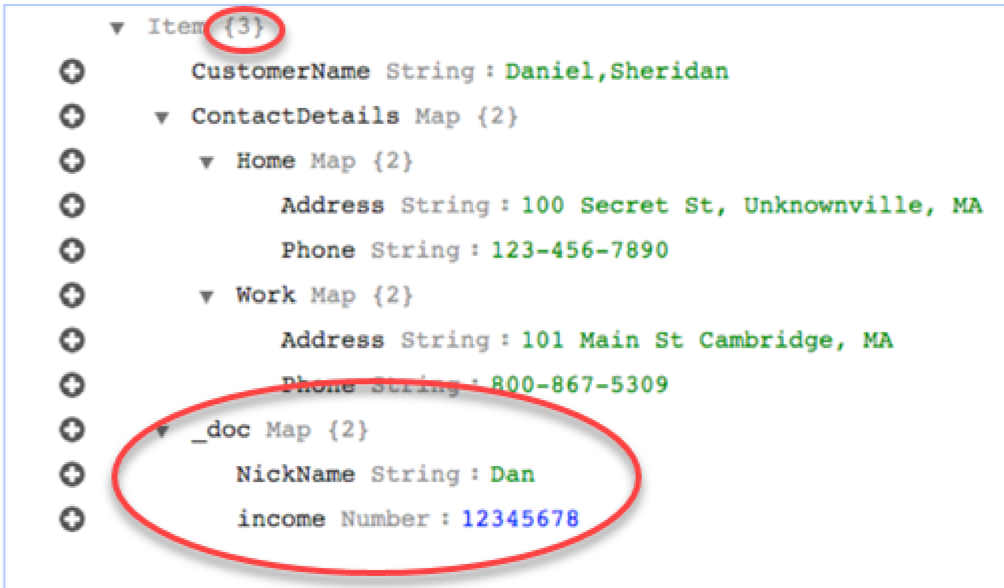
```
            },
            "target-table-name": "customer_t",
            "mapping-parameters": {
                "partition-key-name": "CustomerName",
                "exclude-columns": [
                    "FirstName",
                    "LastName",
                    "HomeAddress",
                    "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 (required). Specifies a DynamoDB json structure of the attribute value. This is useful for comparing an attribute with a value in DynamoDB that you might not know until runtime. You can define an expression attribute value as a placeholder for an actual value.
- expression attribute names (required). This helps avoid potential conflicts with any DynamoDB reserved words, attribute names containing special characters, and similar.
- 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.

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	Las	St	Home ss	HomeF	WorkAddr ss	Worl	DateOfBirth
Primary Key		N/A					
Ran	Mar	5	221B Baker Street	123456 0	31 Spooner Street, Quahog	9876 0	02/29/1988

The structure of the DynamoDB data is as follows:

Custor ame	StoreId	ContactDetails	DateOfBirth
Partition Key	Sort Key	N/A	
Randy sh	5	{ "Name": "Randy", "Home": { "Address": "221B Baker Street", "Phone": 1234567890 }, "Work": { "Address": "31 Spooner Street, Quahog", "Phone": 9876541230 }	02/29/1988

CustomerName	StoreId	ContactDetails	DateOfBirth
		}	

The following JSON shows the object mapping and column mapping used to achieve the DynamoDB structure:

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "test",
                "table-name": "%"
            },
            "rule-action": "include"
        },
        {
            "rule-type": "object-mapping",
            "rule-id": "2",
            "rule-name": "TransformToDDB",
            "rule-action": "map-record-to-record",
            "object-locator": {
                "schema-name": "test",
                "table-name": "customer"
            },
            "target-table-name": "customer_t",
            "mapping-parameters": {
                "partition-key-name": "CustomerName",
                "sort-key-name": "StoreId",
                "exclude-columns": [
                    "FirstName",
                    "LastName",
                    "HomeAddress",
                    "HomePhone",
                    "WorkAddress",
                    "WorkPhone"
                ],
```

```
                "attribute-mappings": [
                    {
                        "target-attribute-name": "CustomerName",
                        "attribute-type": "scalar",
                        "attribute-sub-type": "string",
                        "value": "${FirstName},${LastName}"
                    },
                    {
                        "target-attribute-name": "StoreId",
                        "attribute-type": "scalar",
                        "attribute-sub-type": "string",
                        "value": "${StoreId}"
                    },
                    {
                        "target-attribute-name": "ContactDetails",
                        "attribute-type": "scalar",
                        "attribute-sub-type": "string",
                        "value": "{\"Name\":\"${FirstName}\",\"Home\":{\"Address
\":\"${HomeAddress}\",\"Phone\":\"${HomePhone}\"}, \"Work\":{\"Address\":
\"${WorkAddress}\",\"Phone\":\"${WorkPhone}\"}}"
                    }
                ]
            }
        }
    ]
}
```

Another way to use column mapping is to use DynamoDB format as your document type. The following code example uses *dynamodb-map* as the `attribute-sub-type` for attribute mapping.

```
{
    "rules": [
        {
            "rule-type": "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}"
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    ]
                }
            }
    ]
}
```

As an alternative to `dynamodb-map`, you can use `dynamodb-list` as the attribute-sub-type for attribute mapping, as shown in the following example.

```
{
"target-attribute-name": "ContactDetailsList",
"attribute-type": "document",
"attribute-sub-type": "dynamodb-list",
"value": {
    "L": [
            {
                "N": "${FirstName}"
            },
```

```
            {
                "N": "${HomeAddress}"
            },
            {
                "N": "${HomePhone}"
            },
            {
                "N": "${WorkAddress}"
            },
            {
                "N": "${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.

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 Data Firehose to deliver Kinesis data to an Amazon S3 destination, and then query it using various query engines including Amazon Athena.

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.

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.

> **Note**
>
> The **SSL Mode** option on the AWS DMS console or API doesn’t apply to some data streaming and NoSQL services like Kinesis and DynamoDB. They are secure by default, so AWS DMS shows the SSL mode setting is equal to none (**SSL Mode=None**). You don’t need to provide any additional configuration for your endpoint to make use of SSL. For example, when using Kinesis as a target endpoint, it is secure by default. All API calls to Kinesis use SSL, so there is no need for an additional SSL option in the AWS DMS endpoint. You can securely put data and retrieve data through SSL endpoints using the HTTPS protocol, which AWS DMS uses by default when connecting to a Kinesis Data Stream.

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.

You can set connection settings in the following ways:

- In the AWS DMS console, using endpoint settings.
- In the CLI, using the `kinesis-settings` option of the CreateEndpoint command.

In the CLI, use the following request parameters of the `kinesis-settings` option:

> **Note**
>
> Support for the `IncludeNullAndEmpty` endpoint setting is available in AWS DMS version 3.4.1 and higher. But support for the other following endpoint settings for Kinesis Data Streams targets is available in AWS DMS.

- `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).
- `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`.

- `UseLargeIntegerValue` – Use up to 18 digit int instead of casting ints as doubles, available from AWS DMS version 3.5.4. 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 \
    --region <aws-region> \
    --endpoint-identifier <user-endpoint-identifier> \
    --endpoint-type target \
    --engine-name kinesis \
    --kinesis-settings ServiceAccessRoleArn=arn:aws:iam::<account-id>:role/<kinesis-
role-name>,StreamArn=arn:aws:kinesis:<aws-region>:<account-id>:stream/<stream-
name>,MessageFormat=json-unformatted,
IncludeControlDetails=true,IncludeTransactionDetails=true,IncludePartitionValue=true,PartitionI
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.

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.

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 100 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. The default value is 1 and the maximum value is 512.

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.

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). To provide data for all columns (changed or not), you need to set `REPLICA_IDENTITY` to `FULL` instead of `DEFAULT`. Note that you should choose the `REPLICA_IDENTITY` setting carefully for each table. If you set `REPLICA_IDENTITY` to `FULL`, all of the column values are written to write-ahead logging (WAL) continuously. This may cause performance or resource issues with tables that are updated frequently.
- MySQL generally provides data for all columns except for BLOB and CLOB data types (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**
>
> Only apply `BeforeImageSettings` to AWS DMS tasks that contain a CDC component, such as 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 any column that has a before image value, use `all`. Note that the before image does not contain columns with LOB data types, such as CLOB or BLOB.

```
"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 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",
      "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.

Prerequisites for using a Kinesis data stream as a target for AWS Database Migration Service

IAM role 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.

JSON

```
{
   "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.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:DescribeStream",
        "kinesis:PutRecord",
        "kinesis:PutRecords"
      ],
      "Resource": "*"
    }
  ]
}
```

Accessing a Kinesis data stream as a target for AWS Database Migration Service

In AWS DMS version 3.4.7 and higher, to connect to an Kinesis endpoint, you must do one of the following:

- Configure DMS to use VPC endpoints. For information about configuring DMS to use VPC endpoints, see Configuring VPC endpoints for AWS DMS.

- Configure DMS to use public routes, that is, make your replication instance public. For information about public replication instances, see Public and private replication instances.

Limitations when using Kinesis Data Streams as a target for AWS Database Migration Service

The following limitations apply when using Kinesis Data Streams as a target:

- AWS DMS publishes each update to a single record in the source database as one data record in a given Kinesis data stream regardless of transactions. However, you can include transaction details for each data record by using relevant parameters of the `KinesisSettings` API.
- Full LOB mode is not supported.
- The maximum supported LOB size is 1 MB.
- 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 four 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.
  - `transaction-id`: The CDC transaction ID. All records within the same transaction go to the same partition.
  - `constant`: A fixed literal value for every record regardless of table or data. All records are sent to the same partition key value "constant", providing strict global ordering across all tables.

```
{
    "rule-type": "object-mapping",
    "rule-id": "2",
    "rule-name": "PartitionKeyTypeExample",
    "rule-action": "map-record-to-document",
    "object-locator": {
        "schema-name": "onprem",
        "table-name": "it_system"
    },
    "mapping-parameters": {
        "partition-key-type": "transaction-id | constant | attribute-name | schema-
table"
```

```
    }
}
```

- 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*.
- The `IncludeTransactionDetails` endpoint setting is only supported when the source endpoint is Oracle, SQL Server, PostgreSQL, or MySQL. For other source endpoint types, transaction details will not be included.
- `BatchApply` is not supported for a Kinesis endpoint. Using Batch Apply (for example, the `BatchApplyEnabled` target metadata task setting) for a Kinesis target causes task failure and data loss. Do not enable `BatchApply` when using Kinesis as a target endpoint.
- Kinesis targets are only supported for a Kinesis data stream in the same AWS account and the same AWS Region as the replication instance.
- When migrating from a MySQL source, the BeforeImage data doesn't include CLOB and BLOB data types. For more information, see [Using a before image to view original values of CDC rows for a Kinesis data stream as a target](#).
- AWS DMS doesn't support migrating values of `BigInt` data type with more than 16 digits. To work around this limitation, you can use the following transformation rule to convert the `BigInt` column to a string. For more information about transformation rules, see [Transformation rules and actions](#).

```
{
    "rule-type": "transformation",
    "rule-id": "id",
    "rule-name": "name",
    "rule-target": "column",
    "object-locator": {
        "schema-name": "valid object-mapping rule action",
        "table-name": "",
        "column-name": ""
    },
    "rule-action": "change-data-type",
    "data-type": {
        "type": "string",
        "length": 20
    }
}
```

- When multiple DML operations within a single transaction modify a Large Object (LOB) column on the source database, the target database retains only the final LOB value from the last operation in that transaction. The intermediate LOB values set by earlier operations in the same transaction are overwritten, which can result in potential data loss or inconsistencies. This behavior occurs due to how LOB data is processed during replication.
- AWS DMS does not support source data containing embedded `'\0'` characters when using Kinesis as a target endpoint. Data containing embedded `'\0'` characters will be truncated at the first `'\0'` character.

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. These settings affect values that aren't excluded as part of the `exclude-columns` attribute list. The `map-record-to-record` and `map-record-to-document` values specify how AWS DMS handles these records by default. 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`, note the following:

- This setting only affects columns excluded by the `exclude-columns` list.
- For every such column, AWS DMS creates a corresponding attribute in the target topic.
- AWS DMS creates this corresponding attribute regardless of whether the 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	1234567890	31 Spooner Street, Quahog	9876543210	02/29/1988

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 `"partition-key-type: "constant"`, this sets the partition value to `constant`. 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": "2",
            "rule-name": "TransformToKinesis",
            "rule-action": "map-record-to-document",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "Customer"
            },
            "mapping-parameters": {
                "partition-key-type": "constant",
                "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": "scalar",
                            "attribute-sub-type": "string",
                            "value": {
                                "Home": {
                                    "Address": "${HomeAddress}",
                                    "Phone": "${HomePhone}"
                                },
                                "Work": {
                                    "Address": "${WorkAddress}",
                                    "Phone": "${WorkPhone}"
                                }
                            }
                        },
                        {
                            "target-attribute-name": "DateOfBirth",
                            "attribute-type": "scalar",
                            "attribute-sub-type": "string",
                            "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.

When `partition-key-type` is set to `attribute-name` in a table mapping rule, you must specify `partition-key-name`, which must reference either a column from the source table or a custom column defined in the mapping. Additionally, `attribute-mappings` must be provided to define how source columns map to the target Kinesis Stream.

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.

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 `load`, `insert`, `update`, or `delete`.

For control records, the operation can be `create-table`, `rename-table`, `drop-table`, `change-columns`, `add-column`, `drop-column`, `rename-column`, or `column-type-change`.

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.

Your Kafka cluster can be either an Amazon MSK instance, a cluster running on an Amazon EC2 instance, or an on-premises cluster. An Amazon MSK instance or a cluster on an Amazon EC2 instance can be in the same VPC or a different one. If your cluster is on-premises, you can use your own on-premises name server for your replication instance to resolve the cluster's host name. For information about setting up a name server for your replication instance, see Using your own on-premises name server. For more information about setting up a network, see Setting up a network for a replication instance.

When using an Amazon MSK cluster, make sure that its security group allows access from your replication instance. For information about changing the security group for an Amazon MSK cluster, see Changing an Amazon MSK cluster's security group.

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.

Currently, AWS DMS supports a single topic per task. For a single task with multiple tables, all messages go to a single topic. Each message includes a metadata section that identifies the target schema and table. AWS DMS versions 3.4.6 and higher support multitopic replication using object mapping. For more information, see Multitopic replication using object mapping.

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 locations of one or more brokers in your Kafka cluster in the form of a comma-separated list of each *broker-hostname*:*port*. An example is `"ec2-12-345-678-901.compute-1.amazonaws.com:2345,ec2-10-987-654-321.compute-1.a` This setting can specify the locations of any or all brokers 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

- `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.

> **Note**
>
> You can only use the AWS CLI/SDK to change `MessageMaxBytes` to a non-default value. For example, to modify your existing Kafka endpoint and change `MessageMaxBytes`, use the following command.
>
> ```
> aws dms modify-endpoint --endpoint-arn your-endpoint
> --kafka-settings Broker="broker1-server:broker1-port,broker2-server:broker2-
> port,...",
> Topic=topic-name,MessageMaxBytes=integer-of-max-message-size-in-bytes
> ```

- `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.
- SecurityProtocol – Sets a secure connection to a Kafka target endpoint using Transport Layer Security (TLS). Options include ssl-authentication, ssl-encryption, and sasl-ssl. Using sasl-ssl requires SaslUsername and SaslPassword.
- SslEndpointIdentificationAlgorithm – Sets hostname verification for the certificate. This setting is supported in AWS DMS version 3.5.1 and later. Options include the following:
  - NONE: Disable hostname verification of the broker in the client connection.
  - HTTPS: Enable hostname verification of the broker in the client connection.
- useLargeIntegerValue – Use up to 18 digit int instead of casting ints as doubles, available from AWS DMS version 3.5.4. The default is false.

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. The maximum is 512.

You can improve the performance of change data capture (CDC) for Kafka endpoints by tuning task settings for parallel threads and bulk operations. 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 100 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. The default is 1. The maximum is 512.

When using `ParallelApply*` task settings, the `partition-key-type` default is the `primary-key` of the table, not `schema-name.table-name`.

Connecting to Kafka using Transport Layer Security (TLS)

A Kafka cluster accepts secure connections using Transport Layer Security (TLS). With DMS, you can use any one of the following three security protocol options to secure a Kafka endpoint connection.

SSL encryption (`server-encryption`)

Clients validate server identity through the server's certificate. Then an encrypted connection is made between server and client.

SSL authentication (`mutual-authentication`)

Server and client validate the identity with each other through their own certificates. Then an encrypted connection is made between server and client.

SASL-SSL (`mutual-authentication`)

The Simple Authentication and Security Layer (SASL) method replaces the client's certificate with a user name and password to validate a client identity. Specifically, you provide a user name and password that the server has registered so that the server can validate the identity of a client. Then an encrypted connection is made between server and client.

> **Important**
>
> Apache Kafka and Amazon MSK accept resolved certificates. This is a known limitation of Kafka and Amazon MSK to be addressed. For more information, see Apache Kafka issues, KAFKA-3700.
>
> If you're using Amazon MSK, consider using access control lists (ACLs) as a workaround to this known limitation. For more information about using ACLs, see Apache Kafka ACLs section of *Amazon Managed Streaming for Apache Kafka Developer Guide*.
>
> If you're using a self-managed Kafka cluster, see Comment dated 21/Oct/18 for information about configuring your cluster.

Using SSL encryption with Amazon MSK or a self-managed Kafka cluster

You can use SSL encryption to secure an endpoint connection to Amazon MSK or a self-managed Kafka cluster. When you use the SSL encryption authentication method, clients validate a server's identity through the server's certificate. Then an encrypted connection is made between server and client.

To use SSL encryption to connect to Amazon MSK

- Set the security protocol endpoint setting (`SecurityProtocol`) using the `ssl-encryption` option when you create your target Kafka endpoint.

  The JSON example following sets the security protocol as SSL encryption.

```
"KafkaSettings": {
    "SecurityProtocol": "ssl-encryption",
}
```

To use SSL encryption for a self-managed Kafka cluster

1. If you're using a private Certification Authority (CA) in your on-premises Kafka cluster, upload your private CA cert and get an Amazon Resource Name (ARN).
2. Set the security protocol endpoint setting (`SecurityProtocol`) using the `ssl-encryption` option when you create your target Kafka endpoint. The JSON example following sets the security protocol as `ssl-encryption`.

```
"KafkaSettings": {
    "SecurityProtocol": "ssl-encryption",
}
```

3. If you're using a private CA, set `SslCaCertificateArn` in the ARN you got in the first step above.

Using SSL authentication

You can use SSL authentication to secure an endpoint connection to Amazon MSK or a self-managed Kafka cluster.

To enable client authentication and encryption using SSL authentication to connect to Amazon MSK, do the following:

- Prepare a private key and public certificate for Kafka.

- Upload certificates to the DMS certificate manager.
- Create a Kafka target endpoint with corresponding certificate ARNs specified in Kafka endpoint settings.

To prepare a private key and public certificate for Amazon MSK

1. Create an EC2 instance and set up a client to use authentication as described in steps 1 through 9 in the Client Authentication section of *Amazon Managed Streaming for Apache Kafka Developer Guide*.

   After you complete those steps, you have a Certificate-ARN (the public certificate ARN saved in ACM), and a private key contained within a `kafka.client.keystore.jks` file.

2. Get the public certificate and copy the certificate to the `signed-certificate-from-acm.pem` file, using the command following:

   ```
   aws acm-pca get-certificate --certificate-authority-arn Private_CA_ARN --
   certificate-arn Certificate_ARN
   ```

   That command returns information similar to the following example:

   ```
   {"Certificate": "123", "CertificateChain": "456"}
   ```

   You then copy your equivalent of `"123"` to the `signed-certificate-from-acm.pem` file.

3. Get the private key by importing the `msk-rsa` key from `kafka.client.keystore.jks to keystore.p12`, as shown in the following example.

   ```
   keytool -importkeystore \
   -srckeystore kafka.client.keystore.jks \
   -destkeystore keystore.p12 \
   -deststoretype PKCS12 \
   -srcalias msk-rsa-client \
   -deststorepass test1234 \
   -destkeypass test1234
   ```

4. Use the following command to export `keystore.p12` into `.pem` format.

   ```
   Openssl pkcs12 -in keystore.p12 -out encrypted-private-client-key.pem –nocerts
   ```

The **Enter PEM pass phrase** message appears and identifies the key that is applied to encrypt the certificate.

5. Remove bag attributes and key attributes from the `.pem` file to make sure that the first line starts with the following string.

```
---BEGIN ENCRYPTED PRIVATE KEY---
```

To upload a public certificate and private key to the DMS certificate manager and test the connection to Amazon MSK

1. Upload to DMS certificate manager using the following command.

```
aws dms import-certificate --certificate-identifier signed-cert --certificate-pem
 file://path to signed cert
aws dms import-certificate --certificate-identifier private-key —certificate-pem
 file://path to private key
```

2. Create an Amazon MSK target endpoint and test connection to make sure that TLS authentication works.

```
aws dms create-endpoint --endpoint-identifier $endpoint-identifier --engine-name
 kafka --endpoint-type target --kafka-settings
'{"Broker": "b-0.kafka260.aaaaa1.a99.kafka.us-east-1.amazonaws.com:0000",
 "SecurityProtocol":"ssl-authentication",
"SslClientCertificateArn": "arn:aws:dms:us-east-1:012346789012:cert:",
"SslClientKeyArn": "arn:aws:dms:us-
east-1:0123456789012:cert:","SslClientKeyPassword":"test1234"}'
aws dms test-connection -replication-instance-arn=$rep_inst_arn —endpoint-arn=
$kafka_tar_arn_msk
```

Important

You can use SSL authentication to secure a connection to a self-managed Kafka cluster. In some cases, you might use a private Certification Authority (CA) in your on-premises Kafka cluster. If so, upload your CA chain, public certificate, and private key to the DMS certificate manager. Then, use the corresponding Amazon Resource Name (ARN) in your endpoint settings when you create your on-premises Kafka target endpoint.

To prepare a private key and signed certificate for a self-managed Kafka cluster

1. Generate a key pair as shown in the following example.

```
keytool -genkey -keystore kafka.server.keystore.jks -validity 300 -storepass your-keystore-password
-keypass your-key-passphrase -dname "CN=your-cn-name"
-alias alias-of-key-pair -storetype pkcs12 -keyalg RSA
```

2. Generate a Certificate Sign Request (CSR).

```
keytool -keystore kafka.server.keystore.jks -certreq -file server-cert-sign-request-rsa -alias on-premise-rsa -storepass your-key-store-password
-keypass your-key-password
```

3. Use the CA in your cluster truststore to sign the CSR. If you don't have a CA, you can create your own private CA.

```
openssl req -new -x509 -keyout ca-key -out ca-cert -days validate-days
```

4. Import `ca-cert` into the server truststore and keystore. If you don't have a truststore, use the following command to create the truststore and import `ca-cert` into it.

```
keytool -keystore kafka.server.truststore.jks -alias CARoot -import -file ca-cert
keytool -keystore kafka.server.keystore.jks -alias CARoot -import -file ca-cert
```

5. Sign the certificate.

```
openssl x509 -req -CA ca-cert -CAkey ca-key -in server-cert-sign-request-rsa -out
 signed-server-certificate.pem
-days validate-days -CAcreateserial -passin pass:ca-password
```

6. Import the signed certificate to the keystore.

```
keytool -keystore kafka.server.keystore.jks -import -file signed-certificate.pem -
alias on-premise-rsa -storepass your-keystore-password
-keypass your-key-password
```

7. Use the following command to import the `on-premise-rsa` key from `kafka.server.keystore.jks` to `keystore.p12`.

```
keytool -importkeystore \
-srckeystore kafka.server.keystore.jks \
-destkeystore keystore.p12 \
-deststoretype PKCS12 \
-srcalias on-premise-rsa \
-deststorepass your-truststore-password \
-destkeypass your-key-password
```

8. Use the following command to export `keystore.p12` into `.pem` format.

```
Openssl pkcs12 -in keystore.p12 -out encrypted-private-server-key.pem –nocerts
```

9. Upload `encrypted-private-server-key.pem`, `signed-certificate.pem`, and `ca-cert` to the DMS certificate manager.

10. Create an endpoint by using the returned ARNs.

```
aws dms create-endpoint --endpoint-identifier $endpoint-identifier --engine-name
 kafka --endpoint-type target --kafka-settings
'{"Broker": "b-0.kafka260.aaaaa1.a99.kafka.us-east-1.amazonaws.com:9092",
 "SecurityProtocol":"ssl-authentication",
"SslClientCertificateArn": "your-client-cert-arn","SslClientKeyArn": "your-client-
key-arn","SslClientKeyPassword":"your-client-key-password",
"SslCaCertificateArn": "your-ca-certificate-arn"}'

aws dms test-connection -replication-instance-arn=$rep_inst_arn —endpoint-arn=
$kafka_tar_arn_msk
```

Using SASL-SSL authentication to connect to Amazon MSK

The Simple Authentication and Security Layer (SASL) method uses a user name and password to validate a client identity, and makes an encrypted connection between server and client.

To use SASL, you first create a secure user name and password when you set up your Amazon MSK cluster. For a description how to set up a secure user name and password for an Amazon MSK cluster, see Setting up SASL/SCRAM authentication for an Amazon MSK cluster in the *Amazon Managed Streaming for Apache Kafka Developer Guide*.

Then, when you create your Kafka target endpoint, set the security protocol endpoint setting (`SecurityProtocol`) using the `sasl-ssl` option. You also set `SaslUsername` and `SaslPassword` options. Make sure these are consistent with the secure user name and password that you created when you first set up your Amazon MSK cluster, as shown in the following JSON example.

```
"KafkaSettings": {
    "SecurityProtocol": "sasl-ssl",
    "SaslUsername":"Amazon MSK cluster secure user name",
    "SaslPassword":"Amazon MSK cluster secure password"
}
```

Note

- Currently, AWS DMS supports only public CA backed SASL-SSL. DMS does not support SASL-SSL for use with self-managed Kafka that is backed by private CA.
- For SASL-SSL authentication, AWS DMS supports the SCRAM-SHA-512 mechanism by default. AWS DMS versions 3.5.0 and higher also support the Plain mechanism. To support the Plain mechanism, set the `SaslMechanism` parameter of the `KafkaSettings` API data type to `PLAIN`. The datatype `PLAIN` is supported by Kafka, but not supported by Amazon Kafka (MSK).

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). If logical replication is in use and REPLICA IDENTITY FULL is set for the source table, you can get entire before and after information on the row written to the WALs and available here.
- 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 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](#).

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 Kafka target endpoints don't support IAM access control for Amazon Managed Streaming for Apache Kafka (Amazon MSK).
- Full LOB mode is not supported.
- 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 following example:

```
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 the `auto.create.topics.enable = true` setting to allow AWS DMS to automatically create new topics, 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 four 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.
  - `transaction-id`: The CDC transaction ID. All records within the same transaction go to the same partition.
  - `constant`: A fixed literal value for every record regardless of table or data. All records are sent to the same partition key value "constant", providing strict global ordering across all tables.

```
{
    "rule-type": "object-mapping",
    "rule-id": "2",
    "rule-name": "TransactionIdPartitionKey",
    "rule-action": "map-record-to-document",
    "object-locator": {
        "schema-name": "onprem",
        "table-name": "it_system"
    },
    "mapping-parameters": {
        "partition-key-type": "transaction-id | constant | attribute-name | schema-
table"
    }
}
```

- The `IncludeTransactionDetails` endpoint setting is only supported when the source endpoint is Oracle, SQL Server, PostgreSQL, or MySQL. For other source endpoint types, transaction details will not be included.
- `BatchApply` is not supported for a Kafka endpoint. Using Batch Apply (for example, the `BatchApplyEnabled` target metadata task setting) for a Kafka target might result in loss of data.
- AWS DMS does not support migrating values of `BigInt` data type with more than 16 digits. To work around this limitation, you can use the following transformation rule to convert the `BigInt` column to a string. For more information about transformation rules, see Transformation rules and actions.

```
{
    "rule-type": "transformation",
    "rule-id": "id",
    "rule-name": "name",
    "rule-target": "column",
    "object-locator": {
        "schema-name": "valid object-mapping rule action",
        "table-name": "",
        "column-name": ""
    },
    "rule-action": "change-data-type",
    "data-type": {
        "type": "string",
        "length": 20
    }
}
```

- AWS DMS Kafka target endpoints do not support Amazon MSK servless.
- When defining mapping rules, having both, object mapping rule and a transformation rule is not supported. You must set only one rule.
- AWS DMS supports SASL Authentication for Apache Kafka versions up to 3.8. If you are using Kafka 4.0 or higher, you can only connect without SASL authentication.
- AWS DMS does not support source data containing embedded `'\0'` characters when using Kafka as a target endpoint. Data containing embedded `'\0'` characters will be truncated at the first `'\0'` character.

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.

> **Note**
>
> You don't have to use object mapping. You can use regular table mapping for various transformations. However, the partition key type will follow these default behaviors:
>
> - Primary Key is used as a partition key for Full Load.
> - If no parallel-apply task settings are used, `schema.table` is used as a partition key for CDC.
> - If parallel-apply task settings are used, Primary key is used as a partition key for CDC.

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. These settings affect values that aren't excluded as part of the exclude-columns attribute list. The map-record-to-record and map-record-to-document values specify how AWS DMS handles these records by default. 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, note the following:

- This setting only affects columns excluded by the exclude-columns list.
- For every such column, AWS DMS creates a corresponding attribute in the target topic.
- AWS DMS creates this corresponding attribute regardless of whether the 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	1234567890	31 Spooner Street, Quahog	9876543210	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"
 }
```

Topics

- Restructuring data with attribute mapping
- Multitopic replication using object mapping
- Message format for Apache Kafka

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": "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 `"partition-key-type: "constant"`, this sets the partition value to `constant`. 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-type": "constant",
                "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.
> When `partition-key-type` is set to `attribute-name` in a table mapping rule, you must specify `partition-key-name`, which must reference either a column from the source table or a custom column defined in the mapping. Additionally, `attribute-mappings` must be provided to define how source columns map to the target Kafka topic.

Multitopic replication using object mapping

By default, AWS DMS tasks migrate all source data to one of the Kafka topics following:

- As specified in the **Topic** field of the AWS DMS target endpoint.
- As specified by `kafka-default-topic` if the **Topic** field of the target endpoint isn't populated and the Kafka `auto.create.topics.enable` setting is set to `true`.

With AWS DMS engine versions 3.4.6 and higher, you can use the `kafka-target-topic` attribute to map each migrated source table to a separate topic. For example, the object mapping rules following migrate the source tables `Customer` and `Address` to the Kafka topics `customer_topic` and `address_topic`, respectively. At the same time, AWS DMS migrates all other source tables, including the `Bills` table in the `Test` schema, to the topic specified in the target endpoint.

```
{
    "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": "MapToKafka1",
            "rule-action": "map-record-to-record",
            "kafka-target-topic": "customer_topic",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "Customer"
            },
            "partition-key-type": "constant"
        },
        {
            "rule-type": "object-mapping",
            "rule-id": "3",
            "rule-name": "MapToKafka2",
            "rule-action": "map-record-to-record",
            "kafka-target-topic": "address_topic",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "Address"
            },
            "partition-key-type": "constant"
        },
        {
            "rule-type": "object-mapping",
            "rule-id": "4",
            "rule-name": "DefaultMapToKafka",
            "rule-action": "map-record-to-record",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "Bills"
            }
        }
    ]
}
```

By using Kafka multitopic replication, you can group and migrate source tables to separate Kafka topics using a single replication task.

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 `load`, `insert`, `update`, or `delete`.

For control records, the operation can be `create-table`, `rename-table`, `drop-table`, `change-columns`, `add-column`, `drop-column`, `rename-column`, or `column-type-change`.

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.

The following JSON message example illustrates a data type message with all additional metadata.

```
{
   "data":{
      "id":100000161,
      "fname":"val61s",
      "lname":"val61s",
      "REGION":"val61s"
   },
   "metadata":{
      "timestamp":"2019-10-31T22:53:59.721201Z",
      "record-type":"data",
      "operation":"insert",
      "partition-key-type":"primary-key",
      "partition-key-value":"sbtest.sbtest_x.100000161",
      "schema-name":"sbtest",
      "table-name":"sbtest_x",
      "transaction-id":9324410911751,
      "transaction-record-id":1,
      "prev-transaction-id":9324410910341,
```

```
      "prev-transaction-record-id":10,
      "commit-timestamp":"2019-10-31T22:53:55.000000Z",
      "stream-position":"mysql-bin-
changelog.002171:36912271:0:36912333:9324410911751:mysql-bin-changelog.002171:36912209"
   }
}
```

The following JSON message example illustrates a control type message.

```
{
   "control":{
      "table-def":{
         "columns":{
            "id":{
               "type":"WSTRING",
               "length":512,
               "nullable":false
            },
            "fname":{
               "type":"WSTRING",
               "length":255,
               "nullable":true
            },
            "lname":{
               "type":"WSTRING",
               "length":255,
               "nullable":true
            },
            "REGION":{
               "type":"WSTRING",
               "length":1000,
               "nullable":true
            }
         },
         "primary-key":[
            "id"
         ],
         "collation-name":"latin1_swedish_ci"
      }
   },
   "metadata":{
      "timestamp":"2019-11-21T19:14:22.223792Z",
```

```
        "record-type":"control",
        "operation":"create-table",
        "partition-key-type":"task-id",
        "schema-name":"sbtest",
        "table-name":"sbtest_t1"
    }
}
```

Using an Amazon OpenSearch Service cluster as a target for AWS Database Migration Service

You can use AWS DMS to migrate data to Amazon OpenSearch Service (OpenSearch Service). OpenSearch Service is a managed service that makes it easy to deploy, operate, and scale an OpenSearch Service cluster.

In OpenSearch Service, 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. OpenSearch 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 OpenSearch Service, it creates one index for each table from the source endpoint. The cost for creating an OpenSearch Service 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 OpenSearch stores for each document.

Configure your OpenSearch Service 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 OpenSearch 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 OpenSearch Service 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 OpenSearch Service 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 OpenSearch Service target index. The maximum value for an OpenSearch Service 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 OpenSearch Service 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 OpenSearch Service 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.

For more information on how DMS loads an OpenSearch Service cluster using multithreading, see the AWS blog post [Scale Amazon OpenSearch Service for AWS Database Migration Service migrations].

Multithreaded CDC load task settings

You can improve the performance of change data capture (CDC) for an OpenSearch Service 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 32 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 OpenSearch Service target endpoints is available in AWS DMS versions 3.4.0 and higher.

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 OpenSearch Service 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 OpenSearch Service target endpoint during a CDC load. The default value is 100 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 OpenSearch Service 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 OpenSearch Service index

AWS DMS supports migrating data to OpenSearch Service's scalar data types. When migrating from a relational database like Oracle or MySQL to OpenSearch Service, you might want to restructure how you store this data.

AWS DMS supports the following OpenSearch Service 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 does not support migration of LOB data types.

Prerequisites for using Amazon OpenSearch Service as a target for AWS Database Migration Service

Before you begin work with an OpenSearch Service 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 OpenSearch Service indexes at the target endpoint. The minimum set of access permissions is shown in the following IAM policy.

JSON

```
{
    "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 OpenSearch Service must have the following permissions.

JSON

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

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "es:ESHttpDelete",
        "es:ESHttpGet",
        "es:ESHttpHead",
        "es:ESHttpPost",
        "es:ESHttpPut"
      ],
      "Resource": "*"
    }
  ]
}
```

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 OpenSearch Service domain. An example is `arn:aws:es:us-west-2:123456789012:domain/my-es-domain`

Endpoint settings when using OpenSearch Service as a target for AWS DMS

You can use endpoint settings to configure your OpenSearch Service target database similar to using extra connection attributes. You specify the settings when you create the target endpoint using the AWS DMS console, or by using the `create-endpoint` command in the AWS CLI, with the `--elasticsearch-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following table shows the endpoint settings that you can use with OpenSearch Service as a target.

Attribute name	Valid values	Default value and description
`FullLoadErrorPercentage`	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

Attribute name	Valid values	Default value and description
		errors (10 percent of the row count) when writing to the target endpoint.
`ErrorRetryDuration`	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.
`UseNewMappingType`	`true` or `false`	`false`, but to work using opensearch v2.x it should be set to `true`.

Limitations when using Amazon OpenSearch Service as a target for AWS Database Migration Service

The following limitations apply when using Amazon OpenSearch Service as a target:

- OpenSearch Service uses dynamic mapping (auto guess) to determine the data types to use for migrated data.
- OpenSearch Service 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 OpenSearch Service index requires storage for an additional 6,400,000 bytes.
- With OpenSearch Service, 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 OpenSearch Service 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.

- AWS DMS does not support connection to an Amazon ES cluster that has Fine-grained Access Control enabled with master user and password.
- AWS DMS does not support OpenSearch Service serverless.
- OpenSearch Service does not support writing data to pre-existing indexes.
- The replication task setting, `TargetTablePrepMode:TRUNCATE_BEFORE_LOAD` is not supported for use with a OpenSearch target endpoint.
- When migrating data to Amazon Elasticsearch using AWS DMS, the source data must have a primary key or a unique identifier column. If the source data does not have a primary key or unique identifier, you need to define one using the define-primary-key transformation rule.

Target data types for Amazon OpenSearch 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 OpenSearch Service.

AWS DMS data type	OpenSearch Service 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](#).

Using Amazon DocumentDB as a target for AWS Database Migration Service

For information about what versions of Amazon DocumentDB (with MongoDB compatibility) that AWS DMS supports, see Targets for AWS DMS. 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 AWS managed service such as Amazon RDS, Aurora, or Amazon S3. Or 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.

> **Note**
>
> If your source endpoint is MongoDB or Amazon DocumentDB, run the migration in **Document mode**.

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://amzn-s3-demo-bucket/hr/employee
```

In this case, AWS DMS maps the objects in `amzn-s3-demo-bucket` 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.

Amazon DocumentDB endpoint settings

In AWS DMS versions 3.5.0 and higher, you can improve the performance of change data capture (CDC) for Amazon DocumentDB endpoints by tuning task settings for parallel threads and bulk operations. 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 Amazon DocumentDB 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 DocumentDB target endpoint during a CDC load. The default value is 100 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 DocumentDB endpoint during CDC. The default is 1. The maximum is 512.

> **Note**
>
> For Amazon DocumentDB targets, parallel CDC apply can cause duplicate key errors or stalled CDC apply for workloads that use secondary unique indexes or require strict ordering of changes. Use the default single-threaded CDC apply configuration for these workloads.

For additional details on working with Amazon DocumentDB as a target for AWS DMS, see the following sections:

Topics

> **Note**
>
> For a step-by-step walkthrough of the migration process, see Migrating from MongoDB to Amazon DocumentDB in the AWS Database Migration Service Step-by-Step Migration Guide.

Mapping data from a source to an Amazon DocumentDB target

AWS DMS reads records from the source endpoint, and constructs JSON documents based on the data it reads. For each JSON document, AWS DMS must determine an `_id` field to act as a unique identifier. It then writes the JSON document to an Amazon DocumentDB collection, using the `_id` field as a primary key.

Source data that is a single column

If the source data consists of a single column, the data must be of a string type. (Depending on the source engine, the actual data type might be VARCHAR, NVARCHAR, TEXT, LOB, CLOB, or similar.) AWS DMS assumes that the data is a valid JSON document, and replicates the data to Amazon DocumentDB as is.

If the resulting JSON document contains a field named `_id`, then that field is used as the unique `_id` in Amazon DocumentDB.

If the JSON doesn't contain an `_id` field, then Amazon DocumentDB generates an `_id` value automatically.

Source data that is multiple columns

If the source data consists of multiple columns, then AWS DMS constructs a JSON document from all of these columns. To determine the `_id` field for the document, AWS DMS proceeds as follows:

- If one of the columns is named `_id`, then the data in that column is used as the target`_id`.
- If there is no `_id` column, but the source data has a primary key or a unique index, then AWS DMS uses that key or index value as the `_id` value. The data from the primary key or unique index also appears as explicit fields in the JSON document.
- If there is no `_id` column, and no primary key or a unique index, then Amazon DocumentDB generates an `_id` value automatically.

Coercing a data type at the target endpoint

AWS DMS can modify data structures when it writes to an Amazon DocumentDB target endpoint. You can request these changes by renaming columns and tables at the source endpoint, or by providing transformation rules that are applied when a task is running.

Using a nested JSON document (json_ prefix)

To coerce a data type, you can prefix the source column name with `json_` (that is, `json_`*`columnName`*) either manually or using a transformation. In this case, the column is created as a nested JSON document within the target document, rather than as a string field.

For example, suppose that you want to migrate the following document from a MongoDB source endpoint.

```
{
    "_id": "1",
    "FirstName": "John",
    "LastName": "Doe",
    "ContactDetails": "{"Home": {"Address": "Boston","Phone": "1111111"},"Work":
 { "Address": "Boston", "Phone": "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\": \"1111111\"},
\"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`. To coerce the data type as Nested JSON, the column at source endpoint needs to be renamed as json_ContactDetails" either by adding "*json_*" prefix on the source manually or through transformation rules. For example, you can use the below transformation rule:

```

{
    "rules": [
    {
    "rule-type": "transformation",
    "rule-id": "1",
    "rule-name": "1",
    "rule-target": "column",
    "object-locator": {
    "schema-name": "%",
    "table-name": "%",
    "column-name": "ContactDetails"
     },
    "rule-action": "rename",
    "value": "json_ContactDetails",
    "old-value": null
    }
    ]
}

```

AWS DMS replicates the ContactDetails field as nested JSON, as follows.

```
{
    "_id": "1",
    "FirstName": "John",
    "LastName": "Doe",
    "ContactDetails": {
        "Home": {
```

```
            "Address": "Boston",
            "Phone": "1111111111"
        },
        "Work": {
            "Address": "Boston",
            "Phone": "2222222222"
        }
    }
}
```

Using a JSON array (array_ prefix)

To coerce a data type, you can prefix a column name with `array_` (that is, `array_columnName`), either manually or using a transformation. In this case, AWS DMS considers the column as a JSON array, and creates it as such in the target document.

Suppose that you want to migrate the following document from a MongoDB source endpoint.

```
{
    "_id" : "1",
    "FirstName": "John",
    "LastName": "Doe",

    "ContactAddresses": ["Boston", "New York"],

    "ContactPhoneNumbers": ["1111111111", "2222222222"]
}
```

If you don't coerce any of the source data types, the embedded `ContactDetails` document is migrated as a string.

```
{
    "_id": "1",
    "FirstName": "John",
    "LastName": "Doe",

    "ContactAddresses": "[\"Boston\", \"New York\"]",

    "ContactPhoneNumbers": "[\"1111111111\", \"2222222222\"]"

}
```

However, you can add transformation rules to coerce `ContactAddress` and `ContactPhoneNumbers` to JSON arrays, as shown in the following table.

Original source column name	Renamed source column
`ContactAddress`	`array_ContactAddress`
`ContactPhoneNumbers`	`array_ContactPhoneNumbers`

AWS DMS replicates `ContactAddress` and `ContactPhoneNumbers` as follows.

```
{
    "_id": "1",
    "FirstName": "John",
    "LastName": "Doe",
    "ContactAddresses": [
        "Boston",
        "New York"
    ],
    "ContactPhoneNumbers": [
        "1111111111",
        "2222222222"
    ]
}
```

Connecting to Amazon DocumentDB using TLS

By default, a newly created Amazon DocumentDB cluster accepts secure connections only using Transport Layer Security (TLS). When TLS is enabled, every connection to Amazon DocumentDB requires a public key.

You can 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`.

Connecting to Amazon DocumentDB Elastic Clusters as a target

In AWS DMS versions 3.4.7 and higher, you can create a Amazon DocumentDB target endpoint as an Elastic Cluster. If you create your target endpoint as an Elastic Cluster, you need to attach a new SSL certificate to your Amazon DocumentDB Elastic Cluster endpoint because your existing SSL certificate won't work.

To attach a new SSL certificate to your Amazon DocumentDB Elastic Cluster endpoint

1. In a browser, open https://www.amazontrust.com/repository/SFSRootCAG2.pem and save the contents to a `.pem` file with a unique file name, for example `SFSRootCAG2.pem`. This is the certificate file that you need to import in subsequent steps.
2. Create the Elastic Cluster endpoint and set the following options:

   a. Under **Endpoint Configuration**, choose **Add new CA certificate**.

b. For **Certificate identifier**, enter **`SFSRootCAG2.pem`**.

c. For **Import certificate file**, choose **Choose file**, then navigate to the `SFSRootCAG2.pem` file that you previously downloaded.

d. Select and open the downloaded `SFSRootCAG2.pem` file.

e. Choose **Import certificate**.

f. From the **Choose a certificate** drop down, choose **SFSRootCAG2.pem**.

The new SSL certificate from the downloaded `SFSRootCAG2.pem` file is now attached to your Amazon DocumentDB Elastic Cluster endpoint.

Ongoing replication with Amazon DocumentDB as a target

If ongoing replication (change data capture, CDC) is enabled for Amazon DocumentDB as a target, AWS DMS versions 3.5.0 and higher provide a performance improvement that is twenty times greater than in prior releases. In prior releases where AWS DMS handles up to 250 records per second, AWS DMS now efficiently processes over 5000 records per second. AWS DMS also 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 `HandleSourceTableTruncated` is `true`. For more information, see Task settings for change processing DDL handling.
`DROP TABLE`	Deletes the collection, but only if `HandleSourceTableDropped` is `true`. For more information, see Task settings for change processing DDL handling.
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 `INSERT` 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 `INSERT` 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.

DDL at source	Effect at Amazon DocumentDB target
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 `INSERT` 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.
- The replication task setting, `TargetTablePrepMode:TRUNCATE_BEFORE_LOAD` isn't supported for use with a DocumentDB target endpoint.
- MongoDB capped collections are not supported in Amazon DocumentDB. However, AWS DMS automatically migrates such objects as uncapped collections on target DocumentDB.
- Parallel CDC apply to Amazon DocumentDB targets can cause duplicate key errors or stalled CDC apply for workloads that use secondary unique indexes or require strict ordering of changes. For such workloads, use the default single-threaded CDC apply configuration.

Using endpoint settings with Amazon DocumentDB as a target

You can use endpoint settings to configure your Amazon DocumentDB target database similar to using extra connection attributes. You specify the settings when you create the target endpoint using the AWS DMS console, or by using the `create-endpoint` command in the [AWS CLI](https://docs.aws.amazon.com/cli/latest/reference/dms/index.html), with the `--doc-db-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following table shows the endpoint settings that you can use with Amazon DocumentDB as a target.

Attribute name	Valid values	Default value and description
`replicateShardCollections`	boolean `true` `false`	When `true`, this endpoint setting has the following effects and imposes the following limitations: • AWS DMS is allowed to replicate data to target shard collections. This setting is only applicable if the target DocumentDB endpoint is an Elastic Cluster. • You must set `TargetTablePrepMode` to `DO_NOTHING`. • AWS DMS automatically sets `useUpdateLookup` to `false` during migration.

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.

AWS DMS data type	Amazon DocumentDB data type
BOOLEAN	Boolean
BYTES	Binary data
DATE	Date
TIME	String (UTF8)
DATETIME	Date

AWS DMS data type	Amazon DocumentDB data type
INT1	32-bit integer
INT2	32-bit integer
INT4	32-bit integer
INT8	64-bit integer
NUMERIC	String (UTF8)
REAL4	Double
REAL8	Double
STRING	If the data is recognized as JSON, then AWS DMS migrates it to Amazon DocumentDB as a document. Otherwise, the data is mapped to String (UTF8).
UINT1	32-bit integer
UINT2	32-bit integer
UINT4	64-bit integer
UINT8	String (UTF8)
WSTRING	If the data is recognized as JSON, then AWS DMS migrates it to Amazon DocumentDB as a document. Otherwise, the data is mapped to String (UTF8).
BLOB	Binary
CLOB	If the data is recognized as JSON, then AWS DMS migrates it to Amazon DocumentDB as a document. Otherwise, the data is mapped to String (UTF8).
NCLOB	If the data is recognized as JSON, then AWS DMS migrates it to Amazon DocumentDB as a document. Otherwise, the data is mapped to String (UTF8).

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

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 User 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.

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.
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.

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.

   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.

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 following example 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":"amzn-s3-demo-bucket",
       "S3BucketFolder":"amzn-s3-demo-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.
- `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 following IAM policy and trust documents. 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:::amzn-s3-demo-bucket"
     ]
    },
    {
     "Sid": "AllObjectActions",
     "Effect": "Allow",
     "Action": ["s3:GetObject",
      "s3:PutObject",
      "s3:DeleteObject"
     ],

     "Resource": [
      "arn:aws:s3:::amzn-s3-demo-bucket/"
     ]
    },
    {
     "Sid": "ListObjectsInBucket1",
     "Effect": "Allow",
     "Action": "s3:ListBucket",
     "Resource": [
      "arn:aws:s3:::amzn-s3-demo-bucket",
      "arn:aws:s3:::amzn-s3-demo-bucket/"
     ]
    }
  ]
}
```

The preceding IAM policy allows your account to query the contents of the S3 bucket (`arn:aws:s3:::amzn-s3-demo-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:::amzn-s3-demo-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.

JSON

```
{
  "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.

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 following template 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.

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`

SQL source data types	Gremlin target data types
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	
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 following examples.

  To drop all data (vertices and edges) within the graph, run the following Gremlin command.

  ```
  gremlin> g.V().drop().iterate()
  ```

  To drop vertices that have the label `'customer'`, run the following Gremlin command.

  ```
  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 following Gremlin command.

  ```
  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 following example, 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.
- AWS DMS doesn't support migrating LOB data into a Neptune target.

Using Redis OSS as a target for AWS Database Migration Service

Redis OSS is an open-source in-memory data structure store used as a database, cache, and message broker. Managing data in-memory can result in read or write operations taking less than a millisecond, and hundreds of millions of operations performed each second. As an in-memory data store, Redis OSS powers the most demanding applications requiring sub-millisecond response times.

Using AWS DMS, you can migrate data from any supported source database to a target Redis OSS data store with minimal downtime. For additional information about Redis OSS see, Redis OSS Documentation.

In addition to on-premises Redis OSS, AWS Database Migration Service supports the following:

- Amazon ElastiCache (Redis OSS) as a target data store. ElastiCache (Redis OSS) works with your Redis OSS clients and uses the open Redis OSS data format to store your data.
- Amazon MemoryDB as a target data store. MemoryDB is compatible with Redis OSS and enables you to build applications using all the Redis OSS data structures, APIs, and commands in use today.

For additional information about working with Redis OSS as a target for AWS DMS, see the following sections:

Topics

- Prerequisites for using a Redis OSS cluster as a target for AWS DMS
- Limitations when using Redis as a target for AWS Database Migration Service
- Migrating data from a relational or non-relational database to a Redis OSS target
- Specifying endpoint settings for Redis OSS as a target

Prerequisites for using a Redis OSS cluster as a target for AWS DMS

DMS supports an on-premises Redis OSS target in a standalone configuration, or as a Redis OSS cluster where data is automatically *sharded* across multiple nodes. Sharding is the process of separating data into smaller chunks called shards that are spread across multiple servers or nodes. In effect, a shard is a data partition that contains a subset of the total data set, and serves a slice of the overall workload.

Since Redis OSS is a key-value NoSQL data store, the Redis OSS key naming convention to use when your source is a relational database, is **schema-name.table-name.primary-key**. In Redis OSS, the key and value must not contain the special character %. Otherwise, DMS skips the record.

> **Note**
>
> If you are using ElastiCache (Redis OSS) as a target, DMS supports *cluster mode enabled* configurations only. For more information about using ElastiCache (Redis OSS) version 6.x or higher to create a cluster mode enabled target data store, see Getting started in the *Amazon ElastiCache (Redis OSS) User Guide*.

Before you begin a database migration, launch your Redis OSS cluster with the following criteria.

- Your cluster has one or more shards.
- If you're using an ElastiCache (Redis OSS) target, ensure that your cluster doesn't use IAM role-based access control. Instead, use Redis OSS Auth to authenticate users.
- Enable Multi-AZ (Availability Zones).
- Ensure the cluster has sufficient memory available to fit the data to be migrated from your database.
- Make sure that your target Redis OSS cluster is clear of all data before starting the initial migration task.

You should determine your security requirements for the data migration prior to creating your cluster configuration. DMS supports migration to target replication groups regardless of their encryption configuration. But you can enable or disable encryption only when you create your cluster configuration.

Limitations when using Redis as a target for AWS Database Migration Service

The following limitations apply when using Redis OSS as a target:

- Since Redis OSS is a key-value no-sql data store, the Redis OSS key naming convention to use when your source is a relational database, is `schema-name.table-name.primary-key`.
- In Redis OSS, the key-value can't contain the special character `%`. Otherwise, DMS skips the record.
- DMS won't migrate rows that contain the `%` character.
- DMS won't migrate fields that contain the `%` character in the field name.
- Full LOB mode is not supported.
- A private Certificate Authority (CA) isn’t supported when using ElastiCache (Redis OSS) as a target.
- AWS DMS does not support source data containing embedded `'\0'` characters when using Redis as a target endpoint. Data containing embedded `'\0'` characters will be truncated at the first `'\0'` character.

Migrating data from a relational or non-relational database to a Redis OSS target

You can migrate data from any source SQL or NoSQL data store directly to a Redis OSS target. Setting up and starting a migration to a Redis OSS target is similar to any full load and change data capture migration using the DMS console or API. To perform a database migration to a Redis OSS target, you do the following.

- Create a replication instance to perform all the processes for the migration. For more information, see Creating a replication instance.
- Specify a source endpoint. For more information, see Creating source and target endpoints.
- Locate the DNS name and port number of your cluster.
- Download a certificate bundle that you can use to verify SSL connections.
- Specify a target endpoint, as described below.
- Create a task or set of tasks to define what tables and replication processes you want to use. For more information, see Creating a task.
- Migrate data from your source database to your target cluster.

You begin a database migration in one of two ways:

1. You can choose the AWS DMS console and perform each step there.
2. You can use the AWS Command Line Interface (AWS CLI). For more information about using the CLI with AWS DMS, see AWS CLI for AWS DMS.

To locate the DNS name and port number of your cluster

- Use the following AWS CLI command to provide the `replication-group-id` with the name of your replication group.

  ```
  aws elasticache describe-replication-groups --replication-group-id myreplgroup
  ```

  Here, the output shows the DNS name in the `Address` attribute and the port number in the `Port` attribute of the primary node in the cluster.

  ```
   ...
  "ReadEndpoint": {
  "Port": 6379,
  "Address": "myreplgroup-
  111.1abc1d.1111.uuu1.cache.example.com"
  }
  ...
  ```

  If you are using MemoryDB as your target, use the following AWS CLI command to provide an endpoint address to your Redis OSS cluster.

  ```
  aws memorydb describe-clusters --clusterid clusterid
  ```

Download a certificate bundle for use to verify SSL connections

- Enter the following `wget` command at the command line. Wget is a free GNU command-line utility tool used to download files from the internet.

```
wget https://s3.aws-api-domain/rds-downloads/rds-combined-ca-bundle.pem
```

Here, *aws-api-domain* completes the Amazon S3 domain in your AWS Region required to access the specified S3 bucket and the rds-combined-ca-bundle.pem file that it provides.

To create a target endpoint using the AWS DMS console

This endpoint is for your Redis OSS target that is already running.

- On the console, choose **Endpoints** from the navigation pane and then choose **Create Endpoint**. The following table describes the settings.

For this option	Do this
Endpoint type	Choose the **Target** endpoint type.
Endpoint identifier	Enter the name of your endpoint. For example, include the type of endpoint in the name, such as **`my-redis-target`** .
Target engine	Choose **Redis OSS** as the type of database engine that you want this endpoint to connect.
Cluster name	Enter the DNS name of your Redis OSS cluster.
Port	Enter the port number of your Redis OSS cluster.
SSL security protocol	Choose either **Plain text** or **SSL encryption**. **Plain text**—This option doesn't provide Transport Layer Security (TLS) encryption for traffic between endpoint and database. **SSL encryption**—If you choose this option, enter an SSL Certificate Authority (CA) certificate ARN to verify the server’s certificate and make an encrypted connection.

For this option	Do this
	For on-premises Redis OSS, DMS supports both public and private Certificate Authority (CA). For ElastiCache (Redis OSS), DMS supports only a public CA.
Authentication type	Choose the type of authentication to perform while connecting to Redis OSS. Options include, **None**, **Authentication role**, and **Authentication token**. If you choose Authentication role, provide an **Authentication username** and an **Authentication password**. If you choose Authentication token, provide an **Authentication password** only.
Replication instance	**[Optional]** Only if you intend to test your connectio n, choose the name of the replication instance you previously entered on the **Create replication instance** page.

When you're finished providing all information for your endpoint, AWS DMS creates your Redis OSS target endpoint for use during database migration.

For information about creating a migration task and starting your database migration, see Creating a task.

Specifying endpoint settings for Redis OSS as a target

To create or modify a target endpoint, you can use the console or the `CreateEndpoint` or `ModifyEndpoint` API operations.

For a Redis OSS target in the AWS DMS console, specify **Endpoint-specific settings** on the **Create endpoint** or **Modify endpoint** console page.

When using `CreateEndpoint` and `ModifyEndpoint` API operations, specify request parameters for the `RedisSettings` option. The example following shows how to do this using the AWS CLI.

```
aws dms create-endpoint --endpoint-identifier my-redis-target
--endpoint-type target --engine-name redis --redis-settings
'{"ServerName":"sample-test-
sample.zz012zz.cluster.eee1.cache.bbbxxx.com","Port":6379,"AuthType":"auth-token",
 "SslSecurityProtocol":"ssl-encryption", "AuthPassword":"notanactualpassword"}'

{
    "Endpoint": {
        "EndpointIdentifier": "my-redis-target",
        "EndpointType": "TARGET",
        "EngineName": "redis",
        "EngineDisplayName": "Redis",
        "TransferFiles": false,
        "ReceiveTransferredFiles": false,
        "Status": "active",
        "KmsKeyId": "arn:aws:kms:us-east-1:999999999999:key/x-b188188x",
        "EndpointArn": "arn:aws:dms:us-
east-1:555555555555:endpoint:ABCDEFGHIJKLMONOPQRSTUVWXYZ",
        "SslMode": "none",
        "RedisSettings": {
            "ServerName": "sample-test-sample.zz012zz.cluster.eee1.cache.bbbxxx.com",
            "Port": 6379,
            "SslSecurityProtocol": "ssl-encryption",
            "AuthType": "auth-token"
        }
    }
}
```

The `--redis-settings` parameters follow:

- `ServerName`–(Required) Of type `string`, specifies the Redis OSS cluster that data will be migrated to, and is in your same VPC.
- `Port`–(Required) Of type `number`, the port value used to access the endpoint.
- `SslSecurityProtocol`–(Optional) Valid values include `plaintext` and `ssl-encryption`. The default is `ssl-encryption`.

  The `plaintext` option doesn't provide Transport Layer Security (TLS) encryption for traffic between endpoint and database.

Use `ssl-encryption` to make an encrypted connection. `ssl-encryption` doesn't require an SSL Certificate Authority (CA) ARN to verify a server's certificate, but one can be identified optionally using the `SslCaCertificateArn` setting. If a certificate authority ARN isn't given, DMS uses the Amazon root CA.

When using an on-premises Redis OSS target, you can use `SslCaCertificateArn` to import public or private Certificate Authority (CA) into DMS, and provide that ARN for server authentication. A private CA isn't supported when using ElastiCache (Redis OSS) as a target.

- `AuthType`–(Required) Indicates the type of authentication to perform when connecting to Redis OSS. Valid values include `none`, `auth-token`, and `auth-role`.

  The `auth-token` option requires an "*`AuthPassword`*" be provided, while the `auth-role` option requires "*`AuthUserName`*" and "*`AuthPassword`*" be provided.

Using Babelfish as a target for AWS Database Migration Service

You can migrate data from a Microsoft SQL Server source database to a Babelfish target using AWS Database Migration Service.

Babelfish for Aurora PostgreSQL extends your Amazon Aurora PostgreSQL-Compatible Edition database with the ability to accept database connections from Microsoft SQL Server clients. Doing this allows applications originally built for SQL Server to work directly with Aurora PostgreSQL with few code changes compared to a traditional migration, and without changing database drivers.

For information about versions of Babelfish that AWS DMS supports as a target, see Targets for AWS DMS. Earlier versions of Babelfish on Aurora PostgreSQL require an upgrade before using the Babelfish endpoint.

> **Note**
>
> The Aurora PostgreSQL target endpoint is the preferred way to migrate data to Babelfish. For more information, see Using Babelfish for Aurora PostgreSQL as a target.

For information about using Babelfish as a database endpoint, see Babelfish for Aurora PostgreSQL in the *Amazon Aurora User Guide for Aurora*

Prerequisites to using Babelfish as a target for AWS DMS

You must create your tables before migrating data to make sure that AWS DMS uses the correct data types and table metadata. If you don't create your tables on the target before running migration, AWS DMS may create the tables with incorrect data types and permissions. For example, AWS DMS creates a timestamp column as binary(8) instead, and doesn't provide the expected timestamp/rowversion functionality.

To prepare and create your tables prior to migration

1. Run your create table DDL statements that include any unique constraints, primary keys, or default constraints.

   Do not include foreign key constraints, or any DDL statements for objects like views, stored procedures, functions, or triggers. You can apply them after migrating your source database.
2. Identify any identity columns, computed columns, or columns containing rowversion or timestamp data types for your tables. Then, create the necessary transformation rules to handle known issues when running the migration task. For more information see, Transformation rules and actions.
3. Identify columns with data types that Babelfish doesn't support. Then, change the affected columns in the target table to use supported data types, or create a transformation rule that removes them during the migration task. For more information see, Transformation rules and actions.

   The following table lists source data types not supported by Babelfish, and the corresponding recommended target data type to use.

Source data type	Recommended Babelfish data type
HEIRARCHYID	NVARCHAR(250)
GEOMETRY	VARCHAR(MAX)
GEOGRAPHY	VARCHAR(MAX)

To set Aurora capacity units (ACUs) level for your Aurora PostgreSQL Serverless V2 source database

You can improve performance of your AWS DMS migration task prior to running it by setting the minimum ACU value.

- From the **Severless v2 capacity settings** window, set **Minimum ACUs** to **2**, or a reasonable level for your Aurora DB cluster.

  For additional information about setting Aurora capacity units, see [Choosing the Aurora Serverless v2 capacity range for an Aurora cluster](#) in the *Amazon Aurora User Guide*

After running your AWS DMS migration task, you can reset the minimum value of your ACUs to a reasonable level for your Aurora PostgreSQL Serverless V2 source database.

Security requirements when using Babelfish as a target for AWS Database Migration Service

The following describes the security requirements for using AWS DMS with a Babelfish target:

- The administrator user name (the Admin user) used to create the database.
- PSQL login and user with the sufficient SELECT, INSERT, UPDATE, DELETE, and REFERENCES permissions.

User permissions for using Babelfish as a target for AWS DMS

> **Important**
>
> For security purposes, the user account used for the data migration must be a registered user in any Babelfish database that you use as a target.

Your Babelfish target endpoint requires minimum user permissions to run an AWS DMS migration.

To create a login and a low-privileged Transact-SQL (T-SQL) user

1. Create a login and password to use when connecting to the server.

```
CREATE LOGIN dms_user WITH PASSWORD = 'password';
```

```
GO
```

2. Create the virtual database for your Babelfish cluster.

```
CREATE DATABASE my_database;
GO
```

3. Create the T-SQL user for your target database.

```
USE my_database
GO
CREATE USER dms_user FOR LOGIN dms_user;
GO
```

4. For each table in your Babelfish database, GRANT permissions to the tables.

```
GRANT SELECT, DELETE, INSERT, REFERENCES, UPDATE ON [dbo].[Categories] TO dms_user;
```

Limitations on using Babelfish as a target for AWS Database Migration Service

The following limitations apply when using a Babelfish database as a target for AWS DMS:

- Only table preparation mode "**Do Nothing**" is supported.
- The ROWVERSION data type requires a table mapping rule that removes the column name from the table during the migration task.
- The sql_variant data type isn't supported.
- Full LOB mode is supported. Using SQL Server as a source endpoint requires the SQL Server Endpoint Connection Attribute setting `ForceFullLob=True` to be set in order for LOBs to be migrated to the target endpoint.
- Replication task settings have the following limitations:

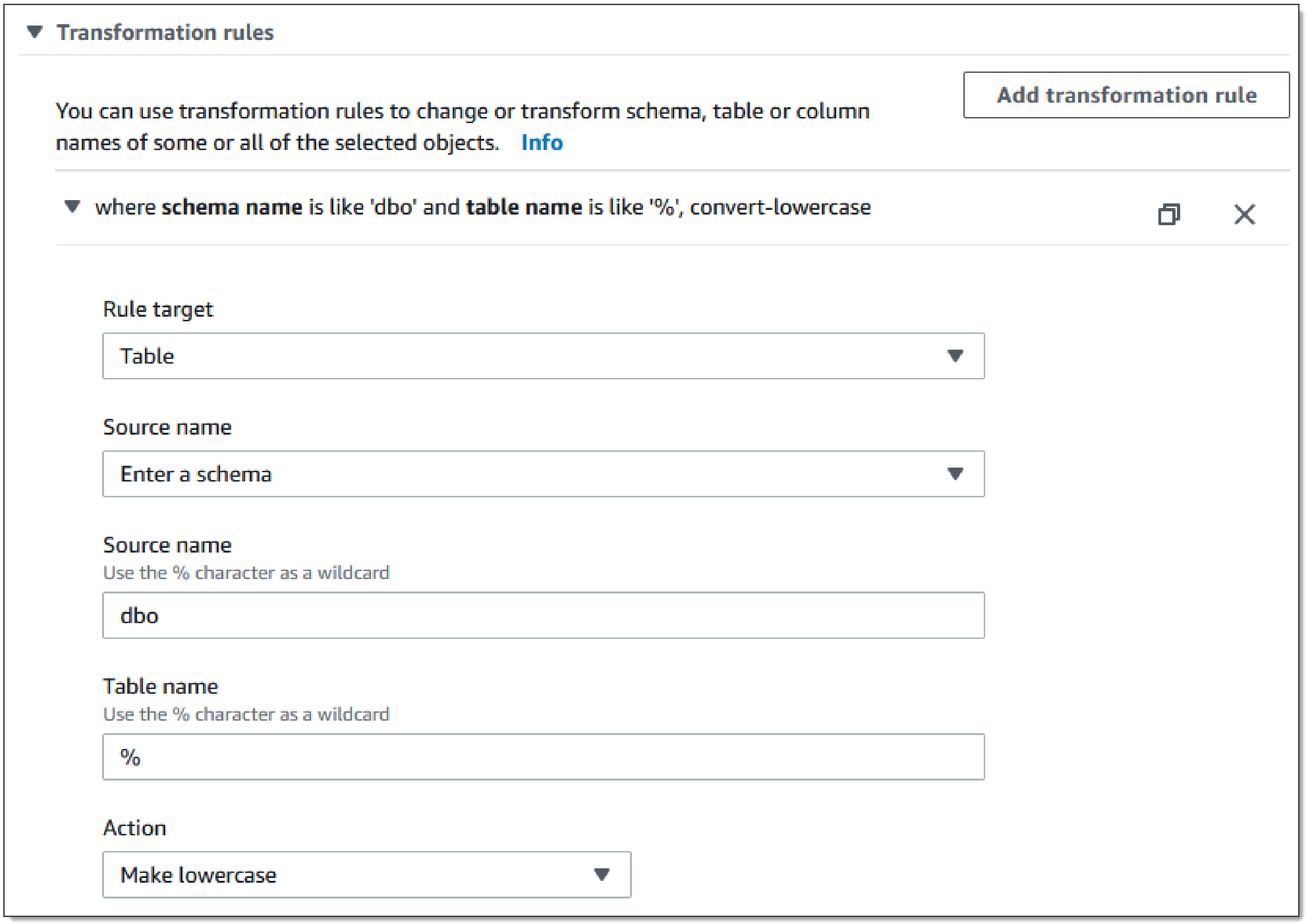
```
{
   "FullLoadSettings": {
      "TargetTablePrepMode": "DO_NOTHING",
      "CreatePkAfterFullLoad": false,
      }.

}
```

- TIME(7), DATETIME2(7), and DATETIMEOFFSET(7) data types in Babelfish limit the precision value for the seconds portion of the time to 6 digits. Consider using a precision value of 6 for your target table when using these data types. For Babelfish versions 2.2.0 and higher, when using TIME(7) and DATETIME2(7), the seventh digit of precision is always zero.
- In DO_NOTHING mode, DMS checks to see if the table already exists. If the table doesn't exist in the target schema, DMS creates the table based on the source table definition, and maps any user defined data types to their base data type.
- An AWS DMS migration task to a Babelfish target doesn't support tables that have columns using ROWVERSION or TIMESTAMP data types. You can use a table mapping rule that removes the column name from the table during the transfer process. In the following transformation rule example, a table named `Actor` in your source is transformed to remove all columns starting with the characters `col` from the `Actor` table in your target.

```
{
  "rules": [{
  "rule-type": "selection",is
  "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%"
  }
 }]
 }
```

- For tables with identity or computed columns, where the target tables use mixed case names like Categories, you must create a transformation rule action that converts the table names to lowercase for your DMS task. The following example shows how to create the transformation rule action, **Make lowercase** using the AWS DMS console. For more information, see Transformation rules and actions.



- Prior to Babelfish version 2.2.0, DMS limited the number of columns that you could replicate to a Babelfish target endpoint to twenty (20) columns. With Babelfish 2.2.0 the limit increased to 100 columns. But with Babelfish versions 2.4.0 and higher, the number of columns that you can replicate increases again. You can run the following code sample against your SQL Server database to determine which tables are too long.

```
USE myDB;
GO
DECLARE @Babelfish_version_string_limit INT = 8000; -- Use 380 for Babelfish versions
 before 2.2.0
WITH bfendpoint
AS (
SELECT
```

```
[TABLE_SCHEMA]
     ,[TABLE_NAME]
   , COUNT( [COLUMN_NAME] ) AS NumberColumns
   , ( SUM( LEN( [COLUMN_NAME] ) + 3)
  + SUM( LEN( FORMAT(ORDINAL_POSITION, 'N0') ) + 3 )
     + LEN( TABLE_SCHEMA ) + 3
  + 12 -- INSERT INTO string
  + 12)  AS InsertIntoCommandLength -- values string
      , CASE WHEN ( SUM( LEN( [COLUMN_NAME] ) + 3)
  + SUM( LEN( FORMAT(ORDINAL_POSITION, 'N0') ) + 3 )
     + LEN( TABLE_SCHEMA ) + 3
  + 12 -- INSERT INTO string
  + 12)  -- values string
   >= @Babelfish_version_string_limit
   THEN 1
   ELSE 0
  END AS IsTooLong
FROM [INFORMATION_SCHEMA].[COLUMNS]
GROUP BY [TABLE_SCHEMA], [TABLE_NAME]
)
SELECT *
FROM bfendpoint
WHERE IsTooLong = 1
ORDER BY TABLE_SCHEMA, InsertIntoCommandLength DESC, TABLE_NAME
;
```

Target data types for Babelfish

The following table shows the Babelfish 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.

AWS DMS data type	Babelfish data type
BOOLEAN	TINYINT
BYTES	VARBINARY(length)
DATE	DATE

AWS DMS data type	Babelfish data type
TIME	TIME
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: • For SQL Server 2008 and higher, use DATETIME2. • For earlier versions, if the scale is 3 or less use DATETIME. In all other cases, use VARCHAR (37). If the column is not a date or time column, use VARCHAR (length).
UINT1	TINYINT
UINT2	SMALLINT
UINT4	INT
UINT8	BIGINT
WSTRING	NVARCHAR(length)

AWS DMS data type	Babelfish data type
BLOB	VARBINARY(max) To use this data type with DMS, you must enable the use of BLOBs for a specific task. DMS supports BLOB data types only in tables that include a primary key.
CLOB	VARCHAR(max) To use this data type with DMS, you must enable the use of CLOBs for a specific task.
NCLOB	NVARCHAR(max) To use this data type with DMS, you must enable the use of NCLOBs for a specific task. During CDC, DMS supports NCLOB data types only in tables that include a primary key.

Using Amazon Timestream as a target for AWS Database Migration Service

You can use AWS Database Migration Service to migrate data from your source database to a Amazon Timestream target endpoint, with support for Full Load and CDC data migrations.

Amazon Timestream is a fast, scalable, and serverless time series database service built for high-volume data ingestion. Time series data is a sequence of data points collected over a time interval, and is used for measuring events that change over time. It is used to collect, store, and analyze metrics from IoT applications, DevOps applications, and analytics applications. Once you have your data in Timestream, you can visualize and identify trends and patterns in your data in near real-time. For information about Amazon Timestream, see What is Amazon Timestream? in the *Amazon Timestream Developer Guide*.

Topics

- Multithreaded CDC load task settings
- Endpoint settings when using Timestream as a target for AWS DMS
- Creating and modifying an Amazon Timestream target endpoint
- Using object mapping to migrate data to a Timestream topic
- Limitations when using Amazon Timestream as a target for AWS Database Migration Service

Prerequisites for using Amazon Timestream as a target for AWS Database Migration Service

Before you set up Amazon Timestream as a target for AWS DMS, make sure that you create an IAM role. This role must allow AWS DMS to gain access to the data being migrated into Amazon Timestream. The minimum set of access permissions for the role that you use to migrate to Timestream is shown in the following IAM policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDescribeEndpoints",
      "Effect": "Allow",
      "Action": [
        "timestream:DescribeEndpoints"
      ],
      "Resource": "*"
    },
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "timestream:ListTables",
        "timestream:DescribeDatabase"
      ],
      "Resource": "arn:aws:timestream:us-east-1:123456789012:database/
DATABASE_NAME"
    },
    {
```

```
        "Sid": "VisualEditor1",
        "Effect": "Allow",
        "Action": [
          "timestream:DeleteTable",
          "timestream:WriteRecords",
          "timestream:UpdateTable",
          "timestream:CreateTable"
        ],
        "Resource": "arn:aws:timestream:us-east-1:123456789012:database/
DATABASE_NAME/table/TABLE_NAME"
    }
  ]
}
```

If you intend to migrate all tables, use * for *TABLE_NAME* in the example above.

Note the following about using Timestream as a target:

- If you intend to ingest historical data with timestamps exceeding 1 year old, we recommend to use AWS DMS to write the data to Amazon S3 in a comma separated value (csv) format. Then, use Timestream's batch load to ingest the data into Timestream. For more information, see Using batch load in Timestream in the Amazon Timestream developer guide.
- For full-load data migrations of data less than 1 year old, we recommend setting the memory store retention period of the Timestream table greater than or equal to the oldest timestamp. Then, once migration completes, edit the table's memory store retention to the desired value. For example, to migrate data with the oldest timestamp being 2 months old, do the following:
  - Set the Timestream target table's memory store retention to 2 months.
  - Start the data migration using AWS DMS.
  - Once the data migration completes, change the retention period of the target Timestream table to your desired value.

  We recommend estimating the memory store cost prior to the migration, using the information on the following pages:
  - Amazon Timestream pricing
  - AWS pricing calculator

- For CDC data migrations, we recommend setting the memory store retention period of the target table such that ingested data falls within the memory store retention bounds. For more information, see Writes Best Practices in the Amazon Timestream developer guide.

Multithreaded full load task settings

To help increase the speed of data transfer, AWS DMS supports a multithreaded full load migration task to a Timestream target endpoint with these task settings:

- `MaxFullLoadSubTasks` – Use this option to indicate the maximum number of source tables to load in parallel. DMS loads each table into its corresponding Amazon Timestream 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 Amazon Timestream target table. The maximum value for a Timestream 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 Amazon Timestream 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 Amazon Timestream targets of various payload sizes, the valid range is 5–512 queues per thread.

Multithreaded CDC load task settings

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 Timestream target endpoint. The default value is 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 Timestream target endpoint during a CDC load. The default value is 100 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 Timestream endpoint during CDC. The default value is 1 and the maximum value is 512.

Endpoint settings when using Timestream as a target for AWS DMS

You can use endpoint settings to configure your Timestream target database similar to using extra connection attributes. You specify the settings when you create the target endpoint using the AWS DMS console, or by using the `create-endpoint` command in the AWS CLI, with the `--timestream-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following table shows the endpoint settings that you can use with Timestream as a target.

Name	Description
`MemoryDuration`	Set this attribute to specify the retention bound to store the data migrated in Timestream's memory store. Time is measured in units of hours. Timestream's memory store is optimized for high ingestion throughput and fast access. Default value: 24 (hours) Valid values: 1 to 8,736 (1 hour to 12 months measured in hours) Example: `--timestream-settings '{"MemoryDuration": 20}'`
`DatabaseName`	Set this attribute to specify the target Timestream database name. Type: string Example: `--timestream-settings '{"DatabaseName": " db_name"}'`
`TableName`	Set this attribute to specify the target Timestream table name.

Name	Description
	Type: string Example: `--timestream-settings '{"TableName": " table_name "}'`
`MagneticDuration`	Set this attribute to specify the magnetic duration applied to the Timestream tables in days. This is the retention bound for the ingested data. Timestream deletes any timestamp exceeding the retention bound. For more information, see Storage in the Amazon Timestream Developer Guide. Example: `--timestream-settings '{"MagneticDuration": "3"}'`
`CdcInsertsAndUpdates`	Set this attribute to `true` to specify that AWS DMS only applies inserts and updates, and not deletes. Timestream does not allow deleting records, so if this value is `false`, AWS DMS nulls out the corresponding record in the Timestream database rather than deleting it. For more information, see Limitations following. Default value: `false` Example: `--timestream-settings '{"CdcInsertsAndUpdates": "true"}'`

Name	Description
`EnableMagneticStoreWrites`	Set this attribute to `true` to enable magnetic store writes. When this value is `false`, AWS DMS does not write records records with a timestamp older than the memory store retention period of the target table, because Timestream does not allow magnetic store writes by default. For more information, see Writes Best Practices in the Amazon Timestream Developer Guide. Default value: `false` Example: `--timestream-settings '{"EnableMagneticStoreWrites": "true"}'`

Creating and modifying an Amazon Timestream target endpoint

Once you have created an IAM role and established the minimum set of access permissions, you can create a Amazon Timestream target endpoint using the AWS DMS console, or by using the `create-endpoint` command in the AWS CLI, with the `--timestream-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following examples show how to create and modify a Timestream target endpoint using the AWS CLI.

Create Timestream target endpoint command

```
aws dms create-endpoint —endpoint-identifier timestream-target-demo
--endpoint-type target —engine-name timestream
--service-access-role-arn arn:aws:iam::123456789012:role/my-role
--timestream-settings
{
    "MemoryDuration": 20,
    "DatabaseName":"db_name",
    "MagneticDuration": 3,
    "CdcInsertsAndUpdates": true,
    "EnableMagneticStoreWrites": true,
}
```

Modify Timestream target endpoint command

```
aws dms modify-endpoint —endpoint-identifier timestream-target-demo
--endpoint-type target —engine-name timestream
--service-access-role-arn arn:aws:iam::123456789012:role/my-role
--timestream-settings
{
    "MemoryDuration": 20,
    "MagneticDuration": 3,
}
```

Using object mapping to migrate data to a Timestream topic

AWS DMS uses table-mapping rules to map data from the source to the target Timestream 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 Timestream topic.

Timestream topics don't have a preset structure other than having a partition key.

Note

You don't have to use object mapping. You can use regular table mapping for various transformations. However, the partition key type will follow these default behaviors:

- Primary Key is used as a partition key for Full Load.
- If no parallel-apply task settings are used, `schema.table` is used as a partition key for CDC.
- If parallel-apply task settings are used, Primary key is used as a partition key for CDC.

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": ""
            }
        }
    ]
}
```

```
{
    "rules": [
        {
            "rule-type": "object-mapping",
            "rule-id": "1",
            "rule-name": "timestream-map",
            "rule-action": "map-record-to-record",
            "target-table-name": "tablename",
            "object-locator": {
                "schema-name": "",
                "table-name": ""
            },
            "mapping-parameters": {
                "timestream-dimensions": [
                    "column_name1",
                     "column_name2"
                ],
                "timestream-timestamp-name": "time_column_name",
                "timestream-multi-measure-name": "column_name1or2",
                "timestream-hash-measure-name":  true or false,
                "timestream-memory-duration": x,
                "timestream-magnetic-duration": y
            }
        }
    ]
}
```

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 Timestream topic. This rule type uses the `taskResourceId.schemaName.tableName` value from the relational database as the partition key in the Timestream 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	1234567890	31 Spooner Street, Quahog	9876543210	02/29/1988

To migrate this information from a schema named `Test` to a Timestream 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": "DefaultMapToTimestream",
            "rule-action": "map-record-to-record",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "Customers"
            }
        }
    ]
}
```

Given a Timestream topic and a partition key (in this case, `taskResourceId.schemaName.tableName`), the following illustrates the resulting record format using our sample data in the Timestream 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"
}
```

Limitations when using Amazon Timestream as a target for AWS Database Migration Service

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

- **Dimensions and Timestamps:** Timestream uses the dimensions and timestamps in the source data like a composite primary key, and also does not allow you to upsert these values. This means that if you change the timestamp or the dimensions for a record in the source database, the Timestream database will try to create a new record. It is thus possible that if you change the dimension or timestamp of a record such that they match those of another existing record, then AWS DMS updates the values of the other record instead of creating a new record or updating the previous corresponding record.

- **DDL Commands:** The current release of AWS DMS only supports CREATE TABLE and DROP TABLE DDL commands.
- **Record Limitations:** Timestream has limitations for records such as record size and measure size. For more information, see Quotas in the Amazon Timestream Developer Guide.
- **Deleting Records and Null Values:** Timestream doesn't support deleting records. To support migrating records deleted from the source, AWS DMS clears the corresponding fields in the records in the Timestream target database. AWS DMS changes the values in the fields of the corresponding target record with **0** for numeric fields, **null** for text fields, and **false** for boolean fields.
- Timestream as a target doesn't support sources that aren't relational databases (RDBMS).
- AWS DMS only supports Timestream as a target in the following regions:
  - US East (N. Virginia)
  - US East (Ohio)
  - US West (Oregon)
  - Europe (Ireland)
  - Europe (Frankfurt)
  - Asia Pacific (Sydney)
  - Asia Pacific (Tokyo)
- Timestream as a target doesn't support setting `TargetTablePrepMode` to `TRUNCATE_BEFORE_LOAD`. We recommend using `DROP_AND_CREATE` for this setting.

Using Amazon RDS for Db2 and IBM Db2 LUW as a target for AWS DMS

You can migrate data to an Amazon RDS for Db2 or an on-premises Db2 database from a Db2 LUW database using AWS Database Migration Service (AWS DMS).

For information about versions of Db2 LUW that AWS DMS supports as a target, see Targets for AWS DMS.

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

Limitations when using Db2 LUW as a target for AWS DMS

The following limitations apply when using Db2 LUW database as a target for AWS DMS. For limitations on using Db2 LUW as a source, see [Limitations when using Db2 LUW as a source for AWS DMS](#).

- AWS DMS only supports Db2 LUW as a target when the source is either Db2 LUW or Db2 for z/OS.
- Using Db2 LUW as a target doesn't support replications with full LOB mode.
- Using Db2 LUW as a target doesn't support the XML datatype in the full load phase. This is a limitation of the IBM dbload utility. For more information, see [The dbload utility](#) in the *IBM Informix Servers* documentation.
- AWS DMS truncates BLOB fields with values corresponding to the double quote character ("). This is a limitation of the IBM dbload utility.
- AWS DMS does not support the parallel full load option when migrating to a Db2 LUW target in DMS version 3.5.3. This option is available from DMS version 3.5.4 or later.

Endpoint settings when using Db2 LUW as a target for AWS DMS

You can use endpoint settings to configure your Db2 LUW target database similar to using extra connection attributes. You specify the settings when you create the target endpoint using the AWS DMS console, or by using the `create-endpoint` command in the [AWS CLI](#), with the `--ibm-db2-settings '{"EndpointSetting": "value", ...}'` JSON syntax.

The following table shows the endpoint settings that you can use with Db2 LUW as a target.

Name	Description
`KeepCsvFiles`	If true, AWS DMS saves any .csv files to the Db2 LUW target that were used to replicate data. DMS uses these files for analysis and troubleshooting..
`LoadTimeout`	The amount of time (in milliseconds) before AWS DMS times out operations performed by DMS on the Db2 target. The default value is 1200 (20 minutes).

Name	Description
`MaxFileSize`	Specifies the maximum size (in KB) of .csv files used to transfer data to Db2 LUW.
`WriteBufferSize`	The size (in KB) of the in-memory file write buffer used when generating .csv files on the local disk on the DMS replication instance. The default value is 1024 (1 MB).

Configuring VPC endpoints for AWS DMS

AWS DMS supports Amazon virtual private cloud (VPC) endpoints as sources and targets. AWS DMS can connect to any AWS source or target database with Amazon VPC endpoints as long as explicitly defined routes to these source and target databases are defined in their AWS DMS VPC.

By supporting Amazon VPC endpoints, AWS DMS makes it easier to maintain end-to-end network security for all replication tasks without additional networking configuration and setup. Using VPC endpoints for all source and target endpoints ensures that all your traffic remains within your VPC and under your control.

For AWS DMS replication instance created in private subnet or AWS DMS serverless replication, to connect to AWS managed databases, it is necessary to set up an Amazon VPC endpoint:

- Amazon S3
- Amazon DynamoDB
- Amazon Kinesis
- Amazon Redshift
- Amazon OpenSearch Service

If you are using AWS Secrets Manager to store connection credentials for DMS to use, you also need to set up a VPC endpoint.

Starting with AWS DMS version 3.4.7, VPC endpoints are required to establish connection between DMS replication instance or Serverless replication and the above Amazon services, when private network is used.

Common AWS DMS prerequisites

Before you configure a VPC endpoint, you must meet the following prerequisites:

- Locate or create the VPC to use with AWS DMS replication instance or AWS DMS serverless replication. If you do not provide this information, DMS attempts to use the default VPC in the region, where it is setup.
- Ensure you have IAM permissions to create VPC endpoint. To connect to Amazon S3 and Amazon DynamoDB, you can create Gateway VPC endpoints that provide reliable connectivity without requiring an internet gateway or a NAT device for your VPC. Gateway endpoints do not use AWS PrivateLink, unlike other types of VPC endpoints. For more information, see Gateway endpoints in the *AWS PrivateLink guide*.
- Configure IAM permissions to use DMS:
  - Configure the `dms-vpc-role` role. For more information, see AWS managed policy: AmazonDMSVPCManagementRole.
  - Configure the `dms-cloudwatch-logs-role` role. For more information, see AWS managed policy: AmazonDMSCloudWatchLogsRole.
  - AWS DMS Serverless requires a service linked role (SLR) to exist in your account. AWS DMS manages the creation and usage of this role. For more information about making sure that you have the necessary SLR, see Service-linked role for AWS DMS Serverless. When you create a replication, AWS DMS Serverless programmatically creates a Serverless service linked role. You can view this role in the IAM console.

Set up an Amazon VPC endpoint with AWS Secrets Manager

You can set up an Amazon VPC endpoint for AWS Secrets Manager to work with AWS DMS. By creating this endpoint, you enable AWS DMS replication instances or serverless replication configurations in private subnets to securely access database credentials stored in Secrets Manager without requiring public internet access.

Prerequisites

Before you configure a VPC endpoint with AWS Secrets Manager in AWS DMS, you must meet the following prerequisites:

- Ensure you configure all the the section called “Common AWS DMS prerequisites”.
- Create and configure source or target database that you want to connect with.

- Create secret in AWS Secrets manager with credentials to access source and target databases. Secret must be located in the same region as AWS DMS replication instance or AWS DMS serverless replication. Depending on the database type, the schema of the secret can vary. For more information, see Working with AWS DMS endpoints.

> **Important**
>
> AWS DMS replication instance and AWS DMS serverless replication do not work with secrets, managed by Amazon RDS. These credentials do not include host and port information, that is required by AWS DMS to establish connections.

- Configuring IAM permissions to manage DMS endpoint is required for some databases: Amazon S3, Amazon Kinesis, Amazon DynamoDB, Amazon Redshift, Amazon OpenSearch Service, Amazon Neptune, and Amazon Timestream. For more information see Working with AWS DMS endpoints.

Create VPC endpoint for AWS Secrets Manager

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. On the VPC console menu bar, choose the same AWS Region as your AWS DMS replication instance.
3. In the VPC navigation pane, choose **Endpoints**.
4. In **Endpoints**, choose **Create endpoint**.
5. Configure the VPC endpoint as follows:

   a. Select **Type** as **AWS Services**.

   b. In the **Service Name** textbox, search for **`secretsmanager`** and select **com.amazonaws.[region].secretsmanager**. Ensure that the **Type** for your selected service is **Interface**.

   c. Under **Network settings** select the VPC that is running in the same region as your DMS replication instance or where you created Serverless replication.

   d. In the **Subnets** section, select your desired subnets where you want DMS to operate. Ensure you only select private subnets. You can identify a private subnet with the subnet ID. For example: `vpc-xxxxxx-subnet-private1-us-west-2a`.

If your DMS replication instance is created without public access, you must choose the route tables associated with the private subnets where your replication instance resides.

> **Note**
>
> Ensure you note the private subnets as you are required to provide them when creating DMS replication subnet group. To connect DMS with the AWS Secrets manager using the VPC endpoints, the subnets specified for VPC endpoint must be the same as the subnets in DMS replication subnet group.

e. Select your desired **Security groups**. The security group rules control the traffic to the endpoint network interface from the resources in your VPC. If you do not specify a security group, the default security group is selected.

6. Select **Full access** under **Policy**. If you want to use custom policy to specify your own access control select **Custom**. You can use a trust policy that conforms with the JSON policy document, `dms-vpc-role`. For more information, see [Creating the IAM roles to use with AWS DMS](#).
7. Select **Create endpoint**.

   You must wait until the status becomes `Available`. Your VPC endpoint now has an ID starting with `vpce-xxxx`.

You have now successfully created a VPC endpoint. You must configure AWS DMS Endpoints, DMS subnet groups. Depending on migration option you choose, configure DMS replication instance or Serverless replication.

Set up an Amazon VPC endpoint with Amazon S3

You can set up an Amazon VPC endpoint for Amazon S3 to work with AWS DMS. By creating this endpoint, you enable AWS DMS replication instances or serverless replication configurations in private subnets to securely access database credentials stored in S3 buckets without requiring public internet access.

Prerequisites

Before you configure a VPC endpoint with Amazon S3 in AWS DMS, you must meet the following prerequisites:

- Ensure you configure all the [the section called "Common AWS DMS prerequisites"].
- Create an Amazon S3 bucket to use as source or target databases with AWS DMS. Do not enable versioning for S3. If you need S3 versioning, use lifecycle policies to actively delete old versions. Otherwise, you can encounter endpoint test connection failures because of an S3 list-object call timeout.
- Configure IAM permissions to manage DMS Amazon S3 endpoint. If you are using AWS DMS Console, the IAM role with the necessary permissions can be created if you have permissions to create IAM roles.

Create a VPC endpoint for Amazon S3

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. On the VPC console menu bar, choose the same AWS Region as your AWS DMS replication instance.
3. In the VPC navigation pane, choose **Endpoints**.
4. In **Endpoints**, choose **Create endpoint**.
5. Configure the VPC endpoint as follows:

    a. Select **Type** as **AWS Services**.

    b. In the **Service Name** textbox, search for **s3** and select **com.amazonaws.[region].s3**. Ensure that the **Type** for your selected service is **Gateway**. You can create a Gateway VPC endpoint when connecting to Amazon S3 and DynamoDB. Gateway endpoints do not use AWS PrivateLink, unlike other types of VPC endpoints.

    c. Under **Network settings** select the VPC that is running in the same region as your DMS replication instance or where you created Serverless replication.

    d. In the **Subnets** section, select your desired subnets where you want DMS to operate. Ensure you only select private subnets. You can identify a private subnet with the subnet ID. For example: `vpc-xxxxxx-subnet-private1-us-west-2a`.

    > **Note**
    >
    > If you have created your DMS replication instance without public access, you must choose the route tables associated with private subnets that are in the same region as your DMS instance. Ensure you note the private subnets as you are

required to provide them when creating DMS replication subnet group. To connect DMS with Amazon S3 using the VPC endpoints, the subnets specified for VPC endpoint must be the same as the subnets in DMS replication subnet group.

6. Select **Full access** under **Policy**. If you want to use custom policy to specify your own access control select **Custom**. You can use a trust policy that conforms with the JSON policy document, `dms-vpc-role`. For more information, see Creating the IAM roles to use with AWS DMS.
7. Select **Create endpoint**.

   You must wait until the status becomes `Available`. Your VPC endpoint now has an ID starting with `vpce-xxxx`.

You have now successfully created a VPC endpoint. You must configure AWS DMS Endpoints, DMS subnet groups. Depending on migration option you choose, configure DMS replication instance or Serverless replication.

Setup an Amazon VPC endpoint for Amazon DynamoDB

When using AWS DMS replication instances in private subnets or AWS DMS serverless replication, you must create a VPC endpoint to establish secure connectivity with Amazon DynamoDB. Without a VPC endpoint configuration, AWS DMS faces connection errors.

When creating the VPC endpoint, you must select **Endpoint type** as **Gateway** or **Interface** in the DMS Console. For more information, see:

- the section called “Common AWS DMS prerequisites”
- Gateway endpoints for Amazon DynamoDB
- How do I troubleshoot connectivity issues with my gateway Amazon VPC endpoints?

Setup an Amazon VPC endpoint for Amazon Kinesis

When using AWS DMS replication instances in private subnets or AWS DMS serverless replication, you must create a VPC endpoint to establish secure connectivity with Amazon Kinesis. Without a VPC endpoint configuration, AWS DMS faces connection errors. For more information, see:

- the section called “Common AWS DMS prerequisites”

- Using Amazon Kinesis Data Streams as a target for AWS Database Migration Service

Setup an Amazon VPC endpoint for Amazon Redshift

When using AWS DMS replication instances in private subnets or AWS DMS serverless replication, you must create a VPC endpoint to establish secure connectivity with Amazon Redshift. Without a VPC endpoint configuration, AWS DMS faces connection errors. For more information, see:

- the section called "Common AWS DMS prerequisites"
- Redshift-managed VPC endpoints

Setup an Amazon VPC endpoint for Amazon OpenSearch Service

When using AWS DMS replication instances in private subnets or AWS DMS serverless replication, you must create a VPC endpoint to establish secure connectivity with Amazon OpenSearch Service. Without a VPC endpoint configuration, AWS DMS faces connection errors. For more information, see:

- the section called "Common AWS DMS prerequisites"
- Configuring VPC access for Amazon OpenSearch Ingestion pipelines

Setup replication instances, DMS subnet groups, and DMS Endpoints

You must configure AWS DMS replication resources after creating VPC endpoints. You can set up replication subnet groups for network isolation, replication instances or serverless replications for processing, and endpoints for connecting to source and target databases to enable secure database migration within your VPC.

Setup an AWS DMS replication instance

To configure an AWS DMS provisioned replication instance you must setup DMS replication subnets groups.

Create DMS replication subnet groups

1. Sign in to the AWS Management Console and open the DMS console.
2. From the left navigation pane, open to **Subnet groups** and select **Create subnet group**.

3. Enter **Name** and **Description**.
4. From the **VPC** dropdown menu, select the VPC that is running in the same region where you want to create your DMS replication instance.
5. From the **Add subnets** dropdown menu, add the private subnets that you have specified when creating your VPC endpoint. You can identify a private subnet with the subnet ID. For example: `vpc-xxxxxx-subnet-private1-us-west-2a`.
6. Click **Create subnet group**.

Create DMS replication instance (provisioned)

1. Navigate to the AWS Management Console to create a replication instance. For more information, see [Creating a replication instance](). To understand more choosing, sizing, and configuring replication instances, see [Working with an AWS DMS replication instance]().
2. In the **Connectivity and security** section, select the VPC from the **Virtual private cloud (VPC) for IPv4** or **Dual-stack mode** where you want to create the AWS DMS replication instance. For more information, see [Setting up a network for a replication instance]().
3. From the **Replication subnet group** dropdown menu, choose the subnet group that you created for your replication instance.

   > **Note**
   >
   > Ensure the subnets specified for the VPC endpoint are identical to the subnets in the DMS replication instance subnet group. You must remove any subnets from your subnet group that are not associated with VPC endpoint.

4. Uncheck the **Public accessible** checkbox to disable public access.
5. In **Advanced settings** section, from the **VPC security groups** dropdown menu, select all the VPC subnet groups associated with your replication instance. These groups must include the subnet group that includes subnets you specified when creating the VPC endpoint.

   If you do not specify the subnet groups, DMS chooses the default **Replication subnet group** or creates it if it does not exist. For more information, see [Security group configuration for AWS DMS]().
6. Complete the replication instance configuration and select **Create replication instance**.

   You must wait until the status becomes `Available`.

Create AWS DMS source and target Endpoints

1. Sign in to the DMS console.
2. Navigate to the **AWS DMS endpoints** and select **Create endpoint**.
3. Create and configure source and target endpoints.
4. In the DMS console, you can choose an existing IAM role or create a new IAM role to access your stored database credentials in the AWS Secrets manager.
5. Click **Run test** to test the endpoint connection in your DMS replication instance. Your replication instance should have `Available` status to run the test with it.
6. Select **Create endpoint**.

Setup an AWS DMS serverless replication

To configure an AWS DMS serverless replication you must setup DMS replication subnets groups.

Create AWS DMS source and target Endpoints

1. Sign in to the DMS console.
2. Navigate to the **AWS DMS endpoints** and select **Create endpoint**.
3. Create and configure source and target endpoints.
4. In the DMS console, you can choose an existing IAM role or create a new IAM role to access your stored database credentials in the AWS Secrets manager.

   > **Note**
   >
   > For AWS DMS serverless replication, you cannot test the connection for the DMS endpoint or use the `TestConnection` API. The connection test is performed during serverless replication launch between DMS instance and your source/target databases. For more information, see AWS DMS Serverless components.

5. Select **Create endpoint**.

Create DMS replication subnet groups

1. Sign in to the AWS Management Console and open the DMS console.
2. From the left navigation pane, open to **Subnet groups** and select **Create subnet group**.

3. Enter **Name** and **Description**.

4. From the **VPC** dropdown menu, select the VPC that is running in the same region as your DMS serverless instance.

5. From the **Add subnets** dropdown menu, add the private subnets that you have specified when creating your VPC endpoint. You can identify a private subnet with the subnet ID. For example: `vpc-xxxxxx-subnet-private1-us-west-2a`.

6. Click **Create subnet group**.

Create DMS serverless replication

1. Navigate to the DMS console to create a serverless instance. For more information, see Creating a serverless replication. To understand more choosing, sizing, and configuring serverless instances, see Working with AWS DMS serverless.

2. In the **Connectivity and security** section, select the VPC from the **Virtual private cloud (VPC)** dropdown menu where you want to create the AWS DMS serverless instance . For more information, see Setting up a network for a replication instance.

3. From the **Subnet group** dropdown menu, choose the subnet group that you created for your serverless instance.

   > **Note**
   >
   > Ensure the subnets specified for the VPC endpoint are identical to the subnets in the DMS serverless instance subnet group. You must remove any subnets from your subnet group that are not associated with your serverless instance.

4. Select **Availability zone**.

5. From the **Maximum DMS capacity units (DCU)** dropdown menu, select the desired DCU capacity.

6. Select **Create task**. This creates a DMS serverless replication configuration that appears in the task list with the status `Start required`.

7. To start serverless replication, choose your task and select **Start** from the **Actions** menu.

Who is impacted when migrating to AWS DMS versions 3.4.7 and higher?

You are impacted if you are using one or more of the previously listed AWS DMS endpoints, and these endpoints are not publicly routable or they don't have VPC endpoints already associated with them.

Who is not impacted when migrating to AWS DMS versions 3.4.7 and higher?

You are not impacted if:

- You aren't using one or more of the previously listed AWS DMS endpoints.
- You are using any of the previously listed endpoints and they are publicly routable.
- You are using any of the previously listed endpoints and they have VPC endpoints associated with them.

Preparing a migration to AWS DMS versions 3.4.7 and higher

To prevent AWS DMS task failures when you are using any of the endpoints described previously, take one of the steps following prior to upgrading AWS DMS to version 3.4.7 or higher:

- Make the impacted AWS DMS endpoints publicly routable. For example, add an Internet Gateway (IGW) route to any VPC already used by your AWS DMS replication instance to make all its source and target endpoints publicly routable.
- Create VPC endpoints to access all source and target endpoints used by AWS DMS as described following.

For any existing VPC endpoints that you use for your AWS DMS source and target endpoints, ensure that they use a trust policy that conforms with the XML policy document, `dms-vpc-role`. For more information on this XML policy document, see Creating the IAM roles to use with AWS DMS.

Otherwise, configure your replication instances as VPC endpoints by adding a VPC endpoint to the VPC containing them. If you configured your replication instances without public endpoints, adding a publicly-accessible VPC endpoint to the VPC that contains your replication instances makes them publicly accessible. You don't need to do anything further to specifically associate your replication instances with the VPC endpoint.

> **Note**
>
> Different services might have unique VPC endpoint configurations. For instance, when using AWS Secrets Manager, you typically don't need to adjust the routing table. Always check the specific requirements for each service.

For more information on configuring VPC endpoints for an AWS DMS replication instance, see Network configurations for database migration. For more information on creating interface VPC endpoints for accessing AWS services generally, see Access an AWS service using an interface VPC endpoint in the *AWS PrivateLink Guide*. For information on AWS DMS regional availability for VPC endpoints, see the AWS Region Table.

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

DMS doesn't capture all supported DDL statements for some source engine types. And DMS handles DDL statements differently when applying them to specific target engines. For information about which DDL statements are supported for a specific source, and how they're applied to a target, see the specific documentation topic for that source and target endpoint.

You can use task settings to configure the way DMS handles DDL behavior during change data capture (CDC). For more information, see Task settings for change processing DDL handling.

Limitations and considerations

Rapid sequences of DDL operations in the source database (such as DDL>DML>DDL) can cause AWS DMS to parse the log incorrectly, leading to data loss or unexpected behavior. To maintain data consistency, wait for AWS DMS to apply each change to the target before performing subsequent operations.

For example, during change data capture (CDC), multiple rapid table rename operations on a source table can trigger errors. If you rename a table and then quickly rename it back to its original name, AWS DMS might report that the table already exists in the target database.

Advanced endpoint configuration

You can configure advanced settings for your endpoints in AWS Database Migration Service (AWS DMS) to setup control over how source and target endpoints behave during the migration process. As part of the advanced setup you can configure AWS DMS VPC peering to enable secure communication between VPCs, DMS Security Groups to control inbound and outbound traffic, Newtwork Access Control lists (NACLs) as additional layer of security, and VPC endpoints for AWS Secrets Manager.

You can set these configurations during endpoint creation or modified later through the AWS DMS Console or API, to fine-tune the migration processes based on specific database engine requirements and performance needs.

Following, you can find out more details about advanced endpoint configuration.

Topics

VPC peering configuration for AWS DMS.

VPC peering enables private network connectivity between two VPCs, allowing AWS DMS replication instances and database endpoints to communicate across different VPCs as if they were in the same network. This is crucial when your DMS replication instance resides in one VPC while source or target databases exist in separate VPCs, enabling direct, secure data migration without traversing the public internet.

When using Amazon RDS, you must configure VPC peering between DMS and RDS if your instances are located in different VPCs.

You must perform the following steps:

Creating a VPC peering connection

1. Navigate to the [Amazon VPC console](https://console.aws.amazon.com/vpc/).
2. In the navigation pane, select **Peering Connections** under **Virtual private cloud**.
3. Click **Create Peering Connection**.
4. Configure the peering connections:

   - Name tag (optional): Enter a name for the peering connection (example: `DMS-RDS-Peering`).

     VPC Requester: Select the VPC that contains your DMS instance.
   - **VPC accepter**: Select the VPC that contains your RDS instance.

   > **Note**
   >
   > If the accepter VPC is associated with a different AWS account, you must have the Account ID and VPC ID for that acount.
5. Click **Create the Peering Connection**.

Accepting the VPC peering connection

1. In the **Peering Connections** list, find the new peering connection with a **Pending Acceptance** status.
2. Select the appropriate peering connection, click **Actions** and select **Accept Request**.

   The peering connection status changes to **Active**.

Updating route tables

To enable traffic between the VPCs, you must update the route table in both your VPCs. To update the route tables in the DMS VPC:

1. Identify CIDR block of the RDS VPC:
   a. Navigate to your VPCs and select your RDS VPC.
   b. Copy the IPv4 CIDR value in **CIDRs** tab.
2. Identify relevant DMS route tables using resource map:
   a. Navigate to your VPCs and select your DMS VPC.
   b. Click the **Resource Map** tab and note the route tables associated with the subnets where your DMS instance is located.
3. Update all route tables in the DMS VPC:
   a. Navigate to the route tables in the Amazon VPC console.
   b. Select the route tables identifies for the DMS VPC. You can open them from the VPC's **Resource map** tab.
   c. Click **Edit routes**.
   d. Click Add route and enter the following information:
      - **Destination**: Enter the IPv4 CIDR block of the RDS VPC (Example: `10.1.0.0/16`).
      - **Target**: Select the peering configuration ID (Example: `pcx-1234567890abcdef`).
   e. Click **Save routes**.

      Your VPC routes are saved for the DMS VPC. Perform the same steps for your RDS VPC.

Update Security Groups

1. Verify the DMS instance Security Group:
   - You must ensure that the outbound rules allow traffic to the RDS instance:
     - **Type**: Custom TCP or the specific database port (Example: 3306 fir MySQL).
     - **Destination**: The CIDR block of the RDS VPC or the security group of the RDS instance.
2. Verify the RDS instance Security Group:

- You must ensure that the inbound rules allow traffic from the DMS instance:
  - **Type**: The specific database port.
  - Source: The CIDR block of the DMS VPC or the security group of the RDS instabce.

> **Note**
>
> You must also ensure the following:
>
> - **Active Peering Connection**: Ensure the VPC peering connection is in the **Active** state before proceeding.
> - **Resource Map**: Use the **Resource map** tab in the Amazon VPC console console to identify which route tables need update.
> - **No Overlapping CIDR Blocks**: The VPCs must have non-overlapping CIDR blocks.
> - **Security Best Practices**: Restrcict Security Group rules to the necessary ports and sources.
>
> For more information, see VPC peering connections in the *Amazon Virtual Private Cloud user guide*.

Security group configuration for AWS DMS

Security group in AWS DMS must allow inbound and outbound connections for your replication instances on the appropriate database port. If you are using Amazon RDS, you must configure the security group between DMS and RDS for your instances.

You must perform the following steps:

Configure the RDS instance security group

1. Navigate to the Amazon VPC console.
2. In the navigation pane on the left under **Security**, select **Security Groups**.
3. Select the RDS Security Group associated with your RDS instance.
4. Edit the inbound rules:
   a. Click **Actions** and select **Edit inbound rules**.

b. Click **Add Rule** to create a new rule.

c. Configure the rule as follows:

   - **Type**: Select your database type (Example: MySQL/Aurora for port 3306, PostgreSQL for port 5432).
   - **Protocol**: This auto-populates based on your database type.
   - **Port Range**: this auto-populates based on your database type.
   - **Source**: Choose **Custom**, and paste the security group ID associated with your DMS instance. This allows traffic from any resource within that security group. You can also specify the IP range (CIDR block) of your DMS instance.

d. Click **Save rules**.

Configure the DMS replication instance security group

1. Navigate to the [Amazon VPC console](#).
2. In the navigation pane on the left under **Security**, select **Security Groups**.
3. In the **Security Group** list find and select the security group associated with your DMS replication instance.
4. Edit the outbound rules:

   a. Click **Actions** and select **Edit outbound rules**.

   b. Click **Add Rule** to create a new rule.

   c. Configure the rule as follows:

      - Type: Select your database type (Example: MySQL/Aurora, PostgreSQL).
      - Protocol: This auto-populates based on your database type.
      - Port Range: this auto-populates based on your database type.
      - Source: Choose **Custom**, and paste the security group ID associated with your RDS instance. This allows traffic from any resource within that security group. You can also specify the IP range (CIDR block) of your RDS instance.

   d. Click **Save rules**.

Additional Considerations

You must consider the following additional configuration information:

- **Use Security Group References**: Referencing security groups in the source or destional instances allows for dynamic management and is more secure than using IP addresses as it automatically included all resources within the group.
- **Database Ports**: Ensure you are using the correct port for your database.
- **Security Best Practices**: Only open the necessary ports to minimize security risks. you must also regular review of your security group rules to ensure they meed your security standards and requirements.

Network Access Control List (NACL) configuration for AWS DMS

When using Amazon RDS as a replication source, you should update the Network Access Control Lists (NACLs) for your DMS and RDS instance. Ensure that the NACLs are associated with the subnets where these instances reside. This allows inbound and outbound traffic on the specific database port.

To update the Network Access Control Lists, you must perform the following steps:

> **Note**
>
> If your DMS and RDS instances are in the same subnet, you only need to update that subnet's NACL.

Identify the relevant NACLs

1. Navigate to the Amazon VPC console.
2. In the navigation pane on the left under **Security**, select **Network ACLs**.
3. Select the relevant NACLs associated with the subnets where your DMS and RDS instances reside.

Update the NACLs for the DMS instance subnet

1. Identify the NACL associated with your DMS instance's subnet. To do so, you can browse through the subnets in the Amazon VPC console, find the DMS subnet, and note the associated NACL ID.
2. Edit the inbound rules:
    a. Click the **Inbound Rules** tab for the selected NACL.
    b. Select **Edit inbound rules**.
    c. Add a new rule:
        - **Rule #**: Choose a unique number (Example: 100).
        - **Type**: Select **Custom TCP Rule**.
        - **Protocol**: TCP
        - **Port Range**: Enter your database port (Example: 3306 for MySQL).
        - **Source**: Enter the CIDR block of the RDS subnet (Example: 10.1.0.0/16).
        - **Allow/Deny**: Select **Allow**.
3. Edit the outbound rules:
    a. Click the **Outbound Rules** tab for the selected NACL.
    b. Click **Edit outbound rules**.
    c. Add a new rule:
        - **Rule #**: Use the same number as used in the inbound rules.
        - **Type**: All traffic.
        - **Destination**: 0.0.0.0/0
        - **Allow/Deny**: Select **Allow**.
4. Click **Save changes**.
5. Perform the same steps to update the NACLs associated with the RDS instance's subnet.

Verify the NACL rules

You must ensure the following criteria for regarding the NACL rules.:

- **Order of rules**: NACLs processes rules in the ascending order based on th erule number. Ensure that all the rules set as "**Allow**" have lower rule numbers than all the rules set as "**Deny**" as that might block traffic.
- **Stateless nature**: NACLs are stateless. You must explicity allow both inbound and outbound traffic.
- **CIDR blocks**: You must ensure that the CIDR blocks you use accurately represent the subnets of your DMS and RDS instances.

Configuring AWS DMS secrets manager VPC Endpoint

You must create a VPC endpoint to access the AWS Secrets Manager from a replication instance in a private subnet. This allows the replication instance access the Secrets Manager directly through the private network without sending traffic over the public internet.

To configure, you must follow the following steps:

Create a security group for the VPC endpoint.

1. Navigate to the [Amazon VPC console](https://console.aws.amazon.com/vpc/).
2. In the navigation pane on the left, select **Security groups**, and choose **Create security group**.
3. Configure security group details:
   - **Security group name**: Example: `SecretsManagerEndpointSG`
   - **Description**: Enter an appropriate description. (Example: Security group for secrets manager VPC endpoint).
   - **VPC**: Select the VPC where your replication instance and endpoints reside.
4. Click **Add Rule** to set inbound rules and configure the following:
   - Type: HTTPS (As the secrets manager uses HTTPS on port 443).
   - Source: Choose **Custom**, and enter the securty group ID of your replication instance. This ensures that any instance associated with that security group can access the VPC endpoint.
5. Review the changes and click **Create security group**.

Create a VPC endpoint for secrets manager

> **Note**
>
> Create an interface VPC endpoint as outline in the Creating an Interface Endpoint documentation topic in the *Amazon Virtual Private Cloud user guide*. When following this procedure, ensure the following:
>
> - For **Service Category**, you should select **AWS services.**
> - For **Service name**, search `seretsmanager` and select the secretes manager service.

1. Select **VPC and Subnets** and configure the following:
   - **VPC**: Ensure it is the same VPC as your replication instance.
   - **Subnets**: Select the subnets where your replication instance resides.
2. In **Additional Settings**, ensure that the **Enable DNS name** is enabled by default for the interface endpoints
3. Under **Security group**, select the appropriate security group name. Example: `SecretsManagerEndpointSG` as created earlier).
4. Review all the settings and Click **Create endpoint**.

Retrieve the VPC endpoint DNS name

1. Access the VPC endpoint details:
   a. Navigate to the Amazon VPC console and choose **Endpoints**.
   b. Select the appropriate endpoint you created.
2. Copy the DNS name:
   a. Under the **Details** tab, navigate to the **DNS Names** section.
   b. Copy the first DNS name listed. (Example: `vpce-0abc123def456789g-secretsmanager.us-east-1.vpce.amazonaws.com`). This is the regional DNS name.

Update your DMS endpoint

1. Navigate to the AWS DMS console.

2. Modify the DMS endpoint:

   a. In the navigation pane on the left, select **Endpoints**.

   b. Choose the appropriate endpoint you want to configure.

   c. Click **Actions** and select **Modify**.

3. Configure endpoint settings:

   a. Navigate to **Endpoint settings** and select **Use endpoint connection attributes** checkbox.

   b. In the **Connection attributes** field, add: secretsManagerEndpointOverride=<copied DNS name>.

   > **Note**
   >
   > If you have multiple connection attributes, you can separate them with a semicolon ";". For example: datePartitionEnabled=false;secretsManagerEndpointOverride=vpce-0abc12
   > secretsmanager.us-east-1.vpce.amazonaws.com

4. Click **Modify endpoint** to save your changes.

Additional considerations

You must consider the following additional configuration information:

Replication instance security group:

- Ensure that the security group associated with your replication instance allows outbound traffic to the VPC endpoint on port 443 (HTTPS).

VPC DNS settings:

- Confirm that **DNS resolution** and **DNS hotnames** are enabled in your VPC. This allows your instances to resolve the VPC endpoint DNS names. You can confirm that by navigating to VPCs in the [Amazon VPC console]() and select your VPC to verify that **DNS resolution** and **DNS hotnames** are set to "**Yes**".

Testing connectivity:

- From your replication instance, you can perform a DNS lookup to ensure it resolves the VPC endpoint: `nslookup secretsmanager.<region>amazonaws.com`. It must return the Ip address associated with your VPC endpoint

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.

A task can consist of three major phases:

- Migration of existing data (Full load)
- The application of cached changes
- Ongoing replication (Change Data Capture)

For more information and an overview of how AWS DMS migration tasks migrate data, see High-level view of AWS DMS

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. For information about how to use a task configuration file to set task settings, see Task settings example.
- 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. For information about stopping a task using the AWS DMS API, see StopReplicationTask in the AWS DMS API Reference.

The following are actions that you can do when working with an AWS DMS task.

Task	Relevant documentation
Creating a task When you create a task, you specify the source, target, and replication instance, along with any migration settings.	Creating a task
Creating an ongoing replication task You can set up a task to provide continuous replication between the source and target.	Creating tasks for ongoing replication using AWS DMS
Applying task settings 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. For information about how to use a task configuration file to set task settings, see Task settings example.	Specifying task settings for AWS Database Migration Service tasks
Using table mapping Table mapping specifies additional task settings for tables using several types of rules. These rules allows you to specify the data source, source schema, tables and views, data, any table and data transformations that are to occur during the task, and settings for how these tables	Selection rules Selection rules and actions Transformation rules Transformation rules and actions Table-settings rules Table and collection settings rules and operations

Task	Relevant documentation
and columns are migrated from the source to the target.	
Running premigration task assessments You can enable and run premigrat ion 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 conflicti ng task settings. These premigrat ion assessments run before you run the task to identify potential issues before they occur during a migration .	Enabling and working with premigration assessments for a task
Data validation Data validation is a task setting you can use to have AWS DMS compare the data on your target data store with the data from your source data store.	AWS DMS data validation.
Modifying a task When a task is stopped, you can modify the settings for the task.	Modifying a task

Task	Relevant documentation
Moving a task When a task is stopped, you can move the task to a different replicati on instance.	Moving a task
Reloading tables during a task You can reload a table during a task if an error occurs during the task.	Reloading tables during a task
Applying filters You can use source filters to limit the number and type of records transferred from your source to your target. For example, you can specify that only employees with a location of headquarters are moved to the target database. You apply filters on a column of data.	Using source filters
Monitoring a task There are several ways to get information on the performance of a task and the tables used by the task.	Monitoring AWS DMS tasks
Managing task logs You can view and delete task logs using the AWS DMS API or AWS CLI.	Viewing and managing AWS DMS task logs

Topics

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.
- 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.
- 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. 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.
- Specify any required supplemental data for the task to migrate your data. For more information, see Specifying supplemental data for task settings.

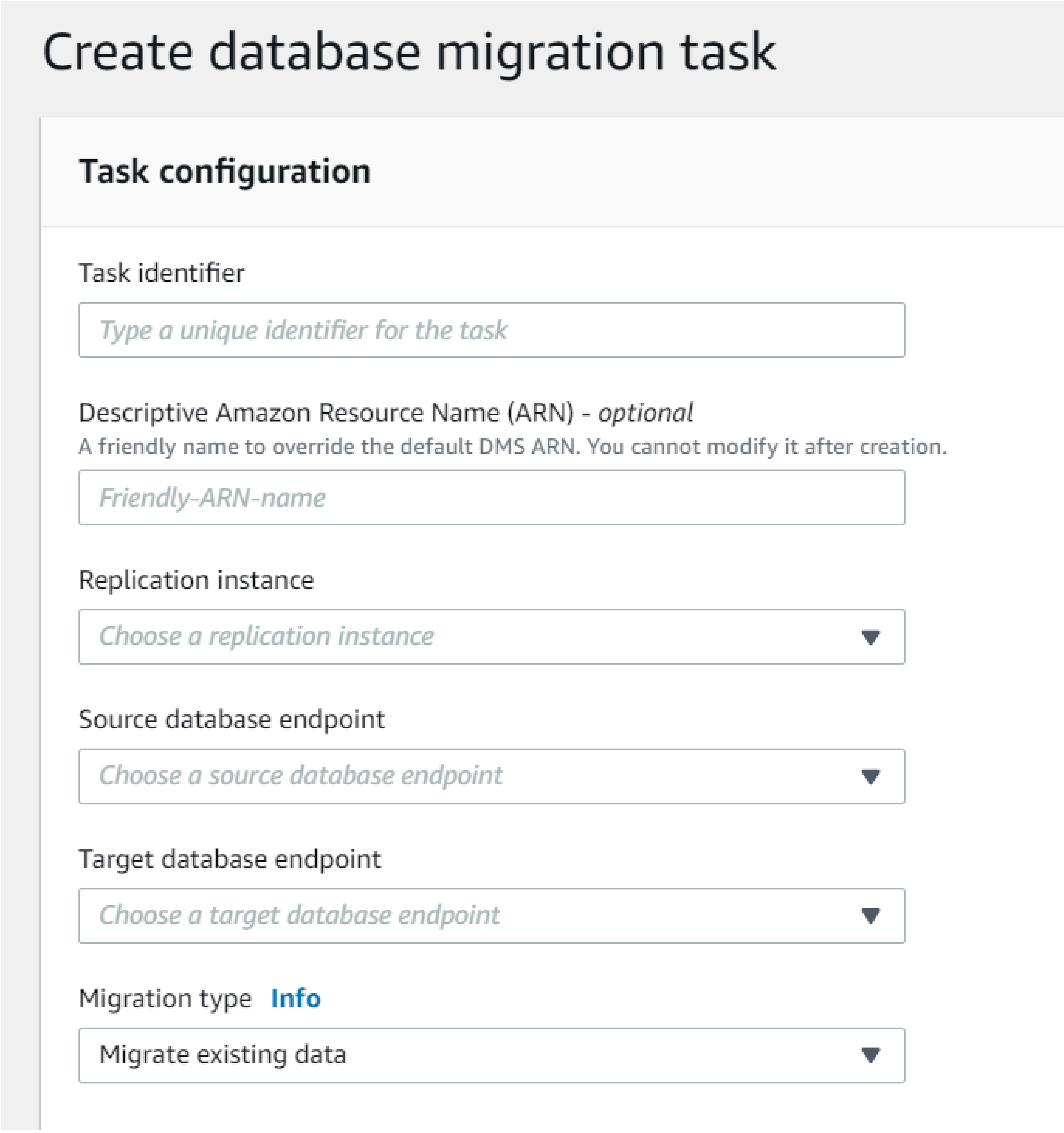
You can choose to start a task as soon as you finish specifying information for that task on the **Create task** page. Alternatively, you can start the task from the Dashboard page later as well.

The following procedure assumes that you have already specified replication instance information and endpoints. For more information about setting up endpoints, see Creating source and target endpoints.

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 about the permissions required, see IAM permissions needed to use AWS DMS.
2. On the navigation pane, choose **Database migration tasks**, and then choose **Create task**.
3. On the **Create database migration task** page, in the **Task configuration** section, specify the task options. The following table describes the settings.



For this option	Do this
Task identifier	Enter a name for the task.

For this option	Do this
Descriptive Amazon Resource Name (ARN) - *optional*	A friendly name to override the default AWS DMS ARN. You can't change this name after you create the task.
Replication instance	Shows the replication instance to be used.
Source database endpoint	Shows the source endpoint to be used.
Target database endpoint	Shows the target endpoint to be used.
Migration type	Choose the migration method you want to use. You can choose to have just the existing data migrated to the target database or have ongoing changes sent to the target database in addition to the migrated data.

4. In the **Task Settings** section, specify values for editing your task, target table preparation mode, stop task, LOB settings, validation, and logging.

For this option	Do this
Editing mode	Choose whether to use the Wizard or the JSON editor to specify your task settings. If you choose Wizard, the following options will be displayed.
CDC start mode for source transactions	This setting is only visible if you choose **Replicate data changes only** for **Migration type** in the preceding section. **Disable custom CDC start mode** – If you choose this option, you can start your task either automatic ally by using the **Automatically on create** option following, or manually by using the console. **Enable custom CDC start mode** – If you choose this option, you can specify a custom UTC start time to start processing changes.

For this option	Do this
Target table preparation mode	This setting is only visible if you choose **Migrate existing data** or **Migrate existing data and replicate ongoing changes** for **Migration type** in the preceding section. **Do nothing** – In **Do nothing** mode, AWS DMS assumes that the target tables have been pre-created on the target. If the tables aren't empty, conflicts might occur during data migration and can result in a DMS task error. If the target table doesn't exist, DMS creates the table for you. Your table structure remains as is and any existing data is left in the table. **Do nothing** 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 pre-create tables, you can use the AWS Schema Conversion Tool (AWS SCT). For more information, see Installing AWS SCT. **Drop tables on target** – In **Drop tables on target** 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.

For this option	Do this
	You might need to perform some configuration on the target database when you use **Drop tables on target** mode. For example, for an Oracle target, AWS DMS 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. **Truncate** – In **Truncate** mode, AWS DMS truncates all target tables before the migration starts. If the target table doesn't exist, DMS creates the table for you. Your table structure remains as is but tables are truncated at the target. **Truncate** 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 Installing AWS SCT. **Note** If your target is MongoDB, **Truncate** mode doesn’t truncate tables at the target. Instead, it drops the collection and loses all the indices. Avoid **Truncate** mode when your target is MongoDB.

For this option	Do this
Stop task after full load completes	This setting is only visible if you choose **Migrate existing data and replicate ongoing changes** for **Migration type** in the preceding section. **Don't stop –** Don't stop the task but immediately apply cached changes and continue on. **Stop before applying cached changes –** Stop the task before the application of cached changes. Using this approach, you can add secondary indexes that might speed the application of changes. **Stop after applying cached changes –** Stop the task after cached changes have been applied. Using this approach, you can add foreign keys if you are using transactional apply.
Include LOB columns in replicati on	**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 **LOB Chunk size** parameter. This mode is slower than using Limited LOB mode. **Limited LOB mode –** Truncate LOBs to the value of the **Max LOB size** parameter. This mode is faster than using Full LOB mode.
Maximum LOB size (kb)	In **Limited LOB Mode**, LOB columns that exceed the setting of **Max LOB size** are truncated to the specified **Max LOB Size** value.
Enable validation	Enables data validation, to verify that the data is migrated accurately from the source to the target. For more information, see AWS DMS data validation.

For this option	Do this
Enable CloudWatch logs	Enables logging by Amazon CloudWatch.

5. In the **Premigration assessment** section, choose whether to run a premigration assessment. A premigration assessment warns you of potential migration issues before starting your database migration task. For more information, see Enabling and working with premigration assessments.
6. In the **Migration task startup configuration** section, specify whether to start the task automatically after creation.
7. In the **Tags** section, specify any tags you need to organize your task. You can use tags to manage your IAM roles and policies, and track your DMS costs. For more information, see Tagging resources.
8. 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. For information about how to use a task configuration file to set task settings, see Task settings example.

There are several main types of task settings, as listed following.

Topics

- [Task settings for change processing DDL handling]
- [Character substitution task settings]
- [Before image task settings]
- [Error handling task settings]
- [Saving task settings]

Task settings	Relevant documentation
Creating a task assessment report You can create a task assessment report that shows any unsupport ed 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.	Enabling and working with premigration assessments for a task
Creating a task When you create a task, you specify the source, target, and replication instance, along with any migration settings.	Creating a task
Creating an ongoing replication task You can set up a task to provide continuous replication between the source and target.	Creating tasks for ongoing replication using AWS DMS
Applying task settings Each task has settings that you can configure according to the needs of your database migration. You create	Specifying task settings for AWS Database Migration Service tasks

Task settings	Relevant documentation
these settings in a JSON file or, with some settings, you can specify the settings using the AWS DMS console.	
Data validation Use data validation to have AWS DMS compare the data on your target data store with the data from your source data store.	AWS DMS data validation
Modifying a task When a task is stopped, you can modify the settings for the task.	Modifying a task
Reloading tables during a task You can reload a table during a task if an error occurs during the task.	Reloading tables during a task
Using table mapping 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 Selection rules and actions Transformation Rules Transformation rules and actions

Task settings	Relevant documentation
Applying filters You can use source filters to limit the number and type of records transferred from your source to your target. For example, you can specify that only employees with a location of headquarters are moved to the target database. You apply filters on a column of data.	Using source filters
Monitoring a task There are several ways to get information on the performance of a task and the tables used by the task.	Monitoring AWS DMS tasks
Managing task logs You can view and delete task logs using the AWS DMS API or AWS CLI.	Viewing and managing AWS DMS task logs

Task settings example

You can use either the AWS Management Console or the AWS CLI to create a replication task. If you use the AWS CLI, you set task settings by creating a JSON file, then specifying the file:// URI of the JSON file as the ReplicationTaskSettings parameter of the CreateReplicationTask operation.

The following example shows how to use the AWS CLI to call the `CreateReplicationTask` operation:

```
aws dms create-replication-task \
--replication-task-identifier MyTask \
--source-endpoint-arn arn:aws:dms:us-
west-2:123456789012:endpoint:ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890ABC \
--target-endpoint-arn arn:aws:dms:us-
west-2:123456789012:endpoint:ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890ABC \
```

```
--replication-instance-arn arn:aws:dms:us-
west-2:123456789012:rep:ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890ABC \
--migration-type cdc \
--table-mappings file://tablemappings.json \
--replication-task-settings file://settings.json
```

The preceding example uses a table mapping file called `tablemappings.json`. For table mapping examples, see Using table mapping to specify task settings.

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": 100,
    "ParallelApplyQueuesPerThread": 1,
    "BatchApplyEnabled": false,
    "TaskRecoveryTableEnabled": false
  },
  "FullLoadSettings": {
    "TargetTablePrepMode": "DO_NOTHING",
    "CreatePkAfterFullLoad": false,
    "StopTaskCachedChangesApplied": false,
    "StopTaskCachedChangesNotApplied": false,
    "MaxFullLoadSubTasks": 8,
    "TransactionConsistencyTimeout": 600,
    "CommitRate": 10000
  },
    "TTSettings" : {
    "EnableTT" : true,
```

```
    "TTS3Settings": {
        "EncryptionMode": "SSE_KMS",
        "ServerSideEncryptionKmsKeyId": "arn:aws:kms:us-west-2:112233445566:key/
myKMSKey",
        "ServiceAccessRoleArn": "arn:aws:iam::112233445566:role/dms-tt-s3-access-role",
        "BucketName": "myttbucket",
        "BucketFolder": "myttfolder",
        "EnableDeletingFromS3OnTaskDelete": false
      },
    "TTRecordSettings": {
        "EnableRawData" : true,
        "OperationsToLog": "DELETE,UPDATE",
        "MaxRecordSize": 64
      }
  },
  "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"
  },

  "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",
    "DataMaskingErrorPolicy": "STOP_TASK",
    "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
  },
  "ValidationSettings": {
    "EnableValidation": false,
    "ValidationMode": "ROW_LEVEL",
    "ThreadCount": 5,
    "PartitionSize": 10000,
    "FailureMaxCount": 1000,
    "RecordFailureDelayInMinutes": 5,
    "RecordSuspendDelayInMinutes": 30,
    "MaxKeyColumnSize": 8096,
    "TableFailureMaxCount": 10000,
    "ValidationOnly": false,
    "HandleCollationDiff": false,
    "RecordFailureDelayLimitInMinutes": 1,
    "SkipLobColumns": false,
    "ValidationPartialLobSize": 0,
    "ValidationQueryCdcDelaySeconds": 0
  }
}
```

Target metadata task settings

Target metadata settings include the following. For information about how to use a task configuration file to set task settings, see Task settings example.

- `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. Renaming a schema in data mapping takes precedence over this setting.
- 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. The recommended value for `LobChunkSize` is 64 kilobytes. Increasing the value for `LobChunkSize` above 64 kilobytes can cause task failures.

- `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 –102400 kilobytes (100 MB). 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. The maximum value for `LobMaxSize` is 102400 kilobytes (100 MB).

  For more information about the criteria for using these task LOB settings, see Setting LOB support for source databases in an AWS DMS task. You can also control the management of LOBs for individual tables. For more information, see Table and collection settings rules and operations.

- `BatchApplyEnabled` – Determines if each transaction is applied individually or if changes are committed in batches. The default value is `false`.

  When `BatchApplyEnabled` is set to `true`, DMS requires a Primary Key (PK) or Unique Key (UK) on the **source** table(s). Without a PK or UK on source tables, only batch inserts are applied but not batch updates and deletes.

  When `BatchApplyEnabled` is set to `true`, AWS DMS generates an error message if a **target** table has a unique constraint and a primary key. Target tables with both a unique constraint and primary key aren't supported when `BatchApplyEnabled` is set to `true`.

  When `BatchApplyEnabled` is set to true and AWS DMS encounters a data error from a table with the default error-handling policy, the AWS DMS task switches from batch mode to one-by-one mode for the rest of the tables. To alter this behavior, you can set the `"SUSPEND_TABLE"` action on the following policies in the `"ErrorBehavior"` group property of the task settings JSON file:

  - `DataErrorPolicy`
  - `ApplyErrorDeletePolicy`
  - `ApplyErrorInsertPolicy`
  - `ApplyErrorUpdatePolicy`

For more information on this "ErrorBehavior" group property, see the example task settings JSON file in Specifying task settings for AWS Database Migration Service tasks. After setting these policies to "SUSPEND_TABLE", the AWS DMS task then suspends data errors on any tables that raise them and continues in batch mode for all tables.

You can use the BatchApplyEnabled parameter 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 about using these settings for a change data capture (CDC) load, see Change processing tuning settings.

- MaxFullLoadSubTasks – indicates the maximum number of tables to load in parallel. The default is 8; the maximum value is 49.
- 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 OpenSearch Service target is 32. You can ask to have this maximum limit increased. ParallelLoadThreads applies to Full Load tasks. For information on the settings for parallel load of individual tables, see Table and collection settings rules and operations.

  This setting applies to the following endpoint engine types:

  - DynamoDB
  - Amazon Kinesis Data Streams
  - Amazon MSK

  - Amazon OpenSearch Service
  - Amazon Redshift

  AWS DMS supports `ParallelLoadThreads` for MySQL as an extra connection attribute. `ParallelLoadThreads` does not apply to MySQL as a task setting.

- `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 OpenSearch 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 and collection settings rules and operations.
- `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 an Amazon DocumentDB, Kinesis, Amazon MSK, OpenSearch, or Amazon Redshift target endpoint. The default is zero (0).

  This setting only applies for CDC-only. This setting does not apply for Full Load.

  This setting applies to the following endpoint engine types:

  - Amazon DocumentDB (with MongoDB compatibility)
  - Amazon Kinesis Data Streams
  - Amazon Managed Streaming for Apache Kafka
  - Amazon OpenSearch Service
  - Amazon Redshift
- `ParallelApplyBufferSize` – Specifies the maximum number of records to store in each buffer queue for concurrent threads to push to an Amazon DocumentDB, Kinesis, Amazon MSK, OpenSearch, or Amazon Redshift target endpoint during a CDC load. The default value is 100. The maximum value is 1000. 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 an Amazon DocumentDB, Kinesis, Amazon MSK, or OpenSearch endpoint during CDC. The default value is 1.

Full-load task settings

Full-load settings include the following. For information about how to use a task configuration file to set task settings, see Task settings example.

- 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.
- Set the `ParallelLoadThreads` option to indicate how many concurrent threads DMS will employ during a full-load process to push data records to a target endpoint. Zero is the default value (0).

  Important

  `MaxFullLoadSubTasks` controls the number of tables or table segments to load in parallel. `ParallelLoadThreads` controls the number of threads that are used by a migration task to execute the loads in parallel. *These settings are multiplicative*. As such, the total number of threads that are used during a full load task is approximately the result of the value of `ParallelLoadThreads` multiplied by the value of `MaxFullLoadSubTasks` (`ParallelLoadThreads` * `MaxFullLoadSubtasks`). If you create tasks with a high number of Full Load sub tasks and a high number of parallel load threads, your task can consume too much memory and fail.

- 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 records that can be transferred together, set the `CommitRate` option. The default value is 10000, and the maximum value is 50000.

Time Travel task settings

To log and debug replication tasks, you can use AWS DMS Time Travel. In this approach, you use Amazon S3 to store logs and encrypt them using your encryption keys. Only with access to your Time Travel S3 bucket, can you retrieve your S3 logs using date-time filters, then view, download, and obfuscate logs as needed. By doing this, you can securely "travel back in time" to investigate database activities. Time Travel works independently from the CloudWatch logging. For more information on CloudWatch logging, see Logging task settings.

You can use Time Travel in all AWS Regions with AWS DMS-supported Oracle, Microsoft SQL Server, and PostgreSQL source endpoints, and AWS DMS-supported PostgreSQL and MySQL target endpoints. You can turn on Time Travel only for full-load and change data capture (CDC) tasks and for CDC-only tasks. To turn on Time Travel or to modify any existing Time Travel settings, ensure that your replication task is stopped.

The Time Travel settings include the `TTSettings` properties following:

- `EnableTT` – If this option is set to `true`, Time Travel logging is turned on for the task. The default value is `false`.

  Type: Boolean

  Required: No
- `EncryptionMode` – The type of server-side encryption being used on your S3 bucket to store your data and logs. You can specify either `"SSE_S3"` (the default) or `"SSE_KMS"`.

You can change `EncryptionMode` from `"SSE_KMS"` to `"SSE_S3"`, but not the reverse.

Type: String

Required: No

- `ServerSideEncryptionKmsKeyId` – If you specify `"SSE_KMS"` for `EncryptionMode`, provide the ID for your custom managed AWS KMS key. Make sure that the key that you use has an attached policy that turns on AWS Identity and Access Management (IAM) user permissions and allows use of the key.

  Only your own custom-managed symmetric KMS key is supported with the `"SSE_KMS"` option.

  Type: String

  Required: Only if you set `EncryptionMode` to `"SSE_KMS"`
- `ServiceAccessRoleArn` – The Amazon Resource Name (ARN) used by the service to access the IAM role. Set the role name to `dms-tt-s3-access-role`. This is a required setting that allows AWS DMS to write and read objects from an S3 bucket.

  Type: String

  Required: If Time Travel is turned on

  Following is an example policy for this role.

  JSON

```
{
 "Version":"2012-10-17",
 "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "kms:GenerateDataKey",
                "kms:Decrypt",
                "s3:ListBucket",
                "s3:DeleteObject"
            ],
            "Resource": [
```

```
                "arn:aws:s3:::S3bucketName*",
                "arn:aws:kms:us-east-1:112233445566:key/1234a1a1-1m2m-1z2z-
d1d2-12dmstt1234"
              ]
        }
    ]
}
```

Following is an example trust policy for this role.

JSON

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

- `BucketName` – The name of the S3 bucket to store Time Travel logs. Make sure to create this S3 bucket before turning on Time Travel logs.

  Type: String

  Required: If Time Travel is turned on

- `BucketFolder` – An optional parameter to set a folder name in the S3 bucket. If you specify this parameter, DMS creates the Time Travel logs in the path `"/`*`BucketName`*`/`*`BucketFolder`*`/`*`taskARN`*`/`*`YYYY`*`/`*`MM`*`/`*`DD`*`/`*`hh`*`"`. If you don't specify this parameter, AWS DMS creates the default path as `"/`*`BucketName`*`/dms-time-travel-logs/`*`taskARN`*`/`*`YYYY`*`/`*`MM`*`/`*`DD`*`/`*`hh`*.

  Type: String

  Required: No

- `EnableDeletingFromS3OnTaskDelete` – When this option is set to `true`, AWS DMS deletes the Time Travel logs from S3 if the task is deleted. The default value is `false`.

  Type: String

  Required: No

- `EnableRawData` – When this option is set to `true`, the data manipulation language (DML) raw data for Time Travel logs appears under the `raw_data` column of the Time Travel logs. For the details, see Using the Time Travel logs. The default value is `false`. When this option is set to `false`, only the type of DML is captured.

  Type: String

  Required: No

- `RawDataFormat` – In AWS DMS versions 3.5.0 and higher, when `EnableRawData` is set to `true`. This property specifies a format for the raw data of the DML in a Time Travel log and can be presented as:

  - `"TEXT"` – Parsed, readable column names and values for DML events captured during CDC as `Raw` fields.
  - `"HEX"` – The original hexidecimal for column names and values captured for DML events during CDC.

  This property applies to Oracle and Microsoft SQL Server database sources.

  Type: String

  Required: No

- `OperationsToLog` – Specifies the type of DML operations to log in Time Travel logs. You can specify one of the following:

  - `"INSERT"`
  - `"UPDATE"`
  - `"DELETE"`
  - `"COMMIT"`
  - `"ROLLBACK"`
  - `"ALL"`

  The default is `"ALL"`.

Type: String

Required: No

- `MaxRecordSize` – Specifies the maximum size of Time Travel log records that are logged for each row. Use this property to control the growth of Time Travel logs for especially busy tables. The default is 64 KB.

  Type: Integer

  Required: No

For more information on turning on and using Time Travel logs, see the following topics.

Topics

- Turning on the Time Travel logs for a task
- Using the Time Travel logs
- How often AWS DMS uploads Time Travel logs to S3

Turning on the Time Travel logs for a task

You can turn on Time Travel for an AWS DMS task using the task settings described previously. Make sure that your replication task is stopped before you turn on Time Travel.

To turn on Time Travel using the AWS CLI

1. Create a DMS task configuration JSON file and add a `TTSettings` section such as the following. For information about how to use a task configuration file to set task settings, see Task settings example.

```
 .
 .
 .
    },
"TTSettings" : {
  "EnableTT" : true,
  "TTS3Settings": {
      "EncryptionMode": "SSE_KMS",
```

```
      "ServerSideEncryptionKmsKeyId": "arn:aws:kms:us-west-2:112233445566:key/
myKMSKey",
      "ServiceAccessRoleArn": "arn:aws:iam::112233445566:role/dms-tt-s3-access-
role",
      "BucketName": "myttbucket",
      "BucketFolder": "myttfolder",
      "EnableDeletingFromS3OnTaskDelete": false
    },
  "TTRecordSettings": {
      "EnableRawData" : true,
      "OperationsToLog": "DELETE,UPDATE",
      "MaxRecordSize": 64
    },
 .
 .
 .
```

2. In an appropriate task action, specify this JSON file using the `--replication-task-settings` option. For example, the CLI code fragment following specifies this Time Travel settings file as part of `create-replication-task`.

```
aws dms create-replication-task
--target-endpoint-arn arn:aws:dms:us-
east-1:112233445566:endpoint:ELS5O7YTYV452CAZR2EYBNQGILFHQIFVPWFRQAY \
--source-endpoint-arn arn:aws:dms:us-
east-1:112233445566:endpoint:HNX2BWIIN5ZYFF7F6UFFZVWTDFFSMTNOV2FTXZA \
--replication-instance-arn arn:aws:dms:us-
east-1:112233445566:rep:ERLHG2UA52EEJJKFYNYWRPCG6T7EPUAB5AWBUJQ \
--migration-type full-load-and-cdc --table-mappings 'file:///FilePath/
mappings.json' \
--replication-task-settings 'file:///FilePath/task-settings-tt-enabled.json' \
--replication-task-identifier test-task
                            .
                            .
                            .
```

Here, the name of this Time Travel settings file is `task-settings-tt-enabled.json`.

Similarly, you can specify this file as part of the `modify-replication-task` action.

Note the special handling of Time Travel logs for the task actions following:

- `start-replication-task` – When you run a replication task, if an S3 bucket used for Time Travel isn't accessible, the task is marked as `FAILED`.
- `stop-replication-task` – When the task stops, AWS DMS immediately pushes all Time Travel logs that are currently available for the replication instance to the S3 bucket used for Time Travel.

While a replication task runs, you can change the `EncryptionMode` value from `"SSE_KMS"` to `"SSE_S3"` but not the reverse.

If the size of Time Travel logs for an ongoing task exceeds 1 GB, DMS pushes the logs to S3 within five minutes of reaching that size. After a task is running, if the S3 bucket or KMS key becomes inaccessible, DMS stops pushing logs to this bucket. If you find your logs aren't being pushed to your S3 bucket, check your S3 and AWS KMS permissions. For more details on how often DMS pushes these logs to S3, see How often AWS DMS uploads Time Travel logs to S3.

To turn on Time Travel for an existing task from the console, use the JSON editor option under **Task Settings** to add a `TTSettings` section.

Using the Time Travel logs

Time Travel log files are comma-separated value (CSV) files with the fields following.

```
log_timestamp
component
dms_source_code_location
transaction_id
event_id
event_timestamp
lsn/scn
primary_key
record_type
event_type
schema_name
table_name
statement
action
result
raw_data
```

After your Time Travel logs are available in S3, you can directly access and query them with tools such as Amazon Athena. Or you can download the logs as you can any file from S3.

The example following shows a Time Travel log where transactions for a table called `mytable` are logged. The line endings for the following log are added for readability.

```
"log_timestamp ","tt_record_type","dms_source_code_location ","transaction_id",
"event_id","event_timestamp","scn_lsn","primary_key","record_type","event_type",
"schema_name","table_name","statement","action","result","raw_data"
"2021-09-23T01:03:00:778230","SOURCE_CAPTURE","postgres_endpoint_wal_engine.c:00819",
"609284109","565612992","2021-09-23 01:03:00.765321+00","00000E9C/D53AB518","","DML",
"UPDATE (3)","dmstest","mytable","","Migrate","","table dmstest.mytable:
UPDATE: id[bigint]:2244937 phone_number[character varying]:'phone-number-482'
age[integer]:82 gender[character]:'f' isactive[character]:'true '
date_of_travel[timestamp without time zone]:'2021-09-23 01:03:00.76593'
description[text]:'TEST DATA TEST DATA TEST DATA TEST DATA'"
```

How often AWS DMS uploads Time Travel logs to S3

To minimize the storage usage of your replication instance, AWS DMS offloads Time Travel logs from it periodically.

The Time travel logs get pushed to your Amazon S3 bucket in the cases following:

- If the current size of logs exceeds 1 GB, AWS DMS uploads the logs to S3 within five minutes. Thus, AWS DMS can make up to 12 calls an hour to S3 and AWS KMS for each running task.
- AWS DMS uploads the logs to S3 every hour, regardless of the size of the logs.
- When a task is stopped, AWS DMS immediately uploads the time travel logs to S3.

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. For information about how to use a task configuration file to set task settings, see Task settings example.

You can turn on 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.

When you set `EnableLogging` to `true`, AWS DMS assigns the CloudWatch group name and stream name as follows. You can't set these values directly.

- **CloudWatchLogGroup**: `dms-tasks-<REPLICATION_INSTANCE_IDENTIFIER>`
- **CloudWatchLogStream**: `dms-task-<REPLICATION_TASK_EXTERNAL_RESOURCE_ID>`

`<REPLICATION_INSTANCE_IDENTIFIER>` is the identifier of the replication instance. `<REPLICATION_TASK_EXTERNAL_RESOURCE_ID>` is the value of the `<resourcename>` section of the Task ARN. For information about how AWS DMS generates resource ARNs, see Constructing an Amazon Resource Name (ARN) for AWS DMS.

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 about 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 types of events:

- `FILE_FACTORY` – The file factory manages files used for batch apply and batch load, and manages Amazon S3 endpoints.
- `METADATA_MANAGER` – The metadata manager manages source and target metadata, partitioning, and table state during replication.
- `SORTER` – The `SORTER` receives incoming events from the `SOURCE_CAPTURE` process. The events are batched in transactions, and passed to the `TARGET_APPLY` service component. If the `SOURCE_CAPTURE` process produces events faster than the `TARGET_APPLY` component can consume them, the `SORTER` component caches the backlogged events to disk or to a swap file. Cached events are a common cause for running out of storage in replication instances.

  The `SORTER` service component manages cached events, gathers CDC statistics, and reports task latency.
- `SOURCE_CAPTURE` – Ongoing replication (CDC) data is captured from the source database or service, and passed to the SORTER service component.
- `SOURCE_UNLOAD` – Data is unloaded from the source database or service during Full Load.

- TABLES_MANAGER — The table manager tracks captured tables, manages the order of table migration, and collects table statistics.
- TARGET_APPLY – Data and data definition language (DDL) statements are applied to the target database.
- TARGET_LOAD – Data is loaded into the target database.
- TASK_MANAGER – The task manager manages running tasks, and breaks tasks down into sub-tasks for parallel data processing.
- TRANSFORMATION – Table-mapping transformation events. For more information, see [Using table mapping to specify task settings](#).
- VALIDATOR/ VALIDATOR_EXT – The VALIDATOR service component verifies that data was migrated accurately from the source to the target. For more information, see [Data validation](#).
- DATA_RESYNC – Common component of Data resync feature that manages data resync flow. For more information, see [the section called "AWS DMS data resync"](#).
- RESYNC_UNLOAD – Data is unloaded from the source database or service during resync process.
- RESYNC_APPLY – Data manipulation language (DML) statements are applied to the target database during resync.

The following logging components generate a large amount of logs when using the LOGGER_SEVERITY_DETAILED_DEBUG log severity level:

- COMMON
- ADDONS
- DATA_STRUCTURE
- COMMUNICATION
- FILE_TRANSFER
- FILE_FACTORY

Logging levels other than DEFAULT are rarely needed for these components during troubleshooting. We do not recommend changing the logging level from DEFAULT for these components unless specifically requested by AWS Support.

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": "FILE_FACTORY",
        "Severity": "LOGGER_SEVERITY_DEFAULT"
      },{
        "Id": "METADATA_MANAGER",
        "Severity": "LOGGER_SEVERITY_DEFAULT"
      },{
        "Id": "SORTER",
        "Severity": "LOGGER_SEVERITY_DEFAULT"
      },{
        "Id": "SOURCE_CAPTURE",
        "Severity": "LOGGER_SEVERITY_DEFAULT"
      },{
        "Id": "SOURCE_UNLOAD",
        "Severity": "LOGGER_SEVERITY_DEFAULT"
      },{
        "Id": "TABLES_MANAGER",
        "Severity": "LOGGER_SEVERITY_DEFAULT"
      },{
        "Id": "TARGET_APPLY",
        "Severity": "LOGGER_SEVERITY_DEFAULT"
```

```
      },{
        "Id": "TARGET_LOAD",
        "Severity": "LOGGER_SEVERITY_INFO"
      },{
        "Id": "TASK_MANAGER",
        "Severity": "LOGGER_SEVERITY_DEBUG"
      },{
        "Id": "TRANSFORMATION",
        "Severity": "LOGGER_SEVERITY_DEBUG"
      },{
        "Id": "VALIDATOR",
        "Severity": "LOGGER_SEVERITY_DEFAULT"
      }
    ],
    "CloudWatchLogGroup": null,
    "CloudWatchLogStream": null
  },
…
```

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 on database targets. For information about how to use a task configuration file to set task settings, see Task settings example.

AWS DMS only creates control tables only during Full Load + CDC or CDC-only tasks, and not during Full Load Only tasks.

For full load and 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 Resource ID of the AWS DMS task. Resource ID can be found in task ARN.
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 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 Resource ID of the AWS DMS task. Resource ID can be found in task ARN.
TASK_STATUS	varchar	One of the following values:

Column	Type	Description
		• FULL LOAD • CHANGE PROCESSING (CDC) • NOT RUNNING 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 PROCESSIN G if CDC is enabled. The task is set to NOT RUNNING before you start the task, or after the task completes.
STATUS_TIME	timestamp	The timestamp of the task status.
PENDING_CHANGES	int	The number of change records that were committed in the source database and cached in the memory and disk of your replication instance.
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.

Column	Type	Description
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.

The Suspended table (`dmslogs.awsdms_suspended_tables`) 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
TABLE_OWNER	nvchar	The table owner.
TABLE_NAME	nvchar	The table name.
SUSPEND_REASON	nvchar	Reason for suspension.

Column	Type	Description
SUSPEND_T IMESTAMP	timestamp	The time the suspension occurred.

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 Resource ID of the AWS DMS task. Resource ID can be found in task ARN.
TIMESLOT_TYPE	varchar	One of the following values: • FULL LOAD • CHANGE PROCESSING (CDC) 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.

Column	Type	Description
RECORDS	int	The number of records processed during the time slot.
TIMESLOT_VOLUME	int	The volume of data processed in MB.

The Validation Failure table (`awsdms_validation_failures_v1`) contains all the data validation failures for a task. For more information see, Data Validation Troubleshooting.

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
  - IBM Db2 LUW, awsdms_control

Stream buffer task settings

You can set stream buffer settings using the AWS CLI, including the following. For information about how to use a task configuration file to set task settings, see Task settings example.

- `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: `[Max LOB size (or LOB chunk size)]*[number of LOB columns]*[number of stream buffers]*[number of tables loading in parallel per task(MaxFullLoadSubTasks)]*3`
- `CtrlStreamBufferSizeInMB` – Use this option to set the size of the control stream buffer. 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. For information about how to use a task configuration file to set task settings, see Task settings example.

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.

  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 turn these constraints off during the migration process and turn them on again as part of the cutover process.

- `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.

For bidirectional replication, the following setting applies only when the target metadata parameter `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 about bidirectional replication, see Performing bidirectional replication.

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, and the maximum value is 200.
- `RecoveryTimeout`– When resuming a task in CDC mode, RecoveryTimeout controls how long (in minutes) the task will wait to encounter the resume checkpoint. If the checkpoint is not encountered within the configured timeframe, the task will fail. The default behavior is to wait indefinitely for the checkpoint event.

Example of how task settings that handle Change Processing Tuning appear in a task setting JSON file:

```
"ChangeProcessingTuning": {
        "BatchApplyPreserveTransaction": true,
        "BatchApplyTimeoutMin": 1,
        "BatchApplyTimeoutMax": 30,
        "BatchApplyMemoryLimit": 500,
        "BatchSplitSize": 0,
        "MinTransactionSize": 1000,
        "CommitTimeout": 1,
        "MemoryLimitTotal": 1024,
        "MemoryKeepTime": 60,
        "StatementCacheSize": 50
        "RecoveryTimeout": -1
}
```

To control the frequency of writes to an Amazon S3 target during a data replication task, you can configure the `cdcMaxBatchInterval` and `cdcMinFileSize` extra connection attributes. This can result in better performance when analyzing the data without any additional overhead operations. For more information, see [Endpoint settings when using Amazon S3 as a target for AWS DMS](#).

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. For information about how to use a task configuration file to set task settings, see Task settings example.

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.
- `ValidationMode` – controls how AWS DMS validates data in the target table against the source table. Starting with replication engine version 3.5.4, DMS automatically sets this to `GROUP_LEVEL` for supported migration paths, delivering enhanced validation performance and significantly faster processing for large datasets. This enhancement applies to migrations for the migration paths listed in AWS DMS data resync. For all other migration paths, the validation mode defaults to `ROW_LEVEL`.

  Note

  Irrespective of the setting, AWS DMS validates all rows configured via table validation.

- `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 between source and target databases. Otherwise, any such differences in column collation are ignored for validation. 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`.
- `RecordFailureDelayInMinutes` – Specifies the delay time in minutes before reporting any validation failure details.

If overall DMS Task CDC latency is greater than the value of `RecordFailureDelayInMinutesthen` it takes the precedence, for example, if `RecordFailureDelayInMinutes` is 5, and CDC Latency is 7 mins then DMS waits 7 minutes to report the validation failure details.

- `RecordFailureDelayLimitInMinutes` – Specifies the delay before reporting any validation failure details. 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 and DMS task CDC Latency value and enables you to set a greater delay before reporting any validation metrics. The default value is 0.
- `RecordSuspendDelayInMinutes` – Specifies the delay time in minutes before tables are suspended from validation due to error threshold set in `FailureMaxCount`.
- `SkipLobColumns` – When this option is set to `true`, AWS DMS skips data validation for all the LOB columns in the table's part of the task validation. The default value is `false`.
- `TableFailureMaxCount` – Specifies the maximum number of rows in one table that can fail validation before validation is suspended for the table. The default value is 1,000.
- `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 performs data validation without performing any migration or replication of data. The default value is `false`. You can't modify the `ValidationOnly` setting after the task is created.

  You must set **TargetTablePrepMode** to `DO_NOTHING` (the default for a validation only task) and set **Migration Type** to one of the following:

  - Full Load — Set the task **Migration type** to **Migrate existing data** in the AWS DMS console. Or, in the AWS DMS API set the migration type to FULL-LOAD.
  - CDC — Set the task **Migration type** to **Replicate data changes only** in the AWS DMS console. Or, in the AWS DMS API set the migration type to CDC.

  Regardless of the migration type chosen, data isn't actually migrated or replicated during a validation only task.

  For more information, see [Validation only tasks](#).

> **Important**
>
> The `ValidationOnly` setting is immutable. It can't be modified for a task after that task is created.

- `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. The 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.
- `PartitionSize` – Specifies the batch size of records to read for comparison from both source and target. The default is 10,000.
- `ValidationQueryCdcDelaySeconds` – The amount of time the first validation query is delayed on both source and target for each CDC update. This might help reduce resource contention when migration latency is high. A validation only task automatically sets this option to 180 seconds. The default is 0.

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 accesses the Oracle endpoint. To do this, run the following statement.

```
grant execute on sys.dbms_crypto to dms_endpoint_user;
```

Data resync settings

The Data resync feature allows you to resync target database with your source database based on data validation report. For more information, see AWS DMS data validation.

You can add additional parameters for `ResyncSettings` in the `ReplicationTaskSettings` endpoint that configures the resync process. For more information, see Task settings example in the Specifying task settings for AWS Database Migration Service tasks.

> **Note**
>
> `ResyncSchedule` and `MaxResyncTime` parameters are required if the resync process is enabled and the task has a CDC component. They are not valid for full-load only tasks.

The Data resync parameter settings and values are as follows:

`EnableResync`

Enables Data resync feature when set to `true`. By default, Data resync is disabled.

Datatype: Boolean

Required: No

Default: `false`

Validation: Should not be null if `ResyncSettings` parameter is present in `TaskSettings`.

`ResyncSchedule`

Time window for the Data resync feature to be in effect. Must be present in Cron format. For more information, see the section called "Cron expression rules".

Datatype: String

Required: No

Validation:

- Must be present in Cron expression format.
- Should not be null for tasks with CDC that has `EnableResync` set to `true`.
- Cannot be set for tasks without CDC component.

`MaxResyncTime`

Maximum time limit in minutes for the Data resync feature to be in effect.

Datatype: Integer

Required: No

Validation:

- Should not be null for tasks with CDC.
- Not required for tasks without CDC.
- Minimum value: `5 minutes`, Maximum value: `14400 minutes` (10 days).

`Validation onlyTaskID`

Unique ID of the validation task. The validation only task ID is appended at the end of an ARN. For example:

- Validation only task ARN: `arn:aws:dms:us-west-2:123456789012:task:6DG4CLGJ5JSJR67CFD7UDXFY7KV6CYGRICL6KWI`
- Validation only task ID: `6DG4CLGJ5JSJR67CFD7UDXFY7KV6CYGRICL6KWI`

Datatype: String

Required: No

Validation: Should not be null for tasks with Data resync feature enabled and validation disabled.

Example:

```
"ResyncSettings": {
    "EnableResync": true,
    "ResyncSchedule": "30 9 ? * MON-FRI",
    "MaxResyncTime": 400,
    "ValidationTaskId": "JXPP94804DJOEWIJD9348R3049"
},
```

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). For information about how to use a task configuration file to set task settings, see Task settings example.

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. For information about how to use a task configuration file to set task settings, see Task settings example.

You can configure character substitution for any task with endpoints from the following source and target databases:

- Source databases:
  - Oracle
  - Microsoft SQL Server
  - MySQL
  - MariaDB
  - PostgreSQL
  - SAP Adaptive Server Enterprise (ASE)
  - IBM Db2 LUW
- Target databases:
  - Oracle
  - Microsoft SQL Server
  - MySQL
  - MariaDB
  - 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 about table mapping, see [Using table mapping to specify task settings](). Character substitution doesn't support LOB data types. This includes any datatype that DMS considers to be a LOB data type. For example, the Extended datatype in Oracle is considered to be a LOB. For more information about source datatypes, see [Source data types for Oracle]() following.

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_U2	macos-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=0	ibm-1390_P110-2003
ibm-1383_P110-1999	ISO_2022,locale=ja,version=1	ibm-1399_P110-2003
ibm-5478_P100-1995	ISO_2022,locale=ja,version=2	ibm-5123_P100-1999
euc-tw-2014	ISO_2022,locale=ja,version=3	ibm-8482_P100-1999
ibm-964_P110-1999	ISO_2022,locale=ja,version=4	ibm-16684_P110-2003
ibm-949_P110-1999	ISO_2022,locale=ko,version=0	ibm-4899_P100-1998
ibm-949_P11A-1999	ISO_2022,locale=ko,version=1	ibm-4971_P100-1999
ibm-970_P110_P110-2006_U2	ISO_2022,locale=zh,version=0	ibm-9067_X100-2005
ibm-971_P100-1995	ISO_2022,locale=zh,version=1	ibm-12712_P100-1998
ibm-1363_P11B-1998	ISO_2022,locale=zh,version=2	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 Apache 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. For information about how to use a task configuration file to set task settings, see Task settings example.

To do so, you use the `BeforeImageSettings` parameter, which adds a new JSON attribute to every update operation with values collected from the source database system.

Make sure to apply `BeforeImageSettings` only to full load plus CDC tasks or CDC only tasks. Full load plus CDC tasks migrate existing data and replicate ongoing changes. CDC only tasks replicate data changes only.

Don't apply `BeforeImageSettings` to tasks that are full load only.

Possible options for `BeforeImageSettings` are the following:

- `EnableBeforeImage` – Turns on 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 any column that has a before image value, use `all`. Note that the before image doesn't support large binary object (LOB) data types such as CLOB and BLOB.

The following shows an example of the use of `BeforeImageSettings`.

```
"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.

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.

Error handling task settings

You can set the error handling behavior of your replication task using the following settings. For information about how to use a task configuration file to set task settings, see Task settings example.

- `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 `DataErrorEscalationCount` 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.
- EventErrorPolicy – Determines the action AWS DMS takes when an error occurs while sending a task-related event. Its possible values are
  - IGNORE – The task continues and any data associated with that event is ignored.
  - STOP_TASK – The task stops and manual intervention is required.
- 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.

  When you set this value to -1, the number of retries that DMS attempts varies based on the returned error type as follows:

  - **Running state, recoverable error**: If a recoverable error such as a lost connection or a target apply fail occurs, DMS retries the task nine times.
  - **Starting state, recoverable error**: DMS retries the task six times.
  - **Running state, fatal error handled by DMS**: DMS retries the task six times.

  - **Running state, fatal error not handled by DMS**: DMS does not retry the task.
  - **Other than above**: AWS DMS retries the task indefinitely.

  Set this value to 0 to never attempt to restart a task.

  We recommend that you set `RecoverableErrorCount` and `RecoverableErrorInterval` to values such that there are sufficient retries at sufficient intervals for your DMS task to recover properly. If a fatal error occurs, DMS stops making restart attempts in most scenarios.

- `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 in a series based on the value of `RecoverableErrorInterval`. For example, if `RecoverableErrorInterval` is set to 5 seconds, then the next retry will happen after 10 seconds, then 20, then 40 seconds and so on. 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.
- `RecoverableErrorStopRetryAfterThrottlingMax`– Default value is set to `true`, and DMS stops resuming the task after the maximum number of seconds that AWS DMS waits between recovery attempts is reached, per `RecoverableErrorStopRetryAfterThrottlingMax`. When set to `false`, DMS keeps resuming the task after the maximum number of seconds that AWS DMS waits between recovery attempts is reached, per `RecoverableErrorStopRetryAfterThrottlingMax` until `RecoverableErrorCount` is reached.
- `ApplyErrorDeletePolicy` – Determines what action AWS DMS takes when there is a conflict with a DELETE operation. The default is `IGNORE_RECORD`. Possible values are the following:
  - `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`. Possible values are the following:
  - `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.

> **Note**
>
> - **In Transactional Apply mode**: In this process, the system first attempts to insert the record. If the insert fails due to a primary key conflict, it deletes the existing record and then inserts the new one.
> - **In Batch Apply mode**: The process removes all existing records in the target batch before inserting the complete set of new records, ensuring a clean replacement of data.
> - This process prevents data duplication, but incurs some performance cost compared to the default policy. The exact performance impact depends on your specific workload characteristics.

- `ApplyErrorUpdatePolicy` – Determines what action AWS DMS takes when there is a missing data conflict with an UPDATE operation. The default is `LOG_ERROR`. Possible values are the following:
  - `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. AWS DMS completely disables LOB column support for the task. Selecting this option requires full supplemental logging to be enabled for all the source table columns when Oracle is the source database.

    > **Note**
    >
    > - **In Transactional Apply mode**: In this process, the system first attempts to update the record. If the update fails due to a missing record on target, it run a delete for the failed record and then inserts the new one. This process requires full supplemental logging for Oracle source databases and DMS disables LOB column support for this task.
    > - **In Batch Apply mode**: The process removes all existing records in the target batch before inserting the complete set of new records, ensuring a clean replacement of data.

- `ApplyErrorEscalationPolicy` – Determines what action AWS DMS takes when the maximum number of errors (set using the `ApplyErrorEscalationCount` parameter) is reached. The default is LOG_ERROR:
  - `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 version 11.x or lower, 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 table mappings defined for a task find no tables when the task starts. The default is `true`.

- `FailOnTransactionConsistencyBreached` – This option applies to tasks using Oracle as a source with CDC. The default is false. 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`.
- `DataMaskingErrorPolicy` – Determines the action AWS DMS takes when the data masking is failed due to incompatible data type or any other reason. The follwing are available options:
  - `STOP_TASK` (Default) – The task stops and manual intervention is required.
  - `IGNORE_RECORD` – The task continues and the data for that record is ignored.
  - `LOG_ERROR` – The task continues and the error is written to the task log. Unmasked data will be loaded in target 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.

> **Note**
>
> Table load errors in Redshift as a target are reported in `STL_LOAD_ERRORS`. For more information, see STL_LOAD_ERRORS in the *Amazon Redshift Database Developer Guide*.

> **Note**
>
> Parameter changes related to recoverable errors only take effect after you stop and resume the DMS task. Changes do not apply to the current execution.

Saving task settings

You can save task settings as a JSON file in case you want to reuse the settings for another task. You can find tasks settings to copy to a JSON file under the **Overview details** section of a task.

> **Note**
>
> While reusing task settings for other tasks, remove any `CloudWatchLogGroup` and `CloudWatchLogStream` attributes. Otherwise, the following error is given: SYSTEM ERROR MESSAGE:Task Settings CloudWatchLogGroup or CloudWatchLogStream cannot be set on create.

For example, the following JSON file contains settings saved for a task.

```
{
    "TargetMetadata": {
        "TargetSchema": "",
        "SupportLobs": true,
        "FullLobMode": false,
        "LobChunkSize": 0,
        "LimitedSizeLobMode": true,
        "LobMaxSize": 32,
        "InlineLobMaxSize": 0,
        "LoadMaxFileSize": 0,
        "ParallelLoadThreads": 0,
        "ParallelLoadBufferSize": 0,
        "BatchApplyEnabled": false,
        "TaskRecoveryTableEnabled": false,
        "ParallelLoadQueuesPerThread": 0,
        "ParallelApplyThreads": 0,
        "ParallelApplyBufferSize": 0,
        "ParallelApplyQueuesPerThread": 0
    },
    "FullLoadSettings": {
        "TargetTablePrepMode": "DO_NOTHING",
        "CreatePkAfterFullLoad": false,
        "StopTaskCachedChangesApplied": false,
        "StopTaskCachedChangesNotApplied": false,
        "MaxFullLoadSubTasks": 8,
        "TransactionConsistencyTimeout": 600,
        "CommitRate": 10000
    },
    "Logging": {
        "EnableLogging": true,
        "LogComponents": [
```

```
            {
                "Id": "TRANSFORMATION",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "SOURCE_UNLOAD",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "IO",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "TARGET_LOAD",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "PERFORMANCE",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "SOURCE_CAPTURE",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "SORTER",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "REST_SERVER",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "VALIDATOR_EXT",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "TARGET_APPLY",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "TASK_MANAGER",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
```

```
            {
                "Id": "TABLES_MANAGER",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "METADATA_MANAGER",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "FILE_FACTORY",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "COMMON",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "ADDONS",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "DATA_STRUCTURE",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "COMMUNICATION",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            },
            {
                "Id": "FILE_TRANSFER",
                "Severity": "LOGGER_SEVERITY_DEFAULT"
            }
        ]
    },
    "ControlTablesSettings": {
        "ControlSchema": "",
        "HistoryTimeslotInMinutes": 5,
        "HistoryTableEnabled": false,
        "SuspendedTablesTableEnabled": false,
        "StatusTableEnabled": false,
        "FullLoadExceptionTableEnabled": false
    },
    "StreamBufferSettings": {
        "StreamBufferCount": 3,
```

```
        "StreamBufferSizeInMB": 8,
        "CtrlStreamBufferSizeInMB": 5
    },
    "ChangeProcessingDdlHandlingPolicy": {
        "HandleSourceTableDropped": true,
        "HandleSourceTableTruncated": true,
        "HandleSourceTableAltered": true
    },
    "ErrorBehavior": {
        "DataErrorPolicy": "LOG_ERROR",
        "DataTruncationErrorPolicy": "LOG_ERROR",
        "DataErrorEscalationPolicy": "SUSPEND_TABLE",
        "DataErrorEscalationCount": 0,
        "TableErrorPolicy": "SUSPEND_TABLE",
        "TableErrorEscalationPolicy": "STOP_TASK",
        "TableErrorEscalationCount": 0,
        "RecoverableErrorCount": -1,
        "RecoverableErrorInterval": 5,
        "RecoverableErrorThrottling": true,
        "RecoverableErrorThrottlingMax": 1800,
        "RecoverableErrorStopRetryAfterThrottlingMax": true,
        "ApplyErrorDeletePolicy": "IGNORE_RECORD",
        "ApplyErrorInsertPolicy": "LOG_ERROR",
        "ApplyErrorUpdatePolicy": "LOG_ERROR",
        "ApplyErrorEscalationPolicy": "LOG_ERROR",
        "ApplyErrorEscalationCount": 0,
        "ApplyErrorFailOnTruncationDdl": false,
        "FullLoadIgnoreConflicts": true,
        "FailOnTransactionConsistencyBreached": false,
        "FailOnNoTablesCaptured": true
    },
    "ChangeProcessingTuning": {
        "BatchApplyPreserveTransaction": true,
        "BatchApplyTimeoutMin": 1,
        "BatchApplyTimeoutMax": 30,
        "BatchApplyMemoryLimit": 500,
        "BatchSplitSize": 0,
        "MinTransactionSize": 1000,
        "CommitTimeout": 1,
        "MemoryLimitTotal": 1024,
        "MemoryKeepTime": 60,
        "StatementCacheSize": 50
    },
    "PostProcessingRules": null,
```

```
    "CharacterSetSettings": null,
    "LoopbackPreventionSettings": null,
    "BeforeImageSettings": null,
    "FailTaskWhenCleanTaskResourceFailed": false
}
```

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 data types 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 LOB size for DMS to accept. That enables 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 can gain significant performance over full LOB mode. We recommend that you use limited LOB mode whenever possible. The maximum value for this parameter is 102400 KB (100 MB).

    > **Note**
    >
    > Using the Max LOB size (K) option with a value greater than 63KB impacts the performance of a full load configured to run in limited LOB mode. During a full load, DMS allocates memory by multiplying the Max LOB size (k) value by the Commit rate, and the product is multiplied by the number of LOB columns. When DMS cannot pre-allocate that memory, it consumes SWAP memory which negatively impacts performance of the full load tasks. If you experience performance issues when

using limited LOB mode, consider decreasing the commit rate until you achieve an acceptable level of performance. During a CDC mode, DMS allocates memory by multiplying the number of LOB Columns by the Max LOB Size parameter specified in the Limited LOB task settings, and then by the record size. DMS CDC process is single threaded per DMS task. For more information, see Change processing tuning settings. To validate limited LOB size, you must set `ValidationPartialLobSize` to the same value as `LobMaxSize` (K).

- **Inline LOB mode** – In inline LOB mode, you set the maximum LOB size that DMS transfers inline. LOBs smaller than the specified size are transferred inline. LOBs larger than the specified size are replicated using full LOB mode. You can select this option to replicate both small and large LOBs when most of the LOBs are small. DMS doesn't support inline LOB mode for endpoints that don't support Full LOB mode, like S3 and Redshift.

> **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 32 K. Therefore, a limited LOB size of less than 32 K is optimal when Oracle is your source database.

- When a task is configured to run in limited LOB mode, the **Max LOB size (K)** option sets the maximum size LOB that AWS DMS accepts. Any LOBs that are larger than this value is truncated to this value.
- When a task is configured to use full LOB mode, AWS DMS retrieves LOBs in pieces. The **LOB chunk size (K)** option determines the size of each piece. When setting this option, pay particular attention to the maximum packet size allowed by your network configuration. If the LOB chunk size exceeds your maximum allowed packet size, you might see disconnect errors. The recommended value for `LobChunkSize` is 64 kilobytes. Increasing the value for `LobChunkSize` above 64 kilobytes can cause task failures.
- When a task is configured to run in inline LOB mode, the `InlineLobMaxSize` setting determines which LOBs DMS transfers inline.

> **Note**
>
> A primary key is mandatory for tables containing LOB columns during Change Data Capture (CDC) operations. DMS uses this key to look up LOB values in the source table.

This requirement only applies to CDC tasks - full-load tasks can read and copy entire LOB columns directly from source to target without restrictions.

For information on the task settings to specify these options, see Target metadata task settings

SQL Commands to check max LOB column length on source table

Use the following SQL commands to check the Max LOB column length and accordingly configure the DMS limited LOB settings to avoid any data truncation in the migration:

Oracle

```
SELECT dbms_lob.getlength(<COL_NAME>) as LOB_LENGTH
FROM <TABLE_NAME>
ORDER BY dbms_lob.getlength(<COL_NAME>) DESC
FETCH FIRST 10 ROWS ONLY;

Select ((max(length(<COL_NAME>)))/(1024)) from <TABLE_NAME>
```

SQL Server

```
Select top 10 datalength(<COL_NAME>) as fieldsize from <TABLE_NAME> order by
 datalength(<COL_NAME>) desc;
```

MySQL

```
Select (max(length(<COL_NAME>))/(1024)) as "Size in KB" from <TABLE_NAME>;
```

PostgreSQL

```
Select max((octet_length(<COL_NAME>))/(1024.0)) as "Size in KB" from <TABLE_NAME>;
```

Db2 LUW

```
-- Method 1: Using SYSCAT.COLUMNS (converting to KB)
```

```
SELECT TABSCHEMA, TABNAME, COLNAME, LENGTH/1024 as LENGTH_KB, TYPENAME FROM
 SYSCAT.COLUMNS WHERE TYPENAME IN ('BLOB', 'CLOB', 'DBCLOB') ORDER BY LENGTH DESC;

-- Method 2: For specific table with KB conversion

  SELECT COLNAME, LENGTH/1024 as LENGTH_KB, TYPENAME FROM SYSCAT.COLUMNS WHERE
 TABSCHEMA = 'YOUR_SCHEMA'AND TABNAME = 'YOUR_TABLE'AND TYPENAME IN ('BLOB', 'CLOB',
 'DBCLOB');

-- Method 3: Using SYSIBM.SYSCOLUMNS

SELECT TBCREATOR, TBNAME, NAME, LENGTH/1024 as LENGTH_KB, COLTYPE FROM
 SYSIBM.SYSCOLUMNS WHERE COLTYPE IN ('BLOB', 'CLOB', 'DBCLOB') ORDER BY LENGTH DESC;

SYBASE :

SELECT c.name as column_name, t.name as data_type, (c.length)/1024 as length_KB FROM
 syscolumns c JOIN systypes t ON c.usertype = t.usertype WHERE object_name(c.id) =
 'YOUR_TABLE_NAME'AND t.name IN ('text', 'image', 'unitext') ORDER BY c.length DESC;
```

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 each individual source transaction is wholly contained within a single task.

You can use multiple tasks to perform a migration if no individual transaction is split across different tasks.

Creating tasks for ongoing replication using AWS DMS

You can create an AWS DMS task that captures ongoing changes from 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.

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. For specific settings for each endpoint, see the following:
  - **Amazon RDS for Oracle:** Configuring an AWS-managed Oracle source for AWS DMS.
  - **Amazon RDS for MySQL and Aurora MySQL:** Using an AWS-managed MySQL-compatible database as a source for AWS DMS.
  - **Amazon RDS for SQL Server:** Setting up ongoing replication on a cloud SQL Server DB instance.
  - **Amazon RDS for PostgreSQL and Aurora PostgreSQL:** PostgreSQL automatically keeps the required WAL.

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**
>
> Only by setting the `StartFromContext` extra connection attribute to the required timestamp does Db2 as a source offer a customized CDC start time.
> 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
  - MariaDB

> **Note**
>
> The following database endpoints do not support CDC native start point functionality:
>
> - Amazon Aurora MySQL
> - Amazon Aurora PostgreSQL
> - Amazon DocumentDB (with MongoDB compatibility)
> - Amazon S3
> - IBM Db2 for z/OS
> - IBM Db2 LUW
> - Microsoft Azure SQL Database
> - Microsoft Azure SQL Managed Instance
> - MongoDB
> - SAP Sybase ASE

When the task is created, AWS DMS marks the CDC start point, and it can't be changed. To use a different CDC start point, create a new task.

Note

When specifying a CDC (Change Data Capture) start point, the premigration assessment will still perform a complete analysis of existing metadata and parameters in the current environment. This ensures that all current configurations and settings are evaluated, regardless of the designated CDC starting point.
Important: If no assessments have been performed within the last 7 days for a given task, the premigration assessment will automatically execute in both resume and reload modes to ensure data completeness and consistency.

Technical Details:

- Metadata analysis remains comprehensive.
- Parameter evaluation covers current state.
- CDC start point does not limit assessment scope.
- Full system configuration review is maintained.
- Auto-execution in resume and reload modes if:

  Last assessment > 7 days ago.

  No previous assessment records found.

This helps ensure accurate migration planning while maintaining data consistency between the specified CDC point and current system state.

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 already been applied to the target. You can look up the native start point for the ongoing replication-only task. To avoid any data inconsistencies, carefully choose the start point for the replication-only task. DMS captures transactions that started after the chosen CDC start point.

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. 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 about checkpoints in general, see Using a checkpoint as a CDC start point.

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 IAM user, make sure that you have the appropriate permissions to access AWS DMS. For more information about the permissions required, see IAM permissions needed to use AWS DMS.
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 Enabling CDC with an AWS-managed PostgreSQL DB instance with AWS DMS.

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, run the following command.

```
SELECT CURRENT_SCN FROM V$DATABASE
```

If you use the SCN or timestamp to start a CDC task, you miss the results of any open transactions and fail to migrate these results. *Open transactions* are transactions that were started before the start position of the task and committed after task start position. 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. With version 3.5.1 and higher, AWS DMS supports open transactions for a CDC-only task using the `openTransactionWindow` endpoint setting if you use the SCN or Timestamp to start the task.

When using the `openTransactionWindow` setting, you must provide the window, in number of minutes, to handle the open transactions. AWS DMS shifts the capture position and finds the new position to start the data capture. AWS DMS uses the new start position for scanning any open transactions from the required Oracle redo or archived redo logs.

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 higher 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 three ways:

- Run the API operation `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. This information is lost if the task is deleted.
- 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`

- From the navigation pane, choose **Database migration tasks**, and choose your parent task from the list that appears on the Database migration tasks page. Your parent task page opens, showing the overview details. Find the checkpoint value under Change data capture (CDC), Change data capture (CDC) start position, and Change data capture (CDC) recovery checkpoint. The checkpoint value appears similar to the following:

  `checkpoint:V1#1#000004AF/B00000D0#0#0#*#0#0`

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.

> **Note**
>
> It can take up to 40 minutes to initialize all the resources the first time you start a new AWS DMS Serverless replication. Note that the `server_time` option is only applicable once the resource initialization has completed.

Starting a task with target reload

You can start an AWS DMS migrate existing data and replicate ongoing changes task with the reload option (“reload-target” in DMS API). In this case the migration will start from the beginning, reload each tables data and continue data replication using full-load and CDC-enabled tasks settings.

To use task start with reload option, the following conditions must apply:

- The task must be stopped.
- The migration method for the task must be either full load or full load with CDC.

DMS applies the TargetTablePrepMode setting before reloading the tables. If you set `TargetTablePrepMode` to `DO_NOTHING`, you must manually truncate the tables first.

> **Note**
>
> When a DMS task is started with the target reload option, the premigration assessment will perform a complete analysis of existing metadata and parameters in the current environment. This ensures that all current configurations and settings are evaluated, regardless of the actual task status.

> **Important**
>
> If no assessments have been performed within the last 7 days for a given task, the premigration assessment will automatically execute to ensure data completeness and consistency.

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 supports bidirectional replication on these database engines:

- Oracle
- SQL Server
- MySQL
- PostgreSQL
- Amazon Aurora MySQL-Compatible Edition
- Aurora PostgreSQL-Compatible Edition

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.

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.
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.
- `setUpMsCdcForTables` must be set to `true` for SQL Server source endpoint to set bidirectional replication.

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 console by selecting the task and choosing **Modify**. You can also use the CLI command or API operation ModifyReplicationTask.

There are a few limitations to modifying a task. These include the following:

- 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 run must have a status of **Stopped** or **Failed** to be modified.

Moving a task

You can move a task to a different replication instance when any of the following situations apply to your use case.

- You're currently using an instance of a certain type and you want to switch to a different instance type.
- Your current instance is overloaded by many replication tasks, and you want to split the load across multiple instances.
- Your instance storage is full, and you want to move tasks off that instance to a more powerful instance as an alternative to scaling storage or compute.
- You want to use a newly released feature of AWS DMS, but don't want to create a new task and restart the migration. Instead, you prefer to spin up a replication instance with a new AWS DMS version that supports the feature, and move the existing task to that instance.

You can move a task in the console by selecting the task and choosing **Move**. You can also run the CLI command or API operation `MoveReplicationTask` to move the task. You can move a task that has any database engine as its target endpoint.

Make sure that the target replication instance has enough storage space to accommodate the task that's being moved. Otherwise, scale the storage to make space for your target replication instance before moving the task.

Also, make sure that your target replication instance is created with the same or higher AWS DMS engine version as the current replication instance.

> **Note**
>
> - You can't move a task to the same replication instance where it currently resides.
> - You can't modify the settings of a task while it's moving.
> - A task you have run must have a status of **Stopped**, **Failed**, or **Failed-move** before you can move it.

There are two task statuses that relate to moving a DMS task, **Moving** and **Failed-move**. For more information about those task status, see [Task status](#).

After moving a task, you can enable and run premigration assessments to check for blocking issues before running the moved task.

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.

Reloading tables does not stop the 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.

> **Note**
>
> DMS applies the `TargetTablePrepMode` setting before reloading the table. If you set `TargetTablePrepMode` to `DO_NOTHING`, you must manually truncate the table first.

AWS Management Console

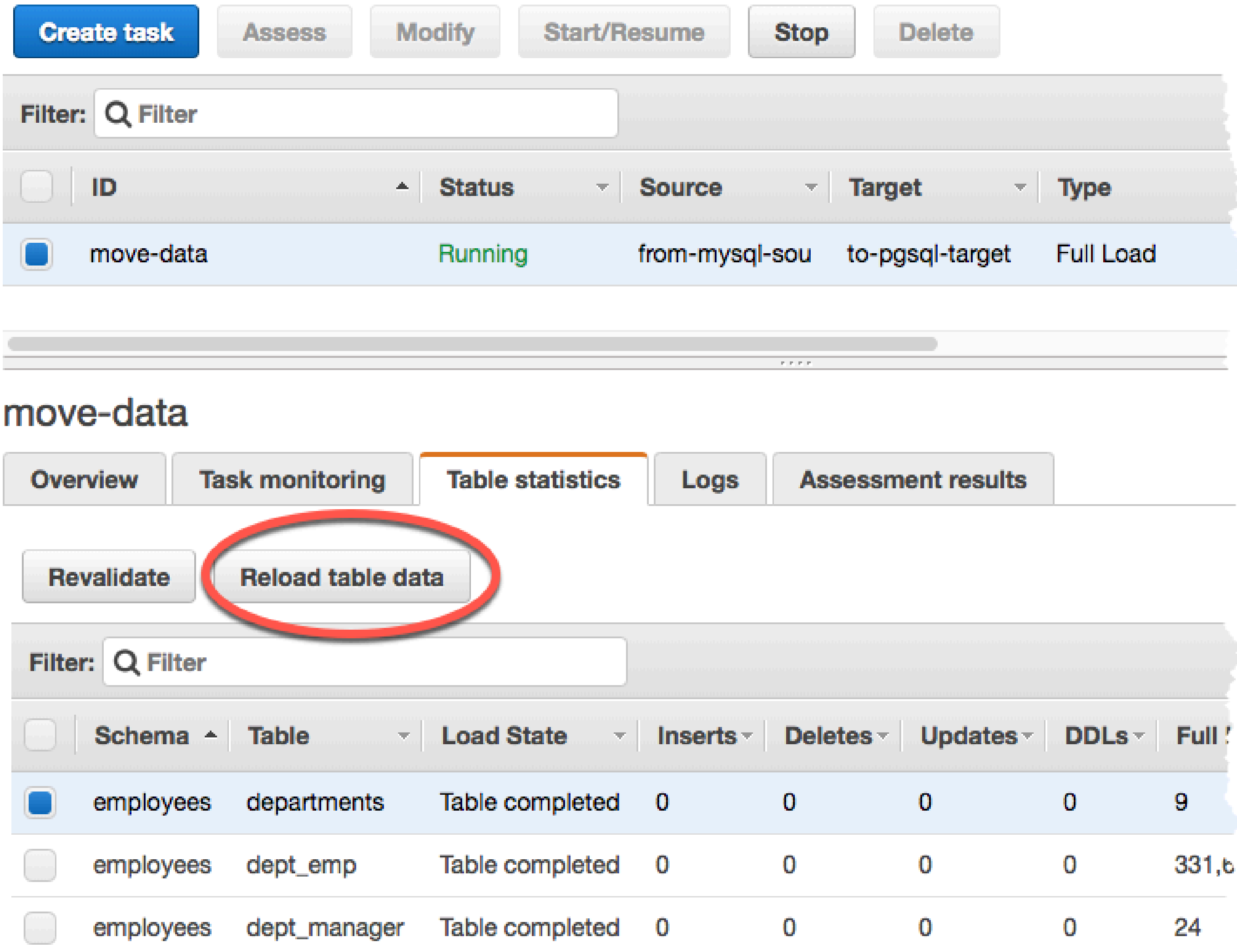
To reload a table using the AWS DMS console

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 IAM user, make sure that you have the appropriate permissions to access AWS DMS. For more information about the permissions required, see IAM permissions needed to use AWS DMS.
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.



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

> **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.

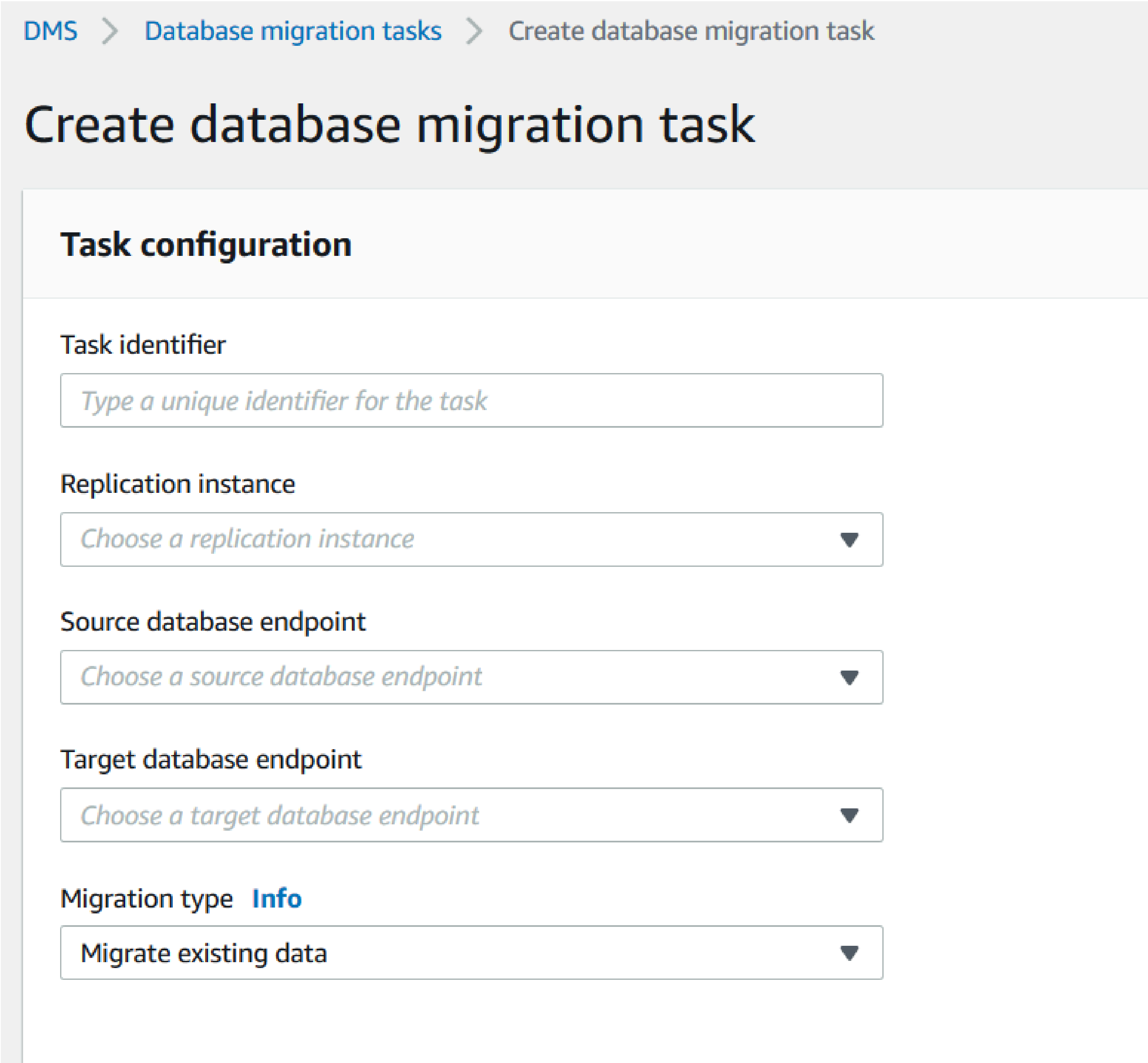
> **Note**
>
> AWS DMS doesn't support more than one transformation rule per schema level, table level, or column level.

The following procedure shows how to set up selection rules, based on a table called **`Customers`** in a schema called **`EntertainmentAgencySample`**.

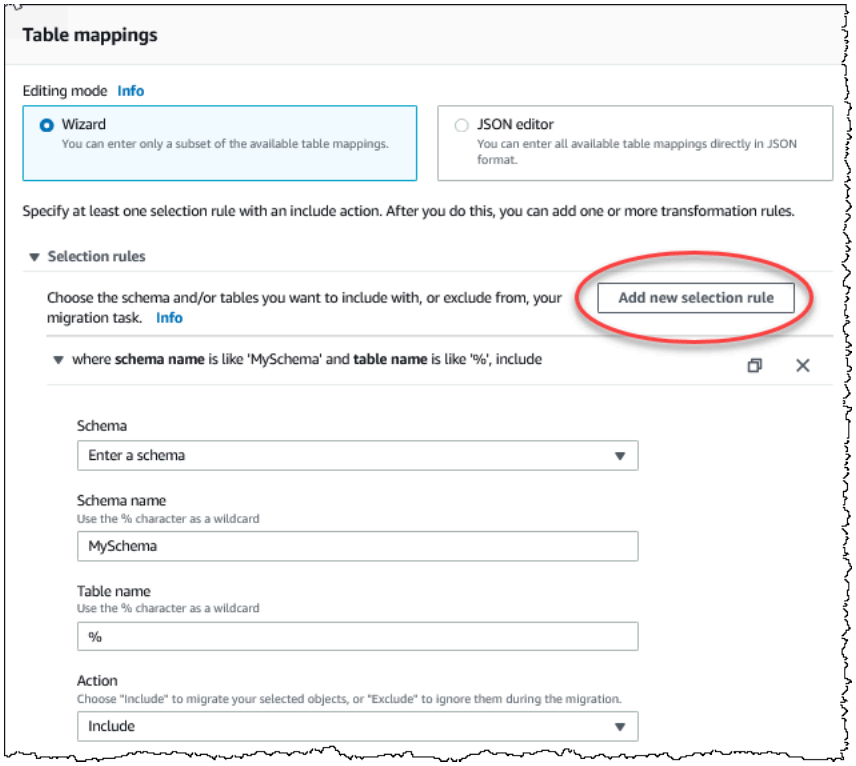
To specify a table selection, filter criteria, and transformations using the console

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 IAM user, make sure that you have the appropriate permissions to access AWS DMS. For more information about the permissions required, see [IAM permissions needed to use AWS DMS]().
2. On the **Dashboard** page, choose **Database migration tasks**.
3. Choose **Create Task**.
4. In the **Task configuration** section, enter the task information, including **Task identifier**, **Replication instance**, **Source database endpoint**, **Target database endpoint**, and **Migration type**.



5. In the **Table mapping** section, enter the schema name and table name. You can use "%" as a wildcard value when specifying the schema name or the table name. For information about other wildcards you can use, see the section called "Wildcards in table mapping". Specify the action to be taken, to include or exclude data defined by the filter.

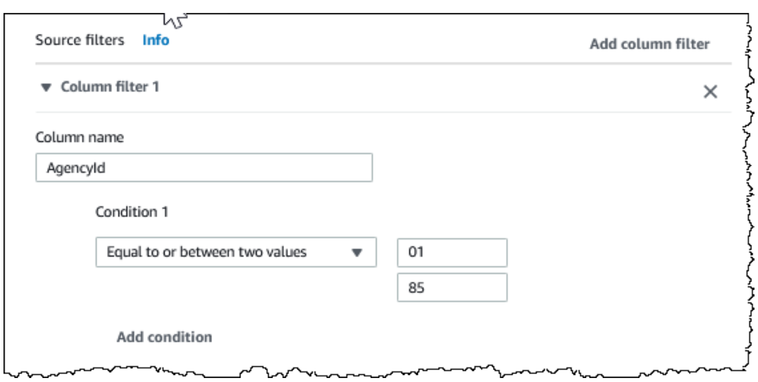


6. Specify filter information using the **Add column filter** and the **Add condition** links.

   a. Choose **Add column filter** to specify a column and conditions.

   b. Choose **Add condition** to add additional conditions.

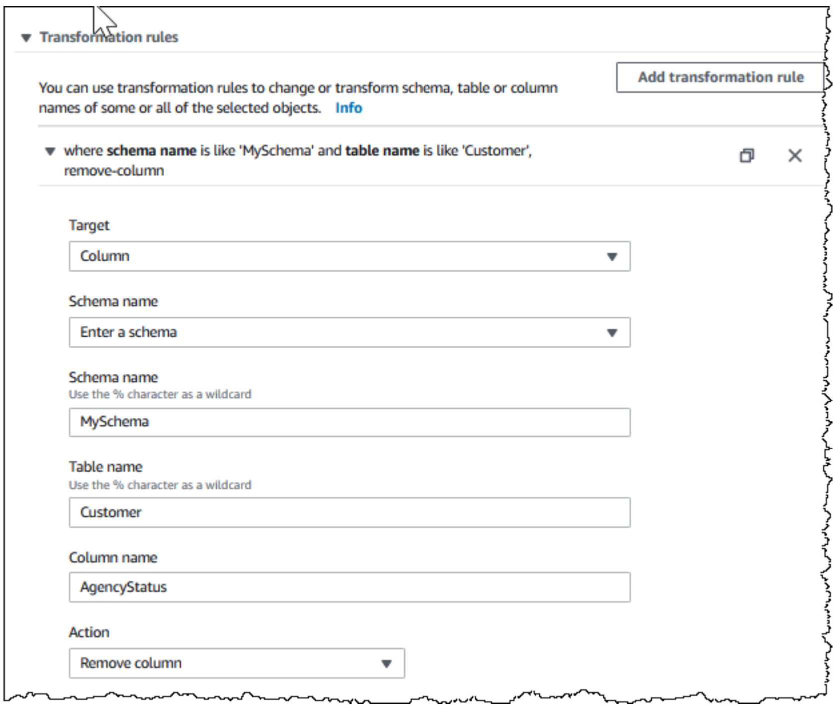
   The following example shows a filter for the **`Customers`** table that includes **`AgencyIDs`** between **`01`** and **`85`**.



7. When you have created the selections you want, choose **Add new selection rule**.

8. After you have created at least one selection rule, you can add a transformation to the task. Choose **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.



10. Choose **Add transformation rule**.
11. Choose **Create task**.

> **Note**
>
> AWS DMS doesn't support more than one transformation rule per schema level, table level, or 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.

AWS DMS can only process table mapping JSON files up to 2 MB in size. We recommend that you keep the mapping rule JSON file size below the 2 MB limit while working with DMS tasks. This prevents unexpected errors during task creation or modification. When a mapping rule file exceeds the 2 MB limit, we recommend that you split the tables across multiple tasks to reduce the size of the mapping rule file so that it stays below this limit.

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.
- `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.

  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.
- `table-settings` rules – Specify how DMS tasks load the data for individual tables. For more information, see Table and collection settings rules and operations.

> **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.
> 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.
> - Amazon Kinesis – For more information, see Using object mapping to migrate data to a Kinesis data stream.

- Apache Kafka – For more information, see Using object mapping to migrate data to a Kafka topic.

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.

> **Warning**
>
> Do not to include any sensitive data within these rules.

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. If you create the rule using the console, the console creates this value for you.
`rule-name`	An alphanumeric value.	A unique name to identify the rule. If you create the rule using the console, the console creates this value for you.
`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

Parameter	Possible values	Description
		an explicitly specified table and schema.

Parameter	Possible values	Description
`object-locator`	An object with the following parameters: • `schema-name` – The name of the schema. • `table-name` – The name of the table. • (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`. 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. Not all target endpoints accept views as a source of replication, even in full load (e.g. Amazon OpenSearch Service). Check the limitations of your target endpoint. **Warning** DMS selection rules are case-sensitive. However, the selection result also depends on the source endpoint database configuration. If the source endpoint	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. For information about other wildcards you can use, see [the section called "Wildcards in table mapping"]. Thus, you can match these items: • A single table, view, or collection in a single schema • A single table, view, or collection in some or all schemas • Some or all tables and views in a single schema, or collections in a single database • Some or all tables and views in some or all schemas, or collections in some or all databases 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:

Parameter	Possible values	Description
	is configured as case-insensitive, the case of the object locator value does not matter. Ensure that correct object identifiers are used in DMS selection rules on a case-insensitive endpoint.	• Oracle • Microsoft SQL Server • PostgreSQL • IBM Db2 LUW • IBM Db2 z/OS • SAP Adaptive Server Enterprise (ASE) • MySQL • AURORA MySQL • MariaDB **Note** 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. The supported sources for databases containing collections include: • MongoDB • Amazon DocumentDB
`load-order`	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.

Parameter	Possible values	Description
`filters`	An array of objects.	One or more objects for filtering the source. You specify object parameter s to filter on a single column in the source. You specify multiple objects to filter on multiple columns. For more information, see Using source filters.

Example Migrate all tables in a schema

The following example migrates all tables from a schema named `Test` in your source to your target endpoint.

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

Example Migrate some tables in a schema

The following example migrates all tables except those starting with `DMS` from a schema named `Test` in your source to your target endpoint.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
```

```
            "rule-name": "1",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "%"
            },
            "rule-action": "include"
        },
        {
            "rule-type": "selection",
            "rule-id": "2",
            "rule-name": "2",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "DMS%"
            },
            "rule-action": "exclude"
        }
    ]
}
```

Example Migrate 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

Tables and views are migrated according to their load-order values, with higher values receiving priority in the migration sequence. The following example migrates two tables, `loadfirst` with a priority value of 2 and `loadsecond` with a priority value of 1, the migration task would first process the `loadfirst` table before proceeding to the `loadsecond` table. This prioritization mechanism ensures that dependencies between database objects are respected during the migration process.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "loadsecond"
            },
            "rule-action": "include",
            "load-order": "1"
        },
        {
            "rule-type": "selection",
            "rule-id": "2",
            "rule-name": "2",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "loadfirst"
            },
            "rule-action": "include",
            "load-order": "2"
        }
    ]
}
```

> **Note**
>
> `load-order` is applicable for table initialization. The load of a successive table won't wait for a previous table load to complete if `MaxFullLoadSubTasks` is greater than 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"
        }
    ]
}
```

Wildcards in table mapping

This section describes wildcards you can use when specifying the schema and table names for table mapping.

Wildcard	Matches
%	Zero or more characters
_	A single character
[_]	A literal underscore character
[ab]	A set of characters. For example, [ab] matches either 'a' or 'b'.
[a-d]	A range of characters. For example,[a-d] matches either 'a', 'b', 'c', or 'd'.

For Oracle source and target endpoints, you can use the `escapeCharacter` extra connection attribute to specify an escape character. An escape character allows you to use a specified wildcard character in expressions as if it was not wild. For example, `escapeCharacter=#` allows you to use '#' to make a wildcard character act as an ordinary character in an expression as in the this sample code.

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "542485267",
```

```
            "rule-name": "542485267",
            "object-locator": { "schema-name": "ROOT", "table-name": "TEST#_T%" },
            "rule-action": "include",
            "filters": []
        }
    ]
}
```

Here, the '#' escape character makes the '_' wildcard character act as a normal character. AWS DMS selects tables in the schema named `ROOT`, where each table has a name with `TEST_T` as its prefix.

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.

Limitations

- You cannot apply more than one transformation rule action against the same object (schema, table, column, table-tablespace, or index-tablespace). You can apply several transformation rule actions on any level as long as each transformation action is applied against a different object. However, this restriction is not applicable when using data masking transformation rules where you can have another transformation like `ADD-COLUMN` or `CHANGE-DATA-TYPE` for the same column.
- Table names and column names in transformation rules are case-sensitive. For example, you must provide table names and column names for an Oracle or Db2 database in upper-case.
- Transformations are not supported for column names with Right-to-Left languages.
- Transformations cannot be performed on columns that contain special characters (e.g. #, \, /, -) in their name.
- The only supported transformation for columns that are mapped to BLOB/CLOB data types is to drop the column on the target.
- AWS DMS doesn't support replicating two source tables to a single target table. AWS DMS replicates records from table to table, and from column to column, according to the replication task's transformation rules. The object names must be unique to prevent overlapping.

  For example, a source table has a column named `ID` and the corresponding target table has a pre-existing column called `id`. If a rule uses an `ADD-COLUMN` statement to add a new column called `id`, and a SQLite statement to populate the column with custom values, this creates a duplicate, ambiguous object named `id` and is not supported.

- When creating a transformation rule, we recommend using the `data-type` parameter only when the selection rules specify multiple columns, for instance, when you set `column-name` to `%`. We don't recommend using `data-type` for selecting a single column.
- AWS DMS does not support transformation rules where source and target objects (tables) are on the same database/schema. Using the same table as both source and target in a transformation rule can lead to unexpected and potentially harmful results, including but not limited to unintended alterations to the table data, modification of table structures or even tables getting dropped.

Values

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. If you specify multiple transformation rules for the same object (schema, table, column, inter-table space, or index table space), AWS DMS applies the transform ation rule with the lower rule-id.
`rule-name`	An alphanumeric value.	A unique name to identify the rule.
`object-lo cator`	An object with the following parameters: • `schema-name` – The name of the schema. For MongoDB and	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

Parameter	Possible values	Description
	Amazon DocumentDB endpoints , this is the name of the database holding a set of collections. • `table-name` – The name of the table, view, or collection. • `table-tablespace-name` – The name of an existing table tablespace. • `index-tablespace-name` – The name of an existing index tablespace. • `column-name` – The name of an existing column. • `data-type` – The name of an existing column data type.	part of the value of each `object-locator` parameter, except `data-type` . Thus, you can match these items: • A single table or view in a single schema • A single table or view in some or all schemas • Some or all tables and views in a single schema • Some or all tables and views in some or all schemas • One or more columns in the specified table or tables, view or views, and schema or schemas. • We recommend using the `data-type` parameter only when the selection rules specify multiple columns, for instance, when you set `column-name` to `%`. We don't recommend using this parameter for a single column. 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 *either* of the following items:

Parameter	Possible values	Description
		• One, some, or all table tablespac es • One, some, or all index tablespac es

Parameter	Possible values	Description
`rule-action`	`add-column` , `include-column` , `remove-column` `rename` `convert-lowercase` , `convert-uppercase` `add-prefix` , `remove-prefix` , `replace-prefix` `add-suffix` , `remove-suffix` , `replace-suffix` `define-primary-key` `change-data-type` `add-before-image-columns` `data-masking-digits-mask` `data-masking-digits-randomize` `data-masking-hash-mask`	The transformation you want to apply to the object. All transform ation rule actions are case-sensitive. The `add-column` value of the `rule-action` parameter adds a column to a table. But you can't add a new column with the same name as an existing column of the same 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. The `include-column` value of the `rule-action` parameter changes the mode of the table to *drop all columns by default* and *include the columns specified*. Multiple columns are included in the target by invoking the `include-column` rule multiple times. You can't use a `define-primary-key` rule when the rule has a wildcard (%) in a schema or table name. For an existing task, transformation rule actions which alter the target

Parameter	Possible values	Description
		table schema such as `remove-column`, `rename`, or `add-prefix` will not take effect until you restart the task. If you resume the task after adding the transformation rule, you may see unexpected behavior for the altered column, which might include missing column data. A task restart is required to ensure the transformation rule works properly. The `data-masking-digits-mask`, `data-masking-digits-randomize`, and `data-masking-hash-mask` are for masking sensitive information contained in one or more columns of the table when loading to target. These transformations are only available for column rule targets. For more details, see [Using data masking to hide sensitive information]()
`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.

Parameter	Possible values	Description
`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`	`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`. 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`. 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", "wstring", "blob", "nclob", "clob", "boolean", "set", "list" "map", "tuple"` **Note** AWS DMS can apply transformations from one type to another ONLY in supported formats. E.g. DATE should be represented	The following is an example of a `data-type` parameter to specify the existing data type to be replaced. `{ "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": "test", "table-name": "table_t", "column-name": "col10" }, "rule-action": "change-data-type", "data-type": { "type": "string", "length": "4092", "scale": "" } }] }`

Parameter	Possible values	Description
	in `YYYY:MM:DD/YYYY-MM-DD.` DATETIME should be represented in `YYYY:MM:DD HH:MM:SS/YYYY-MM-DD HH:MM:SS`. TIME should be represented in `HH:MM:SS`. `precision` – If the added column or replacement data type has a precision, an integer value to specify the precision. `scale` – If the added column or replacement data type has a scale, an integer value or date time value to specify the scale. `length` – The length of new column data (when used with `add-column`)	Here, the `col10` column of the `table_t` table is changed to the `string` data type.

Parameter	Possible values	Description
`expression`	An alphanumeric value that follows SQLite syntax.	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. When used with the `rule-action` set to `add-column`, `expression` specifies data that makes up a new column. Note that only expressions are supported for this parameter. Operators and commands are not supported. For more information about using expressions for transformation rules, see Using transformation rule expressions to define column content. For more information about SQLite expressions, see Using SQLite functions to build expressions.

Parameter	Possible values	Description
`primary-key-def`	An object with the following parameters: • `name` – The name of a new primary key or unique index for the table or view. • (Optional) `origin` – The type of unique key to define: `primary-key` (the default) or `unique-index`. • `columns` – An array of strings listing the names of columns in the order they appear in the primary key or unique index.	This parameter can define the name, type, and content of a unique key on the transformed table or view. It does so when the `rule-action` is set to `define-primary-key` and the `rule-target` is set to `table`. By default, the unique key is defined as a primary key.

Parameter	Possible values	Description
`before-image-def`	An object with the following parameters: • `column-prefix` – A value prepended to a column name. The default value is `BI_`. • `column-suffix` – A value appended to the column name. The default is empty. • `column-filter` – Requires one of the following values: `pk-only` (default), `non-lob` (optional) and `all` (optional).	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 `rule-action` is set to `add-before-image-columns` and the `rule-target` is set to `column`. Don't set both `column-prefix` and `column-suffix` to empty strings. For `column-filter`, select: • `pk-only` – To add only columns that are part of table primary keys. • `non-lob` – To add only columns that are not of LOB type. • `all` – To add any column that has a before-image value. **Note** The `before-image-def` parameter does not support large binary object (LOB) data types such as CLOB and BLOB. If the data type is set as LOB, a void column is created in the table.

Parameter	Possible values	Description
		For more information about before-image support for AWS DMS target endpoints, see: • Using a before image to view original values of CDC rows for a Kinesis data stream as a target • Using a before image to view original values of CDC rows for Apache Kafka as a target

Examples

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": "column",
            "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": "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 following statement populates a `BI_emp_no` column in the corresponding row with 1.

```
UPDATE employees SET emp_no = 3 WHERE BI_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
- Flagging target records using an expression
- Replicating source table headers using expressions
- Using SQLite functions to build expressions
- Adding metadata to a target table using expressions

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`. The schema-name, table-name, and expression parameters refer to objects in the source database table. Value and the data-type block refer to objects in the target database table.

```
{
    "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 on both source and target database.

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	An empty string.	STRING

Header	Value in ongoing replication	Value in full load	Data type
	source. This value might be the system change number (SCN) or the log sequence number (LSN), depending on the source endpoint.		
AR_H_TIMESTAMP	A timestamp indicating the time of the change.	A timestamp indicating the current time data arrives at the target.	DATETIME (scale=7)
AR_H_COMMIT_TIMESTAMP	A timestamp indicating the time of the commit.	A timestamp indicating the current time.	DATETIME (scale=7)
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. This header is supported on the SQL Server and Oracle (version 11.2.0.3 and higher) source endpoints only.	An empty string.	STRING

Header	Value in ongoing replication	Value in full load	Data type
AR_H_CHANGE_SEQ	A unique incrementing number from the source database that consists of a timestamp and an auto incrementing number. The value depends on the source database system.	An empty string.	STRING

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
      }
    }
```

The following example adds a new column to the target that has a unique incrementing number from the source. This value represents a 35 digit unique number at task level. The first 16 digits are part of a timestamp, and the last 19 digits are the record_id number incremented by the DBMS.

```
{
"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_CHANGE_SEQ",
"data-type": {
"type": "string",
"length": 50
}
}
```

Using SQLite functions to build expressions

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.

> **Note**
>
> Instead of the concept of tables and views, MongoDB and DocumentDB databases store data records as documents that are gathered together in *collections*. So then, when migrating from a MongoDB or DocumentDB source, consider the range segmentation type of parallel load settings for selected *collections* rather than tables and views.

Topics

- Using a CASE expression
- Examples

Following, you can find string functions that you can use to build transformation rule expressions.

String functions	Description
`lower(x)`	The `lower(x)` function returns a copy of string *x* with all characters converted to lowercase. The default, built-in `lower` function works for ASCII characters only.
`upper(x)`	The `upper(x)` function returns a copy of string *x* with all characters converted to uppercase. The default, built-in `upper` function works for ASCII characters only.
`ltrim(x,y)`	The `ltrim(x,y)` function returns a string formed by removing all characters that appear in y from the left side of x. If there is no value for y, `ltrim(x)` removes spaces from the left side of x.
`replace(x,y,z)`	The `replace(x,y,z)` function returns a string formed by substituting string z for every occurrence of string y in string x.
`rtrim(x,y)`	The `rtrim(x,y)` function returns a string formed by removing all characters that appear in y from the right side of x. If there is no value for y, `rtrim(x)` removes spaces from the right side of x.
`substr(x,y,z)`	The `substr(x,y,z)` function returns a substring of the input string *x* that begins with the *y*th character, and which is *z* characters long. If *z* is omitted, `substr(x,y)` returns all characters through the end of string *x* beginning with the *y*th character. The leftmost character of *x* is number 1. If *y* is negative, the first character of the substring is found by counting from the right rather than the left. If *z* is negative, then the `abs(z)` character s preceding the *y*th character are returned. If *x* is a string, then the characters' indices refer to actual UTF-8 characters. If *x* is a BLOB, then the indices refer to bytes.

String functions	Description
`trim(x,y)`	The `trim(x,y)` function returns a string formed by removing all characters that appear in *y* from both sides of *x*. If there is no value for *y*, `trim(x)` removes spaces from both sides of *x*.

Following, you can find LOB functions that you can use to build transformation rule expressions.

LOB functions	Description
`hex(x)`	The `hex` function receives a BLOB as an argument and returns an uppercase hexadecimal string version of the BLOB content.
`randomblob ( N)`	The `randomblob( N)` function returns an *N*-byte BLOB that contains pseudorandom bytes. If *N* is less than 1, a 1-byte random BLOB is returned.
`zeroblob(N)`	The `zeroblob( N)` function returns a BLOB that consists of *N* bytes of 0x00.

Following, you can find numeric functions that you can use to build transformation rule expressions.

Numeric functions	Description
`abs(x)`	The `abs(x)` function returns the absolute value of the numeric argument *x*. The `abs(x)` function returns NULL if *x* is NULL. The `abs(x)` function returns 0.0 if *x* is a string or BLOB that can't be converted to a numeric value.
`random()`	The `random` function returns a pseudorandom integer between -9,223,372,036,854,775,808 and +9,223,372,036,854 ,775,807.

Numeric functions	Description
`round (x,y)`	The `round (x,y)` function returns a floating-point value *x* rounded to *y* digits to the right of the decimal point. If there is no value for *y*, it's assumed to be 0.
`max (x,y...)`	The multiargument `max` function returns the argument with the maximum value, or returns NULL if any argument is NULL. The `max` function searches its arguments from left to right for an argument that defines a collating function. If one is found, it uses that collating function for all string comparisons. If none of the arguments to `max` define a collating function, the `BINARY` collating function is used. The `max` function is a simple function when it has two or more arguments, but it operates as an aggregate function if it has a single argument.
`min (x,y...)`	The multiargument `min` function returns the argument with the minimum value. The `min` function searches its arguments from left to right for an argument that defines a collating function. If one is found, it uses that collating function for all string comparisons. If none of the arguments to `min` define a collating function, the `BINARY` collating function is used. The `min` function is a simple function when it has two or more arguments, but it operates as an aggregate function if it has a single argument.

Following, you can find NULL check functions that you can use to build transformation rule expressions.

NULL check functions	Description
`coalesce (x,y...)`	The `coalesce` function returns a copy of its first non-NULL argument, but it returns NULL if all arguments are NULL. The coalesce function has at least two arguments.

NULL check functions	Description
`ifnull(x,y)`	The `ifnull` function returns a copy of its first non-NULL argument, but it returns NULL if both arguments are NULL. The `ifnull` function has exactly two arguments. The `ifnull` function is the same as `coalesce` with two arguments.
`nullif(x,y)`	The `nullif(x,y)` function returns a copy of its first argument if the arguments are different, but it returns NULL if the arguments are the same. The `nullif(x,y)` function searches its arguments from left to right for an argument that defines a collating function. If one is found, it uses that collating function for all string comparisons. If neither argument to nullif defines a collating function, then the `BINARY` collating function is used.

Following, you can find date and time functions that you can use to build transformation rule expressions.

Date and time functions	Description
`date(timestring , modifier, modifier...)`	The `date` function returns the date in the format YYYY-MM-DD.
`time(timestring , modifier, modifier...)`	The `time` function returns the time in the format HH:MM:SS.
`datetime( timestring , modifier, modifier...)`	The `datetime` function returns the date and time in the format YYYY-MM-DD HH:MM:SS.
`julianday( timestring , modifier, modifier...)`	The `julianday` function returns the number of days since noon in Greenwich on November 24, 4714 B.C.
`strftime( format, timestring , modifier, modifier...)`	The `strftime` function returns the date according to the format string specified as the first argument, using one of the following variables:

Date and time functions	Description
	%d: day of month %H: hour 00–24 %f: ** fractional seconds SS.SSS %j: day of year 001–366 %J: ** Julian day number %m: month 01–12 %M: minute 00–59 %s: seconds since 1970-01-01 %S: seconds 00–59 %w: day of week 0–6 sunday==0 %W: week of year 00–53 %Y: year 0000–9999 %%: %

Following, you can find a hash function that you can use to build transformation rule expressions.

Hash function	Description
`hash_sha256(` *x* `)`	The `hash` function generates a hash value for an input column (using the SHA-256 algorithm) and returns the hexadecimal value of the generated hash value. To use the `hash` function in an expression, add `hash_sha256(` *x* `)` to the expression and replace *x* with the source column name.

Using a CASE expression

The SQLite CASE expression evaluates a list of conditions and returns an expression based on the result. Syntax is shown following.

```
    CASE case_expression
     WHEN when_expression_1 THEN result_1
     WHEN when_expression_2 THEN result_2
     ...
     [ ELSE result_else ]
    END

# Or

     CASE
     WHEN case_expression THEN result_1
     WHEN case_expression THEN result_2
     ...
     [ ELSE result_else ]
    END
```

Examples

Example of adding a new string column to the target table using a case condition

The following example transformation rule adds a new string column, emp_seniority, to the target table, employee. It uses the SQLite round function on the salary column, with a case condition to check if the salary equals or exceeds 20,000. If it does, the column gets the value SENIOR, and anything else has the value JUNIOR.

```
  {
      "rule-type": "transformation",
      "rule-id": "2",
      "rule-name": "2",
      "rule-action": "add-column",
      "rule-target": "column",
      "object-locator": {
        "schema-name": "public",
        "table-name": "employee"
      },
      "value": "emp_seniority",
      "expression": " CASE WHEN round($emp_salary)>=20000 THEN ‘SENIOR’ ELSE ‘JUNIOR’ 
 END",
```

```
      "data-type": {
        "type": "string",
        "length": 50
      }

  }
```

Example of adding a new string column to the target table using a SUBSTR function

The following example transformation rule adds a new string column using SQLite operators or functions to define the data in a column. This approach involves using SQLite functions to transform the GUID data loaded from Oracle to UUID format before inserting it into the Postgresql target table.

Following rule uses the SQLite substring (SUBSTR), hexadecimal function (HEX), and lowercase (LOWER) functions to break the GUID data into several groups separated by hyphens, specifically a group of 8 digits followed by three groups of 4 digits followed by a group of 12 digits, for a total of 32 digits representing the 128 bits.

Here is the sample source data and output on target post processing through transformation rule:

Source Table (Oracle GUID format)

T_COL2

```
06F6949D234911EE80670242AC120002
1A2B3C4D5E6F11EE80670242AC120003
F5E4D3C2B1A011EE80670242AC120004
```

Target Table (PostgreSQL UUID format)

T_COL2_TMP

```
06f6949d-2349-11ee-8067-0242ac120002
1a2b3c4d-5e6f-11ee-8067-0242ac120003
f5e4d3c2-b1a0-11ee-8067-0242ac120004
```

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

```
  "rule-action": "add-column",
  "rule-target": "column",
  "object-locator": {
    "schema-name": "SPORTS",
    "table-name": "TEST_TBL_2"
  },
  "value": "t_col2_tmp",
  "expression": "CASE LOWER(SUBSTR(HEX($T_COL2), 1, 8) || '-' ||
 SUBSTR(HEX($T_COL2), 9, 4) || '-' || SUBSTR(HEX($T_COL2), 13, 4) || '-' ||
 SUBSTR(HEX($T_COL2), 17, 4) || '-' || SUBSTR(HEX($T_COL2), 21, 12)) WHEN '----'
 THEN NULL ELSE LOWER(SUBSTR(HEX($T_COL2), 1, 8) || '-' || SUBSTR(HEX($T_COL2), 9,
 4) || '-' || SUBSTR(HEX($T_COL2), 13, 4) || '-' || SUBSTR(HEX($T_COL2), 17, 4) ||
 '-' || SUBSTR(HEX($T_COL2), 21, 12)) END",
  "data-type": {
    "type": "string",
    "length": 60
  }
}
```

Example of adding a new date column to the target table

The following example adds a new date column, `createdate`, to the target table, `employee`. When you use the SQLite date function `datetime`, the date is added to the newly created table for each row inserted.

```
{
    "rule-type": "transformation",
    "rule-id": "2",
    "rule-name": "2",
    "rule-action": "add-column",
    "rule-target": "column",
    "object-locator": {
      "schema-name": "public",
      "table-name": "employee"
    },
    "value": "createdate",
    "expression": "datetime ()",
    "data-type": {
      "type": "datetime",
      "precision": 6
    }
}
```

Example of adding a new numeric column to the target table

The following example adds a new numeric column, `rounded_emp_salary`, to the target table, `employee`. It uses the SQLite `round` function to add the rounded salary.

```
{
    "rule-type": "transformation",
    "rule-id": "2",
    "rule-name": "2",
    "rule-action": "add-column",
    "rule-target": "column",
    "object-locator": {
      "schema-name": "public",
      "table-name": "employee"
    },
    "value": "rounded_emp_salary",
    "expression": "round($emp_salary)",
    "data-type": {
      "type": "int8"
    }
}
```

Example of adding a new string column to the target table using the hash function

The following example adds a new string column, `hashed_emp_number`, to the target table, `employee`. The SQLite `hash_sha256(x)` function creates hashed values on the target for the source column, `emp_number`.

```
{
    "rule-type": "transformation",
    "rule-id": "2",
    "rule-name": "2",
    "rule-action": "add-column",
    "rule-target": "column",
    "object-locator": {
      "schema-name": "public",
      "table-name": "employee"
    },
    "value": "hashed_emp_number",
    "expression": "hash_sha256($emp_number)",
    "data-type": {
      "type": "string",
```

```
        "length": 64
      }
  }
```

Adding metadata to a target table using expressions

You can add the metadata information to the target table by using the expressions following:

- `$AR_M_SOURCE_SCHEMA` – The name of the source schema.
- `$AR_M_SOURCE_TABLE_NAME` – The name of the source table.
- `$AR_M_SOURCE_COLUMN_NAME` – The name of a column in the source table.
- `$AR_M_SOURCE_COLUMN_DATATYPE` – The data type of a column in the source table.

Example of adding a column for a schema name using the schema name from the source

The example following adds a new column named `schema_name` to the target by using the schema name from the source.

```
  {
      "rule-type": "transformation",
      "rule-id": "2",
      "rule-name": "2",
      "rule-action": "add-column",
      "rule-target": "column",
      "object-locator": {
        "schema-name": "%",
        "table-name": "%"
      },
      "rule-action": "add-column",
      "value":"schema_name",
      "expression": "$AR_M_SOURCE_SCHEMA",
      "data-type": {
         "type": "string",
         "length": 50
      }
  }
```

Table and collection settings rules and operations

Use table settings to specify any settings that you want to apply to a selected table or view for a specified operation. Table-settings rules are optional, depending on your endpoint and migration requirements.

Instead of using tables and views, MongoDB and Amazon DocumentDB databases store data records as documents that are gathered together in *collections*. A single database for any MongoDB or Amazon DocumentDB endpoint is a specific set of collections identified by the database name.

When migrating from a MongoDB or Amazon DocumentDB source, you work with parallel load settings slightly differently. In this case, consider the autosegmentation or range segmentation type of parallel load settings for selected collections rather than tables and views.

Topics

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, view, or collection 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.
`object-locator`	An object with the following parameters:	The name of a specific schema and table or view

Parameter	Possible values	Description
	• `schema-name` – The name of the schema. For MongoDB and Amazon DocumentDB endpoints, this is the name of the database holding a set of collections. • `table-name` – The name of the table, view, or collection.	or the name of a specific database and collection (no wildcards).

Parameter	Possible values	Description
`parallel-load`	An object with the following parameters: • `type` – Specifies whether parallel loading is turned on. 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 MongoDB and Amazon DocumentDB endpoints, partitions are segments. AWS DMS can calculate these automatically given associated autosegmentation parameters. Or you can specify these manually using range segmentation parameters. 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. • `partitions` – When `type` is `partitions-list`, this value	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: • By segments specified by all available partitions or subpartitions. • By selected partitions and subpartitions. • By autosegmentation or range-based segments that you specify. For more information about parallel load, see [Using parallel load for selected tables, views, and collections]().

Parameter	Possible values	Description
	specifies all the partitions to load in parallel. • `subpartitions` – For Oracle endpoints only, when `type` is `partitions-list` this value specifies all the subpartitions to load in parallel. • `columns` – When `type` is `ranges`, this value specifies the names of columns used to identify range-based segments to load in parallel. • `boundaries` – When `type` is `ranges`, this value specifies the values of the `columns` used to identify range-based segments to load in parallel.	

Parameter	Possible values	Description
`type`	One of the following for `parallel-load` : • `partitions-auto` – All partitions of the table or view are loaded in parallel. Every partition is allocated to its own thread. This is a required setting for MongoDB and Amazon DocumentDB source endpoints to use the autosegmentation option of a parallel full load. • `subpartitions-auto` – (Oracle endpoints only) All subpartitions of the table or view are loaded in parallel. Every subpartition is allocated to its own thread. • `partitions-list` – 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 (`partitions`) and subpartition names (`subpartitions`).	The mechanism to identify the table, view, or collection partitions, subpartitions, or segments to load in parallel.

Parameter	Possible values	Description
	• `ranges` – All range-specified segments of the table, view, or collection are loaded in parallel. Each table, view, or collection segment that you identify is allocated to its own thread. You specify these segments by column names (`columns`) and column values (`boundaries`). PostgreSQL endpoints support only this type of a parallel load. MongoDB and Amazon DocumentDB as a source endpoints support both this range segmentation type and the autosegmentation type of a parallel full load (`partitions-auto`). • `none` – The table, view, or collection is loaded in a single-threaded task (the default), regardless of its partitions or subpartitions. For more information, see [Creating a task]().	
`number-of-partitions`	(Optional) When `type` is `partitions-auto` for specified collections of a MongoDB or Amazon DocumentDB endpoint, this parameter specifies the total number of partitions (segments) used for migration. The default is 16.	Specifies the exact number of partitions to load in parallel.

Parameter	Possible values	Description
`collection-count-from-metadata`	(Optional) When `type` is `partitions-auto` for specified collections of a MongoDB or Amazon DocumentDB endpoint and this parameter is set to `true`, AWS DMS uses an estimated collection count for determining the number of partitions. If this parameter is set to `false`, AWS DMS uses the actual collection count. The default is `true`.	Specifies whether to use an estimate collection count or the actual collection count to calculate the number of partitions to load in parallel.
`max-records-skip-per-page`	(Optional) When `type` is `partitions-auto` for specified collections of a MongoDB or Amazon DocumentDB endpoint, this is the number of records to skip at once when determining the boundaries for each partition. AWS DMS uses a paginated skip approach to determine the minimum boundary for a partition. The default is 10,000.	Specifies the number of records to skip at once when determining the boundaries for each partition. Setting a relatively large value from the default might result in cursor timeouts and task failures. Setting a relatively low value from the default results in more operations per page and a slower full load.

Parameter	Possible values	Description
`batch-size`	(Optional) When `type` is `partitions-auto` for specified collections of a MongoDB or Amazon DocumentDB endpoint, this integer value limits the number of documents returned in one round-trip batch. If the batch size is zero (0), the cursor uses the server-defined maximum batch size. The default is 0.	Specifies the maximum number of documents returned in one batch. Each batch requires a round trip to the server.
`partitions`	When `type` is `partitions-list` , this is an array of strings that specify the names of partitions to load in parallel.	The names of partitions to load in parallel.
`subpartitions`	(Oracle endpoints only) When `type` is `partitions-list` , this is an array of strings that specifies the names of subpartitions to load in parallel.	The names of subpartitions to load in parallel.
`columns`	When `type` is `ranges`, an array of strings set to the names of columns that identify range-based table, view, or collection segments to load in parallel.	The names of columns that identify range-based table, view, or collection segments to load in parallel.

Parameter	Possible values	Description
`boundaries`	When `type` is `ranges`, an array of column-value arrays. Each column-value array contains column values in the quantity and order specified by `columns`. A column-value array specifies the upper boundary of a table, view, or collection segment. Each additional column-value array adds the upper boundary for one additional table, view, or collectio n segment. All such range-based table, view, or collection segments load in parallel.	Column values that identify range-based table, view, or collectio n partitions to load in parallel.
`lob-settings`	An object with the following parameters: • `mode` – Specifies the migration handling mode for LOBs. • `bulk-max-size` – Specifies the maximum size of LOBs, depending on the `mode` setting.	A value that specifies LOB handling for the table or view identified by the `object-locator` option. The specified LOB handling overrides any task LOB settings for this table or view only. For more information about using the LOB settings parameters, see Specifying LOB settings for a selected table or view.

Parameter	Possible values	Description
`mode`	Specifies the migration handling for LOBs in the specified table or view using the following values: • `limited` – (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 `bulk-max-size` value (zero is invalid). All migrated LOBs greater than `bulk-max-size` are truncated to the size that you set. • `unlimited` – 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 `bulk-max-size` value of zero, all LOBs are migrated in *standard* full LOB mode. In this form of `unlimited` mode, all LOBs are migrated separately from other column data types using a lookup from the source table or view. If you specify a `bulk-max-size` value greater than zero, all LOBs are migrated in *combination* full LOB mode. In this form of `unlimited` mode, LOBs greater than `bulk-max-size`	The mechanism used to migrate LOBs.

Parameter	Possible values	Description
	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 `unlimited` mode, regardless of the form you use. • `none` – All table or view LOBs are migrated according to the task LOB settings. For more information about the task LOB settings, see Target metadata task settings. For more information about how to migrate LOBs and how to specify these task LOB settings, see Setting LOB support for source databases in an AWS DMS task.	
`bulk-max-size`	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.

Wildcards in table-settings are restricted

Using the percent wildcard ("%") in "table-settings" rules is not supported for source databases as shown following.

```
{
    "rule-type": "table-settings",
    "rule-id": "8",
    "rule-name": "8",
    "object-locator": {
        "schema-name": "ipipeline-prod",
        "table-name": "%"
    },
    "parallel-load": {
        "type": "partitions-auto",
        "number-of-partitions": 16,
        "collection-count-from-metadata": "true",
        "max-records-skip-per-page": 1000000,
        "batch-size": 50000
    }
  }
```

If you use "%" in the "table-settings" rules as shown, then AWS DMS returns the exception following.

```
Error in mapping rules. Rule with ruleId = x failed validation. Exact
schema and table name required when using table settings rule.
```

In addition, AWS recommends that you don't load a great number of large collections using a single task with `parallel-load`. Note that AWS DMS limits resource contention as well as the number of segments loaded in parallel by the value of the `MaxFullLoadSubTasks` task settings parameter, with a maximum value of 49.

Instead, specify all collections for your source database for the largest collections by specifying each "schema-name" and "table-name" individually. Also, scale up your migration properly. For example, run multiple tasks across a sufficient number of replication instances to handle a great number of large collections in your database.

Using parallel load for selected tables, views, and collections

To speed up migration and make it more efficient, you can use parallel load for selected relational tables, views, and collections. In other words, you can migrate a single segmented table, view, or collection 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, views, and collections in parallel from the source endpoint. You can then have multiple threads migrate and load the same tables, views, and collections in parallel to the target endpoint. For some database engines, you can segment the tables and views by existing partitions or subpartitions. For other database engines, you can have AWS DMS automatically segment collections according to specific parameters (autosegmentation). Otherwise, you can segment any table, view, or collection 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 LUW
- SAP Adaptive Server Enterprise (ASE)
- MongoDB (only supports the autosegmentation and range segmentation options of a parallel full load)
- Amazon DocumentDB (only supports the autosegmentation and range segmentation options of a parallel full load)

For MongoDB and Amazon DocumentDB endpoints, AWS DMS supports the following data types for columns that are partition keys for the range segmentation option of a parallel full load.

- Double
- String
- ObjectId
- 32 bit integer
- 64 bit integer

Parallel load for use with table-setting rules are supported for the following target endpoints:

- Oracle
- Microsoft SQL Server
- MySQL
- PostgreSQL
- Amazon S3
- SAP Adaptive Server Enterprise (ASE)
- Amazon Redshift
- MongoDB (only supports the autosegmentation and range segmentation options of a parallel full load)
- Amazon DocumentDB (only supports the autosegmentation and range segmentation options of a parallel full load)
- Db2 LUW

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 the supported targets of a parallel-load task, define more segments using column-value boundaries.

> **Important**
>
> `MaxFullLoadSubTasks` controls the number of tables or table segments to load in parallel. `ParallelLoadThreads` controls the number of threads that are used by a migration task to execute the loads in parallel. *These settings are multiplicative*. As such, the total number of threads that are used during a full load task is approximately the result of the value of `ParallelLoadThreads` multiplied by the value of `MaxFullLoadSubTasks` (`ParallelLoadThreads` * `MaxFullLoadSubtasks`).
> If you create tasks with a high number of Full Load sub tasks and a high number of parallel load threads, your task can consume too much memory and fail.

To specify the maximum number of threads per table for Amazon DynamoDB, Amazon Kinesis Data Streams, Apache Kafka, or Amazon Elasticsearch Service targets, use the `ParallelLoadThreads` target metadata task setting.

To specify the buffer size for a parallel load task when `ParallelLoadThreads` is used, use the `ParallelLoadBufferSize` target metadata task setting.

The availability and settings of `ParallelLoadThreads` and `ParallelLoadBufferSize` depend on the target endpoint.

For more information about the `ParallelLoadThreads` and `ParallelLoadBufferSize` settings, see Target metadata task settings. For more information about the `MaxFullLoadSubTasks` setting, see Full-load task settings. 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, view, or collection 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, view, or collection for parallel load:

- By partitions (or segments) – Load all existing table or view partitions (or segments) using the `partitions-auto` type. Or load only selected partitions using the `partitions-list` type with a specified partitions array.

  For MongoDB and Amazon DocumentDB endpoints only, load all or specified collections by segments that AWS DMS automatically calculates also using the `partitions-auto` type and additional optional `table-settings` parameters.
- (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, view, or collection segments that you define by using column-value boundaries. To do so, use the `ranges` type with specified `columns` and `boundaries` arrays.

  > **Note**
  >
  > PostgreSQL endpoints support only this type of a parallel load. MongoDB and Amazon DocumentDB as a source endpoints support both this range segmentation type and the autosegmentation type of a parallel full load (`partitions-auto`).

To identify additional tables, views, or collections 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, view, or collection 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, view, or collection partition (or segment) or subpartition is then automatically allocated to its own thread.

  For some endpoints, parallel load includes partitions or subpartitions only if they are already defined for the table or view. For MongoDB and Amazon DocumentDB source endpoints, you can have AWS DMS automatically calculate the partitions (or segments) based on optional additional parameters. These include `number-of-partitions`, `collection-count-from-metadata`, `max-records-skip-per-page`, and `batch-size`.

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.

   > **Important**
   >
   > For Oracle endpoints, make sure partitions and subpartitions aren't overlapping when choosing them for parallel load. If you use overlapping partitions and subpartitions to load data in parallel, it duplicates entries, or it fails due to a primary key duplicate violation.

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, view, or collection 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.

      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.

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 that 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).

> **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-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 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"
            }
        }
     ]
}
```

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"
            }
```

```
        }
    ]
}
```

Using data masking to hide sensitive information

To conceal sensitive data stored in one or more columns of the tables being migrated, you can leverage Data Masking transformation rule actions. Starting from version 3.5.4, AWS DMS allows the use of data masking transformation rule actions in table mapping, enabling you to alter the contents of one or more columns during the migration process. AWS DMS loads the modified data into the target tables.

AWS Database Migration Service provides three options for data masking transformation rule actions:

- Data Masking: Digits Mask
- Data Masking: Digits Randomize
- Data Masking: Hashing Mask

These data masking transformation rule actions can be configured in the table mapping of your replication task, similar to other transformation rules. The rule target should be set to the column level.

Masking numbers in column data with a masking character

The "Data Masking: Digits Mask" transformation rule action allows you to mask numerical data in one or more columns by replacing digits with a single ASCII printable character that you specify (excluding empty or whitespace characters).

Here's an example that masks all digits in the `cust_passport_no` column of the `customer_master` table with the masking character `'#'` and loads the masked data into the target table:

```
            {
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
```

```
            "object-locator": {
                "schema-name": "cust_schema",
                "table-name": "%"
            },
            "rule-action": "include"
        },
        {
            "rule-type": "transformation",
            "rule-id": "2",
            "rule-name": "2",
            "rule-target": "column",
            "object-locator": {
                "schema-name": "cust_schema",
                "table-name": "customer_master",
                "column-name": "cust_passport_no"
            },
            "rule-action": "data-masking-digits-mask",
            "value": "#"
        }
    ]
}
```

For example, if the column `cust_passport_no` in the source table contains the record "C6BGJ566669K", the AWS DMS task will write this data to the target table as `"C#BGJ######K"`.

Replacing numbers in the column with random numbers

The transformation rule "Data Masking: Digits Randomize" allows you to replace each numerical digit in one or more columns with a random number. In the following example, AWS DMS replaces every digit in the `cust_passport_no` column of the source table `customer_master` with a random number and writes the modified data to the target table:

```
            {
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "cust_schema",
                "table-name": "%"
```

```
            },
            "rule-action": "include"
        },
        {
            "rule-type": "transformation",
            "rule-id": "2",
            "rule-name": "2",
            "rule-target": "column",
            "object-locator": {
                "schema-name": "cust_schema",
                "table-name": "customer_master",
                "column-name": "cust_passport_no"
            },
            "rule-action": "data-masking-digits-randomize"
        }
    ]
}
```

For example, the AWS DMS task will transform the value `"C6BGJ566669K"` in the `cust_passport_no` column of the source table to `"C1BGJ842170K"` and write it to the target database.

Replacing column data with hash value

The transformation rule "Data Masking: Hashing Mask" allows you to replace the column data with a hash generated using the `SHA256` algorithm. The length of the hash will always be 64 characters, hence the target table column length should be 64 characters at minimum. Alternatively, you can add a `change-data-type` transformation rule action to the column to increase the width of the column in the target table.

The following example generates a 64-character long hash value for the data in the `cust_passport_no` column of the source table `customer_master` and loads the transformed data to the target table after increasing the column length:

```
{
"rules": [
{
"rule-type": "selection",
"rule-id": "1",
"rule-name": "1",
```

```
"object-locator": {
"schema-name": "cust_schema",
"table-name": "%"
},
"rule-action": "include"
},
{
"rule-type": "transformation",
"rule-id": "2",
"rule-name": "2",
"rule-target": "column",
"object-locator": {
"schema-name": "cust_schema",
"table-name": "customer_master",
"column-name": "cust_passport_no"
},
"rule-action": "change-data-type",
"data-type": {
"type": "string",
"length": "100",
"scale": ""
}
},
{
"rule-type": "transformation",
"rule-id": "3",
"rule-name": "3",
"rule-target": "column",
"object-locator": {
"schema-name": "cust_schema",
"table-name": "customer_master",
"column-name": "cust_passport_no"
},
"rule-action": "data-masking-hash-mask"
}
]
}
```

For example, if the column `cust_passport_no` of the source table contains value “`C6BGJ566669K`”, AWS DMS task will write a hash “`7CB06784764C9030CCC41E25C15339FEB293FFE9B329A72B5FED564E99900C75`” to the target table.

Limitations

- Each Data Masking transformation rule option is supported for specific AWS DMS data types only:
  - Data Masking: Digits Mask is supported for columns of data types: `WSTRING` and `STRING`.
  - Data Masking: Digits Randomize is supported for columns of data types: `WSTRING, STRING; NUMERIC, INT1, INT2, INT4, and INT8` with unsigned counterparts.
  - Data Masking: Hashing Mask is supported for columns of data types: `WSTRING` and `STRING`.

  To learn more about the mapping of AWS DMS data types to your source engine's data types, refer to the data type mapping of your source engine with AWS DMS data types. See source data types for Source data types for Oracle, Source data types for SQL Server, Source data types for PostgreSQL, and Source data types for MySQL.
- Using a Data Masking rule action for a column with an incompatible data type will cause an error in the DMS task. Refer to DataMaskingErrorPolicy in DMS task settings to specify the error handling behavior. For more information about `DataMaskingErrorPolicy`, see Error handling task settings.
- You may add a change-data-type transformation rule action to change the data type of the column to a compatible type if your source column type is not supported for the masking option you plan to use. The `rule-id` of the `change-data-type` transformation should be a smaller number than the rule-id of the masking transformation so that the data type change happens before masking.
- Use Data Masking: Hashing Mask action for masking Primary Key/ Unique Key/ Foreign Key columns, as the generated hash value will be unique and consistent. Other two masking options cannot guarantee uniqueness.
- While Data Masking: Digits Mask and Data Masking: Digits Randomize affect only the digits in the column data and does not affect the length of data, Data Masking: Hashing Mask modifies the entire column, length of data changes to 64 characters. Hence, the target table to be created accordingly or a change-data-type transformation rule should be added for the column which is being masked.
- Columns with Data Masking transformation rule action specified are excluded from data validation in AWS DMS. If the Primary Key/ Unique Key columns are masked, data validation will not be run for this table; validation status of such table will be equal to `No Primary key`.

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 that is associated with either one value, two values, or no value, depending on the operator.
- Column names, table names, view names, and schema names are case-sensitive. Oracle and Db2 should always use UPPER case.
- Filters only support tables with exact names. Filters do not support wildcards.

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.

The following additional concerns apply when using source filters:

- We recommend that you create an index using the columns included in the filtering definition and the primary key.

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
`filter-type`	`source`
`column-name`	A parameter with the name of the source column to which you want the filter applied. The name is case-sensitive.
`filter-conditions`	An array of one or more objects containing a `filter-operator` parameter and zero or more associated value parameters, depending on the `filter-operator` value.
`filter-operator`	A parameter with one of the following values: • `lte` – less than or equal to one value • `ste` – less than or equal to one value (`lte` alias) • `gte` – greater than or equal to one value • `eq` – equal to one value • `noteq` – not equal to one value

Parameter	Value
	• `between` – equal to or between two values • `notbetween` – not equal to or between two values • `null` – NULL values • `notnull` – no NULL values
`value` or `start-value` and `end-value` or no values	Zero or more value parameters associated with `filter-operator` : • If `filter-operator` is `lte`, `ste`, `gte`, `eq`, or `noteq`, use `value` to specify one value parameter. • If `filter-operator` is `between` or `notbetween` , use `start-value` and `end-value` to specify two value parameters. • If `filter-operator` is `null` or `notnull`, specify no value parameters.

The following examples show some common ways to use source filters.

Example Single filter

The following filter replicates all employees where `empid >= 100` to the target database.

```
{
    "rules": [{
        "rule-type": "selection",
        "rule-id": "1",
        "rule-name": "1",
        "object-locator": {
            "schema-name": "test",
            "table-name": "employee"
        },
        "rule-action": "include",
        "filters": [{
            "filter-type": "source",
            "column-name": "empid",
            "filter-conditions": [{
               "filter-operator": "gte",
```

```
                "value": "100"
            }]
        }]
    }]
}
```

Example Multiple filter operators

The following filter applies multiple filter operators to a single column of data. The filter replicates all employees where `(empid <= 10)` OR `(empid is between 50 and 75)` OR `(empid >= 100)` to the target database.

```
{
    "rules": [{
        "rule-type": "selection",
        "rule-id": "1",
        "rule-name": "1",
        "object-locator": {
            "schema-name": "test",
            "table-name": "employee"
        },
        "rule-action": "include",
        "filters": [{
            "filter-type": "source",
            "column-name": "empid",
            "filter-conditions": [{
                "filter-operator": "lte",
                "value": "10"
            }, {
                "filter-operator": "between",
                "start-value": "50",
                "end-value": "75"
            }, {
                "filter-operator": "gte",
                "value": "100"
            }]
        }]
    }]
}
```

Example Multiple filters

The following filters apply multiple filters to two columns in a table. The filter replicates all employees where (`empid <= 100`) AND (`dept = tech`) to the target database.

```
{
    "rules": [{
        "rule-type": "selection",
        "rule-id": "1",
        "rule-name": "1",
        "object-locator": {
            "schema-name": "test",
            "table-name": "employee"
        },
        "rule-action": "include",
        "filters": [{
            "filter-type": "source",
            "column-name": "empid",
            "filter-conditions": [{
                "filter-operator": "lte",
                "value": "100"
            }]
        }, {
            "filter-type": "source",
            "column-name": "dept",
            "filter-conditions": [{
                "filter-operator": "eq",
                "value": "tech"
            }]
        }]
    }]
}
```

Example Filtering NULL values

The following filter shows how to filter on empty values. It replicates all employees where `dept = NULL` 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": "dept",
            "filter-conditions": [{
                "filter-operator": "null"
            }]
        }]
    }]
}
```

Example Filtering using NOT operators

Some of the operators can be used in the negative form. The following filter replicates all employees where `(empid is < 50) OR (empid is > 75)` 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": "notbetween",
                "start-value": "50",
                "end-value": "75"
```

```
            }]
        }]
    }]
}
```

Example Using Mixed filters operators

Start with AWS DMS version 3.5.0, you can mix inclusive operators and negative operators.

The following filter replicates all employees where `(empid != 50) AND (dept is not NULL)` 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": "noteq",
                "value": "50"
            }]
        }, {
            "filter-type": "source",
            "column-name": "dept",
            "filter-conditions": [{
                "filter-operator": "notnull"
            }]
        }]
    }]
}
```

Note the following when using `null` with other filter operators:

- Using inclusive, negative and `null` filter conditions together within the same filter will not replicate records with `NULL` values.
- Using negative and `null` filter conditions together without inclusive filter conditions within the same filter will not replicate any data.
- Using negative filter conditions without a `null` filter condition set explicitly will not replicate records with `NULL` values.

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.SSS 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 filter 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": "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 and fix issues before you run a new or modified task. This allows you to avoid delays related to task failures caused by missing requirements or known limitations.

AWS DMS provides access to two different options for premigration assessments:

- **Data type assessment**: A legacy report that provides a limited scope of assessments.
- **Premigration assessment run**: Contains various types of individual assessments, including data type assessment results.

> **Note**
>
> If you choose a premigration assessment run, you don't need to choose a data type assessment separately.

These options are described in the following topics:

- Specifying, starting, and viewing premigration assessment runs: A premigration (recommended) 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 source and/or target database from the perspective of criteria 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 or can't 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 about periodic updates, see Specifying individual assessments.

- Starting and viewing data type assessments (Legacy): A data type (legacy) 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 problematic data types found in every schema and table in the source database that is selected for migration.

Creating prerequisites for premigration assessments

This section describes the Amazon S3 and IAM resources you need to create a premigration assessment.

> **Important**
>
> The following prerequisites are only required if you supply your own Amazon S3 bucket and IAM role.

Create an S3 bucket

AWS DMS stores premigration assessment reports in an S3 bucket. To create the S3 bucket, do the following:

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose **Create bucket**.
3. On the **Create bucket** page, enter a globally unique name that includes your sign-in name for the bucket, such as dms-bucket-*yoursignin*.
4. Choose the AWS Region for the DMS migration task.
5. Leave the remaining settings as they are, and choose **Create bucket**.

Create IAM resources

DMS uses an IAM role and policy to access the S3 bucket to store premigration assessment results.

To create the IAM policy, do the following:

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. In the **Create policy** page, choose the **JSON** tab.
5. Paste the following JSON code into the editor, replacing the example code. Replace *amzn-s3-demo-bucket* with the name of the Amazon S3 bucket that you created in the previous section.

   JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:PutObject",
            "s3:DeleteObject",
            "s3:GetObject",
            "s3:PutObjectTagging"
         ],
         "Resource":[
            "arn:aws:s3:::amzn-s3-demo-bucket/*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:ListBucket",
            "s3:GetBucketLocation"
         ],
         "Resource":[
            "arn:aws:s3:::amzn-s3-demo-bucket"
```

```
            ]
        }
    ]
}
```

6. Choose **Next: Tags**, then choose and **Next: Review**.
7. Enter `DMSPremigrationAssessmentS3Policy` for **Name***, and then choose **Create policy**.

To create the IAM role, do the following:

1. In the IAM console, in the navigation pane, choose **Roles**.
2. Choose **Create role**.
3. On the **Select trusted entity** page, for **Trusted entity type**, choose **AWS Service**. For **Use cases for other AWS services**, choose **DMS.**
4. Check the **DMS** check box, and then choose **Next.**
5. On the **Add permissions** page, choose **DMSPremigrationAssessmentS3Policy**. Choose **Next**.
6. On the **Name, review, and create** page, enter `DMSPremigrationAssessmentS3Role` for **Role name**, then choose **Create role**.
7. On the **Roles** page, enter `DMSPremigrationAssessmentS3Role` for **Role name**. Choose **DMSPremigrationAssessmentS3Role**.
8. On the **DMSPremigrationAssessmentS3Role** page, choose the **Trust relationships** tab. Choose **Edit trust policy**.
9. On the **Edit trust policy** page, paste the following JSON into the editor, replacing the existing text.

   JSON

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

```
        }
    ]
}
```

This policy grants the `sts:AssumeRole` permission to DMS to put the premigration assessment run results into the S3 bucket.

10. Choose **Update policy**.

Specifying, starting, and viewing premigration assessment runs

A premigration assessment 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.

Specifying individual assessments

When creating a new assessment run, you can choose to run some or all of the individual assessments that are applicable to your task configuration.

AWS DMS supports premigration assessment runs for the following relational source and target database engines:

- Oracle assessments
- Sql Server assessments
- MySQL assessments (includes MariaDB and Amazon Aurora MySQL-Compatible Edition)
- PostgreSQL assessments (includes Amazon Aurora PostgreSQL-Compatible Edition)
- the section called “MariaDB assessments”
- the section called “Db2 LUW Assessments”

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:

   - To create a new task and assess it, 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, the **Premigration assessment run** checkbox is checked. This page contains the 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.

   - To assess an existing task, 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.

2. Enter a unique name for your assessment run, or leave the default value.
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.
4. Search for and choose an Amazon S3 bucket and folder in your account to store your assessment result report. For information about setting up resources for assessment runs, see Creating prerequisites for premigration assessments.
5. Select or enter an IAM role with full account access to your chosen Amazon S3 bucket and folder. For information about setting up resources for assessment runs, see Creating prerequisites for premigration assessments.
6. Optionally choose a setting to encrypt the assessment result report in your Amazon S3 bucket. For information about S3 bucket encryption, see Setting default server-side encryption behavior for Amazon S3 buckets .
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 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 **Premigration assessments** section on that page showing results of the the assessment runs, listed by name, in reverse chronological order. The latest result appears at the top of the list. Choose the name of the assessment run whose results you want to view.

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. This status takes priority over all other statuses, including error conditions.
- `"error-provisioning"` – An internal error occurred while resources were provisioned (during `provisioning` status). This status is only assigned when no individual assessments have a failed status, as provisioning errors may have prevented assessments from running that could have resulted in failed validations.
- `"error-executing"` – An internal error occurred while individual assessments ran (during `running` status). This status is only assigned when no individual assessments have a failed

status, as error conditions may have prevented assessments from completing that could have resulted in failed validations.

- `"invalid state"` – The assessment run is in an unknown state.
- `"passed"` – All individual assessments have completedsuccessfully with no failed, warning, or error statuses.
- `"provisioning"` – Resources required to run individual assessments are being provisioned.
- `"running"` – Individual assessments are being ran.
- `"starting"` – The assessment run is starting, but resources are not yet being provisioned for individual assessments.
- `"warning"` – At least one individual assessment completed with a `warning` status, and no assessments have failed or error statuses.

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 with a warning status.
- `"skipped"` – The individual assessment were skipped during the assessment run.

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 of the assessment run. 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 start and view the results of premigration assessment runs for an existing migration task, you can run the following CLI commands and AWS DMS API 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 doesn't delete, a single premigration assessment run.
- CLI: `delete-replication-task-assessment-run`, API: `DeleteReplicationTaskAssessmentRun` – Deletes the record of a single premigration assessment run.

Individual assessments

This section describes individual premigration assessments.

To create an individual premigration assessment using the AWS DMS API, use the listed API key for the `IncludeOnly` parameter of the StartReplicationTaskAssessmentRun action.

Topics

Assessments for all endpoint types

This section describes individual premigration assessments for all endpoint types.

Topics

Unsupported data types

API key: `unsupported-data-types-in-source`

Checks for data types in the source endpoint that DMS doesn't support. Not all data types can be migrated between engines.

Large objects (LOBs) are used but target LOB columns are not nullable

API key: `full-lob-not-nullable-at-target`

Checks for the nullability of a LOB column in the target when the replication usese full LOB mode or inline LOB mode. DMS requires a LOB column to be null when using these LOB modes. This assessment requires the source and target databases to be relational.

Source table with Large objects (LOBs) but without primary keys or unique constraints

API key: `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. A table must have a primary key or a unique key for DMS to migrate LOBs. This assessment requires the source database to be relational.

Source table without primary key for CDC or full load and CDC tasks only

API key: `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. A lack of a primary key or a unique key can cause performance issues during the CDC migration. This assessment requires the source database to be relational, and the migration type to include CDC.

Target table without primary keys for CDC tasks only

API key: `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. A lack of a primary key or a unique key can cause full table scans in the target when DMS applies updates and deletes. This can result in performance issues during the CDC migration. This assessment requires the target database to be relational, and the migration type to include CDC.

Unsupported source primary key types - composite primary keys

API key: `unsupported-source-pk-type-for-elasticsearch-target`

Checks for the presence of composite primary keys in source tables when migrating to Amazon OpenSearch Service. The primary key of the source table must consist of a single column. This assessment requires the source database to be relational, and the target database to be DynamoDB.

> **Note**
>
> DMS supports migrating a source database to an OpenSearch Service target where the source primary key consists of multiple columns.

Oracle assessments

For more information about permissions when using Oracle as a source, see User account privileges required on a self-managed Oracle source for AWS DMS or User account privileges required on an AWS-managed Oracle source for AWS DMS.

Note

This section describes individual premigration assessments for migration tasks that use Oracle as a source or a target for AWS DMS.
If you are using self-managed Oracle database as a source for AWS DMS, please use following permission set:

```
grant select on gv_$parameter to dms_user;
                    grant select on v_$instance to dms_user;
                    grant select on v_$version to dms_user;
                    grant select on gv_$ASM_DISKGROUP to dms_user;
                    grant select on gv_$database to dms_user;
                    grant select on DBA_DB_LINKS to to dms_user;
                    grant select on gv_$log_History to dms_user;
                    grant select on gv_$log to dms_user;
                    grant select on dba_types to dms_user;
                    grant select on dba_users to dms_user;
                    grant select on dba_directories to dms_user;
                    grant execute on SYS.DBMS_XMLGEN to dms_user;
```

Additional permissions is required if you are using a self-managed Oracle database as a source for AWS DMS Serverless:

```
grant select on dba_segments to dms_user;
                    grant select on v_$tablespace to dms_user;
                    grant select on dba_tab_subpartitions to dms_user;
                    grant select on dba_extents to dms_user;
```

If you are using an AWS-managed Oracle database as a source for AWS DMS, use the following set of permissions:

```
EXEC RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('V_$PARAMETER', 'dms_user',
 'SELECT');
                    EXEC RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('V_$INSTANCE',
 'dms_user', 'SELECT');
```

```
                    EXEC RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('V_
$VERSION','dms_user', 'SELECT');
                    EXEC RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('GV_
$ASM_DISKGROUP','dms_user', 'SELECT');
                    EXEC RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('GV_
$DATABASE','dms_user', 'SELECT');
                    EXEC
 RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('DBA_DB_LINKS','dms_user', 'SELECT');
                    EXEC RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('GV_
$LOG_HISTORY','dms_user', 'SELECT');
                    EXEC RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('GV_
$LOG','dms_user', 'SELECT');
                    EXEC
 RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('DBA_TYPES','dms_user', 'SELECT');
                    EXEC
 RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('DBA_USERS','dms_user', 'SELECT');
                    EXEC
 RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('DBA_DIRECTORIES','dms_user',
 'SELECT');
                    GRANT SELECT ON RDSADMIN.RDS_CONFIGURATION to dms_user;
                    GRANT EXECUTE ON SYS.DBMS_XMLGEN TO dms_user;
```

Additional permissions is required if you are using an AWS-managed Oracle database as a source for AWS DMS Serverless:

```
EXEC RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('DBA_SEGMENTS','dms_user',
 'SELECT');
                    EXEC
 RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('DBA_TAB_SUBPARTITIONS','dms_user',
 'SELECT');
                    EXEC
 RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('DBA_EXTENTS','dms_user', 'SELECT');
                    EXEC RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('V_
$TABLESPACE','dms_user', 'SELECT');
```

If you are using a self-managed Oracle database as a target for AWS DMS, use the following set of permissions:

```
grant select on v_$instance to dms_user;
                    grant execute on SYS.DBMS_XMLGEN to dms_user;
```

If you are using an AWS-managed Oracle database as a target for AWS DMS, use the following set of permissions:

```
EXEC RDSADMIN.RDSADMIN_UTIL.GRANT_SYS_OBJECT('V_$INSTANCE', 'dms_user',
 'SELECT');
                    GRANT EXECUTE ON SYS.DBMS_XMLGEN TO dms_user;
```

Topics

- Validate whether index-organized tables with an overflow segment are being used on the tables included in the task scope.
- Validate if multilevel nesting tables are used on the tables included in the task scope.
- Validate if invisible columns are used on the tables included in the task scope.
- Validate if materialized views based on a ROWID column are used on the tables included in the task scope.
- Validate if Active Data Guard DML Redirect feature is used.
- Validate if Hybrid Partitioned Tables are used.
- Validate if schema-only Oracle accounts are used
- Validate if Virtual Columns are used
- Validate whether table names defined in the task scope contain apostrophes.
- Validate whether the columns defined in the task scope have XMLType, Long, or Long Raw datatypes and verify the LOB mode configuration in the task settings.
- Validate whether the source Oracle version is supported by AWS DMS.
- Validate whether the target Oracle version is supported by AWS DMS.
- Validate whether the DMS user has the required permissions to use data validation.
- Validate if the DMS user has permissions to use Binary Reader with Oracle ASM
- Validate if the DMS user has permissions to use Binary Reader with Oracle non-ASM
- Validate if the DMS user has permissions to use Binary Reader with CopyToTempFolder method
- Validate if the DMS user has permissions to use Oracle Standby as a Source
- Validate if the DMS source is connected to an application container PDB
- Validate if the table has XML datatypes included in the task scope.
- Validate whether archivelog mode is enabled on the source database.
- Validates the archivelog retention for RDS Oracle.
- Validate if the table has Extended datatypes included in the task scope.
- Validate the length of the object name included in the task scope.
- Validate if the DMS source is connected to an Oracle PDB
- Validate if the table has spatial columns included in the task scope.
- Validate if the DMS source is connected to an Oracle standby.
- Validate if the source database tablespace is encrypted using TDE.

- Validates if the source database uses Automatic Storage Management (ASM)
- Validate if batch apply is enabled and whether the table on the target Oracle database has parallelism enabled at the table or index level
- Recommend "Bulk Array Size" parameter by validating the tables in the task scope
- Validate if HandleCollationDiff task setting is Configured
- Validate if table has primary key or unique index and its state is VALID when DMS validation is enabled
- Validate if Binary Reader is used for Oracle Standby as a source
- Validate if the AWS DMS user has the required directory permissions to replicate data from an Oracle RDS Standby database.
- Validate the type of Oracle Standby used for replication
- Validate if required directories are created for RDS Oracle standby
- Validate if Primary Key or Unique Index exists on target for Batch Apply
- Validate if both Primary Key and Unique index exist on target for Batch Apply
- Validate if unsupported HCC levels are used for Full Load
- Validate if unsupported HCC levels are used for Full Load with CDC
- Validate if unsupported HCC compression used for CDC
- CDC Recommendation based on source compression method
- Check if batch apply is enabled and validate whether the table has more than 999 columns
- Check Transformation Rule for Digits Randomize
- Check Transformation Rule for Digits mask
- Check Transformation Rule for Hashing mask
- Verify that Data Validation task settings and Data Masking Digit randomization are not enabled simultaneously
- Verify that Data Validation task settings and Data Masking Hashing mask are not enabled simultaneously
- Verify that Data Validation task settings and Data Masking Digit mask are not enabled simultaneously
- Validate that replication to a streaming target does not include LOBs or extended data type columns
- Validate that CDC-only task is configured to use the OpenTransactionWindow endpoint setting

- Validate that at least one selected object exists in the source database
- Validate that target foreign key constraints are disabled for migration
- Validate that the Oracle database and AWS DMS versions are compatible
- Validate that secondary constraints and indexes (non-primary) are present in the source database
- Validate that session timeout settings (IDLE_TIME) are set to UNLIMITED
- Validate that the AWS DMS user has all required permissions on the source database
- Validate that XMLTYPE or LOB columns exist when Oracle LogMiner is used
- Validate that the target endpoint is not a read replica
- Validate that the Oracle target does not have CONTEXT indexes when using direct path load
- Validate that FailTasksOnLobTruncation is enabled when using limited LOB mode with existing LOB columns
- Validate that EnableHomogenousPartitionOps endpoint setting is enabled

Validate that limited LOB mode only is used when `BatchApplyEnabled` is enabled

API key: `oracle-batch-apply-lob-mode`

This premigration assessment validates whether tables in the DMS task includes LOB columns. If LOB columns are included in the scope of the task, you must use `BatchApplyEnabled` together with limited LOB mode only.

For more information, see Target metadata task settings.

Validate if tables on the source has columns without scale specified for the Number data type

API key: `oracle-number-columns-without-scale`

This premigration assessment validates whether the DMS task includes columns of NUMBER data type without scale specified. We recommend that you set the endpoint setting `NumberDataTypeScale` to the value specified in the assessment report.

For more information, see Endpoint settings when using Oracle as a source for AWS DMS.

Validate triggers on the target database

API key: `oracle-target-triggers-are-enabled`

This premigration assessment validates whether triggers are enabled on the target database. The assessment will fail if triggers are enabled. We recommend that you disable or remove the triggers during the migration.

For more information, see [For more information, see DMS best practices].

Validate if source has archivelog `DEST_ID` set to 0

API key: `oracle-zero-archive-log-dest-id`

This premigration assessment validates whether endpoint extra connection attribute `useZeroDestid=true` is set for source if archived log `DEST_ID` is set to 0.

For more information, see [How to handle AWS DMS replication when used with Oracle database in fail-over scenarios].

Validate if secondary indexes are enabled on the target database during full-load

API key: `oracle-check-secondary-indexes`

This premigration assessment validates whether secondary indexes are enabled during a full-load on the target database. We recommend that you disable or remove the secondary indexes during full-load.

For more information, [Best practices for AWS Database Migration Service].

Validate if tables used in the DMS task scope with BatchApplyEnabled have more than 999 columns

API key: `oracle-batch-apply-lob-999`

Tables with batch optimized apply mode enabled can't have more than a total of 999 columns. Tables that have more than 999 columns will cause AWS DMS to process the batch one by one, which increases latency. DMS uses the formula **2 * columns_in_original_table + columns_in_primary_key <= 999** to calculate the total number of columns per table supported in batch-optimized apply mode.

For more information, see [Limitations on Oracle as a target for AWS Database Migration Service].

Check supplemental logging on database level

API key: `oracle-supplemental-db-level`

This premigration assessment validates if minimum supplemental logging is enabled at the database level. You must enable supplemental logging to use an Oracle database as a source for migration.

To enable supplemental logging, use the following query:

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

For more information, see Setting up supplemental logging.

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

Validate if required DB link is created for Standby

API key: `oracle-validate-standby-dblink`

This premigration assessment validates if Dblink is created for the Oracle standby database source. AWSDMS_DBLINK is a prerequisite for using a standby database as a source. When using Oracle Standby as a source, AWS DMS does not validate open transactions by default.

For more information, see Working with a self-managed Oracle database as a source for AWS DMS.

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

Oracle validation for LOB datatype and if binary reader is configured

API key: `oracle-binary-lob-source-validation`

This premigration assessment validates if Oracle LogMiner is used for an Oracle database endpoint version 12c or later. AWS DMS does not support Oracle LogMiner for migrations of LOB columns from Oracle databases version 12c. This assessment also checks for the presence of LOB columns and provides appropriate recommendations.

To configure your migration to not use Oracle LogMiner, add the following configuration to your source endpoint:

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

For more information, see Using Oracle LogMiner or AWS DMS Binary Reader for CDC.

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

Validate if the database is CDB

API key: `oracle-validate-cdb`

This premigration assessment validates if the database is a container database. AWS DMS doesn't support the multi-tenant container root database (CDB$ROOT).

> **Note**
>
> This assessment is only required for Oracle versions 12.1.0.1 or later. This assessment is not applicable for Oracle versions prior to 12.1.0.1.

For more information, see [Limitations on using Oracle as a source for AWS DMS](#).

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

Check the Oracle Database Edition

API key: `oracle-check-cdc-support-express-edition`

This premigration assessment validates if the Oracle source database is Express Edition. AWS DMS doesn't support CDC for Oracle Express Edition (Oracle Database XE) version 18.0 and later.

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

Validate Oracle CDC method for DMS

API key: `oracle-recommendation-cdc-method`

This premigration assessment validates redo log generation for the last seven days, and makes a recommendation whether to use AWS DMS Binary Reader or Oracle LogMiner for CDC.

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

For more information about deciding which CDC method to use, see Using Oracle LogMiner or AWS DMS Binary Reader for CDC.

Validate Oracle RAC configuration for DMS

API key: `oracle-check-rac`

This premigration assessment validates if the oracle database is a Real Application Cluster. Real Application Cluster databases must be configured correctly. If the database is based on RAC, we recommend that you use AWS DMS Binary Reader for CDC rather than Oracle LogMiner.

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

For more information, see Using Oracle LogMiner or AWS DMS Binary Reader for CDC.

Validate if DMS user has permissions on target

API key: `oracle-validate-permissions-on-target`

This premigration assessment validates whether DMS users have all the required permissions on the target database.

Validate if supplemental logging is required for all columns

API key: `oracle-validate-supplemental-logging-all-columns`

This premigration assessment validates, for the tables mentioned in the task scope, whether supplemental logging has been added to all columns of tables without a primary or unique key. Without supplemental logging on all columns for a table lacking a primary or unique key, the before-and-after image of the data won't be available in the redo logs. DMS requires supplemental logging for tables without a primary or unique key to generate DML statements.

Validate if supplemental logging is enabled on tables with Primary or Unique keys

API key: `oracle-validate-supplemental-logging-for-pk`

This premigration assessment validates whether supplemental logging is enabled for tables with a primary key or unique index and also checks if `AddSupplementalLogging` is enabled at the endpoint level. To ensure DMS can replicate changes, you can either manually add supplemental logging on the table level based on the primary key or unique key or utilize the endpoint setting

`AddSupplementalLogging = true` with a DMS user having the ALTER permission on any replicated table.

Validate if there are SecureFile LOBs and the task is configured for Full LOB mode

API key: `oracle-validate-securefile-lobs`

This premigration assessment checks for the presence of SecureFile LOBs in tables within the task scope and verifies their LOB settings. Consider assigning LOB tables to a separate task to enhance performance, as running tasks in full LOB mode may result in slower performance.

Validate whether Function-Based Indexes are being used within the tables included in the task scope.

API key: `oracle-validate-function-based-indexes`

This premigration assessment checks for function-based indexes on tables within the task scope. Note that AWS DMS doesn't support replicating function-based indexes. Consider creating the indexes after your migration on your target database.

Validate whether global temporary tables are being used on the tables included in the task scope.

API key: `oracle-validate-global-temporary-tables`

This premigration assessment checks whether global temporary tables are used within the task table-mapping scope. Note that AWS DMS doesn't support migrating or replicating global temporary tables.

Validate whether index-organized tables with an overflow segment are being used on the tables included in the task scope.

API key: `oracle-validate-iot-overflow-segments`

Validate whether index-organized tables with an overflow segment are being used on the tables included in the task scope. AWS DMS doesn't support CDC for index-organized tables with an overflow segment.

Validate if multilevel nesting tables are used on the tables included in the task scope.

API key: `oracle-validate-more-than-one-nesting-table-level`

This premigration assessment checks the nesting level of the nested table used on the task scope. AWS DMS supports only one level of table nesting.

Validate if invisible columns are used on the tables included in the task scope.

API key: `oracle-validate-invisible-columns`

This premigration assessment validates whether the tables used in the task scope have invisible columns. AWS DMS doesn't migrate data from invisible columns in your source database. To migrate the columns that are invisible, you need to modify them to be visible.

Validate if materialized views based on a ROWID column are used on the tables included in the task scope.

API key: `oracle-validate-rowid-based-materialized-views`

This premigration assessment validates whether the materialized views used in the migration are created based on the ROWID column. AWS DMS doesn't support the ROWID data type or materialized views based on a ROWID column.

Validate if Active Data Guard DML Redirect feature is used.

API key: `oracle-validate-adg-redirect-dml`

This premigration assessment validates whether the Active Data Guard DML Redirect feature is used. When using Oracle 19.0 as the source, AWS DMS doesn't support the Data Guard DML Redirect feature.

Validate if Hybrid Partitioned Tables are used.

API key: `oracle-validate-hybrid-partitioned-tables`

This premigration assessment validates whether hybrid partitioned tables are used for the tables defined in the task scope.

Validate if schema-only Oracle accounts are used

API key: `oracle-validate-schema-only-accounts`

This premigration assessment validates whether Schema-Only Accounts are found within the task scope.

Validate if Virtual Columns are used

API key: `oracle-validate-virtual-columns`

This premigration assessment validates whether the Oracle Instance has Virtual Columns in tables within the task scope.

Validate whether table names defined in the task scope contain apostrophes.

API key: `oracle-validate-names-with-apostrophes`

This premigration assessment validates whether the tables used in the task scope contain apostrophes. AWS DMS doesn't replicate tables with names containing apostrophes. If identified, consider renaming such tables. Alternatively, you could create a view or materialized view without apostrophes to load these tables.

Validate whether the columns defined in the task scope have `XMLType`, `Long`, or `Long Raw` datatypes and verify the LOB mode configuration in the task settings.

API key: `oracle-validate-limited-lob-mode-for-longs`

This premigration assessment validates whether the tables defined in the task scope have the datatypes `XMLType`, `Long`, or `Long Raw`, and checks if the task setting is configured to use Limited Size LOB Mode. AWS DMS doesn't support replicating these datatypes using FULL LOB mode. Consider changing the task setting to use Limited Size LOB mode upon identifying tables with such datatypes.

Validate whether the source Oracle version is supported by AWS DMS.

API key: `oracle-validate-supported-versions-of-source`

This premigration assessment validates if the source Oracle instance version is supported by AWS DMS.

Validate whether the target Oracle version is supported by AWS DMS.

API key: `oracle-validate-supported-versions-of-target`

This premigration assessment validates if the target Oracle instance version is supported by AWS DMS.

Validate whether the DMS user has the required permissions to use data validation.

API key: `oracle-prerequisites-privileges-of-validation-feature`

This premigration assessment validates whether the DMS user has the necessary privileges to use DMS Data Validation. You can ignore enabling this validation if you do not intend to use data validation.

Validate if the DMS user has permissions to use Binary Reader with Oracle ASM

API key: `oracle-prerequisites-privileges-of-binary-reader-asm`

This premigration assessment validates whether the DMS user has the necessary privileges to use Binary Reader on the Oracle ASM instance. You can ignore enabling this assessment if your source is not an Oracle ASM instance or if you are not using Binary Reader for CDC.

Validate if the DMS user has permissions to use Binary Reader with Oracle non-ASM

API key: `oracle-prerequisites-privileges-of-binary-reader-non-asm`

This premigration assessment validates whether the DMS user has the necessary privileges to use Binary Reader on the Oracle non-ASM instance. This assessment is only valid if you have an Oracle non-ASM instance.

Validate if the DMS user has permissions to use Binary Reader with CopyToTempFolder method

API key: `oracle-prerequisites-privileges-of-binary-reader-copy-to-temp-folder`

This premigration assessment validates whether the DMS user has the necessary privileges to use the Binary Reader with the 'Copy to Temp Folder' method. This assessment is relevant only if you are planning to use CopyToTempFolder to read CDC changes while using the Binary Reader, and have an ASM instance connected to the source. You can ignore enabling this assessment if you don't intend to use the CopyToTempFolder feature.

We recommend not using the CopyToTempFolder feature because it is deprecated.

Validate if the DMS user has permissions to use Oracle Standby as a Source

API key: `oracle-prerequisites-privileges-of-standby-as-source`

This premigration assessment validates whether the DMS user has the necessary privileges to use a StandBy Oracle Instance as a source. You can ignore enabling this assessment if you don't intend to use a StandBy Oracle Instance as a source.

Validate if the DMS source is connected to an application container PDB

API key: `oracle-check-app-pdb`

This premigration assessment validates whether the DMS source is connected to an application container PDB. DMS doesn't support replication from an application container PDB.

Validate if the table has XML datatypes included in the task scope.

API key: `oracle-check-xml-columns`

This premigration assessment validates whether the tables used in the task scope have XML datatypes. It also checks if the task is configured for Limited LOB mode when the table contains an XML datatype. DMS only supports Limited LOB mode for migrating Oracle XML Columns.

Validate whether archivelog mode is enabled on the source database.

API key: `oracle-check-archivelog-mode`

This premigration assessment validates whether archivelog mode is enabled on the source database. Enabling archive log mode on the source database is necessary for DMS to replicate changes.

Validates the archivelog retention for RDS Oracle.

API key: `oracle-check-archivelog-retention-rds`

This premigration assessment validates whether archivelog retention on your RDS Oracle database is configured for at least 24 hours.

Validate if the table has Extended datatypes included in the task scope.

API key: `oracle-check-extended-columns`

This premigration assessment validates whether the tables used in the task scope have extended datatypes. Note that extended datatypes are supported only with DMS version 3.5 onwards.

Validate the length of the object name included in the task scope.

API key: `oracle-check-object-30-bytes-limit`

This premigration assessment validates whether the length of the object name exceeds 30 bytes. DMS doesn't support long object names (over 30 bytes).

Validate if the DMS source is connected to an Oracle PDB

API key: `oracle-check-pdb-enabled`

This premigration assessment validates whether the DMS source is connected to a PDB. DMS supports CDC only when using the Binary Reader with Oracle PDB as the source. The assessment also evaluates if the task is configured to use the binary reader when DMS is connected to Oracle PDB.

Validate if the table has spatial columns included in the task scope.

API key: `oracle-check-spatial-columns`

This premigration assessment validates whether the table has spatial columns included in the task scope. DMS supports Spatial datatypes only using Full LOB mode. The assessment also evaluates whether the task is configured to use Full LOB mode when DMS identifies spatial columns.

Validate if the DMS source is connected to an Oracle standby.

API key: `oracle-check-standby-db`

This premigration assessment validates whether the source is connected to an Oracle standby. DMS supports CDC only when using the binary reader with Oracle Standby as the source. The assessment also evaluates if the task is configured to use binary reader when DMS is connected to Oracle Standby.

Validate if the source database tablespace is encrypted using TDE.

API key: `oracle-check-tde-enabled`

This premigration assessment validates whether the source has TDE Encryption enabled on the tablespace. DMS supports TDE only with encrypted tablespaces when using Oracle LogMiner for RDS Oracle.

Validates if the source database uses Automatic Storage Management (ASM)

API key: `oracle-check-asm`

This premigration assessment detects if your source database uses ASM. For optimal performance, configure Binary Reader with `parallelASMReadThreads` and `readAheadBlocks` parameters in your endpoint settings. These settings enhance data extraction performance when working with ASM storage

For more information, see [Using Oracle LogMiner or AWS DMS Binary Reader for CDC].

Validate if batch apply is enabled and whether the table on the target Oracle database has parallelism enabled at the table or index level

API key: `oracle-check-degree-of-parallelism`

AWS DMS validates that the table in the target database has any parallelism enabled. Having parallelism enabled on the target database causes the batch process to fail. Therefore, it is required to disable parallelism at the table or index level when using the batch apply feature.

Recommend "Bulk Array Size" parameter by validating the tables in the task scope

API key: `oracle-check-bulk-array-size`

This assessment recommends setting the `BulkArraySize` ECA (Extra Connection Attribute) if there are no tables with LOB (Large Object) data types found within the task scope. Setting the `BulkArraySize` ECA can improve performance of the full load phase of the migration. You can set the bulk array size using the ECA on the Source/Target endpoint to get optimal performance during the full load phase of the migration.

Validate if HandleCollationDiff task setting is Configured

API key: `oracle-check-handlecollationdiff`

This assessment validates if DMS task is configured for validation and recommend `HandleCollationDiff` task setting to avoid any incorrect validation results while validating data between Oracle and PostgreSQL.

For more information, see the section called "Data validation task settings".

Validate if table has primary key or unique index and its state is VALID when DMS validation is enabled

API key: `oracle-check-pk-validity`

Data validation requires that the table has a primary key or unique index on both source and target.

For more information, see *Data validation*.

Validate if Binary Reader is used for Oracle Standby as a source

API key: `oracle-check-binary-reader`

This assessment validates if the source database is a standby database and is using the Binary Reader for Change Data Capture (CDC).

For more information, see the section called “Using Oracle as a source”.

Validate if the AWS DMS user has the required directory permissions to replicate data from an Oracle RDS Standby database.

API key: `oracle-check-directory-permissions`

This assessment validates if the AWS DMS user has the required read privileges on the `ARCHIVELOG_DIR_%` and `ONLINELOG_DIR_%` directories when the source database is an Oracle RDS Standby.

For more information, see the section called “Working with an AWS-managed Oracle source”.

Validate the type of Oracle Standby used for replication

API key: `oracle-check-physical-standby-with-apply`

This assessment validates the type of Oracle standby database used for the AWS DMS replication. AWS DMS only supports Physical standby databases, which must be opened in Read Only mode with the redo logs being applied automatically. AWS DMS does not support Snapshot or Logical standby databases for replication.

For more information, see the section called “Using Oracle Standby as a source with Binary Reader”.

Validate if required directories are created for RDS Oracle standby

API key: `oracle-check-rds-standby-directories`

This assessment validates if the required oracle directories are created for archive logs and online logs on the RDS Standby instance.

For more information, see the section called “Using an Oracle Standby source with Binary Reader for CDC”.

Validate if Primary Key or Unique Index exists on target for Batch Apply

API key: `oracle-check-batch-apply-target-pk-ui-absence`

Batch apply is only supported on tables with Primary Keys or Unique Indexes on the target table. Tables without Primary Keys or Unique Indexes causes the batch to fail, and changes are processed one by one. It is advisable to move such tables to their own tasks and utilize transactional apply mode instead. Alternatively, you can create a unique key on the target table.

For more information, see the section called "Using Oracle as a target".

Validate if both Primary Key and Unique index exist on target for Batch Apply

API key: `oracle-check-batch-apply-target-pk-ui-simultaneously`

Batch apply is only supported on tables with Primary Keys or Unique Indexes on the target table. Tables with Primary Keys and Unique Indexes simultaneously causes the batch to fail, and changes are processed one by one. It is advisable to move such tables to their own tasks and utilize transactional apply mode instead. Alternatively, you can drop a unique key(s) or primary key on the target table and rebuild it if you are doing migration.

For more information, see the section called "Using Oracle as a target".

Validate if unsupported HCC levels are used for Full Load

API key: `oracle-check-binary-reader-hcc-full-load`

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.

For more information, see the section called "Compression methods on Oracle as a source".

Validate if unsupported HCC levels are used for Full Load with CDC

API key: `oracle-check-binary-reader-hcc-full-load-and-cdc`

The Oracle source endpoint is configured to use Binary Reader, HCC with Query low is supported for Full Load task only.

the section called "Compression methods on Oracle as a source"

Validate if unsupported HCC compression used for CDC

API key: `oracle-check-binary-reader-hcc-cdc`

The Oracle source endpoint is configured to use Binary Reader. Binary Reader doesn't support Query low for tasks with CDC.

For more information, see the section called "Using Oracle LogMiner or AWS DMS Binary Reader for CDC".

CDC Recommendation based on source compression method

API key: `oracle-recommend-cdc-method-by-compression`

Compressed objects are detected. Please navigate into the Result section of the specific assessment for further recommendation.

For more information, see the section called "Using Oracle LogMiner or AWS DMS Binary Reader for CDC".

Check if batch apply is enabled and validate whether the table has more than 999 columns

API key: `oracle-batch-apply-lob-999`

DMS uses the `2 * columns_in_original_table + columns_in_primary_key` formula to determine the number of columns on customer table. Based on this formula, we have identified tables with more than 999 columns. This impacts the batch process, causing it to fail and switch to one-by-one mode.

For more information, see the section called "Limitations on Oracle as a target".

Check Transformation Rule for Digits Randomize

API key: `oracle-datamasking-digits-randomize`

This assessment validates whether columns used in table mappings are compatible with the Digits Randomize transformation rule. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying digits randomize transformations does not guarantee any uniqueness.

Check Transformation Rule for Digits mask

API key: `oracle-datamasking-digits-mask`

This assessment validates whether any columns used in the table mapping are not supported by the Digits Mask transformation rule. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying Digits Mask transformations to such columns could cause DMS task failures since uniqueness cannot be guaranteed.

Check Transformation Rule for Hashing mask

API key: `oracle-datamasking-hash-mask`

This assessment validates whether any of the columns used in the table mapping are not supported by the Hashing Mask transformation rule. It also checks if the length of the source column exceeds 64 characters. Ideally, the target column length should be greater than 64 characters to support hash masking. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying digits randomize transformations does not guarantee any uniqueness.

Verify that Data Validation task settings and Data Masking Digit randomization are not enabled simultaneously

API key: `all-to-all-validation-with-datamasking-digits-randomize`

This premigration assessment verifies that Data Validation setting and Data Masking Digit randomization are not simultaneously enabled, as these features are incompatible.

Verify that Data Validation task settings and Data Masking Hashing mask are not enabled simultaneously

API key: `all-to-all-validation-with-datamasking-hash-mask`

This premigration assessment verifies that Data Validation setting and Data Masking Hashing mask are not simultaneously enabled, as these features are incompatible.

Verify that Data Validation task settings and Data Masking Digit mask are not enabled simultaneously

API key: `all-to-all-validation-with-digit-mask`

This premigration assessment verifies that Data Validation setting and Data Masking Digit mask are not simultaneously enabled, as these features are incompatible.

Validate that replication to a streaming target does not include LOBs or extended data type columns

API key: `oracle-validate-lob-to-streaming-target`

This assessment identifies potential data loss when migrating LOB or extended data types to streaming target endpoints (such as S3, Kinesis, or Kafka). Oracle database does not track changes to these data types in its log files causing DMS to write `NULL` values to the streaming target. To prevent data loss, you can implement a '`before`' trigger on the source database which forces Oracle to log these changes.

Validate that CDC-only task is configured to use the `OpenTransactionWindow` endpoint setting

API key: `oracle-check-cdc-open-tx-window`

For CDC-only tasks, use `OpenTransactionWindow` to avoid missing data. For more information, see Creating tasks for ongoing replication using AWS DMS.

Validate that at least one selected object exists in the source database

API key: `all-check-source-selection-rules`

This premigration assessment verifies that at least one object specified in the selection rules exists in the source database, including pattern matching for wildcard-based rules.

Validate that target foreign key constraints are disabled for migration

API key: `oracle-target-foreign-key-constraints-check`

This premigration assessment detects active foreign key constraints on the target database that can cause migration failures (ORA-02291).

Validate that the Oracle database and AWS DMS versions are compatible

API key: `oracle-dms-compatibility-version-check`

This premigration assessment detects if your Oracle database version is incompatible with your AWS DMS version. This mismatch can cause task failures due to unsupported Oracle Redo compatibility settings.

Validate that secondary constraints and indexes (non-primary) are present in the source database

API key: `all-check-secondary-constraints`

This premigration assessment verifies that secondary constraints and indexes (foreign keys, check constraints, non-clustered indexes) are present in the source database.

Validate that session timeout settings (IDLE_TIME) are set to UNLIMITED

API key: `oracle-check-idle-time`

This premigration assessment verifies that the Oracle database `IDLE_TIME` parameter is set to `UNLIMITED` for the AWS DMS user. Limited session timeout can cause migration task failures due to connection timeouts.

Validate that the AWS DMS user has all required permissions on the source database

API key: `oracle-validate-permissions-on-source`

This premigration assessment verifies that the AWS DMS user has been configured with all required permissions on the source database.

Validate that XMLTYPE or LOB columns exist when Oracle LogMiner is used

API key: `oracle-update-lob-columns`

This premigration assessment warns that `XMLTYPE` or LOB columns exist in the source database when Oracle LogMiner is used.

Validate that the target endpoint is not a read replica

API key: `all-check-target-read-replica`

This premigration assessment verifies that the target endpoint is not configured as a read replica. AWS DMS requires write access to the target database and cannot replicate to read-only replicas.

Validate that the Oracle target does not have CONTEXT indexes when using direct path load

API key: `oracle-check-direct-path-context-indexes`

This premigration assessment validates whether target Oracle tables contain CONTEXT indexes. AWS DMS does not support CONTEXT indexes when using direct path in full-load mode. To avoid failure, disable direct path full-load mode or remove CONTEXT indexes before the load.

For more information, see [Limitations on using Oracle as a source for AWS DMS].

Validate that FailTasksOnLobTruncation is enabled when using limited LOB mode with existing LOB columns

API key: `oracle-pg-lobs-with-failtasksonlobtruncation`

This premigration assessment validates whether the `FailTasksOnLobTruncation` extra connection attribute is set to true when LOB columns are present in selected tables and limited LOB mode is specified. This setting fails the task if any LOB data is truncated during migration, preventing silent data loss.

For more information, see Endpoint settings when using Oracle as a source for AWS DMS.

Validate that `EnableHomogenousPartitionOps` endpoint setting is enabled

API key: `oracle-homogenous-partition-ops`

This premigration assessment validates that the `EnableHomogenousPartitionOps` endpoint setting is enabled for Oracle homogeneous migrations. This setting is required for AWS DMS to replicate Oracle partition and subpartition DDL operations.

For more information, see Limitations on using Oracle as a source for AWS DMS.

Sql Server assessments

This section describes individual premigration assessments for migration tasks that use a Microsoft SQL Server source endpoint.

Topics

- Check if the source database has backups stored at a URL or on Windows Azure.
- Check if the source database has backups on multiple disks
- Check if the source database has at least one full backup
- Check if the source database has sparse columns and columnar structure compression.
- Check if the source database instance has server level auditing for SQL Server 2008 or SQL Server 2008 R2
- Check if the source database has geometry columns for full LOB mode
- Check if the source database has columns with the Identity property.
- Check if the DMS user has FULL LOAD permissions
- Check if the DMS user has FULL LOAD and CDC or CDC only permissions
- Check whether MS-Replication is enabled for CDC on on-premises or EC2 databases.
- Check if the DMS user has the VIEW DEFINITION permission.
- Check if the DMS user has the VIEW DATABASE STATE permission on the MASTER database for users without the Sysadmin role.
- Check if the DMS user has the VIEW SERVER STATE permission.
- Validate if text repl size parameter is not unlimited
- Validate if Primary Key or Unique Index exist on target for Batch Apply
- Validate if both Primary Key and Unique index exist on target when batch apply enabled
- Validate if table has primary key or unique index when DMS validation is enabled
- Validate if AWS DMS user has necessary privileges to the target
- Recommendation on using MaxFullLoadSubTasks setting
- Check Transformation Rule for Digits Randomize
- Check Transformation Rule for Digits mask
- Check Transformation Rule for Hashing mask
- Verify that Data Validation task settings and Data Masking Digit randomization are not enabled simultaneously
- Verify that Data Validation task settings and Data Masking Hashing mask are not enabled simultaneously
- Verify that Data Validation task settings and Data Masking Digit mask are not enabled simultaneously
- Validate that at least one selected object exists in the source database

- Validate that secondary constraints and indexes (non-primary) are present in the source database
- Validate that target endpoint is not a read replica
- Validate backup chain
- Check database user permissions for applying EXCLUSIVE_AUTOMATIC_TRUNCATION safeguard policy
- Validate that secondary node connection and required safeguard attributes for AWS DMS source endpoint
- Validate that endpoint has all required extra connection attributes (ECAs) when AWS DMS is connected to secondary node

Validate if secondary indexes are enabled on the target database during full-load

API key: `sqlserver-check-secondary-indexes`

This premigration assessment validates whether secondary indexes are enabled during full-load on the target database. We recommend that you disable or remove secondary indexes.

For more information, Best practices for AWS Database Migration Service.

Validate that limited LOB mode only is used when `BatchApplyEnabled` is set to true

API key: `sqlserver-batch-apply-lob-mode`

This premigration assessment validates whether the DMS task includes LOB columns. If LOB columns are included into the scope of the task, you must use `BatchApplyEnabled` together with limited LOB mode only. We recommend that you create separate tasks for such tables and use transactional apply mode instead.

For more information, see How can I use the DMS batch apply feature to improve CDC replication performance?.

Validate if target database has any triggers enabled on tables in the scope of the task

API key: `sqlserver-check-for-triggers`

AWS DMS identified triggers in the target database that can impact the performance of the full-load DMS task and latency on target. Make sure that these triggers are disabled during a task run and enabled during the cut-over period.

Check if tables in task scope contain computed columns

API key: `sqlserver-check-for-computed-fields`

This premigration assessment checks for the presence of computed columns. AWS DMS doesn't support replicating changes from SQL Server computed columns.

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if tables in task scope have column store indexes

API key: `sqlserver-check-for-columnstore-indexes`

This premigration assessment checks for the presence of tables with columnstore indexes. AWS DMS doesn't support replicating changes from SQL Server tables with columnstore indexes.

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if memory optimized tables are a part of the task scope

API key: `sqlserver-check-for-memory-optimized-tables`

This premigration assessment checks for the presence of memory-optimized tables. AWS DMS doesn't support replicating changes from memory-optimized tables.

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if temporal tables are a part of the task scope

API key: `sqlserver-check-for-temporal-tables`

This premigration assessment checks for the presence of temporal tables. AWS DMS doesn't support replicating changes from temporal tables.

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if delayed durability is enabled at the database level

API key: `sqlserver-check-for-delayed-durability`

This premigration assessment checks for the presence of delayed durability. AWS DMS doesn't support replicating changes from transactions that use delayed durability.

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if accelerated data recovery is enabled at the database level

API key: `sqlserver-check-for-accelerated-data-recovery`

This premigration assessment checks for the presence of accelerated data recovery. AWS DMS doesn't support replicating changes from databases with accelerated data recovery.

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if table mapping has more than 10K tables with primary keys

API key: `sqlserver-large-number-of-tables`

This premigration assessment checks for the presence of more than 10,000 tables with primary keys. Databases configured with MS-Replication can experience task failures if there are too many tables with primary keys.

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

For more information about configuring MS-Replication, see Capturing data changes for ongoing replication from SQL Server.

Check if the source database has tables or schema names with special characters.

API key: `sqlserver-check-for-special-characters`

This premigration assessment verifies whether the source database has table or schema names that include a character from the following set:

```
\\ -- \n \" \b \r ' \t ;
```

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if the source database has column names with masked data

API key: `sqlserver-check-for-masked-data`

This premigration assessment verifies whether the source database has masked data. AWS DMS migrates masked data without masking.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if the source database has encrypted backups

API key: `sqlserver-check-for-encrypted-backups`

This premigration assessment verifies whether the source database has encrypted backups.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if the source database has backups stored at a URL or on Windows Azure.

API key: `sqlserver-check-for-backup-url`

This premigration assessment verifies whether the source database has backups stored at a URL or on Windows Azure.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if the source database has backups on multiple disks

API key: `sqlserver-check-for-backup-multiple-stripes`

This premigration assessment verifies whether the source database has backups on multiple disks.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if the source database has at least one full backup

API key: `sqlserver-check-for-full-backup`

This premigration assessment verifies whether the source database has at least one full backup. SQL Server must be configured for full backup, and you must run a backup before replicating data.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if the source database has sparse columns and columnar structure compression.

API key: `sqlserver-check-for-sparse-columns`

This premigration assessment verifies whether the source database has sparse columns and columnar structure compression. DMS doesn't support sparse columns and columnar structure compression.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if the source database instance has server level auditing for SQL Server 2008 or SQL Server 2008 R2

API key: `sqlserver-check-for-audit-2008`

This premigration assessment verifies whether the source database has enabled server-level auditing for SQL Server 2008 or SQL Server 2008 R2. DMS has a related known issue with SQL Server 2008 and 2008 R2.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if the source database has geometry columns for full LOB mode

API key: `sqlserver-check-for-geometry-columns`

This premigration assessment verifies whether the source database has geometry columns for full Large Object (LOB) mode when using SQL Server as a source. We recommend using limited LOB mode or setting the `InlineLobMaxSize` task setting to use inline LOB mode when your database includes geometry columns.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if the source database has columns with the Identity property.

API key: `sqlserver-check-for-identity-columns`

This premigration assessment verifies whether the source database has a column with the `IDENTITY` property. DMS doesn't migrate this property to the corresponding target database column.

For more information, see [Limitations on using SQL Server as a source for AWS DMS](#).

Check if the DMS user has FULL LOAD permissions

API key: `sqlserver-check-user-permission-for-full-load-only`

This premigration assessment verifies whether the DMS task's user has permissions to run the task in FULL LOAD mode.

For more information, see [Limitations on using SQL Server as a source for AWS DMS](#).

Check if the DMS user has FULL LOAD and CDC or CDC only permissions

API key: `sqlserver-check-user-permission-for-cdc`

This premigration assessment verifies whether the DMS User has permissions to run the task in `FULL LOAD and CDC` or `CDC only` modes.

For more information, see [Limitations on using SQL Server as a source for AWS DMS](#).

Check whether MS-Replication is enabled for CDC on on-premises or EC2 databases.

API key: `sqlserver-check-attribute-for-enable-ms-cdc-onprem`

Check whether MS-Replication is enabled for CDC on on-premises or EC2 databases.

For more information about configuring MS-Replication, see [Capturing data changes for self-managed SQL Server on-premises or on Amazon EC2](#).

Check if the DMS user has the VIEW DEFINITION permission.

API key: `sqlserver-check-user-permission-on-view-definition`

This premigration assessment verifies whether the user specified in the endpoint settings has the `VIEW DEFINITION` permission. DMS requires the `VIEW DEFINITION` permission to view object definitions.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if the DMS user has the VIEW DATABASE STATE permission on the MASTER database for users without the Sysadmin role.

API key: `sqlserver-check-user-permission-on-view-database-state`

This premigration assessment verifies whether the user specified in the endpoint settings has the `VIEW DATABASE STATE` permission. DMS requires this permission to access database objects in the MASTER database. DMS also requires this permission when the user doesn't have sysadmin privileges. DMS requires this permission to create functions, certificates, and logins, and to grant credentials.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Check if the DMS user has the VIEW SERVER STATE permission.

API key: `sqlserver-check-user-permission-on-view-server-state`

This premigration assessment checks if the user specified in the extra connection attributes (ECA) has the `VIEW SERVER STATE`permission. `VIEW SERVER STATE` is a server-level permission that allows a user to view server-wide information and state. This permission provides access to dynamic management views (DMVs) and dynamic management functions (DMFs) that expose information about the SQL Server instance. This permission is required for the DMS user to have access to CDC resources. This permission is required to run a DMS task in `FULL LOAD and CDC` or `CDC only` modes.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Validate if text repl size parameter is not unlimited

API Key:`sqlserver-check-for-max-text-repl-size`

Setting Max text repl size parameter on the database could potentially cause data migration error for LOB columns. DMS highly recommends setting it to -1.

For more information, see the section called “Troubleshooting issues with Microsoft SQL Server”.

Validate if Primary Key or Unique Index exist on target for Batch Apply

API Key:`sqlserver-check-batch-apply-target-pk-ui-absence`

Batch apply is only supported on tables with Primary Keys or Unique Indexes on the target table. Tables without Primary Keys or Unique Indexes causes the batch to fail, and changes are processed one by one. It is advisable to move such tables to their own tasks and utilize transactional apply mode instead. Alternatively, you can create a unique key on the target table.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Validate if both Primary Key and Unique index exist on target when batch apply enabled

API Key:`sqlserver-check-batch-apply-target-pk-ui-simultaneously`

Batch apply is only supported on tables with Primary Keys or Unique Indexes on the target table. Tables with Primary Keys and Unique Indexes simultaneously causes the batch to fail, and changes are processed one by one. It is advisable to move such tables to their own tasks and utilize transactional apply mode instead. Alternatively, you can drop unique keys or primary key on the target table and rebuild it when migrating.

For more information, see Limitations on using SQL Server as a source for AWS DMS.

Validate if table has primary key or unique index when DMS validation is enabled

API Key:`sqlserver-check-pk-validity`

Data validation requires that the table has a primary key or unique index on both source and target.

For more information, see *Data validation*.

Validate if AWS DMS user has necessary privileges to the target

API Key:`sqlserver-check-target-privileges`

The AWS DMS user must have must have at least the db_owner user role on the target database.

For more information, see Security requirements when using SQL Server as a target for AWS Database Migration Service.

Recommendation on using MaxFullLoadSubTasks setting

API Key:`sqlserver-tblnum-for-max-fullload-subtasks`

This assessment checks the number of tables included in the task and recommends increasing the `MaxFullLoadSubTasks` parameter for optimal performance during the full load process.

By default, AWS DMS migrates 8 tables simultaneously. Changing the `MaxFullLoadSubTasks` parameter to a higher value improves the full load performance.

For more information, see Full-load task settings.

Check Transformation Rule for Digits Randomize

API key: `sqlserver-datamasking-digits-randomize`

This assessment validates whether columns used in table mappings are compatible with the Digits Randomize transformation rule. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying digits randomize transformations does not guarantee any uniqueness.

Check Transformation Rule for Digits mask

API key: `sqlserver-datamasking-digits-mask`

This assessment validates whether any columns used in the table mapping are not supported by the Digits Mask transformation rule. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying Digits Mask transformations to such columns could cause DMS task failures since uniqueness cannot be guaranteed.

Check Transformation Rule for Hashing mask

API key: `sqlserver-datamasking-hash-mask`

This assessment validates whether any of the columns used in the table mapping are not supported by the Hashing Mask transformation rule. It also checks if the length of the source column exceeds 64 characters. Ideally, the target column length should be greater than 64 characters to support hash masking. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying digits randomize transformations does not guarantee any uniqueness.

Verify that Data Validation task settings and Data Masking Digit randomization are not enabled simultaneously

API key: `all-to-all-validation-with-datamasking-digits-randomize`

This premigration assessment verifies that Data Validation setting and Data Masking Digit randomization are not simultaneously enabled, as these features are incompatible.

Verify that Data Validation task settings and Data Masking Hashing mask are not enabled simultaneously

API key: `all-to-all-validation-with-datamasking-hash-mask`

This premigration assessment verifies that Data Validation setting and Data Masking Hashing mask are not simultaneously enabled, as these features are incompatible.

Verify that Data Validation task settings and Data Masking Digit mask are not enabled simultaneously

API key: `all-to-all-validation-with-digit-mask`

This premigration assessment verifies that Data Validation setting and Data Masking Digit mask are not simultaneously enabled, as these features are incompatible.

Validate that at least one selected object exists in the source database

API key: `all-check-source-selection-rules`

This premigration assessment verifies that at least one object specified in the selection rules exists in the source database, including pattern matching for wildcard-based rules.

Validate that secondary constraints and indexes (non-primary) are present in the source database

API key: `all-check-secondary-constraints`

This premigration assessment verifies that secondary constraints and indexes (foreign keys, check constraints, non-clustered indexes) are present in the source database.

Validate that target endpoint is not a read replica

API key: `all-check-target-read-replica`

This premigration assessment verifies that the target endpoint is not configured as a read replica. AWS DMS requires write access to the target database and cannot replicate to read-only replicas.

Validate backup chain

API key: `sqlserver-check-for-backup-broken-chain`

This premigration assessment verifies that the source database backup chain is not broken. A broken backup chain can prevent AWS DMS from accessing transaction logs required for CDC replication.

Check database user permissions for applying `EXCLUSIVE_AUTOMATIC_TRUNCATION` safeguard policy

API key: `sqlserver-safeguard-permissions`

This premigration assessment verifies whether the database user has the required permissions to use the `EXCLUSIVE_AUTOMATIC_TRUNCATION` safeguard policy. The user must grant SELECT permissions on the `dbo.syscategories` and `dbo.sysjobs` system objects to the dmsuser.

For more information, see [Endpoint settings when using SQL Server as a source for AWS DMS].

Validate that secondary node connection and required safeguard attributes for AWS DMS source endpoint

API key: `sqlserver-check-sec-node-sg-policy`

This premigration assessment verifies that the source endpoint has the required extra connection attributes (ECAs) configured when connecting to a secondary node with safeguards enabled.

For more information, see [Endpoint settings when using SQL Server as a source for AWS DMS].

Validate that endpoint has all required extra connection attributes (ECAs) when AWS DMS is connected to secondary node

API key: `sqlserver-check-sec-node-without-eca`

This premigration assessment verifies that all required extra connection attributes (ECAs) are configured when the source endpoint connects to a secondary node

For more information, see [Working with self-managed SQL Server AlwaysOn availability groups].

MySQL assessments

This section describes individual premigration assessments for migration tasks that use a MySQL, Aurora MySQL-Compatible Edition or Aurora MySQL-Compatible Edition Serverless source endpoint.

Topics

- Verify that Data Validation task settings and Data Masking Digit randomization are not enabled simultaneously
- Verify that Data Validation task settings and Data Masking Hashing mask are not enabled simultaneously
- Verify that Data Validation task settings and Data Masking Digit mask are not enabled simultaneously
- Check if source Amazon Aurora MySQL instance is not a read replica
- Check if binary log retention time is set properly
- Check if source tables do not have invisible columns.
- Validate if the database binlog format is set to ROW to support DMS CDC
- Validate that at least one selected object exists in the source database
- Validate that tables with generated columns exist in the source database
- Validate that skipTableSuspensionForPartitionDdl is enabled for partitioned tables
- Validate that max_allowed_packet size can handle source LOB columns
- Validate that secondary constraints and indexes (non-primary) are present in the source database

Validate if Binary Log transaction compression is disabled

API key: `mysql-check-binlog-compression`

This premigration assessment validates whether binary Log transaction compression is disabled. AWS DMS doesn't support binary log transaction compression.

For more information, see Limitations on using a MySQL database as a source for AWS DMS.

Validate if DMS user has REPLICATION CLIENT and REPLICATION SLAVE permissions for the source database

API key: `mysql-check-replication-privileges`

This premigration assessment validates whether the DMS user specified in the source endpoint connection settings has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions for the source database if the DMS task migration type is CDC or full-load + CDC.

For more information, see Using any MySQL-compatible database as a source for AWS DMS.

Validate if DMS user has SELECT permissions for the source database tables

API key: `mysql-check-select-privileges`

This premigration assessment validates whether the DMS user specified in the source endpoint connection settings has SELECT permissions for the source database tables.

For more information, see [Using any MySQL-compatible database as a source for AWS DMS](#).

Validate if the server_id is set to 1 or greater in the source database

API key: `mysql-check-server-id`

This premigration assessment validates whether the `server_id` server variable is set to 1 or greater in the source database for CDC migration type.

For more information about sources for AWS DMS, see [Using a self-managed MySQL-compatible database as a source for AWS DMS](#).

Validate if DMS user has necessary permissions for the MySQL database as a target

API key: `mysql-check-target-privileges`

This premigration assessment validates whether the DMS user specified in the target endpoint connection settings has the necessary permissions for the MySQL database as a target.

For more information about MySQL source endpoint prerequisites, see [Using any MySQL-compatible database as a source for AWS DMS](#).

Validate if automatic removal of binary logs is set for the source database

API key: `mysql-check-expire-logs-days`

This premigration assessment validates whether your database is configured to automatically remove binary logs. The values of either `EXPIRE_LOGS_DAYS` or `BINLOG_EXPIRE_LOGS_SECONDS` global system variables should be greater than zero to prevent overuse of disk space during migration.

For more information about sources for AWS DMS, see [Using a self-managed MySQL-compatible database as a source for AWS DMS](#).

Validate that limited LOB mode only is used when `BatchApplyEnabled` is set to true

API key: `mysql-batch-apply-lob-mode`

This premigration assessment validates whether the DMS task includes LOB columns. If LOB columns are included into the scope of the task, you must use `BatchApplyEnabled` together with limited LOB mode only.

For more information about the `BatchApplyEnabled` setting, see [How can I use the DMS batch apply feature to improve CDC replication performance?](#).

Validate if a table uses a storage engine other than Innodb

API key: `mysql-check-table-storage-engine`

This premigration assessment validates whether the storage engine used for any table in the Source MySQL database is an engine other than Innodb. 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 on the target database and configure your DMS task to use `TRUNCATE_BEFORE_LOAD` or `DO_NOTHING` as the full-load task setting. For more information about full-load task settings, see [Full-load task settings](#).

> **Note**
>
> This premigration assessment is not available for Aurora MySQL-Compatible Edition or Aurora MySQL-Compatible Edition Serverless.

For more information about MySQL endpoint limitations, see [Limitations on using a MySQL database as a source for AWS DMS](#).

Validate if auto-increment is enabled on any tables used for migration

API key: `mysql-check-auto-increment`

This premigration assessment validates whether the source tables that are used in the task have auto-increment enabled. DMS doesn't migrate the AUTO_INCREMENT attribute on a column to a target database.

For more information about MySQL endpoint limitations, see [Limitations on using a MySQL database as a source for AWS DMS](#). For information about handling identity columns in MySQL, see [Handle IDENTITY columns in AWS DMS: Part 2](#).

Validate if the database binlog image is set to FULL to support DMS CDC

API key: `mysql-check-binlog-image`

This premigration assessment checks whether the source database's binlog image is set to FULL. In MySQL, the binlog_row_image variable determines how a binary log event is written when using the ROW format. To ensure compatibility with DMS and support CDC, set the binlog_row_image variable to FULL. This setting ensures that DMS receives sufficient information to construct the full Data Manipulation Language (DML) for the target database during migration.

To set the binlog image to FULL, do the following:

- For Amazon RDS, this value is FULL by default.
- For databases hosed on-premises or on Amazon EC2, set the binlog_row_image value in my.ini (Microsoft Windows) or my.cnf (UNIX).

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

Validate if the source database is a MySQL Read-Replica

API key: mysql-check-database-role

This premigration assessment verifies whether the source database is a read replica. To enable CDC support for DMS when connected to a read replica, set the log_slave_updates parameter to True. For more information about using a self-managed MySQL database, see Using a self-managed MySQL-compatible database as a source for AWS DMS.

To set the log_slave_updates value to True, do the following:

- For Amazon RDS, use the database's parameter group. For information about using RDS database parameter groups, see Working with parameter groups in the *Amazon RDS User Guide*.
- For databases hosed on-premises or on Amazon EC2, set the log_slave_updates value in my.ini (Microsoft Windows) or my.cnf (UNIX).

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

Validate if a table has partitions, and recommend target_table_prep_mode for full-load task settings

API key: mysql-check-table-partition

This premigration assessment checks for the presence of tables with partitions in the source database. DMS creates tables without partitions on the MySQL target. To migrate partitioned tables to a partitioned table on the target, you must do the following:

- Pre-create the partitioned tables in the target MySQL database.
- Configure your DMS task to use `TRUNCATE_BEFORE_LOAD` or `DO_NOTHING` as the full-load task setting.

For more information about MySQL endpoint limitations, see Limitations on using a MySQL database as a source for AWS DMS.

Validate if DMS supports the database version

API key: `mysql-check-supported-version`

This premigration assessment verifies whether the source database version is compatible with DMS. For more information about supported MySQL versions, see Source endpoints for data migration.

Validate if the target database is configured to set `local_infile` to 1

API key: `mysql-check-target-localinfile-set`

This premigration assessment checks whether the `local_infile` parameter in the target database is set to 1. DMS requires the 'local_infile' parameter to be set to 1 during full load in your target database. For more information, see Migrating from MySQL to MySQL using AWS DMS.

This assessment is only valid for a full-load or full-load and CDC task.

Validate if target database has tables with foreign keys

API key: `mysql-check-fk-target`

This premigration assessment checks whether a full load or full and CDC task migrating to a MySQL database has tables with foreign keys. The default setting in DMS is to load tables in alphabetical order. Tables with foreign keys and referential integrity constraints can cause the load to fail, as the parent and child tables may not be loaded at the same time.

For more information about referential integrity in DMS, see **Working with indexes, triggers, and referential integrity constraints** in the Improving the performance of an AWS DMS migration topic.

Validate if source tables in the task scope have cascade constraints

API key: `mysql-check-cascade-constraints`

This premigration assessment checks if any of the MySQL source tables have cascade constraints. Cascade constraints are not migrated or replicated by DMS tasks, because MySQL doesn't record the changes for these events in the binlog. While AWS DMS doesn't support these constraints, you can use workarounds for relational database targets.

For information about supporting cascase constrains and other constraints, see Indexes, Foreign Keys, or Cascade Updates or Deletes Not Migrated in the **Troubleshooting migration tasks in AWS DMS** topic.

Validate if the timeout values are appropriate for a MySQL source or target

API key: `mysql-check-target-network-parameter`

This premigration assessment checks whether a task’s MySQL endpoint has the `net_read_timeout`, `net_write_timeout` and `wait_timeout` settings set to at least 300 seconds. This is needed to prevent disconnects during the migration.

For more information, see Connections to a target MySQL instance are disconnected during a task.

Validate `max_statement_time` database parameter

API key: `mysql-check-max-statement-time`

Check source parameter - `max_Statement_time` for MySQL based sources. If there are tables larger than 1 billion, validate the value `max_Statement_time` and recommend setting to higher value to avoid any potential data loss.

Validate if Primary Key or Unique Index exist on target for Batch Apply

API key: `mysql-check-batch-apply-target-pk-ui-absence`

Batch apply is only supported on tables with Primary Keys or Unique Indexes on the target table. Tables without Primary Keys or Unique Indexes causes the batch to fail, and changes are processed one by one. It is advisable to move such tables to their own tasks and utilize transactional apply mode instead. Alternatively, you can create a unique key on the target table.

For more information, see Using a MySQL-compatible database as a target for AWS Database Migration Service.

Validate if both Primary Key and Unique index exist on target for Batch Apply

API key: `mysql-check-batch-apply-target-pk-ui-simultaneously`

Batch apply is only supported on tables with Primary Keys or Unique Indexes on the target table. Tables with Primary Keys and Unique Indexes simultaneously causes the batch to fail, and changes are processed one by one. It is advisable to move such tables to their own tasks and utilize transactional apply mode instead. Alternatively, you can drop a unique key(s) or primary key on the target table and rebuild it if you are doing migration.

For more information, see Using a MySQL-compatible database as a target for AWS Database Migration Service.

Validate if secondary indexes are enabled during full load on the target database

API key: `mysql-check-secondary-indexes`

Consider disabling or removing the secondary indexes from the target database. Secondary indexes can affect your migration performance during full load. It is advisable to enable secondary indexes before applying the cached changes.

For more information, see Best practices for AWS Database Migration Service.

Validate if table has primary key or unique index when DMS validation is enabled

API key: `mysql-check-pk-validity`

Data validation requires that the table has a primary key or unique index.

For more information, see AWS DMS data validation.

Recommendation on using `MaxFullLoadSubTasks` setting

API key: `mysql-tblnum-for-max-fullload-subtasks`

This assessment checks the number of tables included in the task and recommends increasing the `MaxFullLoadSubTasks` parameter for optimal performance during the full load process. By default, AWS DMS migrates 8 tables simultaneously. Changing the `MaxFullLoadSubTasks` parameter to a higher value improves the full load performance.

For more information, see Full-load task settings.

Check Transformation Rule for Digits Randomize

API key: `mysql-datamasking-digits-randomize`

This assessment validates whether columns used in table mappings are compatible with the Digits Randomize transformation rule. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying digits randomize transformations does not guarantee any uniqueness.

Check Transformation Rule for Digits mask

API key: `mysql-datamasking-digits-mask`

This assessment validates whether any columns used in the table mapping are not supported by the Digits Mask transformation rule. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying Digits Mask transformations to such columns could cause DMS task failures since uniqueness cannot be guaranteed.

Check Transformation Rule for Hashing mask

API key: `mysql-datamasking-hash-mask`

This assessment validates whether any of the columns used in the table mapping are not supported by the Hashing Mask transformation rule. It also checks if the length of the source column exceeds 64 characters. Ideally, the target column length should be greater than 64 characters to support hash masking. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying digits randomize transformations does not guarantee any uniqueness.

Verify that Data Validation task settings and Data Masking Digit randomization are not enabled simultaneously

API key: `all-to-all-validation-with-datamasking-digits-randomize`

This premigration assessment verifies that Data Validation setting and Data Masking Digit randomization are not simultaneously enabled, as these features are incompatible.

Verify that Data Validation task settings and Data Masking Hashing mask are not enabled simultaneously

API key: `all-to-all-validation-with-datamasking-hash-mask`

This premigration assessment verifies that Data Validation setting and Data Masking Hashing mask are not simultaneously enabled, as these features are incompatible.

Verify that Data Validation task settings and Data Masking Digit mask are not enabled simultaneously

API key: `all-to-all-validation-with-digit-mask`

This premigration assessment verifies that Data Validation setting and Data Masking Digit mask are not simultaneously enabled, as these features are incompatible.

Check if source Amazon Aurora MySQL instance is not a read replica

API key: `mysql-check-aurora-read-only`

This premigration assessment validates whether migrating between two Amazon Aurora MySQL clusters, the source endpoint must be a read/write instance, not a replica instance.

Check if binary log retention time is set properly

API key: `mysql-check-binlog-retention-time`

This premigration assessment validates whether the value of 'binlog retention hours' is larger than 24 hours.

Check if source tables do not have invisible columns.

API key: `mysql-check-invisible-columns`

This premigration assessment validates whether source tables do not have invisible columns. AWS DMS does not migrate data from invisible columns in your source database.

Validate if the database binlog format is set to ROW to support DMS CDC

API key: `mysql-check-binlog-format`

This premigration assessment validates whether the source database binlog format is configured to ROW to support Change Data Capture (CDC). To set the binlog format to ROW, do the following:

- For Amazon RDS, use the database's parameter group. For information, see [Configuring MySQL binary logging for Single-AZ databases](#) in the Amazon Relational Database Service User Guide.

- For databases hosed on-premises or on Amazon EC2, set the `binlog_format` value in `my.ini` (Microsoft Windows) or `my.cnf` (UNIX).

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration. For more information about self-hosted MySQL servers, see Using a self-managed MySQL-compatible database as a source for AWS DMS.

Validate that at least one selected object exists in the source database

API key: `all-check-source-selection-rules`

This premigration assessment verifies that at least one object specified in the selection rules exists in the source database, including pattern matching for wildcard-based rules.

Validate that tables with generated columns exist in the source database

API key: `mysql-check-generated-columns`

This premigration assessment checks whether any of the MySQL source tables have generated columns. AWS DMS tasks don't migrate or replicate generated columns. For information about how to migrate generated columns, see Limitations on using a MySQL database as a source for AWS DMS.

Validate that skipTableSuspensionForPartitionDdl is enabled for partitioned tables

API key: `mysql-check-skip-table-suspension-partition-ddl`

This premigration assessment detects partitioned tables in the source database and verifies the `skipTableSuspensionForPartitionDdl` parameter setting. Failure to set this parameter may result in unnecessary table suspensions during migration. For more details, refer to the following link: Limitations on using a MySQL database as a source for AWS DMS.

Validate that max_allowed_packet size can handle source LOB columns

API key: `mysql-check-max-allowed-packet-lob`

AWS DMS detects LOB columns in source tables that exceed your current `max_allowed_packet` setting. This mismatch can cause replication failures during data migration. For more information, see Troubleshooting issues with MySQL.

Validate that secondary constraints and indexes (non-primary) are present in the source database

API key: `all-check-secondary-constraints`

This premigration assessment verifies that secondary constraints and indexes (foreign keys, check constraints, non-clustered indexes) are present in the source database.

MariaDB assessments

This section describes individual premigration assessments for migration tasks that use a MariaDB source endpoint.

To create an individual premigration assessment using the AWS DMS API, use the listed API key for the `Include` parameter of the StartReplicationTaskAssessmentRun action.

Topics

- Validate if target database has tables with foreign keys
- Validate if source tables in the task scope have cascade constraints
- Validate if source tables in the task scope have generated columns
- Validate if the timeout values are appropriate for a MariaDB source
- Validate if the timeout values are appropriate for a MariaDB target
- Validate max_statement_time database parameter
- Validate if Primary Key or Unique Index exist on target for Batch Apply
- Validate if both Primary Key and Unique index exist on target for Batch Apply
- Validate if secondary indexes are enabled during full load on the target database
- Validate if table has primary key or unique index when DMS validation is enabled
- Recommendation on using MaxFullLoadSubTasks setting
- Check Transformation Rule for Digits Randomize
- Check Transformation Rule for Digits mask
- Check Transformation Rule for Hashing mask
- Verify that Data Validation task settings and Data Masking Digit randomization are not enabled simultaneously
- Verify that Data Validation task settings and Data Masking Hashing mask are not enabled simultaneously
- Verify that Data Validation task settings and Data Masking Digit mask are not enabled simultaneously
- Check if binary log retention time is set properly
- Check if source tables do not have invisible columns
- Validate that at least one selected object exists in the source database
- Validate that skipTableSuspensionForPartitionDdl is enabled for partitioned tables
- Validate that secondary constraints and indexes (non-primary) are present in the source database

Validate if the `server_id` is set to 1 or greater in the source database

API key: `mariadb-check-server-id`

This premigration assessment validates whether the `server_id` server variable is set to 1 or greater in the source database for CDC migration type.

For more information about MariaDB endpoint limitations, see [Using a self-managed MySQL-compatible database as a source for AWS DMS].

Validate if automatic removal of binary logs is set for the source database

API key: `mariadb-check-expire-logs-days`

This premigration assessment validates whether your database is configured to automatically remove binary logs. The values of either `EXPIRE_LOGS_DAYS` or `BINLOG_EXPIRE_LOGS_SECONDS` global system variables should be greater than zero to prevent overuse of disk space during migration.

For more information about MariaDB endpoint limitations, see [Using a self-managed MySQL-compatible database as a source for AWS DMS].

Validate that limited LOB mode only is used when BatchApplyEnabled is set to true

API key: `mariadb-batch-apply-lob-mode`

When LOB columns are included in the replication, you can use `BatchApplyEnabled` in limited LOB mode only. Using other options of the LOB mode will cause the batch to fail, and AWS DMS will process the changes one by one. We recommend that you move these tables to their own tasks and use transactional apply mode instead.

For more information about the `BatchApplyEnabled` setting, see [How can I use the DMS batch apply feature to improve CDC replication performance?].

Validate if Binary Log transaction compression is disabled

API key: `mariadb-check-binlog-compression`

This premigration assessment validates whether Binary Log transaction compression is disabled. AWS DMS doesn't support binary log transaction compression.

For more information, see [Limitations on using a MySQL database as a source for AWS DMS].

Validate if DMS user has REPLICATION CLIENT and REPLICATION SLAVE privileges for the source database

API key: `mariadb-check-replication-privileges`

This premigration assessment validates whether the DMS user specified in the source endpoint connection settings has `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges for the source database, if the DMS task migration type is CDC or full-load + CDC.

For more information, see Using any MySQL-compatible database as a source for AWS DMS.

Validate if DMS user has SELECT permissions for the source database tables

API key: `mariadb-check-select-privileges`

This premigration assessment validates whether the DMS user specified in the source endpoint connection settings has `SELECT` permissions for the source database tables.

For more information, see Using any MySQL-compatible database as a source for AWS DMS.

Validate if DMS user has necessary privileges for the MySQL-compatible database as a target

API key: `mariadb-check-target-privileges`

This premigration assessment validates whether the DMS user specified in the target endpoint connection settings has necessary privileges for the MySQL-compatible database as a target.

For more information, see Using any MySQL-compatible database as a source for AWS DMS.

Validate if a table uses a storage engine other than Innodb

API key: `mariadb-check-table-storage-engine`

This premigration assessment validates whether the storage engine used for any table in the Source MariaDB database is an engine other than Innodb. 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 on the target database and configure your DMS task to use `TRUNCATE_BEFORE_LOAD` or `DO_NOTHING` as the full-load task setting. For more information about full-load task settings, see Full-load task settings.

For more information about MariaDB endpoint limitations, see Limitations on using a MySQL database as a source for AWS DMS.

Validate if auto-increment is enabled on any tables used for migration

API key: `mariadb-check-auto-increment`

This premigration assessment validates whether the source tables that are used in the task have auto-increment enabled. DMS doesn't migrate the AUTO_INCREMENT attribute on a column to a target database.

For more information about MariaDB endpoint limitations, see Limitations on using a MySQL database as a source for AWS DMS. For information about handling identity columns in MariaDB, see Handle IDENTITY columns in AWS DMS: Part 2.

Validate if the database binlog format is set to `ROW` to support DMS CDC

API key: `mariadb-check-binlog-format`

This premigration assessment validates whether the source database binlog format is set to `ROW` to support DMS Change Data Capture (CDC).

To set the binlog format to `ROW`, do the following:

- For Amazon RDS, use the database's parameter group. For information about using an RDS parameter group, see Configuring MySQL binary logging in the *Amazon RDS User Guide*.
- For databases hosed on-premises or on Amazon EC2, set the `binlog_format` value in `my.ini` (Microsoft Windows) or `my.cnf` (UNIX).

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

For more information about self-hosted MariaDB servers, see Using a self-managed MySQL-compatible database as a source for AWS DMS.

Validate if the database binlog image is set to `FULL` to support DMS CDC

API key: `mariadb-check-binlog-image`

This premigration assessment checks whether the source database's binlog image is set to `FULL`. In MariaDB, the `binlog_row_image` variable determines how a binary log event is written when using the `ROW` format. To ensure compatibility with DMS and support CDC, set the `binlog_row_image` variable to `FULL`. This setting ensures that DMS receives sufficient information to construct the full Data Manipulation Language (DML) for the target database during migration.

To set the binlog image to `FULL`, do the following:

- For Amazon RDS, this value is `FULL` by default.
- For databases hosed on-premises or on Amazon EC2, set the `binlog_row_image` value in `my.ini` (Microsoft Windows) or `my.cnf` (UNIX).

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

For more information about self-hosted MariaDB servers, see [Using a self-managed MySQL-compatible database as a source for AWS DMS](#).

Validate if the source database is a MariaDB Read-Replica

API key: `mariadb-check-database-role`

This premigration assessment verifies whether the source database is a read replica. To enable CDC support for DMS when connected to a read replica, set the `log_slave_updates` parameter to `True`. For more information about using a self-managed MySQL database, see [Using a self-managed MySQL-compatible database as a source for AWS DMS](#).

To set the `log_slave_updates` value to `True`, do the following:

- For Amazon RDS, use the database's parameter group. For information about using RDS database parameter groups, see [Working with parameter groups](#) in the *Amazon RDS User Guide*.
- For databases hosed on-premises or on Amazon EC2, set the `log_slave_updates` value in `my.ini` (Microsoft Windows) or `my.cnf` (UNIX).

This assessment is only valid for a full-load and CDC migration, or a CDC-only migration. This assessment is not valid for a full-load only migration.

Validate if a table has partitions, and recommend `TRUNCATE_BEFORE_LOAD` or `DO_NOTHING` for full-load task settings

API key: `mariadb-check-table-partition`

This premigration assessment checks for the presence of tables with partitions in the source database. DMS creates tables without partitions on the MariaDB target. To migrate partitioned tables to a partitioned table on the target, you must do the following:

- Pre-create the partitioned tables in the target MariaDB database.

- Configure your DMS task to use TRUNCATE_BEFORE_LOAD or DO_NOTHING as the full-load task setting.

For more information about MariaDB endpoint limitations, see Limitations on using a MySQL database as a source for AWS DMS.

Validate if DMS supports the database version

API key: mariadb-check-supported-version

This premigration assessment verifies whether the source database version is compatible with DMS. CDC is not supported with Amazon RDS MariaDB versions 10.4 or lower, or MySQL versions greater than 10.11. For more information about supported MariaDB versions, see Source endpoints for data migration.

Validate if the target database is configured to set `local_infile` to 1

API key: mariadb-check-target-localinfile-set

This premigration assessment checks whether the `local_infile` parameter in the target database is set to 1. DMS requires the 'local_infile' parameter to be set to 1 during full load in your target database. For more information, see Migrating from MySQL to MySQL using AWS DMS.

This assessment is only valid for a full-load task.

Validate if target database has tables with foreign keys

API key: mariadb-check-fk-target

This premigration assessment checks whether a full load or full and CDC task migrating to a MariaDB database has tables with foreign keys. The default setting in DMS is to load tables in alphabetical order. Tables with foreign keys and referential integrity constraints can cause the load to fail, as the parent and child tables may not be loaded at the same time.

For more information about referential integrity in DMS, see **Working with indexes, triggers, and referential integrity constraints** in the Improving the performance of an AWS DMS migration topic.

Validate if source tables in the task scope have cascade constraints

API key: mariadb-check-cascade-constraints

This premigration assessment checks if any of the MariaDB source tables have cascade constraints. Cascade constraints are not migrated or replicated by DMS tasks, because MariaDB doesn't record the changes for these events in the binlog. While AWS DMS doesn't support these constraints, you can use workarounds for relational database targets.

For information about supporting cascase constrains and other constraints, see Indexes, Foreign Keys, or Cascade Updates or Deletes Not Migrated in the **Troubleshooting migration tasks in AWS DMS** topic.

Validate if source tables in the task scope have generated columns

API key: `mariadb-check-generated-columns`

This premigration assessment checks whether any of the MariaDB source tables have generated columns. DMS tasks don't migrate or replicate generated columns.

For information about how to migrate generated columns, see Connections to a target MySQL instance are disconnected during a task.

Validate if the timeout values are appropriate for a MariaDB source

API key: `mariadb-check-source-network-parameter`

This premigration assessment checks whether a task's MariaDB source endpoint has the `net_read_timeout`, `net_write_timeout` and `wait_timeout` settings set to at least 300 seconds. This is needed to prevent disconnects during the migration.

For more information, see Connections to a target MySQL instance are disconnected during a task.

Validate if the timeout values are appropriate for a MariaDB target

API key: `mariadb-check-target-network-parameter`

This premigration assessment checks whether a task's MariaDB target endpoint has the `net_read_timeout`, `net_write_timeout` and `wait_timeout` settings set to at least 300 seconds. This is needed to prevent disconnects during the migration.

For more information, see Connections to a target MySQL instance are disconnected during a task.

Validate `max_statement_time` database parameter

API key:`mariadb-check-max-statement-time`

AWS DMS validates that the database source parameter `max_statement_time` is set to a value other than 0. It is required to set this parameter to 0 to accommodate the DMS full load process. You can consider changing the parameter value after the full load is completed, as setting it to a value other than 0 may result is loss of data.

Validate if Primary Key or Unique Index exist on target for Batch Apply

API key: `mariadb-check-batch-apply-target-pk-ui-absence`

Batch apply is only supported on tables with Primary Keys or Unique Indexes on the target table. Tables without Primary Keys or Unique Indexes will cause the batch to fail, and changes will be processed one by one. It is advisable to move such tables to their own tasks and utilize transactional apply mode instead. Alternatively, you can create a unique key on the target table.

For more information, see the section called "Using MySQL as a target".

Validate if both Primary Key and Unique index exist on target for Batch Apply

API key: `mariadb-check-batch-apply-target-pk-ui-simultaneously`

Batch apply is only supported on tables with Primary Keys or Unique Indexes on the target table. Tables with Primary Keys and Unique Indexes simultaneously cause the batch to fail, and changes are processed one by one. It is advisable to move such tables to their own tasks and utilize transactional apply mode instead. Alternatively, you can drop a unique key(s) or primary key on the target table and rebuild it if you are doing migration.

For more information, see the section called "Using MySQL as a target".

Validate if secondary indexes are enabled during full load on the target database

API key: `mariadb-check-secondary-indexes`

You must consider disabling or removing the secondary indexes from the target database. Secondary indexes can affect your migration performance during full load. It is advisable to enable secondary indexes before applying the cached changes.

For more information, see *Best practices*.

Validate if table has primary key or unique index when DMS validation is enabled

API key: `mariadb-check-pk-validity`

Data validation requires that the table has a primary key or unique index on both source and target.

For more information, see *Data validation*.

Recommendation on using `MaxFullLoadSubTasks` setting

This assessment checks the number of tables included in the task and recommends increasing the `MaxFullLoadSubTasks` parameter for optimal performance during the full load process. By default, AWS DMS migrates 8 tables simultaneously. Changing the `MaxFullLoadSubTasks` parameter to a higher value will improve the full load performance.

For more information, see the section called "Full-load task settings".

Check Transformation Rule for Digits Randomize

API key: `mariadb-datamasking-digits-randomize`

This assessment validates whether columns used in table mappings are compatible with the Digits Randomize transformation rule. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying digits randomize transformations does not guarantee any uniqueness.

Check Transformation Rule for Digits mask

API key: `mariadb-datamasking-digits-mask`

This assessment validates whether any columns used in the table mapping are not supported by the Digits Mask transformation rule. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying Digits Mask transformations to such columns could cause DMS task failures since uniqueness cannot be guaranteed.

Check Transformation Rule for Hashing mask

API key: `mariadb-datamasking-hash-mask`

This assessment validates whether any of the columns used in the table mapping are not supported by the Hashing Mask transformation rule. It also checks if the length of the source column exceeds 64 characters. Ideally, the target column length should be greater than 64 characters to support hash masking. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying digits randomize transformations does not guarantee any uniqueness.

Verify that Data Validation task settings and Data Masking Digit randomization are not enabled simultaneously

API key: `all-to-all-validation-with-datamasking-digits-randomize`

This premigration assessment verifies that Data Validation setting and Data Masking Digit randomization are not simultaneously enabled, as these features are incompatible.

Verify that Data Validation task settings and Data Masking Hashing mask are not enabled simultaneously

API key: `all-to-all-validation-with-datamasking-hash-mask`

This premigration assessment verifies that Data Validation setting and Data Masking Hashing mask are not simultaneously enabled, as these features are incompatible.

Verify that Data Validation task settings and Data Masking Digit mask are not enabled simultaneously

API key: `all-to-all-validation-with-digit-mask`

This premigration assessment verifies that Data Validation setting and Data Masking Digit mask are not simultaneously enabled, as these features are incompatible.

Check if binary log retention time is set properly

API key: `mariadb-check-binlog-retention-time`

This premigration assessment validates whether the value of '`binlog retention hours`' is larger than 24 hours.

Check if source tables do not have invisible columns

API key: `mariadb-check-invisible-columns`

This premigration assessment validates whether source tables do not have invisible columns. AWS DMS does not migrate data from invisible columns in your source database.

Validate that at least one selected object exists in the source database

API key: `all-check-source-selection-rules`

This premigration assessment verifies that at least one object specified in the selection rules exists in the source database, including pattern matching for wildcard-based rules.

Validate that `skipTableSuspensionForPartitionDdl` is enabled for partitioned tables

API key: `mariadb-check-skip-table-suspension-partition-ddl`

This premigration assessment detects partitioned tables in the source database and verifies the `skipTableSuspensionForPartitionDdl` parameter setting. Failure to set this parameter may result in unnecessary table suspensions during migration.

For more information, see Limitations on using a MySQL database as a source for AWS DMS.

Validate that secondary constraints and indexes (non-primary) are present in the source database

API key: `all-check-secondary-constraints`

This premigration assessment verifies that secondary constraints and indexes (foreign keys, check constraints, non-clustered indexes) are present in the source database.

PostgreSQL assessments

This section describes individual premigration assessments for migration tasks that use a PostgreSQL source endpoint.

Topics

- Validate that limited LOB mode only is used when BatchApplyEnabled is set to true
- Validate if source database version is supported by DMS for migration
- Validate the logical_decoding_work_mem parameter on the source database
- Validate whether the source database has any long running transactions
- Validate the source database parameter max_slot_wal_keep_size
- Check if the source database parameter postgres-check-maxwalsenders is set to support CDC.
- Check if the source database is configured for PGLOGICAL
- Validate if the source table primary key is of LOB Datatype
- Validate if the source table has a primary key
- Validate if prepared transactions are present on the source database
- Validate if wal_sender_timeout is set to a minimum required value to support DMS CDC
- Validate if wal_level is set to logical on the source database
- Validate if both Primary Key and Unique index exist on target for Batch Apply
- Recommend Max LOB setting when LOB objects are found
- Validate if table has primary key or unique index and its state is well when DMS validation is enabled
- Validate if AWS DMS user has necessary privileges to the target
- Validates availability of free replication slots for CDC
- Verify DMS User Full Load Permissions
- Check Transformation Rule for Digits Randomize
- Check Transformation Rule for Digits mask
- Check Transformation Rule for Hashing mask
- Verify that Data Validation task settings and Data Masking Digit randomization are not enabled simultaneously
- Verify that Data Validation task settings and Data Masking Hashing mask are not enabled simultaneously
- Verify that Data Validation task settings and Data Masking Digit mask are not enabled simultaneously
- Validate that at least one selected object exists in the source database
- Validate that target PostgreSQL database contains generated columns

- Validate that materialized views exist in homogeneous PostgreSQL migrations
- Validate that REPLICA IDENTITY FULL conflicts with pglogical plugin usage
- Validate that secondary constraints and indexes (non-primary) are present in the source database
- Validate CHAR/VARCHAR columns compatibility for migration to Oracle
- Validate that idle_in_transaction_session_timeout setting is configured on the source database
- Validate that AWS DMS user has required roles for AWS-managed PostgreSQL databases

Validate if DDL event trigger is set to ENABLE ALWAYS

API key: `postgres-check-ddl-event-trigger`

This premigration assessment validates whether the DDL event trigger is set to `ENABLE ALWAYS`. When your source database is also a target for another third–party replication system, DDL changes might not migrate during CDC. This situation can prevent DMS from triggering the the `awsdms_intercept_ddl` event. To work around the situation, modify the trigger on your source database like in the follwoing example:

```
alter event trigger awsdms_intercept_ddl enable always;
```

For more information, see Limitations on using a PostgreSQL database as a DMS source.

Validate if PostGIS columns exist in the source database

API key: `postgres-check-postgis-data-type`

This premigration assessment validates whether the columns of PostGIS data type that exist in case source and target engines are different. AWS DMS supports the PostGIS data type only for homogeneous (like-to-like) migrations.

For more information, see Limitations on using a PostgreSQL database as a DMS source.

Validate if foreign key constraint is disabled on the target tables during the full-load process

API key: `postgres-check-session-replication-role`

This premigration assessment validates whether the `session_replication_role parameter` is set to `REPLICA` on the target to disable foreign key constraints during full-load phase. For full-load migration types, you should disabled foreign key constraints.

For more information about PostgreSQL endpoint limitations, see [Using a PostgreSQL database as a target for AWS Database Migration Service](#).

Validate if tables with similar names exist

API key: `postgres-check-similar-table-name`

This premigration assessment validates whether there are tables with similar names on the source. Having multiple tables with the same name written in different case can cause unpredictable behaviors during replication.

For more information about PostgreSQL endpoint limitations, see [Limitations on using a PostgreSQL database as a DMS source](#).

Validate if there are tables with ARRAY data type without a primary key

API key: `postgres-check-table-with-array`

This premigration assessment validates whether there are tables with array data type without a primary key. A table with an `ARRAY` data type missing a primary key is ignored during the full-load.

For more information about PostgreSQL endpoint limitations, see [Limitations on using a PostgreSQL database as a DMS source](#).

Validate if primary keys or unique indexes exist on the target tables when BatchApplyEnabled is enabled

API key: `postgres-check-batch-apply-target-pk-ui-absence`

Batch apply is only supported on tables with primary keys or unique indexes on the target table. Tables without primary keys or unique indexes will cause the batch to fail, and AWS DMS will process the changes one by one. We recommend that you create separate tasks for such tables and use transactional apply mode instead. Alternatively, you can create a unique key on the target table.

For more information, see [Using a PostgreSQL database as a target for AWS Database Migration Service](#).

Validate if any table of the target database has secondary indexes for the full-load migration task

API key: `postgres-check-target-secondary-indexes`

This premigration assessment validates whether there are tables with secondary indexes in the scope of the full-load migration task. We recommend that you drop the secondary indexes for the duration of the full-load task.

For more information, see [Using a PostgreSQL database as a target for AWS Database Migration Service].

Validate that limited LOB mode only is used when BatchApplyEnabled is set to true

API key: `postgres-batch-apply-lob-mode`

When LOB columns are included in the replication, you can use `BatchApplyEnabled` in limited LOB mode only. Using other options of the LOB mode will cause the batch to fail, and AWS DMS will process changes one by one. We recommend that you move these tables to their own tasks and use transactional apply mode instead.

For more information about the `BatchApplyEnabled` setting, see [How can I use the DMS batch apply feature to improve CDC replication performance?].

Validate if source database version is supported by DMS for migration

API key: `postgres-check-dbversion`

This premigration assessment verifies whether the source database version is compatible with AWS DMS.

Validate the `logical_decoding_work_mem` parameter on the source database

API key: `postgres-check-for-logical-decoding-work-mem`

This premigration assessment recommends tuning the `logical_decoding_work_mem` parameter on the source database. On a highly transactional database where you might have long running transactions or many sub-transactions, it may result in increased logical decoding memory consumption and the need to spill to disk. This results in high DMS source latency during replication. In such scenarios, you might need to tune `logical_decoding_work_mem`. This parameter is supported in PostgreSQL versions 13 and greater.

Validate whether the source database has any long running transactions

API key: `postgres-check-longrunningtxn`

This premigration assessment verifies whether the source database has any long running transactions which lasted more than 10 minutes. Starting the task might fail, because by default, DMS checks for any open transactions while starting the task.

Validate the source database parameter `max_slot_wal_keep_size`

API key: `postgres-check-maxslot-wal-keep-size`

This premigration assessment verifies the value configured for `max_slot_wal_keep_size`. When `max_slot_wal_keep_size` is set to a non-default value, the DMS task may fail due to the removal of required WAL files.

Check if the source database parameter `postgres-check-maxwalsenders` is set to support CDC.

API key: `postgres-check-maxwalsenders`

This premigration assessment verifies the value configured for `max_wal_senders` on the source database. DMS requires `max_wal_senders` to be set greater than 1 to support Change Data Capture (CDC).

Check if the source database is configured for `PGLOGICAL`

API key: `postgres-check-pglogical`

This premigration assessment verifies if the `shared_preload_libraries` value is set to `pglogical` to support `PGLOGICAL` for CDC. Note that you can ignore this assessment if you are planning to use test decoding for logical replication.

Validate if the source table primary key is of LOB Datatype

API key: `postgres-check-pk-lob`

This premigration assessment verifies if a table's primary key is of Large Object (LOB) datatype. DMS does not support replication if the source table has an LOB column as a primary key.

Validate if the source table has a primary key

API key: `postgres-check-pk`

This premigration assessment verifies if primary keys exist for the tables used in the task scope. DMS doesn’t support replication for tables without primary keys, unless the replica identity is set to `full` on the source table.

Validate if prepared transactions are present on the source database

API key: `postgres-check-preparedtxn`

This premigration assessment verifies if there are any prepared transactions present on the source database. Replication slot creation might stop responding if there are any prepared transactions on the source database.

Validate if `wal_sender_timeout` is set to a minimum required value to support DMS CDC

API key: `postgres-check-walsenderstimeout`

This premigration assessment verifies if `wal_sender_timeout` is set to a minimum of 10000 milliseconds (10 seconds). A DMS task with CDC requires a minimum of 10000 milliseconds (10 seconds), and fails if the value is less than 10000.

Validate if `wal_level` is set to logical on the source database

API key: `postgres-check-wallevel`

This premigration assessment verifies if `wal_level` is set to logical. For DMS CDC to work, this parameter needs to be enabled on the source database.

Validate if both Primary Key and Unique index exist on target for Batch Apply

API key: `postgres-check-batch-apply-target-pk-ui-simultaneously`

Batch apply is only supported on tables with Primary Keys or Unique Indexes on the target table. Tables with Primary Keys and Unique Indexes simultaneously cause the batch to fail, and changes are processed one by one. It is advisable to move such tables to their own tasks and utilize transactional apply mode instead. Alternatively, you can drop a unique key(s) or primary key on the target table and rebuild it if you are doing migration.

For more information, see [the section called "Enabling CDC using a self-managed PostgreSQL database as a AWS DMS source"].

Recommend Max LOB setting when LOB objects are found

API key: `postgres-check-limited-lob-size`

The LOB Size Calculation for PostgreSQL is different from other Engine. Ensure you are setting the right Maximum LOB size on your task setting to avoid any data truncation.

For more information, see *Data validation*.

Validate if table has primary key or unique index and its state is well when DMS validation is enabled

API key: `postgres-check-pk-validity`

Data validation requires that the table has a primary key or unique index.

For more information, see *Data validation*.

Validate if AWS DMS user has necessary privileges to the target

API key: `postgres-check-target-privileges`

The AWS DMS user must have at least the db_owner user role on the target database.

For more information, see the section called "Security requirements for PostgreSQL as a target".

Validates availability of free replication slots for CDC

API key: `postgres-check-replication-slots-count`

This assessment validates whether replication slots are available for CDC to replicate changes.

Verify DMS User Full Load Permissions

API key: `postgres-check-select-object-privileges`

This assessment validates whether the DMS user has the necessary SELECT privileges on tables required for Full Load operations.

Check Transformation Rule for Digits Randomize

API key: `postgres-datamasking-digits-randomize`

This assessment validates whether columns used in table mappings are compatible with the Digits Randomize transformation rule. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying digits randomize transformations does not guarantee any uniqueness.

Check Transformation Rule for Digits mask

API key: `postgres-datamasking-digits-mask`

This assessment validates whether any columns used in the table mapping are not supported by the Digits Mask transformation rule. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying Digits Mask transformations to such columns could cause DMS task failures since uniqueness cannot be guaranteed.

Check Transformation Rule for Hashing mask

API key: `postgres-datamasking-hash-mask`

This assessment validates whether any of the columns used in the table mapping are not supported by the Hashing Mask transformation rule. It also checks if the length of the source column exceeds 64 characters. Ideally, the target column length should be greater than 64 characters to support hash masking. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying digits randomize transformations does not guarantee any uniqueness.

Verify that Data Validation task settings and Data Masking Digit randomization are not enabled simultaneously

API key: `all-to-all-validation-with-datamasking-digits-randomize`

This premigration assessment verifies that Data Validation setting and Data Masking Digit randomization are not simultaneously enabled, as these features are incompatible.

Verify that Data Validation task settings and Data Masking Hashing mask are not enabled simultaneously

API key: `all-to-all-validation-with-datamasking-hash-mask`

This premigration assessment verifies that Data Validation setting and Data Masking Hashing mask are not simultaneously enabled, as these features are incompatible.

Verify that Data Validation task settings and Data Masking Digit mask are not enabled simultaneously

API key: `all-to-all-validation-with-digit-mask`

This premigration assessment verifies that Data Validation setting and Data Masking Digit mask are not simultaneously enabled, as these features are incompatible.

Validate that at least one selected object exists in the source database

API key: `all-check-source-selection-rules`

This premigration assessment verifies that at least one object specified in the selection rules exists in the source database, including pattern matching for wildcard-based rules.

Validate that target PostgreSQL database contains generated columns

API key: `postgres-check-target-generated-cols`

This premigration assessment validates whether the target PostgreSQL database contains any generated columns (including both STORED and VIRTUAL types) that may require special handling during migration. Generated columns, which compute their values from other columns, need specific verification to ensure compatibility with the target PostgreSQL version and proper data consistency after migration.

Validate that materialized views exist in homogeneous PostgreSQL migrations

API key: `postgres-check-materialized-views`

When migrating between PostgreSQL databases, AWS DMS cannot migrate materialized views. Materialized views must be manually created on your target database after migration.

For more information, see Limitations on using a PostgreSQL database as a DMS source.

Validate that REPLICA IDENTITY FULL conflicts with pglogical plugin usage

API key: `postgres-check-pglogical-replica-identity-full`

This premigration assessment detects tables using REPLICA IDENTITY FULL. While REPLICA IDENTITY FULL is supported using the test_decoding plugin, using it with pglogical will prevent updates from being replicated correctly. Either change the REPLICA IDENTITY setting to DEFAULT/INDEX, or switch to a test_decoding plugin to maintain REPLICA IDENTITY FULL

For more information, see Enabling change data capture (CDC) using logical replication.

Validate that secondary constraints and indexes (non-primary) are present in the source database

API key: `all-check-secondary-constraints`

This premigration assessment verifies that secondary constraints and indexes (foreign keys, check constraints, non-clustered indexes) are present in the source database.

Validate CHAR/VARCHAR columns compatibility for migration to Oracle

API key: `postgres-to-oracle-check-varchar-columns`

This premigration assessment verifies that NCHAR/NVARCHAR2 data type columns used in the target database are compatible with CHAR/VARCHAR columns in the source database.

Validate that `idle_in_transaction_session_timeout` setting is configured on the source database

API key: `postgres-check-idle-in-transaction-session-timeout`

This premigration assessment verifies that the `idle_in_transaction_session_timeout` parameter is not set to 0 on the source database.

Validate that AWS DMS user has required roles for AWS-managed PostgreSQL databases

API key: `postgres-check-rds-roles`

This premigration assessment verifies that the AWS DMS user has been configured with all required roles for AWS-managed PostgreSQL databases. Insufficient roles can cause migration task failures.

Db2 LUW Assessments

This section describes individual premigration assessments for migration tasks that use a Db2 LUW source endpoint.

Topics

- Check Transformation Rule for Digits mask
- Check Transformation Rule for Hashing mask
- Verify that Data Validation task settings and Data Masking Digit randomization are not enabled simultaneously
- Verify that Data Validation task settings and Data Masking Hashing mask are not enabled simultaneously
- Verify that Data Validation task settings and Data Masking Digit mask are not enabled simultaneously
- Verify that target tables have the correct index configuration (Primary Key or Unique Index, not both) for Batch Apply compatibility
- Validate that only 'Limited LOB mode' is used when BatchApplyEnabled is set to true
- Validate if secondary indexes are disabled on the target database during full-load
- Validate that at least one selected object exists in the source database
- Validate that secondary constraints and indexes (non-primary) are present in the source database

Validate if the IBM Db2 LUW database is configured to be recoverable.

API key: `db2-check-archive-config-param`

This premigration assessment validates whether the Db2 LUW database has either or both of the database configuration parameters `LOGARCHMETH1` and `LOGARCHMETH2` set to **ON**.

Validate if the DMS user has the required permissions on the source database to perform a full-load

API key: `db2-check-full-load-privileges`

This premigration assessment validates whether DMS user has all the required permissions on the source database for full-load operations.

Validate if the DMS user has the required permissions on the source database to perform CDC

API key: `db2-check-cdc-privileges`

This premigration assessment validates whether DMS user has all the required permissions on the source database for CDC operations.

Validate if the source IBM Db2 LUW source table has Db2 XML data type

API key: `db2-check-xml-data-type`

This premigration assessment validates if the source IBM Db2 LUW table has Db2 XML data type.

Validate if the source IBM Db2 LUW version is supported by AWS DMS

API key: `db2-validate-supported-versions-source`

This premigration assessment validates if the source IBM Db2 LUW version is supported by AWS DMS.

Validate if the target IBM Db2 LUW version is supported by AWS DMS

API key: `db2-validate-supported-versions-target`

This premigration assessment validates if the target IBM Db2 LUW version is supported by AWS DMS.

Check Transformation Rule for Digits Randomize

API key: `db2-datamasking-digits-randomize`

This assessment validates whether columns used in table mappings are compatible with the Digits Randomize transformation rule. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying digits randomize transformations does not guarantee any uniqueness.

Check Transformation Rule for Digits mask

API key: `db2-datamasking-digits-mask`

This assessment validates whether any columns used in the table mapping are not supported by the Digits Mask transformation rule. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying Digits Mask transformations to such columns could cause DMS task failures since uniqueness cannot be guaranteed.

Check Transformation Rule for Hashing mask

API key: `db2-datamasking-hash-mask`

This assessment validates whether any of the columns used in the table mapping are not supported by the Hashing Mask transformation rule. It also checks if the length of the source column exceeds 64 characters. Ideally, the target column length should be greater than 64 characters to support hash masking. Additionally, the assessment checks if any columns selected for transformation are part of primary keys, unique constraints, or foreign keys, as applying digits randomize transformations does not guarantee any uniqueness.

Verify that Data Validation task settings and Data Masking Digit randomization are not enabled simultaneously

API key: `all-to-all-validation-with-datamasking-digits-randomize`

This premigration assessment verifies that Data Validation setting and Data Masking Digit randomization are not simultaneously enabled, as these features are incompatible.

Verify that Data Validation task settings and Data Masking Hashing mask are not enabled simultaneously

API key: `all-to-all-validation-with-datamasking-hash-mask`

This premigration assessment verifies that Data Validation setting and Data Masking Hashing mask are not simultaneously enabled, as these features are incompatible.

Verify that Data Validation task settings and Data Masking Digit mask are not enabled simultaneously

API key: `all-to-all-validation-with-digit-mask`

This premigration assessment verifies that Data Validation setting and Data Masking Digit mask are not simultaneously enabled, as these features are incompatible.

Verify that target tables have the correct index configuration (Primary Key or Unique Index, not both) for Batch Apply compatibility

API key: `db2-check-batch-apply-target-pk-ui-absence`

Batch Apply requires the target tables to have either Primary or Unique keys, but not both. If a table contains both Primary and Unique Key, the apply mode changes from batch to transactional.

Validate that only 'Limited LOB mode' is used when `BatchApplyEnabled` is set to true

API key: `db2-check-for-batch-apply-lob-mode`

This premigration assessment validates whether DMS task includes LOB columns. If LOB columns are included in the scope of the task, you must use ‘Limited LOB mode’ in order to be able to use `BatchApplyEnabled=true`.

Validate if secondary indexes are disabled on the target database during full-load

API key: `db2-check-secondary-indexes`

This premigration assessment validates whether secondary indexes are disabled during a full-load on the target database. You must disable or remove the secondary indexes during full-load.

Validate that at least one selected object exists in the source database

API key: `all-check-source-selection-rules`

This premigration assessment verifies that at least one object specified in the selection rules exists in the source database, including pattern matching for wildcard-based rules.

Validate that secondary constraints and indexes (non-primary) are present in the source database

API key: `all-check-secondary-constraints`

This premigration assessment verifies that secondary constraints and indexes (foreign keys, check constraints, non-clustered indexes) are present in the source database.

Starting and viewing data type assessments (Legacy)

> **Note**
>
> This section describes legacy content. We recommend that you use premigration assessment runs, described prior in [Specifying, starting, and viewing premigration assessment runs](#).
> Data type assessments are not available in the console. You can only run data type assessments using the API or CLI, and you can only view the results of a data type assessment in the task's S3 bucket.
> The Pre-migration Assessment will automatically run under these conditions:
>
> - During Start Task: If you haven't manually run the assessment during task creation.
> - During Resume Task: If no completed assessment exists within the past 7 days.

A data type assessment identifies data types in a source database that might not get migrated correctly because the target doesn't support them. 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. If your migration task has unsupported data types, AWS DMS creates a report that you can look at to see if your migration task has any unsupported data types. AWS DMS doesn't create a report if your migration task doesn't have 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.

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 following example 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"
]
},
{
"table-name":"table2",
"columns":[
"column11",
"column12"
]
}
]
}
]
```

```
            }
            ]
            }
```

AWS DMS 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. Because AWS DMS creates a bucket in your account, make sure that it doesn't exceed your bucket limit. Otherwise, the data type assessment fails.

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. If a **ResourceNotFoundFault** error occurs when you run `StartReplicationTaskAssessment`, see ResourceNotFoundFault in the Troubleshooting section for information about creating the `dms-access-for-tasks` role manually.

Troubleshooting assessment runs

Following, you can find topics about troubleshooting issues with running assessment reports with AWS Database Migration Service. These topics can help you to resolve common issues.

Topics

ResourceNotFoundFault when running StartReplicationTaskAssessment

You may encounter the following exception when running the StartReplicationTaskAssessment action.

```
An error occurred (ResourceNotFoundFault) when calling the
 StartReplicationTaskAssessment operation: Task assessment has not been run or dms-
access-for-tasks IAM Role not configured correctly
```

If you encounter this exception, create the **dms-access-for-tasks** role by doing the following:

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. On the **Select trusted entity** page, for **Trusted entity type**, choose **Custom trust policy**.
5. Paste the following JSON in the editor, replacing the existing text.

   JSON

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

   The preceding policy grants the `sts:AssumeRole` permission to AWS DMS. When you add the **AmazonDMSRedshiftS3Role** policy, DMS can to create the S3 bucket in your account, and put the data type assessment results into this S3 bucket.
6. Choose **Next**.

7. On the **Add permissions** page, search for and add the **AmazonDMSRedshiftS3Role** policy. Choose **Next**.
8. On the **Name, review, and create** page, name the role **dms-access-for-tasks**. Choose **Create role**.

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.

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 Amazon SNS events and notifications in AWS Database Migration Service

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 and Table state during tasks. For more information about control tables, see Control table task settings.

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. For more information about monitoring AWS DMS metrics, see AWS Database Migration Service metrics. For more information about using AWS DMS task logs, see Viewing and managing AWS DMS task logs.

Time Travel logs

To log and debug replication tasks, you can use AWS DMS Time Travel. In this approach, you use Amazon S3 to store logs and encrypt them using your encryption keys. You can retrieve your S3 logs using date-time filters, then view, download, and obfuscate logs as needed. By doing this, you can "travel back in time" to investigate database activities.

You can use Time Travel with DMS-supported PostgreSQL source endpoints and DMS-supported PostgreSQL and MySQL target endpoints. You can turn on Time Travel only for full-load and CDC tasks and for CDC only tasks. To turn on Time Travel or to modify any existing Time Travel settings, ensure that your task is stopped.

For more information about Time Travel logs, see Time Travel task settings. For best practices for using Time Travel logs, see Troubleshooting replication tasks with Time Travel.

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.

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

- Table state during tasks
- Monitoring replication tasks using Amazon CloudWatch
- AWS Database Migration Service metrics
- Viewing and managing AWS DMS task logs
- Logging AWS DMS API calls with AWS CloudTrail
- AWS DMS Context logging
- Enhanced monitoring dashboard
- Viewing AWS DMS events

Task status

The task status indicates the condition of the task. The following table shows the possible statuses a task can have:

Task status	Description
Creating	AWS DMS is creating the task.
Running	The task is performing the migration duties specified.
Stopped	The task is stopped.
Stopping	The task is being stopped. This is usually an indication of user intervention in the task.
Deleting	The task is being deleted, usually from a request for user intervention.
Failed	The task has failed. For more information, see the task log files.
Error	The task stopped due to an error. A brief description of the task error is provided in the last failure message section of the **Overview** tab.
Running with errors	The task is running with an error status. This usually indicates that one or more of the tables in the task

Task status	Description
	couldn't be migrated. The task continues to load other tables according to the selection rules.
Starting	The task is connecting to the replication instance and to the source and target endpoints. Any filters and transformations are being applied.
Ready	The task is ready to run. This status usually follows the "creating" status.
Modifying	The task is being modified, usually due to a user action that modified the task settings.
Moving	The task is in the process of being moved to another replication instance. The replication remains in this state until the move is complete. Deleting the task is the only operation allowed on the replication task while it's being moved.
Failed-move	The task move has failed for any reason, such as not having enough storage space on the target replication instance. When a replication task is in this state, it can be started, modified, moved, or deleted.
Testing	The database migration specified for this task is being tested in response to running either the StartReplicationTaskAssessmentRun , or the StartReplicationTaskAssessment operation.

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.

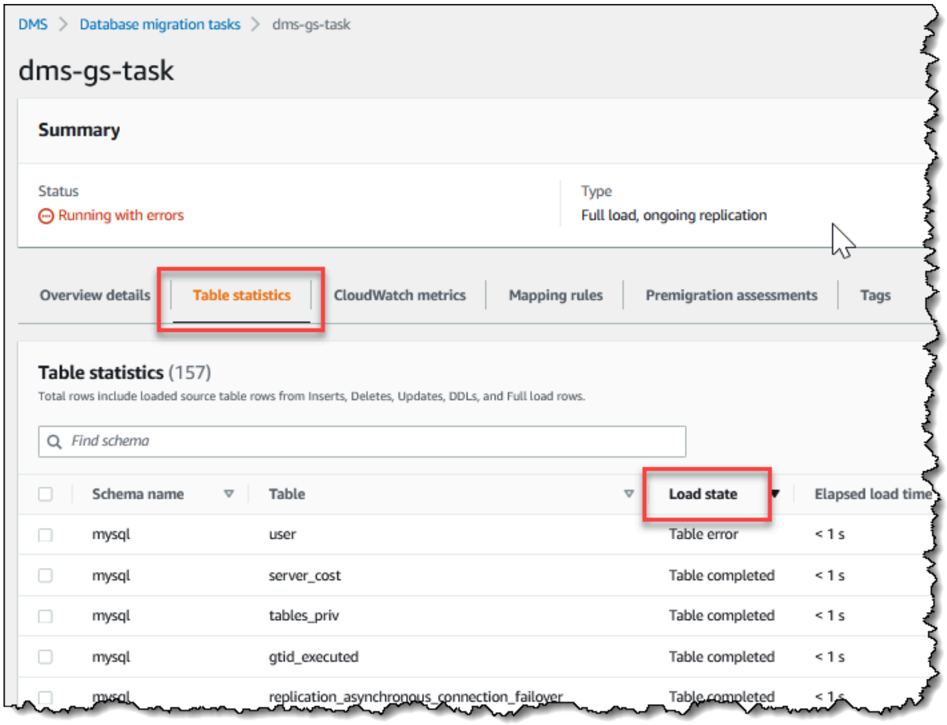
Note that the "last updated" column the DMS console only indicates the time that AWS DMS last updated the table statistics record for a table. It does not indicate the time of the last update to the table.

In addition to using the DMS console, you can *output* a description of current replication tasks, including task status, by using the `aws dms describe-replication-tasks` command in the AWS CLI, as shown in the following example.

```
{
    "ReplicationTasks": [
        {
            "ReplicationTaskIdentifier": "moveit2",
            "SourceEndpointArn": "arn:aws:dms:us-
east-1:123456789012:endpoint:6GGI6YPWWGAYUVLKIB732KEVWA",
            "TargetEndpointArn": "arn:aws:dms:us-
east-1:123456789012:endpoint:EOM4SFKCZEYHZBFGAGZT3QEC5U",
            "ReplicationInstanceArn": "arn:aws:dms:us-
east-1:123456789012:rep:T3OM7OUB5NM2LCVZF7JPGJRNUE",
            "MigrationType": "full-load",
            "TableMappings": ...output omitted... ,
            "ReplicationTaskSettings": ...output omitted... ,
            "Status": "stopped",
            "StopReason": "Stop Reason FULL_LOAD_ONLY_FINISHED",
            "ReplicationTaskCreationDate": 1590524772.505,
            "ReplicationTaskStartDate": 1590619805.212,
            "ReplicationTaskArn": "arn:aws:dms:us-
east-1:123456789012:task:K55IUCGBASJS5VHZJIINA45FII",
            "ReplicationTaskStats": {
                "FullLoadProgressPercent": 100,
                "ElapsedTimeMillis": 0,
                "TablesLoaded": 0,
                "TablesLoading": 0,
                "TablesQueued": 0,
                "TablesErrored": 0,
                "FreshStartDate": 1590619811.528,
                "StartDate": 1590619811.528,
                "StopDate": 1590619842.068
            }
        }
    ]
}
```

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:



State	Description
Table does not exist	AWS DMS cannot find the table on the source endpoint.
Before load	The full load process has been enabled, but it hasn't started yet.
Full load	The full load process is in progress.
Table completed	Full load has completed.
Table cancelled	Loading of the table has been cancelled.
Table error	An error occurred when loading the table.

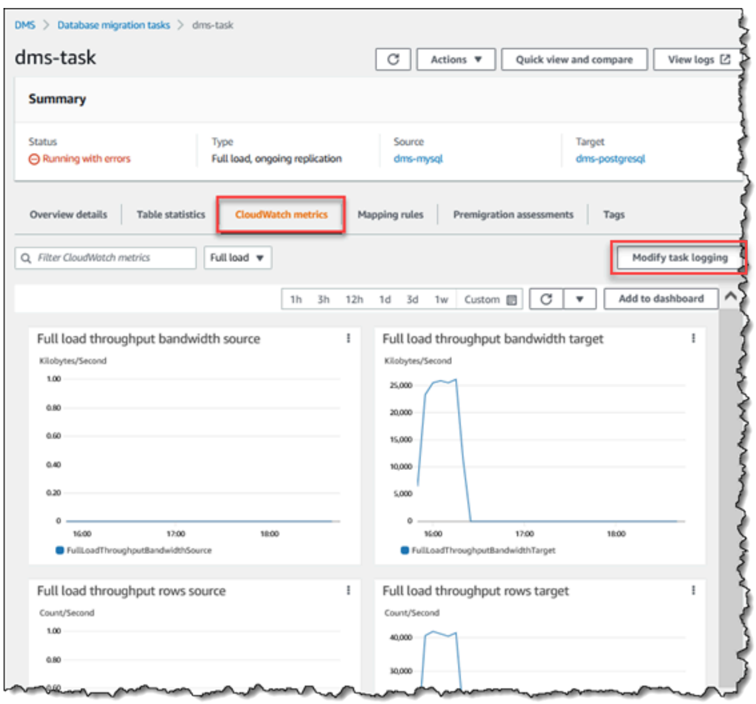
Monitoring replication tasks using Amazon CloudWatch

You can use Amazon CloudWatch alarms or events to more closely track your migration. For more information about Amazon CloudWatch, see What are Amazon CloudWatch, Amazon CloudWatch Events, and Amazon CloudWatch Logs? in the Amazon CloudWatch User Guide. Note that there is a charge for using Amazon CloudWatch.

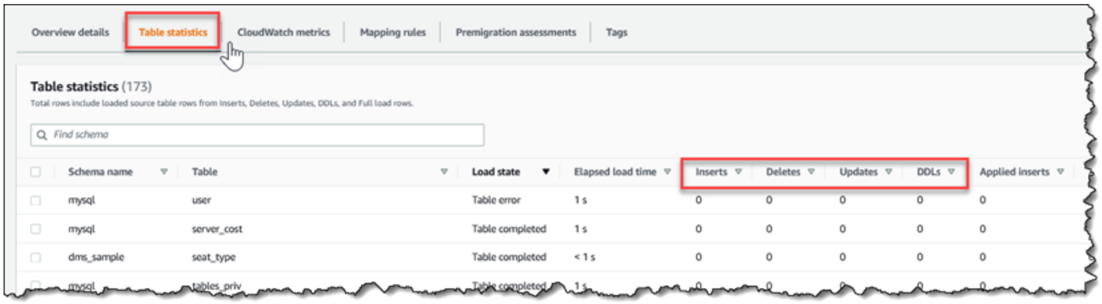
If your replication task doesn't create CloudWatch logs, see AWS DMS does not create CloudWatch logs in the troubleshooting guide.

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 **CloudWatch metrics** tab.

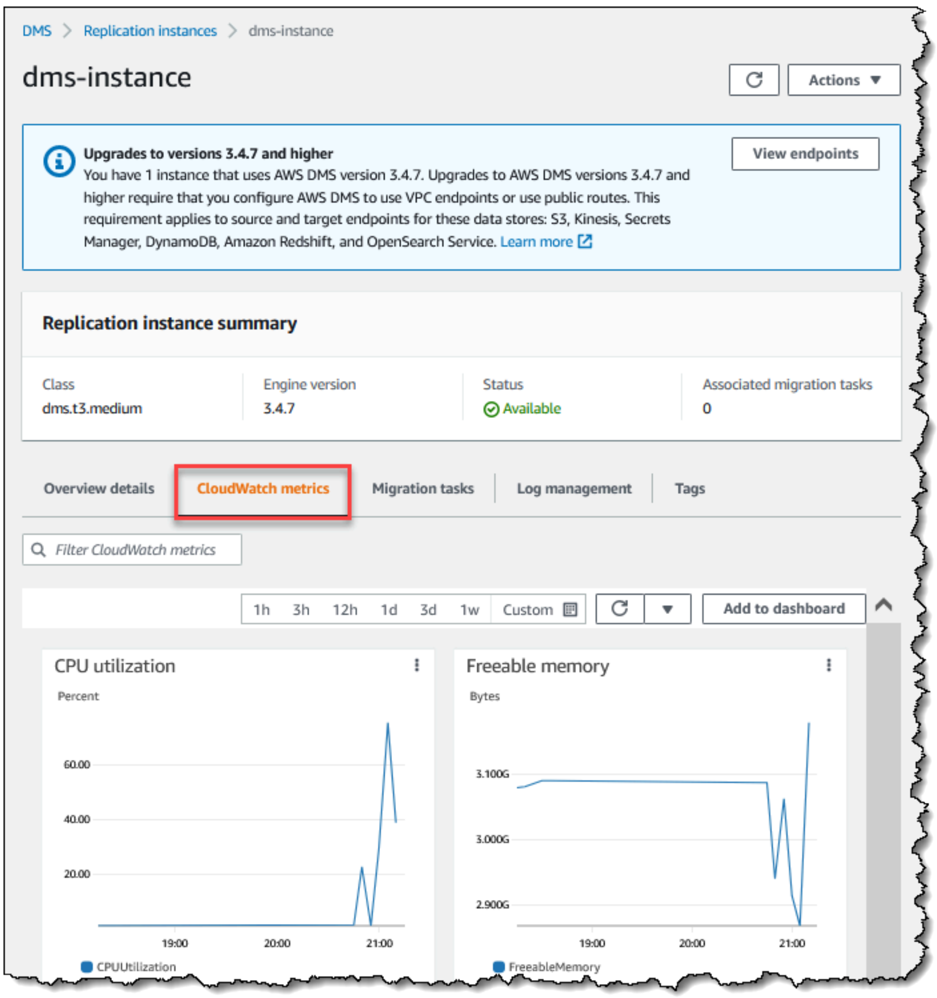
To view and modify the CloudWatch task log settings, choose **Modify task logging**. For more information, see Logging task settings.



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.



In addition, if you select a replication instance from the **Replication Instance** page, you can view performance metrics for the instance by choosing the **CloudWatch metrics** tab.



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.
- **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.
- **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 for the replication host and the source endpoint, as well as for the replication host and the target endpoint. The **CDCLatencySource** and **CDCLatencyTarget** can be used to identify the cause of latency in a task by comparing these related statistics. For example, if the **CDCLatencySource** value is nearly the same as the **CDCLatencyTarget** value, you should first check the source side. However, if the **CDCLatencyTarget** is higher than the **CDCLatencySource**, you should focus on checking the target side latency.

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 is 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.

To roughly estimate the actual memory requirements for an AWS DMS migration task, you can use the following parameters.

LOB columns

An average number of LOB columns in each table in your migration scope.

Maximum number of tables to load in parallel

The maximum number of tables that AWS DMS loads in parallel in one task.

The default value is 8.

LOB chunk size

The size of the LOB chunks, in kilobytes, that AWS DMS uses to replicate data to the target database.

Commit rate during full load

The maximum number of records that AWS DMS can transfer in parallel.

The default value is 10,000.

LOB size

The maximum size of an individual LOB, in kilobytes.

Bulk array size

The maximum number of rows that are fetched or processed by your endpoint driver. This value depends on the driver settings.

The default value is 1,000.

After you determine these values, you can use one of the following methods to estimate the amount of required memory for your migration task. These methods depend on the option that you choose for **LOB column settings** in your migration task.

- For **Full LOB mode**, use the following formula.

```
Required memory = (LOB columns) * (Maximum number of tables to load in parallel) * (LOB chunk size) * (Commit rate during full load)
```

Consider an example where your source tables include on average 2 LOB columns, and the size of the LOB chunks is 64 KB. If you use the default values for `Maximum number of tables to load in parallel` and `Commit rate during full load`, then the amount of required memory for your task is as follows.

```
Required memory = 2 * 8 * 64 * 10,000 = 10,240,000 KB
```

> **Note**
>
> To reduce the value of **Commit rate during full load**, open the AWS DMS console, choose **Database migration tasks**, and create or modify a task. Expand **Advanced settings**, and enter your value for **Commit rate during full load**.

- For **Limited LOB mode**, use the following formula.

  ```
  Required memory = (LOB columns) * (Maximum number of tables to load in parallel) * (LOB size) * (Bulk array size)
  ```

  Consider an example where your source tables include on average 2 LOB columns, and the maximum size of an individual LOB is 4,096 KB. If you use the default values for `Maximum number of tables to load in parallel` and `Bulk array size`, then the amount of required memory for your task is as follows.

  ```
  Required memory = 2 * 8 * 4,096 * 1,000 = 65,536,000 KB
  ```

For AWS DMS to perform conversions optimally, the CPU must be available when the conversions happen. Overloading the CPU and not having enough CPU resources can result in slow migrations. AWS DMS can be CPU-intensive, especially when performing heterogeneous migrations and replications such as migrating from Oracle to PostgreSQL.

Replication instance metrics

Replication instance monitoring includes Amazon CloudWatch metrics for the following statistics.

Metric	Description
AvailableMemory	An estimate of how much memory is available for starting new applications, without swapping. For more information, see

Metric	Description
	MemAvailable value in /proc/memInfo section of the Linux man-pages. Units: Bytes
CPUAllocated	The percentage of CPU maximally allocated for the task (0 means no limit). AWS DMS raises this metric against the combined dimensions of ReplicationInstanceIdentifer and ReplicationTaskIdentifier in the CloudWatch console. Use the ReplicationInstanceIdentifier, ReplicationTaskIdentifier category to view this metric. Units: Percent
CPUUtilization	The percentage of allocated vCPU (virtual CPU) currently in use on the instance. Units: Percent
DiskQueueDepth	The number of outstanding read/write requests (I/Os) waiting to access the disk. Units: Count
FreeStorageSpace	The amount of available storage space. Units: Bytes
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

Metric	Description
FreeableMemory	The amount of available random access memory. Units: Bytes
MemoryAllocated	The maximum allocation of memory for the task (0 means no limits). AWS DMS raises this metric against the combined dimensions of `ReplicationInstanceIdentifer` and `ReplicationTaskIdentifier` in the CloudWatch console. Use the `ReplicationInstanceIdentifier, ReplicationTaskIdentifier` category to view this metric. Units: MiB
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

Metric	Description
SwapUsage	The amount of swap space used on the replication instance. Units: Bytes
NetworkTransmitThroughput	The outgoing (Transmit) network traffic on the replication instance, including customer database traffic. Units: Bytes/second
NetworkReceiveThroughput	The incoming (Receive) network traffic on the replication instance, including customer database traffic. Units: Bytes/second

Replication task metrics

Replication task monitoring includes metrics for the following statistics.

Metric	Description
FullLoadThroughputBandwidthTarget	Outgoing data transmitted from a full load for the target in KB 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.

Metric	Description
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.
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.

Metric	Description
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. CDCSourceLatency can be 0 when there is no replication lag between the source and the replication instance. CDCSourceLatency can also become zero when the replication task attempts to read the next event in the source's transaction log and there are no new events compared to the last time it read from the source. When this happens, the task resets the CDCSourceLatency to 0.
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. Target latency is the difference between the replicati on instance server time and the oldest unconfirmed event id forwarded to a target component. In other words, target latency is the timestamp difference between the replication instance and the oldest event applied but unconfirmed by TRG endpoint (99%). 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 CDCLatenc ySource. 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

Metric	Description
CPUUtilization	The percentage of CPU being used by a task across multiple cores. The semantics of task CPUUtilization is slightly different from replication CPUUtilizaiton. If 1 vCPU is fully used, it indicates 100%, but if multiple vCPUs are in use, the value could be above 100%. Units: Percent
SwapUsage	The amount of swap used by the task. Units: Bytes
MemoryUsage	The control group (cgroup) **memory.usage_in_bytes** consumed by a task. DMS uses cgroups to control the usage of system resources such as memory and CPU. This metric indicates a task's memory usage in Megabytes within the cgroup allocated for that task. The cgroup limits are based on the resources available for your DMS replicati on instance class. **memory.usage_in_bytes** consists of resident set size (RSS), cache, and swap components of memory. The operating system can reclaim cache memory if needed. We recommend that you also monitor the replication instance metric, **AvailableMemory**. AWS DMS raises this metric against the combined dimension s of `ReplicationInstanceIdentifer` and `ReplicationTaskIdentifier` in the CloudWatch console. Use the `ReplicationInstanceIdentifier, ReplicationTaskIdentifier` category to view this metric.

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.

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.

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
   }
}
```

Note

For each replication instance, AWS DMS deletes logs that are older than 10 days.

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": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-08-01T16:42:09Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AKIAIOSFODNN7EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/admin",
                "accountId": "123456789012",
                "userName": "admin"
            }
        }
    },
    "eventTime": "2018-08-02T00:11:44Z",
    "eventSource": "dms.amazonaws.com",
    "eventName": "RebootReplicationInstance",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "72.21.198.64",
    "userAgent": "console.amazonaws.com",
    "requestParameters": {
        "forceFailover": false,
        "replicationInstanceArn": "arn:aws:dms:us-
east-1:123456789012:rep:EX4MBJ2NMRDL3BMAYJOXUGYPUE"
    },
    "responseElements": {
        "replicationInstance": {
            "replicationInstanceIdentifier": "replication-instance-1",
            "replicationInstanceStatus": "rebooting",
            "allocatedStorage": 50,
            "replicationInstancePrivateIpAddresses": [
                "172.31.20.204"
            ],
            "instanceCreateTime": "Aug 1, 2018 11:56:21 PM",
            "autoMinorVersionUpgrade": true,
            "engineVersion": "2.4.3",
```

```
            "publiclyAccessible": true,
            "replicationInstanceClass": "dms.t2.medium",
            "availabilityZone": "us-east-1b",
            "kmsKeyId": "arn:aws:kms:us-east-1:123456789012:key/
f7bc0f8e-1a3a-4ace-9faa-e8494fa3921a",
            "replicationSubnetGroup": {
                "vpcId": "vpc-1f6a9c6a",
                "subnetGroupStatus": "Complete",
                "replicationSubnetGroupArn": "arn:aws:dms:us-
east-1:123456789012:subgrp:EDHRVRBAAAPONQAIYWP4NUW22M",
                "subnets": [
                    {
                        "subnetIdentifier": "subnet-cbfff283",
                        "subnetAvailabilityZone": {
                            "name": "us-east-1b"
                        },
                        "subnetStatus": "Active"
                    },
                    {
                        "subnetIdentifier": "subnet-d7c825e8",
                        "subnetAvailabilityZone": {
                            "name": "us-east-1e"
                        },
                        "subnetStatus": "Active"
                    },
                    {
                        "subnetIdentifier": "subnet-6746046b",
                        "subnetAvailabilityZone": {
                            "name": "us-east-1f"
                        },
                        "subnetStatus": "Active"
                    },
                    {
                        "subnetIdentifier": "subnet-bac383e0",
                        "subnetAvailabilityZone": {
                            "name": "us-east-1c"
                        },
                        "subnetStatus": "Active"
                    },
                    {
                        "subnetIdentifier": "subnet-42599426",
                        "subnetAvailabilityZone": {
                            "name": "us-east-1d"
                        },
```

```
                        "subnetStatus": "Active"
                    },
                    {
                        "subnetIdentifier": "subnet-da327bf6",
                        "subnetAvailabilityZone": {
                            "name": "us-east-1a"
                        },
                        "subnetStatus": "Active"
                    }
                ],
                "replicationSubnetGroupIdentifier": "default-vpc-1f6a9c6a",
                "replicationSubnetGroupDescription": "default group created by console
 for vpc id vpc-1f6a9c6a"
            },
            "replicationInstanceEniId": "eni-0d6db8c7137cb9844",
            "vpcSecurityGroups": [
                {
                    "vpcSecurityGroupId": "sg-f839b688",
                    "status": "active"
                }
            ],
            "pendingModifiedValues": {},
            "replicationInstancePublicIpAddresses": [
                "18.211.48.119"
            ],
            "replicationInstancePublicIpAddress": "18.211.48.119",
            "preferredMaintenanceWindow": "fri:22:44-fri:23:14",
            "replicationInstanceArn": "arn:aws:dms:us-
east-1:123456789012:rep:EX4MBJ2NMRDL3BMAYJOXUGYPUE",
            "replicationInstanceEniIds": [
                "eni-0d6db8c7137cb9844"
            ],
            "multiAZ": false,
            "replicationInstancePrivateIpAddress": "172.31.20.204",
            "patchingPrecedence": 0
        }
    },
    "requestID": "a3c83c11-95e8-11e8-9d08-4b8f2b45bfd5",
    "eventID": "b3c4adb1-e34b-4744-bdeb-35528062a541",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

AWS DMS Context logging

AWS DMS uses context logging to give you information about a migration in progress. Context logging writes information, such as the following, to the task's CloudWatch log:

- Information about the task's connection to the source and target databases.
- Replication task behavior. You can use the task logs to diagnose replication issues.
- SQL statements without data that AWS DMS executes on source and target databases. You can use the SQL logs to diagnose unexpected migration behavior.
- Stream position details for each CDC event.

Context logging is only available in AWS DMS version 3.5.0 or higher.

AWS DMS turns on context logging by default. To control context logging, set the `EnableLogContext` task setting to `true` or `false`, or by modifying the task in the console.

AWS DMS writes context log information to the CloudWatch log's replication task every three minutes. Make sure that your replication instance has sufficient space for its application log. For more information about managing task logs, see Viewing and managing AWS DMS task logs.

Topics

Object Types

AWS DMS produces context logging in CloudWatch for the following object types.

Object Type	Description
`TABLE_NAME`	These log entries contain information about tables that are in scope with the current task mapping rule. You can use these entries to examine the table events for a specific period during migration.

Object Type	Description
SCHEMA_NAME	These log entries contain information about schemas used by the current task mapping rule. You can use these entries to determine which schema AWS DMS is using for a specific period during migration.
TRANSACTION_ID	These entries contain the transaction ID for each DML/ DDL change captured from the source database. You can use these log entries to determine what changes happened during a given transaction.
CONNECTION_ID	These entries contain the connection ID. You can use these log entries to determine which connection AWS DMS uses for each migration step.
STATEMENT	These entries contain the SQL code used to fetch, process, and apply each migration change.
STREAM_POSITION	These entries contain the position in the transaction log file for each migration action on the source database. The format for these entries varies between source database engine types. You can also use this information to determine a starting position for a recovery checkpoint when configuring CDC-only replication.

Logging Examples

This section contains examples of log records that you can use to monitor replication and diagnose replication issues.

Connection log examples

This section contains log samples that include connection IDs.

```
2023-02-22T10:09:29 [SOURCE_CAPTURE  ]I:  Capture record 1 to internal
 queue from Source  {operation:START_REGULAR (43), connectionId:27598,
 streamPosition:0000124A/6800A778.NOW}  (streamcomponent.c:2920)

2023-02-22T10:12:30 [SOURCE_CAPTURE  ]I:  Capture record 0 to internal queue from
 Source  {operation:IDLE (51), connectionId:27598}  (streamcomponent.c:2920)

2023-02-22T11:25:27 [SOURCE_CAPTURE  ]I:  Capture record 0 to internal queue
 from Source  {operation:IDLE (51), columnName:region, connectionId:27598}
 (streamcomponent.c:2920)
```

Task behavior log examples

This section contains log samples about replication task log behavior. You can use this information to diagnose replication issues, such as a task in the IDLE status.

The following SOURCE_CAPTURE logs indicate that there are no events available to read from the source database log file, and contain TARGET_APPLY records that indicate that there are no events received from AWS DMS CDC components to apply to the target database. These events also contain previously applied event-related context details.

```
2023-02-22T11:23:24 [SOURCE_CAPTURE  ]I:  No Event fetched from wal log
 (postgres_endpoint_wal_engine.c:1369)
2023-02-22T11:24:29 [TARGET_APPLY    ]I:  No records received to load
 or apply on target , waiting for data from upstream. The last context
 is  {operation:INSERT (1), tableName:sales_11, schemaName:public,
 txnId:18662441, connectionId:17855, statement:INSERT INTO
 "public"."sales_11"("sales_no","dept_name","sale_amount","sale_date","region") values
 (?,?,?,?,?),
```

SQL statement log examples

This section contains log samples about SQL statements run on source and target databases. The SQL statements you see in the logs only show the SQL statement; they don't show the data. The following TARGET_APPLY log shows an INSERT statement that ran on the target.

```
2023-02-22T11:26:07 [TARGET_APPLY    ]I:  Applied record 2193305 to
 target  {operation:INSERT (1), tableName:sales_111, schemaName:public,
 txnId:18761543, connectionId:17855, statement:INSERT INTO
 "public"."sales_111"("sales_no","dept_name","sale_amount","sale_date","region") values
 (?,?,?,?,?),
```

Limitations

The following limitations apply to AWS DMS context logging:

- While AWS DMS creates minimal logging for all endpoint types, extensive engine-specific context logging is only available for the following endpoint types. We recommend turning on context logging when using these endpoint types.
  - MySQL
  - PostgreSQL
  - Oracle
  - Microsoft SQL Server
  - MongoDB/ Amazon DocumentDB
  - Amazon S3

Enhanced monitoring dashboard

The enhanced monitoring dashboard provides comprehensive visibility into critical metrics related to your monitoring tasks and replication instances. It allows you to filter, aggregate, and visualize metrics for specific resources you want to track. The dashboard directly publishes existing CloudWatch metrics, enabling you to monitor resource performance without altering data point sampling times.

Topics

- Overview of the enhanced monitoring dashboard
- Viewing metrics in the enhanced monitoring dashboard
- Enhanced monitoring dashboard views
- Retention of enhanced monitoring metrics

Overview of the enhanced monitoring dashboard

The enhanced monitoring dashboard is available in the AWS DMS console. It provides a user-friendly interface with intuitive visualizations and charts, that enable you to monitor, analyze, and optimize your data migration processes effectively. With enhanced monitoring, you can streamline monitoring processes and quickly identify potential issues by viewing all relevant information in a centralized location.

On the enhanced monitoring dashboard, you can view metrics for tasks and replication instances and details for endpoints. You can also track the number of active CloudWatch alarms and the service health status for the current Region. The dashboard is available in all Commercial Regions where AWS DMS is available. There is no additional cost to use this dashboard.

> **Note**
>
> The enhanced monitoring dashboard doesn't support AWS DMS serverless replications.

Viewing metrics in the enhanced monitoring dashboard

To view the metrics in the enhanced monitoring dashboard, make sure that you have an IAM role with CloudWatch permissions. In addition, you need the `cloudwatch:DescribeAlarms` and `health:DescribeEvents` permissions to view the metrics. Finally, to access CloudWatch metrics in the enhanced monitoring dashboard, you must also have permission to access the GetMetricData API within your IAM policies. Without these permissions or the permissions for only a subset of APIs, you won't be able to view the metrics in the enhanced monitoring dashboard. For information about the IAM permissions you need, see IAM permissions needed to use AWS DMS.

Enhanced monitoring dashboard views

The AWS DMS enhanced monitoring dashboard has two types of views: Tasks and Replication instance views.

Tasks view

The **Tasks** view in the enhanced dashboard of AWS DMS provides a comprehensive overview of your data migration tasks. This view presents a centralized location where you can monitor and analyze various aspects of your tasks through intuitive charts and visualizations.

The **Tasks** view offers insights into key metrics and statistics related to your migration tasks, which enables you to gain a better understanding of their performance, progress, and overall health. Following are some of the key features and benefits of the **Tasks** view:

- **Task status summary**: This chart displays the distribution of your tasks across different status categories, such as running, stopped, failed, or completed. You can quickly identify tasks that require attention and take appropriate actions.
- **Performance metrics**: These charts illustrate the performance of your tasks, including metrics such as throughput, latency, CPU utilization, and more. These metrics help you identify potential bottlenecks and optimize your migration process.
- **Error analysis**: In case of task failures or errors, the **Tasks** view provides detailed information about the errors encountered and their associated log entries. This information can help you in troubleshooting and resolving issues more efficiently.
- **Historical trends**: The dashboard incorporates historical data, allowing you to analyze how your tasks have performed over a period of time. You can identify patterns, track progress, and make informed decisions based on these historical trends.
- **Filtering and sorting**: The **Tasks** view enables you to filter and sort tasks based on various criteria, such as task name, tags, or specific time ranges. This flexibility allows you to focus on the tasks or aspects that are most relevant to your needs.

The following list describes the metrics that you can see in the **Tasks** view:

- **Full load throughput bandwidth - source**: Represents incoming data transmitted from a full load from the source in KB per second.
- **Full load throughput bandwidth - target**: Represents outgoing data transmitted from a full load for the target in KB per second.
- **Full load throughput rows - source**: Represents incoming changes from a full load from the target in rows per second.
- **Full load throughput rows - target**: Represents outgoing changes from a full load for the target in rows per second.
- **CDC throughput bandwidth - source**: Represents network bandwidth for the source in KB per second.

  CDC throughput bandwidth records bandwidth on sampling points. If no network traffic is found, the value is zero. Because CDC doesn't issue long-running transactions, network traffic might not be recorded.

- **CDC throughput bandwidth - target**: Represents network bandwidth for the target in KB per second.

  CDC throughput bandwidth records bandwidth on sampling points. If no network traffic is found, the value is zero. Because CDC doesn't issue long-running transactions, network traffic might not be recorded.
- **CDC throughput rows - source**: Represents incoming task changes from the source in rows per second.
- **CDC throughput rows - target**: Represents outgoing task changes for the target in rows per second.
- **CDC latency - source**: Represents the gap, in seconds, between the last event captured from the source endpoint and current system time stamp of the AWS DMS instance. If no changes have been captured from the source due to task scoping, AWS DMS sets this value to zero.
- **CDC latency - target**: Represents the gap, in seconds, between the first event timestamp waiting to commit on the target and the current timestamp of the AWS DMS instance. This value occurs if there are transactions that aren't handled by target. Otherwise, target latency is the same as source latency if all transactions are applied. Target latency should never be smaller than the source latency.
- **CPU utilization**: Represents the percentage of CPU being used by a task across multiple cores. The semantics of task `CPUUtilization` is slightly different from the semantics of replication instance `CPUUtilizaiton`. If 1 vCPU is fully used, it indicates 100%, but if multiple vCPUs are in use, the value could exceed 100%.
- **Memory usage**: Represents the control group (cgroup) `memory.usage_in_bytes` consumed by a task. AWS DMS uses cgroups to control the usage of system resources, such as memory and CPU. This metric indicates a task's memory usage in Megabytes within the cgroup allocated for that task.

  The cgroup limits are based on the resources available for your AWS DMS replication instance class. `memory.usage_in_bytes` consists of resident set size (RSS), cache, and swap components of memory. The operating system can reclaim cache memory if needed. We recommend that you also monitor the replication instance metric, AvailableMemory.

  AWS DMS raises this metric against the combined dimensions of `ReplicationInstanceIdentifer` and `ReplicationTaskIdentifier` in the CloudWatch console. Use the `ReplicationInstanceIdentifier`, `ReplicationTaskIdentifier` category to view this metric.

- **Validation record count**: This chart is visible only if validation is enabled for the AWS DMS task. This is a combination of validation metrics available for the AWS DMS task, which includes the following:
  - `ValidationSucceededRecordCount` – Number of rows that AWS DMS validated per minute.
  - `ValidationAttemptedRecordCount` – Number of rows that the 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.

In addition to the preceding metrics, you can customize the **Task** view and include additional metrics by adding them as a widgets. For information about these metrics, we recommend you review the following documentation:

- All available metrics for AWS DMS tasks for migration and replication, see Monitoring AWS DMS tasks.
- All available validation related metrics, see Data validation.

Replication instance view

The **Replication instance** view provides a comprehensive overview of your replication instances, allowing you to monitor and manage your data replication infrastructure effectively. This view presents a centralized location where you can analyze various aspects of your replication instances through intuitive charts and visualizations.

The **Replication instance** view offers insights into key metrics and statistics related to your replication instances, which enables you to gain a better understanding of their performance, resource utilization, and overall health. Following are some of the key features and benefits of the **Replication instance** view:

- **Instance status summary**: This chart displays details of the selected replication instance, such as availability zones, instance class, engine version, allocated storage, and current health status.
- **Resource utilization**: The dashboard presents charts that illustrate the resource utilization of your replication instances, including metrics, such as CPU, memory, and disk usage. These metrics help you identify potential resource constraints and optimize your replication infrastructure.

- **Historical trends**: The dashboard incorporates historical data, allowing you to analyze how your replication instances have performed over a period of time. You can identify patterns, track changes, and make informed decisions based on these historical trends.
- **Filtering and sorting**: The **Replication instance** view enables you to filter and sort replication instances based on various criteria, such as instance name or specific time ranges. This flexibility allows you to focus on the instances or aspects that are most relevant to your needs.

The following list describes the metrics that you can see in the **Replication instance** view:

- **CloudWatch alarms**: Represents the summary of the alarms in the AWS/DMS namespace.
- **CPU utilization**: Represents the percentage of CPU being used by a task across multiple cores. The semantics of task `CPUUtilization` is slightly different from the semantics of replication `CPUUtilizaiton`. If 1 vCPU is fully used, it indicates 100%, but if multiple vCPUs are in use, the value could exceed 100%.
- **Memory usage by tasks per instance**: Represents the resident set size (RSS) occupied by a task. It indicates the portion of memory occupied by a task held in main memory (RAM). Because 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.
- **Memory**: Represents the amount of memory available, in use, or can be freed to use, and the amount of swap space used.
- **Available memory**: Represents an estimate of how much memory is available for starting new applications, without swapping.
- **Free memory**: Represents the amount of physical memory available for use by applications, page cache, and for the kernel's own data structures.
- **Freeable memory**: Freeable memory is not an 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 is a combination of buffers and cache in use on the replication.
- **Swap usage**: Represents the amount of swap space used on the replication instance.
- **Free storage**: Represents the amount of available storage space in bytes.

In addition to the preceding metrics, you can customize the **Replication instance** view and include additional metrics by adding them as a widgets. For information about these metrics, we recommend you review the following documentation:

- All available metrics for AWS DMS tasks for migration and replication, see Monitoring AWS DMS tasks.
- All available validation related metrics, see Data validation.

Retention of enhanced monitoring metrics

By default, enhanced monitoring metrics follow the retention policy for CloudWatch metrics. For information about changing the retention policy, see Change log data retention in CloudWatch Logs in the *Amazon CloudWatch User Guide*.

Viewing AWS DMS events

You can retrieve the following event information for your AWS DMS resources:

- Resource name
- Resource type
- Date and time of the event
- Message summary of the event

You can access events in the Events tab of the AWS Management Console, which shows events from the past 7 days. You can also retrieve events by using the `describe-events` AWS CLI command, or the `DescribeEvents` DMS API operation. If you use the AWS CLI or the DMS API to view events, you can retrieve events for up to the past 14 days.

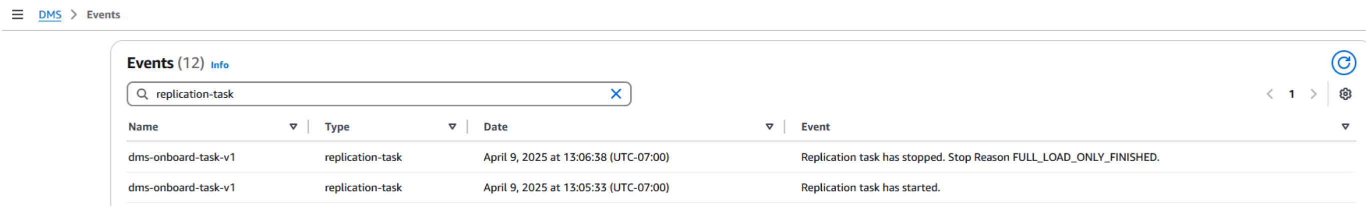
> **Note**
>
> If you need to store events for longer periods of time, you can send AWS DMS events to EventBridge and send the event details to another service for storing. For more information, see Using Amazon EventBridge event rules for AWS DMS.
> For descriptions of the AWS DMS events, see AWS DMS event categories and event messages.

DMS Console

To view all AWS DMS events for the past 7 days in the DMS Console:

1. Sign in to the AWS Management Console and open the DMS console at https://console.aws.amazon.com/dms/v2
2. From the navigation page, select **Events**.
3. Enter a search term to filter your results.

   The following example shows a list of events filtered by the characters **`replication-task`**.



AWS CLI

To view all events generated in the last hour, run [`describe-events`] AWS DMS CLI command with no parameters.

```
aws dms describe-events
```

The following sample output shows that a replication task has been modified:

```
{
    "Events": [
        {
            "SourceIdentifier": "dms-task",
            "SourceType": "replication-task",
            "Message": "Replication task has modified.",
            "EventCategories": [
                "configuration change"
            ],
            "Date": "2025-04-09T12:48:22.231000-07:00"
        }
    ]
}
```

To view all AWS DMS events for the past 14 days (20160 minutes), run the [`describe-events`] AWS CLI command and set the `--duration` parameter to **`20160`**.

```
aws dms describe-events --duration 20160
```

The following example shows the events in the specified time range for the replication instance `dms-replication-instance`.

```
aws dms describe-events \
    --source-identifier dms-replication-instance \
    --source-type replication-instance \
    --start-time 2025-04-10T22:00Z \
    --end-time 2025-04-12T23:59Z
```

The following sample output shows the status of a replication instance:

```
{
    "Events": [
        {
            "SourceIdentifier": "dms-replication-instance",
            "SourceType": "replication-instance",
            "Message": "Replication application shutdown",
            "EventCategories": [],
            "Date": "2025-04-12T03:11:43.270000-07:00"
        },
        {
            "SourceIdentifier": "dms-replication-instance",
            "SourceType": "replication-instance",
            "Message": "Replication application restarted",
            "EventCategories": [],
            "Date": "2025-04-12T03:19:58.214000-07:00"
        }
    ]
}
```

AWS DMS API

You can view all AWS DMS instance events for the past 14 days by running the `DescribeEvents` DMS API operation and setting the `--duration` parameter to **`20160`**.

Working with Amazon EventBridge events and notifications in AWS Database Migration Service

You can use Amazon EventBridge to provide notification of when an AWS DMS event occurs, for example the creation or deletion of a replication instance. EventBridge receives events and routes notification of an event as defined by event rules. You can work with notifications in any form supported by Amazon EventBridge for an AWS Region. For more information about using Amazon EventBridge, see [What is Amazon EventBridge?] in the *Amazon EventBridge User Guide*.

> **Note**
>
> Working with Amazon EventBridge events is supported in AWS DMS version 3.4.5 and higher.

EventBridge receives an event, an indicator of a change in AWS DMS environment, and applies a rule to route the event to a notification mechanism. Rules match events to notification mechanisms based on the structure of the event, called an *event pattern*.

AWS DMS groups events into categories that you can apply an event rule to, so you can be notified when an event in that category occurs. For example, suppose that you apply an EventBridge event rule to the Creation category for a given replication instance. You're then notified whenever a creation-related event occurs that affects your replication instance. If you apply a rule to a Configuration Change category for a replication instance, you're notified when the replication instance's configuration is changed. For a list of the event categories provided by AWS DMS, see the AWS DMS event categories and event messages, following.

> **Note**
>
> To allow publishing from events.amazonaws.com, make sure to update your Amazon SNS topics' access policies. For more information, see [Using resource-based policies for Amazon EventBridge] in the *Amazon EventBridge User Guide*.
> For more information about moving event subscriptions to Amazon EventBridge, see [migrate active event subscriptions from DMS to Amazon EventBridge.]
> For more information on using text messages with Amazon SNS, see [Sending and receiving SMS notifications using Amazon SNS].

Using Amazon EventBridge event rules for AWS DMS

Amazon EventBridge sends event notifications to the addresses that you provide when you create an EventBridge event rule. You might want to create several different rules. For example, you might create one rule receiving all event notifications and another rule that includes only critical events for your production DMS resources. You can also turn on or turn off event notifications in EventBridge.

To create Amazon EventBridge rules that react to AWS DMS events

- Perform the steps described in [Creating Amazon EventBridge rules that react to events](#) in the *Amazon EventBridge User Guide*, and create a rule for AWS DMS events:

    a. Specify a notification action to take when EventBridge receives an event that matches the event pattern in the rule. When an event matches, EventBridge sends the event and invokes the action defined in the rule.

    b. For **Service provider**, choose **AWS**.

    c. For **Service name**, choose **Database Migration Service (DMS)**.

You can then begin to receive event notifications.

The following JSON example shows an EventBridge events model for an AWS DMS service.

```
{
   "version":"0",
   "id":"11a11b11-222b-333a-44d4-01234a5b67890",
   "detail-type":"DMS Replication Task State Change",
   "source":"aws.dms",
   "account":"0123456789012",
   "time":"1970-01-01T00:00:00Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:dms:us-east-1:012345678901:task:AAAABBBB0CCCCDDDDEEEEE1FFFF2GGG3FFFFFF3"
   ],
   "detail":{
      "type":"REPLICATION_TASK",
      "category":"StateChange",
      "eventType":"REPLICATION_TASK_STARTED",
      "eventId":"DMS-EVENT-0069",
```

```
      "resourceLink":"https://console.aws.amazon.com/dms/v2/home?region=us-
east-1#taskDetails/taskName",
      "detailMessage":"Replication task started, with flag = fresh start"
   }
}
```

For the list of categories and events that you can be notified of, see the following section.

AWS DMS event categories and event messages

AWS DMS generates a significant number of events in categories that you can identify. Each category applies to a replication instance or replication task source types.

Topics

- ReplicationInstance event messages
- ReplicationTask event messages
- Replication event messages

ReplicationInstance event messages

The following table shows the possible categories and events for the **ReplicationInstance** source type.

Category	Event ID	Description
Creation	DMS-EVENT-0067	A replication instance is being created.
Deletion	DMS-EVENT-0066	The replication instance is being deleted.
Configura tion Change	DMS-EVENT-0012	The replication instance class for this replication instance is being changed.
Configura tion Change	DMS-EVENT-0018	The storage for the replication instance is being increased.

Category	Event ID	Description
Configura tion Change	DMS-EVENT-0024	The replication instance is transitio ning to a Multi-AZ configuration.
Configura tion Change	DMS-EVENT-0030	The replication instance is transitio ning to a Single-AZ configuration.
Maintenan ce	DMS-EVENT-0026	Offline maintenance of the replication instance is taking place. The replicati on instance is currently unavailable.
Creation	DMS-EVENT-0005	A replication instance has been created.
Deletion	DMS-EVENT-0003	The replication instance has been deleted.
Configura tion Change	DMS-EVENT-0014	The replication instance class for this replication instance has changed.
Configura tion Change	DMS-EVENT-0017	The storage for the replication instance has been increased.
Configura tion Change	DMS-EVENT-0025	The replication instance has finished transitioning to a Multi-AZ configura tion.
Configura tion Change	DMS-EVENT-0029	The replication instance has finished transitioning to a Single-AZ configura tion.
Maintenan ce	DMS-EVENT-0047	Management software on the replicati on instance has been updated.
Maintenan ce	DMS-EVENT-0027	Offline maintenance of the replication instance is complete. The replication instance is now available.

Category	Event ID	Description
Maintenance	DMS-EVENT-0068	Replication instance is in a state that cannot be upgraded.
Failover	DMS-EVENT-0034	If you request Failover too frequently, this event occurs instead of regular failover events.
Failure	DMS-EVENT-0031	Replication instance put into %s state.
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.
Failure		Replication instance put into incompatible-parameters
Failover		Timed out waiting for a state safe to initiate user requested failover
Failover	DMS-EVENT-0013	Failover started for a Multi-AZ replication instance.
Failover	DMS-EVENT-0049	Failover has been completed for a Multi-AZ replication instance.
Failover	DMS-EVENT-0050	Multi-AZ activation has started.
Failover	DMS-EVENT-0051	Multi-AZ activation completed.
StateChange		General and slow query logs have been automatically rotated as %s

Category	Event ID	Description
StateChange		AWS DMS is unable to access the KMS encryption key for application instance %s. This is likely due to the key being disabled or AWS DMS being unable to access it. If this continues the application will be placed into an inaccessible state. Please refer to the troubleshooting section in the AWS DMS documentation for further details.
StateChange		AWS DMS can now successfully access the KMS encryption key for application instance %s.
StateChange		Amazon DMS has been unable to access the KMS encryption key for application instance %s. This application will be placed into an inaccessible state. Please refer to the troubleshooting section in the Amazon DMS documentation for further details.
StateChange		App Restart on HM as part of Replication Instance creation
StateChange		App Shutdown on HM as part of Replication Instance deletion
Failover	DMS-EVENT-0015	Multi-AZ failover to standby complete.
LowStorage	DMS-EVENT-0007	Free storage for the replication instance is low.
LowStorage		Allocated inodes have been exhausted - scale storage to resolve

ReplicationTask event messages

The following table shows the possible categories and events for the **ReplicationTask** source type.

Category	Event ID	Description
Failure	DMS-EVENT-0078	A replication task has failed.
Failure	DMS-EVENT-0082	A call to clean task data has failed.
State Change	DMS-EVENT-0081	Reload of table details has been requested.
State Change		Replication task has been copied.
State Change		Replication task copy has failed.
State Change		Replication task has been moved.
State Change		Replication task move has failed.
State Change		Creation of target task failed.
State Change		Replication task assessment run has started.
State Change		Replication task assessment run has finished successfully.
State Change		Replication task assessment run has finished with failure.
StateChange		Replication task assessment run has finished with warning.

Category	Event ID	Description
StateChange		Replication task assessment run has finished with error.
StateChange		Replication task assessment run %s has been cancelled.
StateChange		Replication task assessment run %s has been deleted.
StateChange		Replication task assessment run has failed to provision resources.
StateChange		Replication task has failed.
Creation		The replication task has been created.
ConfigurationChange		A replication task has been modified.
Failure		A replication task has failed.
StateChange	DMS-EVENT-0091	Reading paused, swap files limit reached.
StateChange	DMS-EVENT-0092	Reading paused, disk usage limit reached.
StateChange	DMS-EVENT-0093	Reading resumed.
StateChange	DMS-EVENT-0069	The replication task has started with taskType: %s, startType: %s
StateChange	DMS-EVENT-0079	Replication task stopped
Deletion	DMS-EVENT-0073	The replication task has been deleted.

Replication event messages

The following table shows the possible categories and events for the **Replication** source type.

Category	Description
State Change	DMS replication scaling up event.
State Change	DMS replication scaling down event.
State Change	DMS replication scaling event completed.
State Change	DMS replication has been created.
State Change	DMS replication is initializing.
State Change	DMS replication is preparing the resources for metadata collection.
State Change	The connections tied to DMS replication is being tested.
State Change	DMS replication is fetching metadata
State Change	DMS replication is calculating capacity
State Change	DMS replication is provisioning its capacity
State Change	DMS replication has been provisioned.
State Change	DMS replication has started
State Change	DMS replication is running.
State Change	DMS replication is being stopped.
State Change	DMS replication has stopped
State Change	DMS replication is being modified.
State Change	DMS replication is being deleted.
State Change	DMS replication is deprovisioning its capacity

Category	Description
State Change	DMS replication has been deprovisioned.
Failure	DMS replication has failed.

Working with Amazon SNS events and notifications in AWS Database Migration Service

Beginning with the release of AWS DMS 3.4.5 and with later versions, we recommend that you use Amazon EventBridge to provide notifications when an AWS DMS event occurs. For more information about using EventBridge events with AWS DMS, see Working with Amazon EventBridge events and notifications in AWS Database Migration Service.

Moving event subscriptions to Amazon EventBridge

You can use the following AWS CLI command to migrate active event subscriptions from DMS to Amazon EventBridge, up to 10 at a time.

```
update-subscriptions-to-event-bridge [--force-move | --no-force-move]
```

By default, AWS DMS only migrates active event subscriptions when your replication instance is current with AWS DMS 3.4.5 and higher. To override this default behavior, use the `--force-move` option. However, some types of events might not be available by using Amazon EventBridge if your replication instances aren't upgraded.

To run the `update-subscriptions-to-event-bridge` CLI command, an AWS Identity and Access Management (IAM) user must have the following policy permissions.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
              "SNS:GetTopicAttributes",
              "SNS:SetTopicAttributes",
              "events:PutTargets",
              "events:EnableRule",
              "events:PutRule"
          ],
          "Resource": "*"
```

```
        }
    ]
}
```

For more information about moving subscriptions to EventBridge, see UpdateSubscriptionsToEventBridge in the *AWS Database Migration Service API Reference*.

Working with Amazon SNS events and notifications

AWS DMS versions 3.4.5 and earlier support working with events and notifications as described following.

AWS Database Migration Service (AWS DMS) can use 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 for SNS notifications, 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 event notifications differ from CloudTrail events in CloudWatch or EventBridge. CloudTrail event notifications can be generated by any API invocation. DMS sends a notification only when a DMS event occurs.

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 with Amazon SNS, 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 using SNS.

AWS DMS event categories and event messages for SNS notifications

> **Important**
>
> Beginning with the release of AWS DMS 3.4.5 and with later versions, we recommend that you use Amazon EventBridge to provide notifications when an AWS DMS event occurs. For more information about using EventBridge events with AWS DMS, see Working with Amazon EventBridge events and notifications in AWS Database Migration Service.

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 finished transitioning to a Multi-AZ configuration.

Category	DMS event ID	Description
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 is created.
Deletion	DMS-EVENT-0066	The replication instance is being deleted.
Deletion	DMS-EVENT-0003	The replication instance is 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	A replication instance is in a state that can't be upgraded.
LowStorage	DMS-EVENT-0007	The replication instance has consumed more than 90% of its allocated storage. You can monitor the storage space for a replication instance using the Free Storage Space metric.
Failover	DMS-EVENT-0013	Failover started for a Multi-AZ replication instance.
Failover	DMS-EVENT-0049	Failover is complete for a Multi-AZ replication instance.

Category	DMS event ID	Description
Failover	DMS-EVENT-0015	Multi-AZ failover to standby is complete.
Failover	DMS-EVENT-0050	Multi-AZ activation has started.
Failover	DMS-EVENT-0051	Multi-AZ activation had 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	The service can't access the AWS 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-0081	A reload of table details has been requested.
State Change	DMS-EVENT-0079	The replication task has stopped.
State Change	DMS-EVENT-0091	Reading paused, swap files limit reached.
State Change	DMS-EVENT-0092	Reading paused, disk usage limit reached.
State Change	DMS-EVENT-0093	Reading resumed.
Failure	DMS-EVENT-0078	The replication task has failed.
Failure	DMS-EVENT-0082	A call to delete the task has failed to clean up task data.

Category	DMS event ID	Description
Configuration Change	DMS-EVENT-0080	The replication task is modified.
Deletion	DMS-EVENT-0073	The replication task is deleted.
Creation	DMS-EVENT-0074	The replication task is created.

The following table shows the possible categories and events for the premigration assessment source type.

Category	DMS event ID	Description
State Change	DMS-EVENT-0083	The premigration assessment has started.
State Change	DMS-EVENT-0084	The premigration assessment has passed.
State Change	DMS-EVENT-0085	The premigration assessment has completed with failures.
State Change	DMS-EVENT-0086	The premigration assessment has completed with warnings.
State Change	DMS-EVENT-0087	The premigration assessmen t encountered and error while running.
State Change	DMS-EVENT-0088	The premigration assessment has been cancelled.
State Change	DMS-EVENT-0089	The premigration assessment has been deleted.

Category	DMS event ID	Description
State Change	DMS-EVENT-0090	The premigration assessmen t encountered an error while provisioning resources.

The following example shows an AWS DMS event subscription with the 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 using SNS

Important

Beginning with the release of AWS DMS 3.4.5 and with later versions, we recommend that you use Amazon EventBridge to provide notifications when an AWS DMS event occurs. For more information about using EventBridge events with AWS DMS, see Working with Amazon EventBridge events and notifications in AWS Database Migration Service.

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 the AWS Management Console

> **Important**
>
> Beginning with the release of AWS DMS 3.4.5 and with later versions, we recommend that you use Amazon EventBridge to provide notifications when an AWS DMS event occurs. For more information about using EventBridge events with AWS DMS, see [Working with Amazon EventBridge events and notifications in AWS Database Migration Service](#).

To subscribe to AWS DMS event notification with Amazon SNS by 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're signed in as an IAM user, make sure that you 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:

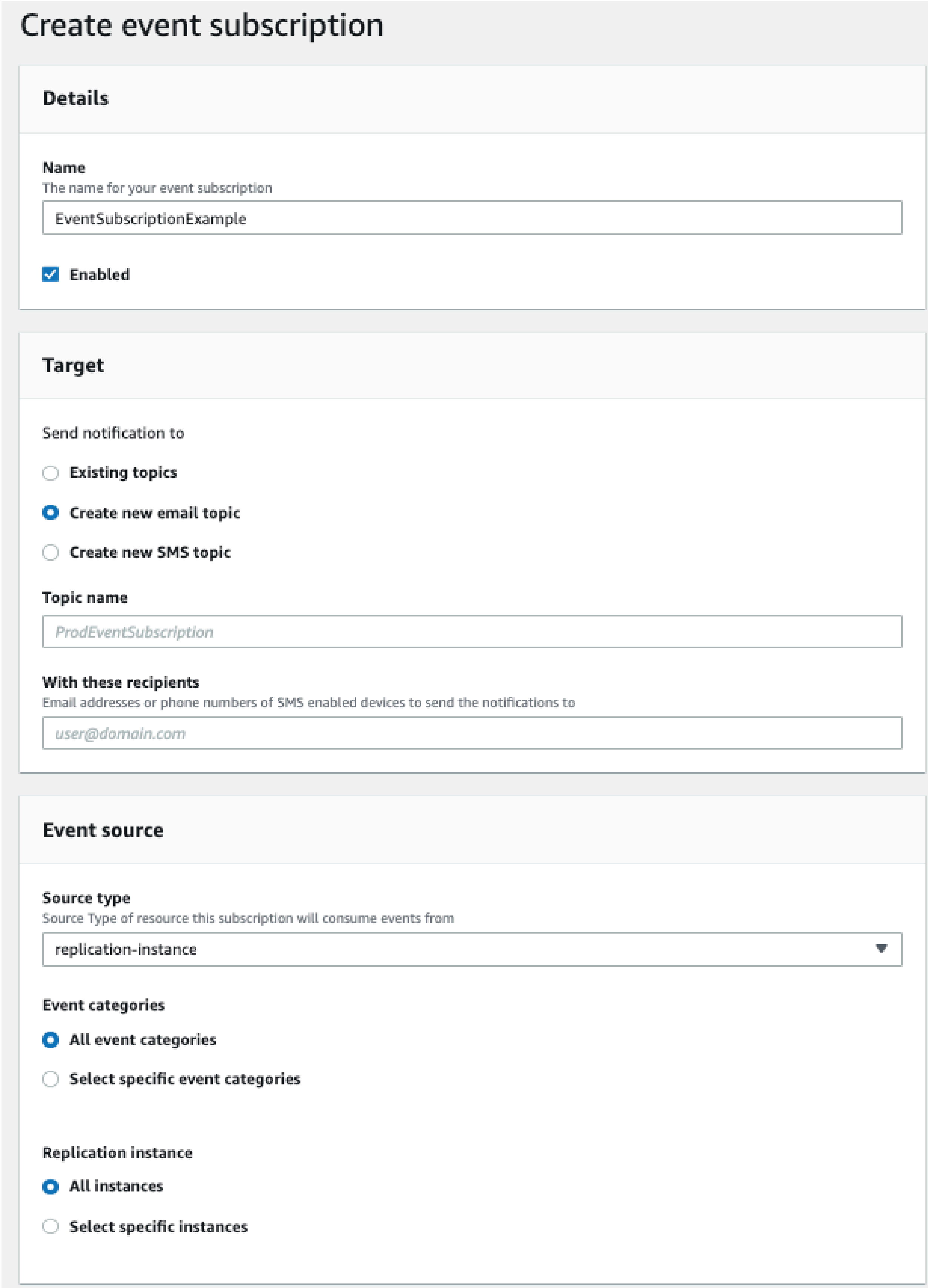
   a. Under **Details**, for **Name**, enter a name for the event notification subscription.

   b. Choose **Enabled** to enable the subscription. If you want to create the subscription but not have notifications sent yet, don't choose **Enabled**.

   c. Under **Target**, choose either **Existing topics**, **Create new email topic** or **Create new SMS topic** to send notifications. Make sure that you either have an existing Amazon SNS topic to send notices to or create the topic. If you 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
Event categories
All event categories
Select specific event categories
Replication instance
All instances
Select specific instances

f. Select **Create event subscription**.

The AWS DMS console indicates that the subscription is being created.

> **Note**
>
> You can also create Amazon SNS event notification subscriptions using the AWS DMS API and CLI. For more information, see the [CreateEventSubscription](#) in the *AWS DMS API Reference* and [create-event-subscription](#) in the *AWS DMS CLI Reference* documentation.

Validating the access policy of your SNS topic

Your SNS access policy requires permissions that allow AWS DMS to publish events to your SNS topic. You can validate and update your access policy as described in the following procedures.

To validate your access policy

1. Open the **Amazon SNS** console.
2. From the navigation panel, choose **Topics** and select the topic that you want to receive DMS notifications about.
3. Select the **Access policy** tab.

You can update your policy if your SNS access policy doesn't allow AWS DMS to publish events to your SNS topic.

To update your access policy

1. From the **Details** section of your topic page, choose **Edit**.
2. Expand the **Access policy** section, and attach the following policy into the JSON editor.

```
{
      "Sid": "dms-allow-publish",
      "Effect": "Allow",
      "Principal": {
        "Service": "dms.amazonaws.com"
      },
      "Action": "sns:Publish",
```

```
        "Resource": "your-SNS-topic-ARN"
      }
```

We recommend that you further restrict the access to your SNS topic by specifying the `aws:SourceArn` condition, which is the DMS EventSubscription Arn that publishes events to the topic.

```
...
"Resource": "your-SNS-topic-ARN"
"Condition": {
    "StringEquals": {
       "aws:SourceArn": "arn:partition:dms:your-AWS-region:your-AWS-account-
ID:es:your-dms-es-arn or *"
 }
```

3. Choose **Save changes**.

AWS DMS data validation

Topics

AWS DMS provides support for data validation to ensure that your data was migrated accurately from the source to the target. If enabled, validation begins immediately after a full load is performed for a table. Validation compares the incremental changes for a CDC-enabled task as they occur.

During data validation, AWS DMS compares each row in the source with its corresponding row at the target, verifies the rows contain the same data, and reports any mismatches. To accomplish this AWS DMS issues appropriate queries to retrieve the data. Note that these queries will consume additional resources at the source and target as well as additional network resources.

For a CDC only task with validation enabled, all pre-existing data in a table is validated before starting validation of new data.

Data validation works with the following source databases wherever AWS DMS supports them as source endpoints:

- Oracle
- PostgreSQL-compatible database (PostgreSQL, Aurora PostgreSQL, or Aurora Serverless for PostgreSQL)

- MySQL-compatible database (MySQL, MariaDB, Aurora MySQL, or Aurora Serverless for MySQL)
- Microsoft SQL Server
- IBM Db2 LUW

Data validation works with the following target databases wherever AWS DMS supports them as target endpoints:

- Oracle
- PostgreSQL-compatible database (PostgreSQL, Aurora PostgreSQL, or Aurora Serverless for PostgreSQL)
- MySQL-compatible database (MySQL, MariaDB, Aurora MySQL, or Aurora Serverless for MySQL)
- Microsoft SQL Server
- IBM Db2 LUW
- Amazon Redshift
- Amazon S3. For information about validating Amazon S3 target data, see Amazon S3 target data validation.

For more information about the supported endpoints, see Working with AWS DMS endpoints.

Data validation requires additional time, beyond the amount required for the migration itself. The extra time required depends on how much data was migrated.

For more information about these settings, see Data validation task settings.

For an example of `ValidationSettings` task settings in a JSON file, see Task settings example.

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_control.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.
  - **Pending validation**—The table is waiting validation.
  - **Preparing table**—Preparing the table enabled in the migration task for validation.
  - **Pending revalidation**—All rows in the table are pending validation after the table was updated.
- **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.
- **ValidationFailed**—The number of records that didn't pass the data validation phase.

For an example of `ValidationSettings` task settings in a JSON file, see [Task settings example](#).

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 open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.

   If you're signed in as an AWS Identity and Access Management (IAM) user, make sure that you have the appropriate permissions to access AWS DMS. the permissions required, see IAM permissions needed to use AWS DMS.
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**.

Using JSON editor to modify validation rules

To add a validation rule to a task using the JSON editor from the AWS DMS Console, do the following:

1. Select **Database migration tasks**.
2. Select your task from the list of migration tasks.
3. If your task is running, select **Stop** from the **Actions** drop down menu.
4. Once the task has stopped, to modify your task, select **Modify** from the **Actions** drop down menu.
5. In the **Table mappings** section, select **JSON editor** and add your validation rule to your table mappings.

For example, you can add the following validation rule to run a replace function on the source. In this case, if the validation rule encounters a null byte, it validates it as a space.

```
{
 "rule-type": "validation",
 "rule-id": "1",
 "rule-name": "1",
 "rule-target": "column",
 "object-locator": {
  "schema-name": "Test-Schema",
  "table-name": "Test-Table",
  "column-name": "Test-Column"
 },
 "rule-action": "override-validation-function",
 "source-function": "REPLACE(${column-name}, chr(0), chr(32))",
 "target-function": "${column-name}"
}
```

Note

`override-validation-function` does not take effect if the column is a part of the primary key.

Validation only tasks

You can create validation only tasks to preview and validate data without performing any migration or data replication. To create a validation only task, set the `EnableValidation` and `ValidationOnly` settings to `true`. When enabling `ValidationOnly`, additional requirements apply. For more information, see Data validation task settings.

For a full load only migration type, a validation only task completes much faster than its CDC equivalent when many failures are reported. But changes to the source or target endpoint are reported as failures for full load mode, a possible disadvantage.

A CDC validation only task delays validation based on average latency, and retries failures multiple times before reporting them. If the majority of data comparisons result in failures, a validation only task for CDC mode is very slow, a potential drawback.

A validation only task must be set up in the same direction as the replication task, especially for CDC. This is because a CDC Validation Only task detects which rows have changed and need to be revalidated based on the change log on the source. If the target is specified as the source, then it only knows about changes sent to the target by DMS and is not guaranteed to catch replication errors.

Full load validation only

Beginning with AWS DMS version 3.4.6 and higher, a full load validation only task quickly compares all rows from the source and target tables in a single pass, immediately reports any failures, and then shuts down. Validation never is suspended due to failures in this mode, it is optimized for speed. But changes to the source or target endpoint are reported as failures.

> **Note**
>
> Beginning with AWS DMS version 3.4.6 and higher, this validation behavior also applies to full load migration task with validation enabled.

CDC validation only

A CDC validation only task validates all existing rows between the source and target tables on a fresh start. In addition, a CDC validation only task runs continuously, re-validates ongoing replication changes, limits the number of failures reported each pass, and retries mismatched rows before failing them. It is optimized to prevent false positives.

Validation for a table (or the entire task) is suspended if the `FailureMaxCount` or `TableFailureMaxCount` thresholds are breached. This also applies for a CDC or Full Load+CDC migration task with validation enabled. And a CDC task with validation enabled delays re-validation for each changed row based on average source and target latency.

But a CDC *validation only task* doesn't migrate data and has no latency. It sets `ValidationQueryCdcDelaySeconds` to 180 by default. And you can increase the amount to account for high latency environments and help prevent false positives.

Validation only use cases

Use cases for splitting the data validation portion of a migration or replication task into a separate *validation only task* includes, but is not limited to, the following:

- *Control exactly when validation occurs* — Validation queries add an additional load to both source and target endpoints. So, migrating or replicating data in one task first, then validating the results in another task can be beneficial.

- *Reduce load on the replication instance* — Splitting data validation to run on its own instance can be advantageous.
- *Quickly obtain how many rows don't match at a given moment in time* — For example, just before or during a maintenance window production cut–over to a target endpoint, you can create a Full Load validation only task to get an answer to your question.
- *When validation failures are expected for a migration task with a CDC component* — For example, if migrating Oracle `varchar2` to PostgreSQL `jsonb`, CDC validation keeps retrying these failed rows and limits the number of failures reported each time. But, you can create a Full Load validation only task and obtain a quicker answer.
- *You've developed a data repair script/utility that reads the validation failure table* — (See also, Troubleshooting). A Full Load validation only task quickly reports failures for the data repair script to act upon.

For an example of `ValidationSettings` task settings in a JSON file, see Task settings example).

Troubleshooting

During validation, AWS DMS creates a new table at the target endpoint: `awsdms_control.awsdms_validation_failures_v1`. If any record enters the *ValidationSuspended* or the *ValidationFailed* state, AWS DMS writes diagnostic information to `awsdms_control.awsdms_validation_failures_v1`. You can query this table to help troubleshoot validation errors.

For information about changing the default schema the table is created in on the target, see Control table task settings.

Following is a description of the `awsdms_control.awsdms_validation_failures_v1` table:

Column name	Data type	Description
TASK_NAME	VARCHAR(128) NOT NULL	AWS DMS task identifier.
TABLE_OWNER	VARCHAR(128) NOT NULL	Schema (owner) of the table.

Column name	Data type	Description
TABLE_NAME	VARCHAR(128) NOT NULL	Table name.
FAILURE_TIME	DATETIME(3) NOT NULL	Time when the failure occurred.
KEY_TYPE	VARCHAR(128) NOT NULL	Reserved for future use (value is always 'Row')
KEY	TEXT NOT NULL	This is the primary key for row record type.
FAILURE_TYPE	VARCHAR(128) NOT NULL	Severity of validation error. Can be either RECORD_DIFF , MISSING_SOURCE , MISSING_TARGET , or TABLE_WARNING .
DETAILS	VARCHAR(8000) NOT NULL	JSON formatted string of all source/target column values which do not match for the given key.

The following is a sample query for a MySQL target that will show you all the failures for a task by querying the `awsdms_control.awsdms_validation_failures_v1` table. Note that the schema name and query syntax will vary across target engine versions. 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.

awsdms_validation_failures_v2 control table

During validation, in AWS DMS version 3.6.1 and above, DMS creates a new table at the PostgreSQL target endpoint: awsdms_validation_failures_v2. This tables consists of failures for all the DMS tasks that have data validation enabled. When the awsdms_validation_failures_v2 table is created, you should not drop or truncate the table as it can cause errors for any tasks with validation and resync enabled. awsdms_validation_failures_v2 table has an auto-increment primary key feature. This table consists of new columns to support the Data resync feature. They are:

RESYNC_RESULT

Values: SUCCESS or FAILURE.

RESYNC_TIME

Timestamp with millisecond precision. Default value is NULL if Data resync is not attempted for this failure.

RESYNC_ACTION

Values: UPSERT or DELETE.

RESYNC_ID

Primary key column with auto-increment enabled.

In the awsdms_validation_failures_v2 table, an index is added to the TASK_NAME, TABLE_OWNER, TABLE_NAME, FAILURE_TYPE, and FAILURE_TIME columns to efficiently read failures for any given table in your target database. Below is an example create statement to create an awsdms_validation_failures_v2 table:

```
CREATE TABLE public.awsdms_validation_failures_v2 (
```

```
    "RESYNC_ID" int8 GENERATED BY DEFAULT AS IDENTITY( INCREMENT BY 1 MINVALUE 1
 MAXVALUE 9223372036854775807 START 1 CACHE 1 NO CYCLE) NOT NULL,
    "TASK_NAME" varchar(128) NOT NULL,
    "TABLE_OWNER" varchar(128) NOT NULL,
    "TABLE_NAME" varchar(128) NOT NULL,
    "FAILURE_TIME" timestamp NOT NULL,
    "KEY_TYPE" varchar(128) NOT NULL,
    "KEY" varchar(7800) NOT NULL,
    "FAILURE_TYPE" varchar(128) NOT NULL,
    "DETAILS" varchar(7000) NOT NULL,
    "RESYNC_RESULT" varchar(128) NULL,
    "RESYNC_TIME" timestamp NULL,
    "RESYNC_ACTION" varchar(128) NULL,
    CONSTRAINT awsdms_validation_failures_v2_pkey PRIMARY KEY ("RESYNC_ID")
);
```

Redshift Validation Performance

Amazon Redshift differs from relational databases in several ways, including columnar storage, MPP, data compression, and other factors. These differences give Redshift a different performance profile from relational databases.

During the full-load replication phase, validation uses range queries, with the data size governed by the `PartitionSize` setting. These range-based queries select all the records from the source table.

For ongoing replication, queries switch between range-based and individual-record fetches. The query type is determined dynamically based on multiple factors, such as the following:

- Query volume
- Types of DML queries on the source table
- Task latency
- Total number of records
- Validation settings such as `PartitionSize`

You may see additional load on your Amazon Redshift cluster due to validation queries. As the above factors vary across use cases, you must review your validation query performance and tune your cluster and table accordingly. Some options to mitigate performance isses include the following:

- Reduce the `PartitionSize` and `ThreadCount` settings to help reduce the workload during full-load validation. Note that this will slow down data validation.
- While Redshift doesn't enforce primary keys, AWS DMS relies on primary keys to uniquely identify records on the target for data validation. If possible, set the primary key to mirror the sort key so that full load validation queries execute more quickly.

Enhanced data validation for AWS Database Migration Service

AWS Database Migration Service has enhanced data validation performance for database migrations, enabling customers to validate large datasets with significantly faster processing times. This enhanced data validation is now available in version 3.5.4 of the replication engine for both full load and full load with CDC migration tasks. Currently, this enhancement supports migration paths from Oracle to PostgreSQL, SQL Server to PostgreSQL, Oracle to Oracle, and SQL Server to SQL Server, with additional migration paths planned for future releases.

Prerequisites

- *Oracle:* grant the `EXECUTE` permission on `SYS.DBMS_CRYPTO` to the user account that accesses the Oracle endpoint:

  ```
  GRANT EXECUTE ON SYS.DBMS_CRYPTO TO dms_endpoint_user;
  ```

- Install the `pgcrypto` extension on the PostgreSQL database:

  For self-managed PostgreSQL instances, you need to install the `contrib` module libraries and create the extension:

  - Install the `contrib` module libraries. For example, on an Amazon EC2 instance with Amazon Linux and PostgreSQL 15:

    ```
    sudo dnf install postgresql15-contrib
    ```

  - Create the `pgcrypto` extension:

    ```
    CREATE EXTENSION IF NOT EXISTS pgcrypto;
    ```

- For Amazon RDS for PostgreSQL instances, configure the SSL mode for the AWS DMS endpoint:
  - By default, Amazon RDS forces an SSL connection. When you create a AWS DMS endpoint for a Amazon RDS for PostgreSQL instance, use the "SSL mode" option = "required".

  - If you want to use the "SSL mode" option = "none", set the `rds.force_ssl` parameter to 0 in the RDS Parameter Group.
- For PostgreSQL 12 and 13, create the `BIT_XOR` aggregate:

```
CREATE OR REPLACE AGGREGATE BIT_XOR(IN v bit) (SFUNC = bitxor, STYPE = bit);
```

Enhanced data validation limitations

This enhanced data validation feature has the following limitations:

- Database endpoint requirements: This improvement is enabled only for database endpoints that meet the following criteria:
  - Use AWS Secrets Manager to store credentials.
  - For Microsoft SQL Server, Kerberos authentication is also supported.
- Database version support:
  - PostgreSQL 12 and higher
  - Oracle 12.1 and higher
  - For Microsoft SQL Server versions lower than 2019, validation of NCHAR and NVARCHAR data types is not supported.

Limitations

- Data validation requires that the table has a primary key or unique index.
  - Primary key columns cannot be of type `CLOB`, `BLOB`, `BINARY`, or `BYTE`.
  - For primary key columns of type `VARCHAR` or `CHAR`, the length must be less than 1024. You must specify the length in the datatype. You can't use unbounded data types as a primary key for data validation.
  - An Oracle key created with the `NOVALIDATE` clause is *not* considered a primary key or unique index.
  - For an Oracle table with no primary key and only a unique key, the columns with the unique constraint must also have a `NOT NULL` constraint.
- Validation of NULL PK/UK values aren't supported.

- 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. This is especially true for Redshift targets. For more information, see Redshift Validation Performance following.
- Data validation isn't supported when consolidating several databases into one.
- For a source or target Oracle endpoint, AWS DMS uses `DBMS_CRYPTO`. If you use data validation on Oracle endpoint 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 validation, then AWS DMS can't validate those rows.
- 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.
- AWS DMS doesn't support data validation of views.
- AWS DMS doesn't support data validation when character substitution task settings are used.
- AWS DMS doesn't support validating the Oracle LONG type.
- AWS DMS doesn't support validating the Oracle Spatial type during heterogeneous migration.
- Data validation ignores those columns in tables for which data masking transformations exists in table mapping.
- Data validation skips an entire table if there is a data masking transformation rule for its PK/UK column. The validation state will show as No primary key for such tables.
- Data validation does not work with Amazon Aurora PostgreSQL Limitless. When attempting to validate tables in a Limitless Database, the validation state displays "No primary key" for these tables.

For limitations when using S3 target validation, see Limitations for using S3 target validation.

Amazon S3 target data validation

AWS DMS supports validating replicated data in Amazon S3 targets. Because AWS DMS stores replicated data as flat files in Amazon S3, we use Amazon Athena `CREATE TABLE AS SELECT` (CTAS) queries to validate data.

Queries on data that is stored in Amazon S3 are computationally intense. Thus, AWS DMS runs validation on Amazon S3 data during change data capture (CDC) only once a day, at midnight (00:00) UTC. Each daily validation that AWS DMS runs is called an *interval validation*. During an interval validation, AWS DMS validates all of the change records that were migrated to the target Amazon S3 bucket for the previous 24 hours. For more information about limitations for interval validation, see Limitations for using S3 target validation.

Amazon S3 target validation uses Amazon Athena, so additional costs apply. For more information, see Amazon Athena Pricing.

> **Note**
>
> S3 target validation requires AWS DMS version 3.5.0 or later.

Topics

- S3 target validation prerequisites
- Permissions for using S3 target validation
- Limitations for using S3 target validation
- Using validation only tasks with S3 target validation

S3 target validation prerequisites

Before using S3 target validation, check the following settings and permissions:

- Set the `DataFormat` value for your endpoint's S3Settings to `parquet`. For more information, see Parquet settings for S3.
- Ensure that the role assigned to the user account that was used to create the migration task has the correct set of permissions. See Permissions following.

For tasks using ongoing replication (CDC), check the following settings:

- Turn on supplemental logging so you have complete records in the CDC data. For information about turning on supplemental logging, see Automatically add supplemental logging to an Oracle source endpoint in the Troubleshooting and diagnostic support section in this guide.
- Set the `TimestampColumnName` parameter for the target endpoint. There are no limitations on the timestamp column name. For more information, see S3Settings.
- Set up date-based folder partitioning for the target. For more information, see Using date-based folder partitioning.

Permissions for using S3 target validation

To set up access for using S3 target validation, ensure that the role assigned to the user account that was used to create the migration task has the following set of permissions. Replace the sample values with your values.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "athena:StartQueryExecution",
                "athena:GetQueryExecution",
                "athena:CreateWorkGroup"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:DeleteDatabase",
                "glue:GetDatabase",
                "glue:GetTables",
                "glue:CreateTable",
                "glue:DeleteTable",
                "glue:GetTable"
            ],
```

```
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucketMultipartUploads",
                "s3:AbortMultipartUpload",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": "*"
        }
    ]
}
```

Limitations for using S3 target validation

View the following additional limitations that apply when using S3 target validation. For limitations that apply to all validations, see Limitations.

- Your `DatePartitionSequence` value needs a Day component. S3 target validation does not support the `YYYYMM` format.
- When interval validation is running during CDC, you may see false validation errors in the `awsdms_validation_failures_v1` table. These errors occur because AWS DMS migrates changes that arrived during the interval validation into the next day's partition folder. Normally, these changes are written into the current day's partition folder. These false errors are a limitation of validating replication from a dynamic source database to a static target, such as Amazon S3. To investigate these false errors, check for records near the end of the validation window (00:00 UTC), which is when these errors typically appear.

  To minimize the number of false errors, ensure that the `CDCLatencySource` for the task is low. For information about monitoring latency, see Replication task metrics.
- Tasks in the `failed` or `stopped` state don't validate the previous day's changes. To minimize validation errors because of unexpected failures, create separate validation only tasks with the same table mappings, and source and target endpoints. For more information about validation only tasks, see Using validation only tasks with S3 target validation.

- The **Validation Status** column in table statistics reflects the state of the most recent interval validation. As a result, a table which has mismatches might show up as validated after the next day's interval validation. Check the `s3_validation_failures folder` in the target Amazon S3 bucket for mismatches that occurred more than a day ago.
- S3 Validation uses the bucketed table feature of Amazon Athena. This allows S3 validation to make a bucketed copy of the target table data. This means that the copy of the table data is divided into subsets that match DMS validation's internal partitioning. Athena bucketed tables have a limit of 100,000 buckets. Any tables that S3 validation attempts to validate that exceed this limit will fail validation. The number of buckets that S3 Validation attempts to create is equal to the following:

  ```
  (#records in the table) / (validation partition size setting)
  ```

  To work around this limitation, increase the validation partition size setting so that the number of buckets created by S3 Validation is less than 100,000. For more information about bucketing, see [Partitioning and bucketing in Athena](#) in the *Amazon Athena User Guide*.
- The table name must not contain special characters except underscore.

  S3 Validation uses Amazon Athena which does not support special characters (other than underscore) in table names. For more information, see CREATE TABLE topic in the *Amazon Athena User Guide*.
- When AWS DMS data validation feature is used with an Amazon S3 target managed by AWS Lake Formation, the validation process fails. This can result in data consistency issues.

Using validation only tasks with S3 target validation

A *validation only task* runs validation on data that is to be migrated without running the migration.

Validation only tasks continue to run, even if the migration task stops, which ensures that AWS DMS doesn't miss the 00:00 UTC interval validation window.

Using validation only tasks with Amazon S3 target endpoints has the following limitations:

- Amazon S3 Validation for Full-Load Tasks with the Validation only setting enabled are supported, but operate differently than Full-Load and Validation only tasks for other endpoints. For S3 as a Target, a task of this type validates against only the Full-Load Data in the S3 target, and will not validate against any data migrated as part of a CDC migration. Only use this feature to validate data created by a Full-Load only task. Using this mode to validate data in a target that has an active CDC task running will not produce an effective validation.

- Validation only tasks only validate changes since the last interval validation window (00:00 UTC). Validation only tasks don't validate full-load data or CDC data from previous days.

AWS DMS data resync

AWS Database Migration Service (AWS DMS) Data resync automatically fixes data inconsistencies identified through data validation between your source and target databases. This feature works as part of your existing DMS migration tasks, ensuring proper updates occur based on your task configurations, connection settings, table mappings, and transformations.

Data resync feature operates by reading validation failures from a control table on the target database and executing appropriate fix-up operations. When a mismatch is detected, the current data is retrived from the source using the primary key stored in the failure record, and it is applied to the target while respecting any configured transformations. For more information, see the section called "`awsdms_validation_failures_v2` control table".

The behavior varies depending on your migration type. For full-load-only tasks, Data resync runs once after the initial load and validation complete. For tasks with change data capture (CDC), Data resync operates according to a configured schedule, temporarily pausing replication and validation while fixes are applied.

During CDC resync operations:

- Replication and validation pause temporarily.
- Data resync processes existing validation failures.
- Normal replication and validation resume.
- The process repeats based on your configured schedule.

Data resync automatically tracks the status of each fix-up operation and provides detailed metrics through table statistics.

Prerequisites:

The Data resync feature needs following prerequisites:

- You must have AWS DMS engine version 3.6.1 or later.

- You must configure schedule and timing duration settings for tasks that have ongoing replication. Full load only tasks do not require these settings.

Limitations

The Data resync feature has following limitations:

- Data resync only supports Oracle and SQL Server as source database.
- Data resync supports PostgreSQL and Amazon Aurora PostgreSQL compatible engine as target database.
- All the tables in your source and target database must have primary keys. Validation does not support tables without a primary key or a unique key. Any tables that do not have a valid primary or unique key are suspended from validation and no validation failures are reported.
- When running Full-load-only tasks, data validation must be enabled.
- Data resync cannot be enabled for Validation only task as they do not replicate any data. You can enable resync on the parent replication task by providing the Validation only `taskID`. For more information, see Validation only tasks.
- If the Validation only task has a `ControlSchema` parameter setting configured in the task settings, then the replication task must also have the same parameter configuration for Data resync to find the correct validation failures.
- You are required to configure schedule and timing duration settings for CDC tasks.
- During the resync window, Data resync can have an impact on the replication latency in DMS.

For more information regarding troubleshooting validations in AWS DMS during Data resync, see the Troubleshooting section under AWS DMS data validation.

Scheduling and timing

For tasks with CDC, you must configure when and how long Data resync operates. This helps prevent impact to your normal replication operations. You specify:

- A schedule using cron format to define when resync operations can occur.
- A maximum duration to ensure resync operations don't extend into peak usage periods.

It is recommended to schedule resync operations during off-peak hours or to a period where there minimal to no changes on the source database.

> **Note**
>
> The scheduled time includes waiting for the target apply stream to be empty, as Data resync and normal replication cannot run simultaneously.

Use cases

The data resync feature enables users to reconcile data inconsistencies between source and target systems. It identifies mismatched records and synchronizes them to maintain data consistency across distributed environments. The following use cases demonstrate common scenarios where the data resync feature resolves data consistency challenges:

Scenario 1: Full load task - run resync using the same DMS task

In your existing DMS full-load migration task, you can do the following:

- Enable validation: `Validation with data migration = true`.
- Enable resync: `Data resync = true`

Scenario 2: Full load and CDC, CDC only task - run resync using the same DMS task

In your existing DMS CDC migration task, you can do the following:

- Enable validation: `Validation with data migration = true`.
- Enable resync: `Data resync = true`
- Specify resync schedule: `"ResyncSchedule": "0 0,2,4,6 * * *"`.
- Specify resync time: `MaxResyncTime": 60`

Scenario 3: Full load and CDC or CDC only task for replication and resync, in combination with a validation only task

To perform validation only operation in another DMS task when using resync, you can do the following:

- Create a validation only DMS CDC task.

> **Note**
>
> You must note down and specify the ID of this task during Data resync.

- In your primary CDC task, disable validation: `Data validation = false`.
- Enable resync: `Data resync = true`
- Specify resync schedule: `"ResyncSchedule": "0 0,2,4,6 * * *"`.
- Specify resync time: `MaxResyncTime": 60`.
- Specify the ID of the validation only DMS CDC task. Validation only task ID is appended at the end of ARN. Example ARN: `arn:aws:dms:us-west-2:123456789012:task:6DG4CLGJ5JSJR67CFD7UDXFY7KV6CYGRICL6KWI` and Example validation only task ID: `6DG4CLGJ5JSJR67CFD7UDXFY7KV6CYGRICL6KWI`.

Best practices

You can leverage the Data resync feature in AWS Database Migration Service to improve durability of your replication tasks and attain consistency. Some of the best practices to use the Data resync feature are:

- As part of the Data resync, records that have mismatches are fixed by fetching it from the source and applying it on the target database. If the source database is updated during the resync window, resync reads the latest record value and applies it on the target. This can cause CDC apply events to fail and introduce temporary inconsistencies on the target database . To avoid this, you must schedule the resync window during off-business hours or periods where the changes on the source database are zero or minimal.
- Set the resync window during periods of minimal source database activity and within your acceptable target latency threshold. Small resync intervals can cause unprocessed validation mismatches to accumulate, while large windows may increase replication latency when many validation failures occur. Monitor validation failure and resync rates to determine optimal resync windows during source inactivity periods. Some examples for setting up the resync windows are:
  - Multiple short window configuration:

```
"ResyncSchedule": "0 0,2,4,6 * * *",
"MaxResyncTime": 60
```

  - Single daily window configuration:

    ```
    "ResyncSchedule": "0 0 * * *",
    "MaxResyncTime": 360
    ```

- Monitoring replication latency in DMS during resync windows and adjust schedule accordingly to mitigate large spikes.
- You can review resync results through table stastics or by querying the `awsdms_validation_failures_v2` table on the target databadse. For more information, see [Monitoring replication tasks using Amazon CloudWatch](#).
- When the task is in an ongoing replication phase, avoid initiating a reload for individual tables during the resync window.
- Best practices for a CDC replication task:
  - All the tables in your database complete loading process.
  - Mismatches are identified in the on going validation process.
  - As per the resync scheduled window, the replication task pauses for a brief period.
  - Data resync fixes the issues idenfified during the validation process.
  - The replication process resumes and repeats as per the schedule.

Data resync configuration and examples

Data resync settings configuration:

You can configure resync for your replication task in DMS. Below is an example of Data resync settings configuration in your task:

```
"ResyncSettings": {
    "EnableResync": true,
    "ResyncSchedule": "0 0,2,4,6 * * *",  // Run at 12AM, 2AM, 4AM, and 6AM daily
    "MaxResyncTime": 60,                  // Run for maximum of 60 minutes, or 1
 hour
    "ValidationTaskId": "TASK-ID-IF-NEEDED" //Optional, used only if validation is
 performed as a separate Validation only task
}
```

Examples of common resync scheduling patterns:

- `0 0 * * *`: Run once everyday at midnight.
- `0 0,12 * * *`: Run twice every day at midnight and noon.
- `0 0,2,4,6, * * *`: Run every two hours between midnight and 6 am.
- `0 1 * * 1`: Run every week on Mondays at 1 am.

> **Note**
>
> You must specifiy a number for each day starting 0 to 6. For more information, see Cron expressions rules.

Monitoring resync operations:

You can monitor the resync operation through table statistics. Here is an example output:

```
{
    "TableStatistics": {
        ...
        "ValidationFailedRecords": 1000,
        ...
        "ResyncRowsAttempted": 1000,
        "ResyncRowsSucceeded": 995,
        "ResyncRowsFailed": 5,
        "ResyncProgress": 99.5, // ratio of ResyncRowsSucceeded/
ValidationFailedRecords
        "ResyncState": "Last resync at: 2024-03-14T06:00:00Z"
    }
}
```

To configure the Data resync feature in AWS DMS, you can review various resync parameters and their respective configuration settings. For more information, see the section called "Data resync settings". For more information regarding data resync logging settings, see the section called "Logging task settings".

Validation and troubleshooting

Validation:

When data valudation is enabled, AWS DMS creates a validation failures table in your target database with the following structure:

```
CREATE TABLE awsdms_validation_failures_v2 (
    "RESYNC_ID" bigint NOT NULL,
    "TASK_NAME" varchar(128) NOT NULL,
    "TABLE_OWNER" varchar(128) NOT NULL,
    "TABLE_NAME" varchar(128) NOT NULL,
    "FAILURE_TIME" timestamp NOT NULL,
    "KEY_TYPE" varchar(128) NOT NULL,
    "KEY" varchar(7800) NOT NULL,
    "FAILURE_TYPE" varchar(128) NOT NULL,
    "DETAILS" varchar(7000) NOT NULL,
    "RESYNC_RESULT" varchar(128) NULL,
    "RESYNC_TIME" timestamp NULL,
    "RESYNC_ACTION" varchar(128) NULL
);
```

You can write a query to this table to understand the data mismatches that are found and how are they resolved.

When validation is enabled, AWS DMS creates a validation failures table in your target database. If you have any issues you can query the `awsdms_control.awsdms_validation_failures_v2` table to understand the data mismatches that are found and how are they resolved. For more information, see [Troubleshooting] section in [AWS DMS Data validation].

Common workflow:

During validation in data resync the standard workflow is as follows:

Full Load Only tasks:

1. All the tables in your database complete loading process.
2. Mismatches are identified in the on going validation process.
3. Data resync fixes the issues idenfified during the validation process.
4. Validation process validates the rectification.

5. Migration task is completed successfully.

CDC tasks:

1. All the tables in your database complete loading process.
2. Mismatches are identified in the on going validation process.
3. As per the resync scheduled window, the replication task pauses for a brief period.
4. Data resync fixes the issues idenfified during the validation process.
5. The replication process resumes and repeats as per the schedule.

Any modification done to the task such as stoping the replication task during the resync operation or reloading and revalidating tables can impact the task's behavior and outcome. Some of the known behavior changes are as follows:

When you stop the replication task while resync operation is in progress:

- The resync operation does not automatically resume. You must restart it again.
- Future resync operations occur as per the configured schedule.
- Any incomplete fixes are attempted in the next resync schedule window.

When you reload a table in your database:

- The resync operation skips any table undergoing reload.
- The previous validation failures for a table that was reloaded are ignored.
- New validation begins after the reload action completes.

When you revalidate a table in your database:

- All the statiscs for your resync operation are reset.
- The previous validation failures for a table that was revalidated are ignored.

Note

When upgrading or moving a task to DMS version 3.6.1 and above, any failures in the `awsdms_control.awsdms_validation_failures_v1` table are not resynced. Only failures in the `awsdms_validation_failures_v2` table are resynced. To resync failures in `awsdms_control.awsdms_validation_failures_v2` table, you must reload the task, reload one or more tables in the task, or re-validate one or more tables. For more information, see the following links:

- To reload a task, see `StartReplicationTask` API reference.
- To reload one or more tables in a task, see `reload-tables` in the *AWS CLI command reference* documentation.
- To re-validate one or more tables, see the `validate-only` option in the `reload-tables` section in the *AWS CLI command reference* documentation.

.

Cron expression rules

To configure Data resync operations during a replication task in AWS DMS you can use cron expressions rules. These rules allow you customise resync time windows and schedule them as per your business needs. You can use various parameters such as minutes, hours, days, months, and days of the week. The cron expression rules for each parameters are:

Minutes:

- Minute range from 0 to 59.
- You can use (-), `or`/`and` to specify the range. Maximum 10 items separated by a comma (`,`).
- **Examples**:
  - `2-5` equals to `2,3,5,5`.
  - `1-2,3-4,5,7-10` is a valid range.
  - `1,2,3,4,5,6,7,8,9,10` is a valid range.
  - `1,2,3,4,5,6,7,8,9,10,11` is not a valid range. The resync operation skips after the 10th range item.
- You can use (*). Example: `*` equals to `0-59`.
- You can use (/) only in combination with (-) or (*).

  Examples:

  - `2-7/2` equals to `2,4,6`.
  - `*/15` equals to `0,15,30,45`.

Hours:

Same as "**Minutes**" but the valid range is from `0` to `23`.

Days:

- Same as "**Minutes**" but the valid range is from `1` to `31`.
- Use of `L` is supported in resync configuration. It is intepretted as last day of the month. You must not use it in combination with another syntax.

Months:

Same as "**Minutes**" but the valid range is from `1` to `12`.

Days of week:

- Same as "**Minutes**" but the valid range is from `0` to `6`.
- You cannot add a string value for the name of the week.

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
- Data providers
- Data migrations
- Endpoints
- Event subscriptions
- Instance profiles
- Migration projects
- Replication instances
- Replication subnet 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 [Using Cost Allocation Tags]() in the *AWS Billing User Guide*.

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]().

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.

When working with XML using the AWS DMS API, tags use the following schema:

```
<Tagging>
  <TagSet>
   <Tag>
    <Key>Project</Key>
    <Value>Trinity</Value>
   </Tag>
   <Tag>
    <Key>User</Key>
    <Value>Jones</Value>
   </Tag>
  </TagSet>
</Tagging>
```

The following table provides a list of the allowed XML tags and their characteristics. Note that values for Key and Value are case dependent. For example, project=Trinity and PROJECT=Trinity are two distinct tags.

Tagging element	Description
TagSet	A tag set is a container for all tags assigned to an Amazon RDS resource. There can be only one tag set per resource. You work with a TagSet only through the AWS DMS API.
Tag	A tag is a user-defined key-value pair. There can be from 1 to 10 tags in a tag set.
Key	A key is the required name of the tag. The string value can be from 1 to 128 Unicode characters in length and cannot be prefixed with "dms:" or "aws:". The string might only contain only the set of Unicode

Tagging element	Description
	letters, digits, white-space, '_', '.', '/', '=', '+', '-' (Java regex: `"^([\\p{L }\\p{Z}\\p{N}_.:/=+\\-]*)$"`). Keys must be unique to a tag set. For example, you cannot have a key-pair in a tag set with the key the same but with different values, such as project/Trinity and project/Xanadu.
Value	A value is the optional value of the tag. The string value can be from 1 to 256 Unicode characters in length and cannot be prefixed with "dms:" or "aws:". The string might only contain only the set of Unicode letters, digits, white-space, '_', '.', '/', '=', '+', '-' (Java regex: `"^([\\p{L }\\p{Z}\\p{N}_.:/=+\\-]*)$"`). 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.

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.
- 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.

- 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.
- 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. 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.
- 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.

Topics

- Using Kerberos Authentication with AWS Database Migration Service

Data protection in AWS Database Migration Service

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

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 AWS KMS keys specifically to encrypt the target data for the following AWS DMS target endpoints:

- Amazon Redshift – For more information about setting `encryptionMode`, see Endpoint settings when using Amazon Redshift as a target for AWS DMS. For more information about creating a custom AWS KMS encryption key, see Creating and using AWS KMS keys to encrypt Amazon Redshift target data.
- Amazon S3 – For more information about setting `encryptionMode`, see Endpoint settings when using Amazon S3 as a target for AWS DMS. For more information about creating a custom AWS KMS encryption key, see Creating AWS KMS keys to encrypt Amazon S3 target objects.

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) or 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

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.

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.

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. For more information about the security considerations when using these network configurations, see Network security for AWS Database Migration Service.

Data protection in DMS Fleet Advisor

DMS Fleet Advisor collects and analyzes your database metadata to determine the right size of the migration target. DMS Fleet Advisor doesn't access data in your tables and doesn't transfer it. Also, DMS Fleet Advisor doesn't track database feature usage and doesn't access your usage statistics.

You control access to your databases when you create database users which DMS Fleet Advisor uses to work with your databases. You grant the required privileges to these users. To use DMS Fleet

Advisor, you grant your database users with read permissions. DMS Fleet Advisor doesn't modify your databases and doesn't require write permissions. For more information, see Creating database users for AWS DMS Fleet Advisor.

You can use data encryption in your databases. AWS DMS also encrypts connections within DMS Fleet Advisor and within its data collectors.

DMS data collector uses the Data Protection application programming interface (DPAPI) to encrypt, protect, and store information about customer's environment and database credentials. DMS Fleet Advisor stores this encrypted data in a file on the server where your DMS data collector works. DMS Fleet Advisor doesn't transfer this data from this server. For more information about DPAPI, see How to: Use Data Protection.

After you install the DMS data collector, you can view all queries that this application runs to collect metrics. You can run the DMS data collector in an offline mode and then review the collected data on your server. Also, you can review this collected data in your Amazon S3 bucket. For more information, see How does DMS data collector work?.

Opting out of using your data for service improvement in AWS Database Migration Service

You can choose to opt out of having your data used to develop and improve AWS DMS by using the AWS Organizations opt-out policy. You can choose to opt out even if AWS DMS does not currently collect any such data. For more information, see AI services opt-out policies in the *AWS Organizations User Guide*.

Presently, AWS Database Migration Service (AWS DMS) does not collect any of the data that it processes on your behalf. To develop and improve DMS and the functionalities of other AWS services, DMS may collect such data in the future. We will update this documentation page when DMS is configured to collect any data. You will have an option to opt out at any time.

> **Note**
>
> For you to use the opt-out policy, your AWS accounts must be centrally managed by AWS Organizations. If you have not created an organization for your AWS accounts, see Managing an organization with AWS Organizations in the *AWS Organizations User Guide*.

Opting out has the following effects:

- AWS DMS deletes the data that it collected and stored for service improvement purposes prior to your opt out (if any).
- After you opt out, AWS DMS no longer collect or store this data for service improvement purposes.

Cross-region inference in AWS Database Migration Service

Certain AWS Database Migration Service features use cross-region AI inference to automatically select the optimal AWS Region within your geography to process inference requests, maximizing available compute resources and model availability, and providing the best customer experience. With cross-region inference, you get:

- Access to the most advanced AI capabilities and features
- Increased throughput and resilience during high demand periods

Cross-region AI inference requests are kept within the AWS Regions that are part of the same geography as your primary AWS Region. For example, a request made from a primary AWS Region in the US is kept within the AWS Regions in the US. Your data remains stored only in your primary AWS Region. All data is transmitted encrypted across Amazon's secure network.

> **Note**
>
> Amazon CloudWatch and AWS CloudTrail logs don't specify the AWS Region in which AI inference occurs.

Cross-region inference in DMS Schema Conversion

When you use Generative AI features in DMS Schema Conversion, anonymized code fragments and related schema metadata might be sent to other AWS Regions within the same geography for AI processing. Your production data remains in your primary AWS Region and is never accessed or transmitted.

> **Important**
>
> Cross-region inference is always enabled when you use Generative AI features in DMS Schema Conversion. To keep schema conversion processing resident in your primary AWS Region, use schema conversion with disabled Generative AI features.

Generative AI features in DMS Schema Conversion are currently available in a limited number of regions. The following table describes what AWS Regions your requests may be routed to depending on your primary AWS Region.

Primary AWS Region	Inference AWS Regions
Asia Pacific (Tokyo) (ap-northeast-1)	Asia Pacific (Tokyo) (ap-northeast-1) Asia Pacific (Osaka) (ap-northeast-3)
Asia Pacific (Osaka) (ap-northeast-3)	Asia Pacific (Tokyo) (ap-northeast-1) Asia Pacific (Osaka) (ap-northeast-3)
Asia Pacific (Sydney) (ap-southeast-2)	Asia Pacific (Sydney) (ap-southeast-2) Asia Pacific (Melbourne) (ap-southeast-4)
Canada (Central) (ca-central-1)	Canada (Central) (ca-central-1) US East (N. Virginia) (us-east-1) US East (Ohio) (us-east-2) US West (Oregon) (us-west-2)
Europe (Frankfurt) (eu-central-1)	Europe (Frankfurt) (eu-central-1) Europe (Stockholm) (eu-north-1) Europe (Milan) (eu-south-1) Europe (Spain) (eu-south-2)

Primary AWS Region	Inference AWS Regions
	Europe (Ireland) (eu-west-1) Europe (Paris) (eu-west-3)
Europe (Stockholm) (eu-north-1)	Europe (Frankfurt) (eu-central-1) Europe (Stockholm) (eu-north-1) Europe (Milan) (eu-south-1) Europe (Spain) (eu-south-2) Europe (Ireland) (eu-west-1) Europe (Paris) (eu-west-3)
Europe (Ireland) (eu-west-1)	Europe (Frankfurt) (eu-central-1) Europe (Stockholm) (eu-north-1) Europe (Milan) (eu-south-1) Europe (Spain) (eu-south-2) Europe (Ireland) (eu-west-1) Europe (Paris) (eu-west-3)
Europe (London) (eu-west-2)	Europe (Frankfurt) (eu-central-1) Europe (Stockholm) (eu-north-1) Europe (Milan) (eu-south-1) Europe (Spain) (eu-south-2) Europe (Ireland) (eu-west-1) Europe (London) (eu-west-2) Europe (Paris) (eu-west-3)

Primary AWS Region	Inference AWS Regions
Europe (Paris) (eu-west-3)	Europe (Frankfurt) (eu-central-1) Europe (Stockholm) (eu-north-1) Europe (Milan) (eu-south-1) Europe (Spain) (eu-south-2) Europe (Ireland) (eu-west-1) Europe (Paris) (eu-west-3)
US East (N. Virginia) (us-east-1)	US East (N. Virginia) (us-east-1) US East (Ohio) (us-east-2) US West (Oregon) (us-west-2)
US East (Ohio) (us-east-2)	US East (N. Virginia) (us-east-1) US East (Ohio) (us-east-2) US West (Oregon) (us-west-2)
US West (Oregon) (us-west-2)	US East (N. Virginia) (us-east-1) US East (Ohio) (us-east-2) US West (Oregon) (us-west-2)

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

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting AWS Database Migration Service identity and access)
- **Service administrator** - determine user access and submit permission requests (see How AWS Database Migration Service works with IAM)
- **IAM administrator** - write policies to manage access (see AWS Database Migration Service identity-based policy examples)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose 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. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control 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 policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries –** Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs) –** Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs) –** Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies –** Advanced policies passed as a parameter when creating a temporary session for a role or federated user. 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

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

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions 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

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

AWS DMS works with the following resources:

- 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 `us-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 Using resource names to control access. 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

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. 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. To see all AWS global condition keys, see AWS global condition context keys 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 defined your own custom tags. For more information, see Using tags to control access.

To see a list of AWS DMS condition keys, see Condition Keys for AWS Database Migration Service in the *IAM User Guide*. To learn with 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.

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 and Creating AWS KMS keys to encrypt Amazon S3 target objects.

Examples

For examples of AWS DMS resource-based policies, see Resource-based policy examples for AWS KMS.

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.

For an example identity-based policy that limits access to a resource based on tags, see Accessing AWS DMS resources based on tags.

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.

For details about creating or managing AWS DMS service-linked roles, see Using 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.
  - OpenSearch as a target – For more information see Prerequisites for using Amazon OpenSearch Service as a target for AWS Database Migration Service.
  - Amazon Kinesis as a target – For more information see Prerequisites for using a Kinesis data stream as a target for AWS Database Migration Service.
  - 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 or Amazon S3 bucket settings.
  - Amazon S3 as a source or as a target – For more information, see Prerequisites when using Amazon S3 as a source for AWS DMS or Prerequisites for using Amazon S3 as a target.

  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 DMS console, 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 AWS DMS.

Choosing an IAM role in AWS DMS

If you use the AWS DMS console, 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 AWS DMS.

Identity and access management for DMS Fleet Advisor

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. DMS Fleet Advisor 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*.

DMS Fleet Advisor uses IAM roles to access Amazon Simple Storage Service. An IAM role is an entity within your AWS account that has specific permissions. For more information, see Create IAM resources.

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
- Using the AWS DMS console
- Allow users to view their own permissions
- Accessing one Amazon S3 bucket
- Accessing AWS DMS resources based on tags

Policy best practices

Identity-based policies 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 with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the AWS DMS console

The following policy 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.

JSON

```
{
    "Version":"2012-10-17",
     "Statement": [
    {
        "Effect": "Allow",
        "Action": "dms:*",
        "Resource": "arn:aws:dms:*:123456789012:*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "kms:ListAliases",
            "kms:DescribeKey"
        ],
        "Resource": "arn:aws:kms:*:123456789012:key/*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "iam:GetRole",
            "iam:CreateRole",
            "iam:AttachRolePolicy"
        ],
        "Resource": "arn:aws:iam::123456789012:role/*"
    },
    {
        "Effect": "Allow",
        "Action": "iam:PassRole",
        "Resource": "arn:aws:iam::123456789012:role/*",
        "Condition": {
            "StringEquals": {
                "iam:PassedToService": "dms.amazonaws.com"
            }
        }
    },
```

```
    {
        "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": "arn:aws:logs:*:123456789012:*"
    }
   ]
}
```

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": "arn:aws:service:region:account:resourcetype/id"
        }
```

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 AWS DMS](#).

```
{
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole",
                "iam:CreateRole",
                "iam:AttachRolePolicy"
            ],
            "Resource": "arn:aws:service:region:account:resourcetype/id"
        }
```

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": "arn:aws:service:region:account:resourcetype/id"
        }
```

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": "arn:aws:service:region:account:resourcetype/id"
        }
```

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

```
{
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:FilterLogEvents",
                "logs:GetLogEvents"
            ],
            "Resource": "arn:aws:service:region:account:resourcetype/id"
        }
```

If you use the AWS DMS console, the AWS Command Line Interface (AWS CLI) or the AWS DMS API for your migration, you need to add several roles to your account. For more information about adding these roles, see Creating the IAM roles to use with AWS DMS.

For more information on the requirements for using this policy to access AWS DMS, see IAM permissions needed to use AWS DMS.

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 AWS-managed `AmazonDMSRedshiftS3Role` policy.

The following example 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.

JSON

```
{
    "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.

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.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dms:*",
      "Resource": "*",
      "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] and [Creating AWS KMS keys to encrypt Amazon S3 target objects].

Topics

A policy for a custom AWS KMS encryption key to encrypt Amazon Redshift target data

The following example shows the JSON for the key policy created for an AWS KMS encryption key that you create to encrypt Amazon Redshift target data.

JSON


```
{
  "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"
      ],
      "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 AWS KMS key, 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 following example shows the JSON for the key policy created for an AWS KMS encryption key that you create to encrypt Amazon S3 target data.

JSON

```
{
    "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"
            ],
            "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 AWS KMS key, such as the key that you just created. In addition, such a user can manage attachment to AWS resources, such as the target endpoint.

Using secrets to access AWS Database Migration Service endpoints

For AWS DMS, a *secret* is an encrypted key that you can use to represent a set of user credentials to authenticate, through *secret authentication*, the database connection for a supported AWS DMS source or target endpoint. For an Oracle endpoint that also uses Oracle Automatic Storage

Management (ASM), AWS DMS requires an additional secret that represents the user credentials to access Oracle ASM.

You can create the secret or secrets that AWS DMS requires for secret authentication using AWS Secrets Manager, a service for securely creating, storing, and retrieving credentials to access applications, services, and IT resources in the cloud and on premise. This includes support for automatic periodic rotation of the encrypted secret value without your intervention, providing an extra level of security for your credentials. Enabling secret value rotation in AWS Secrets Manager also ensures that this secret value rotation happens without any effect on any database migration that relies on the secret. For secretly authenticating an endpoint database connection, create a secret whose identity or ARN you assign to `SecretsManagerSecretId`, which you include in your endpoint settings. For secretly authenticating Oracle ASM as part of an Oracle endpoint, create a secret whose identity or ARN you assign to `SecretsManagerOracleAsmSecretId`, which you also include in your endpoint settings.

> **Note**
>
> You can't use master credentials managed by Amazon RDS Aurora. These credentials don't include host or port information, which AWS DMS needs to establish connections. Instead, create a new user and secret. For information about creating a user and secret, see Using the AWS Management Console to create a secret and secret access role following.

For more information on AWS Secrets Manager, see What Is AWS Secrets Manager? in the *AWS Secrets Manager User Guide*.

AWS DMS supports secret authentication for the following on-premise or AWS-managed databases on supported source and target endpoints:

- Amazon DocumentDB
- IBM Db2 LUW
- Microsoft SQL Server
- MongoDB
- MySQL
- Oracle
- PostgreSQL

- Amazon Redshift
- SAP ASE

For connection to any of these databases, you have the choice of entering one of the following sets of values, but not both, as part of your endpoint settings:

- Clear-text values to authenticate the database connection using the `UserName`, `Password`, `ServerName`, and `Port` settings. For an Oracle endpoint that also uses Oracle ASM, include additional clear-text values to authenticate ASM using the `AsmUserName`, `AsmPassword`, and `AsmServerName` settings.
- Secret authentication using values for the `SecretsManagerSecretId` and `SecretsManagerAccessRoleArn` settings. For an Oracle endpoint using Oracle ASM, include additional values for the `SecretsManagerOracleAsmSecretId` and `SecretsManagerOracleAsmAccessRoleArn` settings. The secret values for these settings can include the following for:
  - `SecretsManagerSecretId` – The full Amazon Resource Name (ARN), partial ARN, or friendly name of a secret that you have created for endpoint database access in the AWS Secrets Manager.
  - `SecretsManagerAccessRoleArn` – The ARN of a secret access role that you have created in IAM to provide AWS DMS access to this `SecretsManagerSecretId` secret on your behalf.
  - `SecretsManagerOracleAsmSecretId` – The full Amazon Resource Name (ARN), partial ARN, or friendly name of a secret that you have created for Oracle ASM access in the AWS Secrets Manager.
  - `SecretsManagerOracleAsmAccessRoleArn` – The ARN of a secret access role that you have created in IAM to provide AWS DMS access to this `SecretsManagerOracleAsmSecretId` secret on your behalf.

> **Note**
>
> You can also use a single secret access role to provide AWS DMS access to both the `SecretsManagerSecretId` secret and the `SecretsManagerOracleAsmSecretId` secret. If you create this single secret access role for both secrets, ensure that you assign the same ARN for this access role to both `SecretsManagerAccessRoleArn` and `SecretsManagerOracleAsmAccessRoleArn`. For example, if your secret access role

for both secrets has its ARN assigned to the variable, ARN2xsecrets, you can set these ARN settings as follows:

```
SecretsManagerAccessRoleArn = ARN2xsecrets;
SecretsManagerOracleAsmAccessRoleArn = ARN2xsecrets;
```

For more information on creating these values, see Using the AWS Management Console to create a secret and secret access role.

After you have created and specified the required secret and secret access-role endpoint settings for your endpoints, update the permissions on the user accounts that will run the `CreateEndpoint` or `ModifyEndpoint` API request with these secret details. Ensure that these account permissions include the `IAM:GetRole` permission on the secret access role and the `SecretsManager:DescribeSecret` permission on the secret. AWS DMS requires these permissions to validate both the access role and its secret.

To provide and verify required user permissions

1. Sign in to the AWS Management Console and open the AWS Identity and Access Management console at https://console.aws.amazon.com/iam/.
2. Choose **Users**, then select the **User ID** used for making `CreateEndpoint` and `ModifyEndpoint` API calls.
3. From the **Permissions** tab, choose **{} JSON**.
4. Make sure the user has the permissions shown following.

```
{
 "Statement": [{
   "Effect": "Allow",
   "Action": [
    "iam:GetRole",
    "iam:PassRole"
   ],
   "Resource": "SECRET_ACCESS_ROLE_ARN"
  },
  {
   "Effect": "Allow",
   "Action": "secretsmanager:DescribeSecret",
```

```
    "Resource": "SECRET_ARN"
   }
  ]
}
```

5. If the user doesn't have those permission, add the permissions.
6. If you're using an IAM Role for making DMS API calls, repeat the steps above for the respective role.
7. Open a terminal and use the AWS CLI to validate that permissions are given correctly by assuming the Role or User used above.

   a. Validate user's permission on the SecretAccessRole using the IAM `get-role` command.

      ```
      aws iam get-role --role-name ROLE_NAME
      ```

      Replace *ROLE_NAME* with the name of `SecretsManagerAccessRole`.

      If the command returns an error message, make sure the permissions were given correctly.

   b. Validate user's permission on the secret using the Secrets Manager `describe-secret` command.

      ```
      aws secretsmanager describe-secret --secret-id SECRET_NAME OR SECRET_ARN --
      region=REGION_NAME
      ```

      User can be the friendly name, partial ARN or the full ARN. For more information, see describe-secret.

      If the command returns an error message, make sure the permissions were given correctly.

Using the AWS Management Console to create a secret and secret access role

You can use the AWS Management Console to create a secret for endpoint authentication and to create the policy and role to allow AWS DMS to access the secret on your behalf.

To create a secret using the AWS Management Console that AWS DMS can use to authenticate a database for source and target endpoint connections

1. Sign in to the AWS Management Console and open the AWS Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Choose **Store a new secret**.
3. Under **Select secret type** on the **Store a new secret** page, choose **Other type of secrets**, then choose **Plaintext**.

> **Note**
>
> This is the only place that you need to enter clear text credentials to connect to your endpoint database from this point forward.

4. In the **Plaintext** field:

   - For a secret whose identity you assign to `SecretsManagerSecretId`, enter the following JSON structure.

     ```
     {
       "username": db_username,
       "password": db_user_password,
       "port": db_port_number,
       "host": db_server_name
     }
     ```

     > **Note**
     >
     > This is the minimum list of JSON members required to authenticate the endpoint database. You can add any additional JSON endpoint settings as JSON members in all lower case that you want. However, AWS DMS ignores any additional JSON members for endpoint authentication.

     Here, *`db_username`* is the name of the user accessing the database, *`db_user_password`* is the password of the database user, *`db_port_number`* is the port number to access the database, and *`db_server_name`* is the database server name (address) on the web, as in the following example.

```
{
  "username": "admin",
  "password": "some_password",
  "port": "8190",
  "host": "oracle101.abcdefghij.us-east-1.rds.amazonaws.com"
}
```

- For a secret whose identity you assign to `SecretsManagerOracleAsmSecretId`, enter the following JSON structure.

```
{
  "asm_user": asm_username,
  "asm_password": asm_user_password,
  "asm_server": asm_server_name
}
```

> **Note**
>
> This is the minimum list of JSON members required to authenticate Oracle ASM for an Oracle endpoint. It is also the complete list that you can specify based on the available Oracle ASM endpoint settings.

Here, *`asm_username`* is the name of the user accessing Oracle ASM, *`asm_user_password`* is the password of the Oracle ASM user, and *`asm_server_name`* is the Oracle ASM server name (address) on the web, including the port, as in the following example.

```
{
  "asm_user": "oracle_asm_user",
  "asm_password": "oracle_asm_password",
  "asm_server": "oracle101.abcdefghij.us-east-1.rds.amazonaws.com:8190/+ASM"
}
```

5. Select an AWS KMS encryption key to encrypt the secret. You can accept the default encryption key created for your service by AWS Secrets Manager or select a AWS KMS key that you create.

6. Specify a name to reference this secret and an optional description. This is the friendly name that you use as the value for `SecretsManagerSecretId` or `SecretsManagerOracleAsmSecretId`.
7. If you want to enable automatic rotation in the secret, you need to select or create an AWS Lambda function with permission to rotate the credentials for the secret as described. However, before setting automatic rotation to use your Lambda function, ensure that the configuration settings for the function add the following four characters to the value of the `EXCLUDE_CHARACTERS` environment variable.

```
;.:+{}*&,%\
```

   AWS DMS doesn't allow these characters in passwords used for endpoint credentials. Configuring your Lambda function to exclude them prevents AWS Secrets Manager from generating these characters as part of its rotated password values. After you set automatic rotation to use your Lambda function, AWS Secrets Manager immediately rotates the secret to validate your secret configuration.

   > **Note**
   >
   > Depending on your database engine configuration, your database might not fetch the rotated credentials. In this case, you need to manually restart the task to refresh the credentials.

8. Review and store your secret in AWS Secrets Manager. You can then look up each secret by its friendly name in AWS Secrets Manager, then retrieve the secret ARN as the value for `SecretsManagerSecretId` or `SecretsManagerOracleAsmSecretId` as appropriate to authenticate access to your endpoint database connection and Oracle ASM (if used).

To create the secret access policy and role to set your `SecretsManagerAccessRoleArn` or `SecretsManagerOracleAsmAccessRoleArn`, which allows AWS DMS to access AWS Secrets Manager to access your appropriate secret

1. Sign in to the AWS Management Console and open the AWS Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.
2. Choose **Policies**, then choose **Create policy**.
3. Choose **JSON** and enter the following policy to enable access to and decryption of your secret.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    }
  ]
}
```

Here, *secret_arn* is the ARN of your secret, which you can get from either `SecretsManagerSecretId` or `SecretsManagerOracleAsmSecretId` as appropriate, and *kms_key_arn* is the ARN of the AWS KMS key that you are using to encrypt your secret, as in the following example.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:us-
east-2:123456789012:secret:MySQLTestSecret-qeHamH"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                        "kms:Decrypt",
                        "kms:DescribeKey"
                      ],
            "Resource": "arn:aws:kms:us-
east-2:123456789012:key/761138dc-0542-4e58-947f-4a3a8458d0fd"
        }
     ]
}
```

> **Note**
>
> If you use the default encryption key created by AWS Secrets Manager, you do not have to specify the AWS KMS permissions for *kms_key_arn*.
> If you want your policy to provide access to both secrets, simply specify an additional JSON resource object for the other *secret_arn*.
> If your secret is in a different account, then the `SecretsManagerAccessRoleArn` role needs an additional policy to verify the cross account secret. For such use cases, add the action `secretsmanager:DescribeSecret` to the policy. For more details on setting up a cross-account secret, see [Permissions to AWS Secrets Manager secrets for users in a different account](#).

4. Review and create the policy with a friendly name and optional description.
5. Choose **Roles**, then choose **Create role**.
6. Choose **AWS service** as the type of trusted entity.
7. Choose **DMS** from the list of services as the trusted service, then choose **Next: Permissions**.
8. Look up and attach the policy you created in step 4, then proceed through adding any tags and review your role. At this point, edit the trust relationships for the role to use your AWS DMS regional service principal as the trusted entity. This principal has the following format.

```
dms.region-name.amazonaws.com
```

Here, *region-name* is the name of your region, such as `us-east-1`. Thus, an AWS DMS regional service principal for this region follows.

```
dms.us-east-1.amazonaws.com
```

9. After editing the trusted entity for the role, create the role with a friendly name and optional description. You can now look up your new role by its friendly name in IAM, then retrieve the role ARN as the `SecretsManagerAccessRoleArn` or `SecretsManagerOracleAsmAccessRoleArn` value to authenticate your endpoint database connection.

To use secrets manager with a replication instance in a private subnet

1. Create a secret manager VPC endpoint and note the DNS for the endpoint. For more information about creating a secrets manager VPC endpoint, see [Connecting to Secrets Manager through a VPC endpoint](#) in the *AWS Secrets Manager User Guide.*
2. For the VPC endpoint security group ingress rules, allow HTTPS traffic from replication instance's private ip address or security groups that is attached to replication instances.
3. For the replication instance security group egress rules, allow all traffic for destination `0.0.0.0/0`.
4. Set the endpoint extra connection attribute, `secretsManagerEndpointOverride=`*`secretsManager endpoint DNS`* to provide the secret manager VPC endpoint DNS, as shown in the following example.

```
secretsManagerEndpointOverride=vpce-1234a5678b9012c-12345678.secretsmanager.eu-west-1.vpce.amazonaws.com
```

Using service-linked roles for AWS DMS

AWS Database Migration Service uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to AWS DMS. Service-linked roles are predefined by AWS DMS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up AWS DMS easier because you don't have to manually add the necessary permissions. AWS DMS defines the permissions of its service-linked roles, and unless defined otherwise, only AWS DMS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy can't be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your AWS DMS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked roles for AWS DMS features

Topics

Service-linked roles for AWS DMS Fleet Advisor

AWS DMS Fleet Advisor uses the service-linked role named **AWSServiceRoleForDMSFleetAdvisor** – DMS Fleet Advisor uses this service-linked role to manage Amazon CloudWatch metrics. This service-linked role is attached to the following managed policy: `AWSDMSFleetAdvisorServiceRolePolicy`. For updates to this policy, see AWS managed policies for AWS Database Migration Service.

The AWSServiceRoleForDMSFleetAdvisor service-linked role trusts the following services to assume the role:

- `dms-fleet-advisor.amazonaws.com`

The role permissions policy named AWSDMSFleetAdvisorServiceRolePolicy allows AWS DMS Fleet Advisor to complete the following actions on the specified resources:

- Action: `cloudwatch:PutMetricData` on `all AWS resources`

  This permission allows principals to publish metric data points to Amazon CloudWatch. AWS DMS Fleet Advisor requires this permission to display charts with database metrics from CloudWatch.

The following code example shows the AWSDMSFleetAdvisorServiceRolePolicy policy that you use to create the AWSDMSFleetAdvisorServiceRolePolicy role.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Resource": "*",
        "Action": "cloudwatch:PutMetricData",
        "Condition": {
            "StringEquals": {
                "cloudwatch:namespace": "AWS/DMS/FleetAdvisor"
            }
        }
    }
}
```

You must configure permissions to allow an IAM entity, such as a user, group, or role, to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for AWS DMS Fleet Advisor

You can use the IAM console to create a service-linked role with the **DMS – Fleet Advisor** use case. In the AWS CLI or the AWS API, create a service-linked role with the `dms-fleet-advisor.amazonaws.com` service name. For more information, see Creating a service-linked role in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

Make sure that you create this role before you create a data collector. DMS Fleet Advisor uses this role to display charts with database metrics in the AWS Management Console. For more information, see Creating a data collector.

Editing a service-linked role for AWS DMS Fleet Advisor

AWS DMS doesn't allow you to edit the AWSServiceRoleForDMSFleetAdvisor service-linked role. After you create a service-linked role, you can't change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for AWS DMS Fleet Advisor

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. Thus, you don't have an unused entity that isn't actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

> **Note**
>
> If the AWS DMS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete AWS DMS resources used by the AWSServiceRoleForDMSFleetAdvisor

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
2. In the navigation pane, choose **Data collectors** under **Discover**. The **Data collectors** page opens.
3. Choose your data collector and choose **Delete**.
4. To confirm deletion, enter the data collector name in the text input field. Next, choose **Delete**.

> **Important**
>
> When you delete a DMS data collector, DMS Fleet Advisor deletes all databases from Inventory that you discovered using this collector.

After you delete all data collectors, you can delete the service-linked role.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForDMSFleetAdvisor service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported regions for AWS DMS Fleet Advisor service-linked roles

AWS DMS Fleet Advisor supports using service-linked roles in all of the regions where the service is available. For more information, see Supported AWS Regions.

Service-linked role for AWS DMS

AWS DMS uses the service-linked role named **AWSServiceRoleForDMSServerless**. AWS DMS uses this service-linked role to create and manage AWS DMS resources on your behalf. AWS DMS uses this role for automatic instance management so that you only have to manage replications.

The AWSServiceRoleForDMSServerless service-linked role trusts the following services to assume the role:

- `dms.amazonaws.com`

You must configure permissions to allow an IAM entity, such as a user, group, or role, to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for AWS DMS

When you start a replication task, or start a premigration assessment, AWS DMS programmatically creates a AWS DMS service linked role. You can view this role in the IAM console. You can also choose to create this role manually. To create the role manually, use the IAM console to create a service-linked role with the **DMS** use case. In the AWS CLI or the AWS API, create a service-linked role using `dms.amazonaws.com` for the service name. For more information, see Creating a service-linked role in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

> **Note**
>
> If you delete a role while you have replications in your account, the replication results in a failure.

Editing a service-linked role for AWS DMS

AWS DMS does not allow you to edit the AWSServiceRoleForDMSServerless service-linked role. After you create a service-linked role, you can't change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for AWS DMS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. Thus, you don't have an unused entity that isn't actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

> **Note**
>
> If the AWS DMS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete AWS DMS resources used by the AWSServiceRoleForDMSServerless

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2.
2. In the navigation pane, choose **Serverless replications** under **Migrate data**. The **Serverless** page opens.
3. Choose your serverless replication and choose **Delete**.
4. To confirm deletion, enter the serverless replication name in the text input field. Next, choose **Delete**.

After you delete all serverless replications, you can delete the service-linked role.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForDMSServerless service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported regions for AWS DMS service-linked roles

AWS DMS supports using service-linked roles in all of the regions where the service is available.

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

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 an 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, your policies must be updated 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 that are 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's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I'm an administrator and want to allow others to access AWS DMS

To allow others to access AWS DMS, you must grant permission to the people or applications that need access. If you are using AWS IAM Identity Center to manage people and applications, you assign permission sets to users or groups to define their level of access. Permission sets automatically create and assign IAM policies to IAM roles that are associated with the person or application. For more information, see Permission sets in the *AWS IAM Identity Center User Guide*.

If you are not using IAM Identity Center, you must create IAM entities (users or roles) for the people or applications that need access. You must then attach a policy to the entity that grants them the correct permissions in AWS DMS. After the permissions are granted, provide the credentials to the user or application developer. They will use those credentials to access AWS. To learn more about creating IAM users, groups, policies, and permissions, see IAM Identities and Policies and permissions in IAM 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.
- 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 Cross account resource access in IAM in the *IAM User Guide*.

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 following policy 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.

JSON

```
{

  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dms:*",
      "Resource": "arn:aws:dms:*:123456789012:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:ListAliases",
        "kms:DescribeKey"
      ],
      "Resource": "arn:aws:kms:*:123456789012:key/*"
    },
```

```
    {
      "Effect": "Allow",
      "Action": [
        "iam:GetRole",
        "iam:PassRole",
        "iam:CreateRole",
        "iam:AttachRolePolicy"
      ],
      "Resource": "arn:aws:iam::123456789012:role/*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "dms.amazonaws.com"
        }
      }
    },
    {
      "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": "arn:aws:logs:*:123456789012:*"
      }
    ]
  }
```

The breakdown of these following permissions 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": "arn:aws:dms:region:account:resourcetype/id"
}
```

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": "arn:aws:service:region:account:resourcetype/id"
        }
```

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 AWS DMS](#).

```
{
            "Effect": "Allow",
            "Action": [
```

```
                "iam:GetRole",
                "iam:PassRole",
                "iam:CreateRole",
                "iam:AttachRolePolicy"
            ],
            "Resource": "arn:aws:service:region:account:resourcetype/id"
        }
```

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": "arn:aws:service:region:account:resourcetype/id"
        }
```

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": "arn:aws:service:region:account:resourcetype/id"
        }
```

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

```
{
            "Effect": "Allow",
```

```
            "Action": [
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:FilterLogEvents",
                "logs:GetLogEvents"
            ],
            "Resource": "arn:aws:service:region:account:resourcetype/id"
        }
```

If you use the AWS DMS console, the AWS Command Line Interface (AWS CLI) or the AWS DMS API for your migration, you need to add several roles to your account. For more information about adding these roles, see [Creating the IAM roles to use with AWS DMS](#).

Creating the IAM roles to use with AWS DMS

If you use the AWS DMS console, 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",
        "PolicyId": "ANPAJHKIGMBQI4AEFFSYO",
        "Arn": "arn:aws:iam::aws:policy/service-role/AmazonDMSVPCManagementRole",
        "Path": "/service-role/",
        "DefaultVersionId": "v4",
        "AttachmentCount": 1,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "Description": "Provides access to manage VPC settings for AWS managed customer
 configurations",
        "CreateDate": "2015-11-18T16:33:19+00:00",
        "UpdateDate": "2024-07-25T15:19:01+00:00",
        "Tags": []
    }
}
```

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 v4
```

The information returned from the command is as follows.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Sid": "ExampleStatementID",
        "Effect": "Allow",
        "Action": [
            "ec2:CreateNetworkInterface",
            "ec2:DeleteNetworkInterface",
            "ec2:DescribeAvailabilityZones",
            "ec2:DescribeDhcpOptions",
            "ec2:DescribeInternetGateways",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeSecurityGroups",
            "ec2:DescribeSubnets",
            "ec2:DescribeVpcs",
            "ec2:ModifyNetworkInterfaceAttribute"
        ],
        "Resource": "*"
      }
    ]
}
```

You can use the same commands to get information about `AmazonDMSCloudWatchLogsRole` and the `AmazonDMSRedshiftS3Role` managed policy.

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 following IAM policy. Name the JSON file `dmsAssumeRolePolicyDocument.json`.

   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 following IAM policy. Name the JSON file `dmsAssumeRolePolicyDocument2.json`.

   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 following IAM policy. Name the JSON file `dmsAssumeRolePolicyDocument3.json`.

   JSON

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

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

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

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

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

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

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that AWS Database Migration Service gives another service to the resource. If the `aws:SourceArn` value doesn't contain the account ID, such as an AWS DMS replication instance name (ARN), you must use both global condition context keys to limit permissions. If you use both global condition context keys and the `aws:SourceArn` value contains the account ID, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement. Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

AWS DMS supports confused deputy options starting from the 3.4.7 version and higher. For more information, see AWS Database Migration Service 3.4.7 release notes. If your replication instance uses AWS DMS version 3.4.6 or lower, make sure that you upgrade to latest version before you set the confused deputy options.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcard characters (*) for the unknown portions of the ARN. For example, `arn:aws:dms:*:*123456789012*:rep:*`.

Topics

- IAM roles to use with AWS DMS API for cross-service confused deputy prevention
- IAM policy to store preflight assessments in Amazon S3 for cross-service confused deputy prevention

- Using Amazon DynamoDB as a target endpoint with AWS DMS for cross-service confused deputy prevention

IAM roles to use with AWS DMS API for cross-service confused deputy prevention

To use the AWS CLI or the AWS DMS API for your database migration, you must add the `dms-vpc-role` and `dms-cloudwatch-logs-role` IAM roles to your AWS account before you can use the features of AWS DMS. For more information, see Creating the IAM roles to use with AWS DMS.

The following example shows policies for using the `dms-vpc-role` role with the `my-replication-instance` replication instance. Use these policies to prevent the confused deputy problem.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
        "Sid": "AllowDMSAssumeRole",
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "AWS:SourceAccount": "111122223333"
         },
         "ArnLike": {
             "aws:SourceArn": "arn:aws:dms:*:123456789012:*"
         }
      }
    }
  ]
}
```

IAM policy to store preflight assessments in Amazon S3 for cross-service confused deputy prevention

To store preassessment results in your S3 bucket, you create an IAM policy that allows AWS DMS to manage objects in Amazon S3. For more information, see Create IAM resources.

The following example shows a trust policy with confused deputy conditions that are set on an IAM role that allows AWS DMS to access all tasks and assessment runs under a specified user account.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDMSAssumeRole",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "AWS:SourceAccount": "111122223333"
        },
        "ArnLike": {
            "aws:SourceArn": [
                "arn:aws:dms:*:123456789012:assessment-run:*",
                "arn:aws:dms:*:123456789012:task:*"
          ]
        }
      }
    }
  ]
}
```

Using Amazon DynamoDB as a target endpoint with AWS DMS for cross-service confused deputy prevention

To use Amazon DynamoDB as a target endpoint for your database migration, you must create the IAM role that allows AWS DMS to assume and grant access to the DynamoDB tables. Then, use this role when you create your target DynamoDB endpoint in AWS DMS. For more information, see Using Amazon DynamoDB as a target.

The following example shows a trust policy with confused deputy conditions that are set on an IAM role that allows all AWS DMS endpoints to access DynamoDB tables.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDMSAssumeRole",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "*",
      "Condition": {
      "StringEquals": {
          "AWS:SourceAccount": "111122223333"
        },
        "ArnLike": {
        "aws:SourceArn": [
            "arn:aws:dms:*:123456789012:assessment-run:*",
            "arn:aws:dms:*:123456789012:task:*"
            ]
         }
      }
    }
  ]
}
```

AWS managed policies for AWS Database Migration Service

Topics

AWS managed policy: AmazonDMSVPCManagementRole

This policy is attached to the `dms-vpc-role` role, which allows AWS DMS to perform actions on your behalf.

This policy grants contributor permissions that allow AWS DMS to manage network resources.

Permissions details

This policy includes the following operations:

- `ec2:CreateNetworkInterface` – AWS DMS needs this permission to create network interfaces. These interfaces are essential for the AWS DMS replication instance to connect to the source and target databases.
- `ec2:DeleteNetworkInterface` – AWS DMS needs this permission to clean up network interfaces that it created once they are no longer needed. This helps in resource management and avoiding unnecessary costs.
- `ec2:DescribeAvailabilityZones` – This permission allows AWS DMS to retrieve information about the availability zones in a region. AWS DMS uses this information to ensure that it provisions resources in the correct zones for redundancy and availability.
- `ec2:DescribeDhcpOptions` – AWS DMS retrieves the DHCP options set details for the specified VPC. This information is required to configure the networking correctly for the replication instances.
- `ec2:DescribeInternetGateways` – AWS DMS may require this permission to understand the internet gateways configured in the VPC. This information is crucial if the replication instance or databases need internet access.

- `ec2:DescribeNetworkInterfaces` – AWS DMS retrieves information about existing network interfaces within the VPC. This information is necessary for AWS DMS to configure the network interfaces correctly and ensure proper network connectivity for the migration process.
- `ec2:DescribeSecurityGroups` – Security groups control the inbound and outbound traffic to instances and resources. AWS DMS needs to describe security groups to correctly configure network interfaces and ensure proper communication between the replication instance and the databases.
- `ec2:DescribeSubnets` – This permission allows AWS DMS to list the subnets in a VPC. AWS DMS uses this information to launch replication instances in the appropriate subnets, ensuring they have the necessary network connectivity.
- `ec2:DescribeVpcs` – Describing VPCs is essential for AWS DMS to understand the network environment where the replication instance and databases reside. This includes knowing the CIDR blocks and other VPC-specific configurations.
- `ec2:ModifyNetworkInterfaceAttribute` – This permission is required for AWS DMS to modify attributes of the network interfaces it manages. This could include adjusting settings to ensure connectivity and security.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
   "Sid": "Statement1",
   "Effect": "Allow",
   "Action": [
    "ec2:CreateNetworkInterface",
    "ec2:DeleteNetworkInterface",
    "ec2:DescribeAvailabilityZones",
    "ec2:DescribeDhcpOptions",
    "ec2:DescribeInternetGateways",
    "ec2:DescribeNetworkInterfaces",
    "ec2:DescribeSecurityGroups",
    "ec2:DescribeSubnets",
    "ec2:DescribeVpcs",
    "ec2:ModifyNetworkInterfaceAttribute"
   ],
   "Resource": "*"
```

```
    }
      ]
  }
```

AWS managed policy: AWSDMSServerlessServiceRolePolicy

This policy is attached to the `AWSServiceRoleForDMSServerless` role, which allows AWS DMS to perform actions on your behalf. For more information, see Service-linked role for AWS DMS.

This policy grants contributor permissions that allow AWS DMS to manage replication resources.

Permissions details

This policy includes the following permissions.

- **AWS DMS** – Allows principals to interact with AWS DMS resources.
- **Amazon S3** – Allows DMS to create an S3 bucket to store a premigration assessment. The S3 bucket is created for one user per Region and its bucket policy limits access to only the service's service role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "id0",
            "Effect": "Allow",
            "Action": [
                "dms:CreateReplicationInstance",
                "dms:CreateReplicationTask"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "dms:req-tag/ResourceCreatedBy": "DMSServerless"
                }
            }
        },
        {
            "Sid": "id1",
            "Effect": "Allow",
            "Action": [
```

```
                "dms:DescribeReplicationInstances",
                "dms:DescribeReplicationTasks"
            ],
            "Resource": "*"
        },
        {
            "Sid": "id2",
            "Effect": "Allow",
            "Action": [
                "dms:StartReplicationTask",
                "dms:StopReplicationTask",
                "dms:ModifyReplicationTask",
                "dms:DeleteReplicationTask",
                "dms:ModifyReplicationInstance",
                "dms:DeleteReplicationInstance"
            ],
            "Resource": [
                "arn:aws:dms:*:*:rep:*",
                "arn:aws:dms:*:*:task:*"
            ],
            "Condition": {
                "StringEqualsIgnoreCase": {
                    "aws:ResourceTag/ResourceCreatedBy": "DMSServerless"
                }
            }
        },
        {
            "Sid": "id3",
            "Effect": "Allow",
            "Action": [
                "dms:TestConnection",
                "dms:DeleteConnection"
            ],
            "Resource": [
                "arn:aws:dms:*:*:rep:*",
                "arn:aws:dms:*:*:endpoint:*"
            ]
        },
        {
            "Sid": "id4",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:DeleteObject",
```

```
                "s3:GetObject",
                "s3:PutObjectTagging"
            ],
            "Resource": [
                "arn:aws:s3:::dms-serverless-premigration-results-*",
                "arn:aws:s3:::dms-premigration-results-*"
            ],
            "Condition": {
                "StringEquals": {
                    "s3:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        },
        {
            "Sid": "id5",
            "Effect": "Allow",
            "Action": [
                "s3:PutBucketPolicy",
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:CreateBucket"
            ],
            "Resource": [
                "arn:aws:s3:::dms-serverless-premigration-results-*",
                "arn:aws:s3:::dms-premigration-results-*"
            ],
            "Condition": {
                "StringEquals": {
                    "s3:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        },
        {
            "Sid": "id6",
            "Effect": "Allow",
            "Action": [
                "dms:StartReplicationTaskAssessmentRun"
            ],
            "Resource": [
                "arn:aws:dms:*:*:task:*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
```

```
                }
            }
        }
    ]
}
```

AWS managed policy: AmazonDMSCloudWatchLogsRole

This policy is attached to the `dms-cloudwatch-logs-role` role, which allows AWS DMS to perform actions on your behalf. For more information, see Using service-linked roles for AWS DMS.

This policy grants contributor permissions that allow AWS DMS to publish replication logs to CloudWatch logs.

Permissions details

This policy includes the following permissions.

- `logs` – Allows principals to publish logs to CloudWatch Logs. This permission is required so that AWS DMS can use CloudWatch to display replication logs.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDescribeOnAllLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "AllowDescribeOfAllLogStreamsOnDmsTasksLogGroup",
            "Effect": "Allow",
            "Action": [
```

```
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:dms-tasks-*",
                "arn:aws:logs:*:*:log-group:dms-serverless-replication-*"
            ]
        },
        {
            "Sid": "AllowCreationOfDmsLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:dms-tasks-*",
                "arn:aws:logs:*:*:log-group:dms-serverless-replication-*:log-
stream:"
            ]
        },
        {
            "Sid": "AllowCreationOfDmsLogStream",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:dms-tasks-*:log-stream:dms-task-*",
                "arn:aws:logs:*:*:log-group:dms-serverless-replication-*:log-
stream:dms-serverless-*"
            ]
        },
        {
            "Sid": "AllowUploadOfLogEventsToDmsLogStream",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:dms-tasks-*:log-stream:dms-task-*",
                "arn:aws:logs:*:*:log-group:dms-serverless-replication-*:log-
stream:dms-serverless-*"
            ]
        }
    ]
```

```
}
```

AWS managed policy: AWSDMSFleetAdvisorServiceRolePolicy

You can't attach AWSDMSFleetAdvisorServiceRolePolicy to your IAM entities. This policy is attached to a service-linked role that allows AWS DMS Fleet Advisor to perform actions on your behalf. For more information, see Using service-linked roles for AWS DMS.

This policy grants contributor permissions that allow AWS DMS Fleet Advisor to publish Amazon CloudWatch metrics.

Permissions details

This policy includes the following permissions.

- `cloudwatch` – Allows principals to publish metric data points to Amazon CloudWatch. This permission is required so that AWS DMS Fleet Advisor can use CloudWatch to display charts with database metrics.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Resource": "*",
        "Action": "cloudwatch:PutMetricData",
        "Condition": {
            "StringEquals": {
                "cloudwatch:namespace": "AWS/DMS/FleetAdvisor"
            }
        }
    }
}
```

AWS managed policy: AmazonDMSRedshiftS3Role

This policy provides permissions that allow AWS DMS to manage S3 settings for Redshift endpoints.

Permissions details

This policy includes the following operations:

- `s3:CreateBucket` - Allows DMS to create S3 buckets with the "dms-" prefix
- `s3:ListBucket` - Allows DMS to list the contents of S3 buckets with the "dms-" prefix
- `s3:DeleteBucket` - Allows DMS to delete S3 buckets with the "dms-" prefix
- `s3:GetBucketLocation` - Allows DMS to retrieve the Region where an S3 bucket is located
- `s3:GetObject` - Allows DMS to retrieve objects from S3 buckets with the "dms-" prefix
- `s3:PutObject` - Allows DMS to add objects to S3 buckets with the "dms-" prefix
- `s3:DeleteObject` - Allows DMS to delete objects from S3 buckets with the "dms-" prefix
- `s3:GetObjectVersion` - Allows DMS to retrieve specific versions of objects in versioned buckets
- `s3:GetBucketPolicy` - Allows DMS to retrieve bucket policies
- `s3:PutBucketPolicy` - Allows DMS to create or update bucket policies
- `s3:GetBucketAcl` - Allows DMS to retrieve bucket access control lists (ACLs)
- `s3:PutBucketVersioning` - Allows DMS to enable or suspend versioning on buckets
- `s3:GetBucketVersioning` - Allows DMS to retrieve the versioning status of buckets
- `s3:PutLifecycleConfiguration` - Allows DMS to create or update lifecycle rules for buckets
- `s3:GetLifecycleConfiguration` - Allows DMS to retrieve lifecycle rules configured for buckets
- `s3:DeleteBucketPolicy` - Allows DMS to delete bucket policies

All these permissions apply only to resources with ARN pattern: `arn:aws:s3:::dms-*`

JSON policy document

JSON

```
{
```

```
  "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-*"
    }
  ]
}
```

AWS DMS updates to AWS managed policies

View details about updates to AWS managed policies for AWS DMS since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the AWS DMS Document history page.

Change	Description	Date
AWSDMSServerlessServiceRolePolicy – Change	AWS DMS updated `AWSDMSServerlessServiceRolePolicy` to	November 5, 2025

Change	Description	Date
	allow DMS to create S3 buckets and put premigration assessment results into those buckets for replication tasks not related to DMS serverless.	
[Service-linked role for AWS DMS Serverless]() – Change	AWS DMS updated `AWSDMSServerlessServiceRolePolicy` to allow `dms:StartReplicationTaskAssessmentRun` to support running premigration assessments. AWS DMS also updated the serverless service-linked role to create S3 buckets and put the premigration assessment results into those buckets.	February 14, 2025
[AWSDMSServerlessServiceRolePolicy]() – Change	AWS DMS added `dms:ModifyReplicationTask` which is required by AWS DMS Serverless to call the `ModifyReplicationTask` operation to modify a replication task. AWS DMS added `dms:ModifyReplicationInstance` which is required by AWS DMS Serverless to call `ModifyReplicationInstance` operation to modify a replication instance.	January 17, 2025

Change	Description	Date
AmazonDMSVPCManagementRole – Change	AWS DMS added `ec2:DescribeDhcpOptions` and `ec2:DescribeNetworkInterfaces` operations to allow AWS DMS to manage network settings on your behalf.	June 17, 2024
AWSDMSServerlessServiceRolePolicy – New policy	AWS DMS added the `AWSDMSServerlessServiceRolePolicy` role to allow AWS DMS to create and manage services on your behalf, such as publishing Amazon CloudWatch metrics.	May 22, 2023
AmazonDMSCloudWatchLogsRole – Change	AWS DMS added the ARN for serverless resources to each of the permissions granted, to allow uploading AWS DMS replication logs from serverless replication configurations to CloudWatch Logs.	May 22, 2023
AWSDMSFleetAdvisorServiceRolePolicy – New policy	AWS DMS Fleet Advisor added a new policy to allow publishing metric data points to Amazon CloudWatch.	March 6, 2023
AWS DMS started tracking changes	AWS DMS started tracking changes for its AWS managed policies.	March 6, 2023

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 following programs:

- 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 on Amazon Web Services 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 CSPM – 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.

Infrastructure security in AWS Database Migration Service

As a managed service, AWS Database Migration Service is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well‐Architected Framework*.

You use AWS published API calls to access AWS DMS through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

You can call these API operations from any network location. AWS DMS also supports resource-based access policies, which can specify restrictions on actions and resources, for example, based on the source IP address. In addition, you can 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.

To confine your communications with AWS DMS within a single VPC, you can create a VPC interface endpoint that enables you to connect to AWS DMS through AWS PrivateLink. AWS PrivateLink helps ensure that any call to AWS DMS and its associated results remain confined to the specific VPC for which your interface endpoint is created. You can then specify the URL for this interface endpoint as an option with every AWS DMS command that you run using the AWS CLI or an SDK. Doing this helps ensure that your entire communications with AWS DMS remain confined to the VPC and are otherwise invisible to the public internet.

To create an interface endpoint to access DMS in a single VPC

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. From the navigation pane, choose **Endpoints**. This opens the **Create endpoints** page, where you can create the interface endpoint from a VPC to AWS DMS.

3. Choose **AWS services**, then search for and choose a value for **Service Name**, in this case AWS DMS in the following form.

   ```
   com.amazonaws.region.dms
   ```

   Here, *region* specifies the AWS Region where AWS DMS runs, for example `com.amazonaws.us-west-2.dms`.

4. For **VPC**, choose the VPC to create the interface endpoint from, for example `vpc-12abcd34`.
5. Choose a value for **Availability Zone** and for **Subnet ID**. These values should indicate a location where your chosen AWS DMS endpoint can run, for example `us-west-2a (usw2-az1)` and `subnet-ab123cd4`.
6. Choose **Enable DNS name** to create the endpoint with a DNS name. This DNS name consists of the endpoint ID (`vpce-12abcd34efg567hij`) hyphenated with a random string (`ab12dc34`). These are separated from the service name by a dot in reverse dot-separated order, with `vpce` added (`dms.us-west-2.vpce.amazonaws.com`).

   An example is `vpce-12abcd34efg567hij-ab12dc34.dms.us-west-2.vpce.amazonaws.com`.

7. For **Security group**, choose a group to use for the endpoint.

   When you set up your security group, make sure to allow outbound HTTPS calls from within it. For more information, see Creating security groups in the *Amazon VPC User Guide*.

8. Choose either **Full Access** or a custom value for **Policy**. For example, you might choose a custom policy similar to the following that restricts your endpoint's access to certain actions and resources.

   ```
   {
     "Statement": [
       {
         "Action": "dms:*",
         "Effect": "Allow",
         "Resource": "*",
         "Principal": "*"
       },
       {
         "Action": [
           "dms:ModifyReplicationInstance",
           "dms:DeleteReplicationInstance"
   ```

```
      ],
      "Effect": "Deny",
      "Resource": "arn:aws:dms:us-west-2:<account-id>:rep:<replication-instance-id>",
      "Principal": "*"
    }
  ]
}
```

Here, the sample policy allows any AWS DMS API call, except for deleting or modifying a specific replication instance.

You can now specify a URL formed using the DNS name created in step 6 as an option. You specify this for every AWS DMS CLI command or API operation to access the service instance using the created interface endpoint. For example, you might run the DMS CLI command `DescribeEndpoints` in this VPC as shown following.

```
$ aws dms describe-endpoints --endpoint-url https://vpce-12abcd34efg567hij-ab12dc34.dms.us-west-2.vpce.amazonaws.com
```

If you enable the private DNS option, you don't have to specify the endpoint URL in the request.

For more information on creating and using VPC interface endpoints (including enabling the private DNS option), see Interface VPC endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

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*:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dms:*"
            ],
            "Effect": "Deny",
            "Resource": "arn:aws:dms:us-
east-1:111122223333: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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dms:*"
            ],
            "Effect": "Deny",
            "Resource": "arn:aws:dms:us-
east-1:111122223333: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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dms:*"
            ],
            "Effect": "Deny",
            "Resource": "arn:aws:dms:us-
east-1:111122223333:task:U03YR4N47DXH3ATT4YMWOIT"
        }
    ]
}
```

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:U03YR4N47DXH3ATT4YMWOIT"
```

```
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:U03YR4N47DXH3ATT4YMWOIT
```

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.

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 preceding, 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":

JSON

```
{
    "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".

JSON

```
{
    "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".

JSON

```
{
```

```
    "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
```

Encryption for AWS DMS homogeneous migrations

AWS DMS homogeneous migrations also encrypt resources used for your data migrations, including storage and other components. If you don't specify a customer managed key, AWS DMS homogeneous migrations automatically use an AWS owned key to encrypt your resources.

When you use a customer managed key, AWS DMS homogeneous migrations create grants on your key to allow the service to encrypt and decrypt resources as needed. These grants are managed automatically as part of the migration lifecycle.

Using a customer managed key with homogeneous migrations

To use a customer managed key for your homogeneous migration, add the following permissions to the IAM permissions that you must grant to the IAM user account to use AWS DMS homogeneous migrations:

```
{
    "Effect": "Allow",
    "Action": [
        "kms:CreateGrant",
```

```
            "kms:DescribeKey",
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:GenerateDataKeyWithoutPlaintext"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "kms:ViaService": [
                    "dms.us-west-2.amazonaws.com",
                    "elasticfilesystem.us-west-2.amazonaws.com"
                ]
            }
        }
}
```

DMS will be creating an internal grant for accessing and managing the encryption keys during the migration process. In order to further scope down the CreateGrant operation, the following limitations can be applied to your customer managed key policy:

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
  },
  "Action": "kms:CreateGrant",
  "Resource": "*",
  "Condition": {
    "ForAllValues:StringEquals": {
      "kms:GrantOperations": [
        "CreateGrant",
        "DescribeKey",
        "Encrypt",
        "Decrypt",
        "RetireGrant",
        "GenerateDataKeyWithoutPlaintext"
      ]
    },
    "StringEquals": {
      "kms:ViaService": [
            "dms.us-west-2.amazonaws.com",
            "elasticfilesystem.us-west-2.amazonaws.com"
      ]
```

```
      }
    }
}
```

Important

Additional encryption context constraints are not supported at this time. Including such constraints will cause the migration to fail.

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 KMS key in your account (`aws/dms`) or you can create a custom KMS key. If you have an existing KMS key, you can also use that key for encryption.

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 KMS key (`aws/dms`) is created when you first launch a replication instance, if you haven't selected a custom KMS key from the **Advanced** section of the **Create Replication Instance** page. If you use the default KMS key, the only permissions you need to grant to the IAM user account you are using for migration are `kms:ListAliases` and `kms:DescribeKey`. For more information about using the default KMS key, see IAM permissions needed to use AWS DMS.

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

- Add the IAM user account used for the migration as a key administrator 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 KMS key](#) 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 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 KMS key aliases. For more information about creating your own AWS KMS keys and giving users access to a KMS key, see the *AWS KMS Developer Guide*.

If you don't specify a 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

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 following options 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.
- Amazon S3 – For more information, see Creating AWS KMS keys to encrypt Amazon S3 target objects.

After you have created your AWS DMS resources with a 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 – 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 multiple VPCs – 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 Direct Connect or a VPN – 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 – 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 – 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 MySQL-Compatible Edition 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.

Topics

To establish a secure connection, you provide the root certificate or the chain of intermediate CA certificates leading to the root (as a certificate bundle) that was used to sign the server's SSL certificate at the 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
MongoDB	Default	Supported	Not Supported	Supported
Db2 LUW	Default	Not Supported	Supported	Not Supported
Db2 for z/OS	Default	Not Supported	Supported	Not Supported

Note

The SSL Mode option on the DMS console or API doesn't apply to some data streaming and NoSQL services like Kinesis, and DynamoDB. They are secure by default, so DMS shows the SSL mode setting is equal to none (**SSL Mode=None**). You don't need to provide any additional configuration for your endpoint to make use of SSL. For example, when using Kinesis as a target endpoint, it is secure by default. All API calls to Kinesis use SSL, so there is no need for an additional SSL option in the DMS endpoint. You can securely put data and retrieve data through SSL endpoints using the HTTPS protocol, which DMS uses by default when connecting to a Kinesis Data Stream.

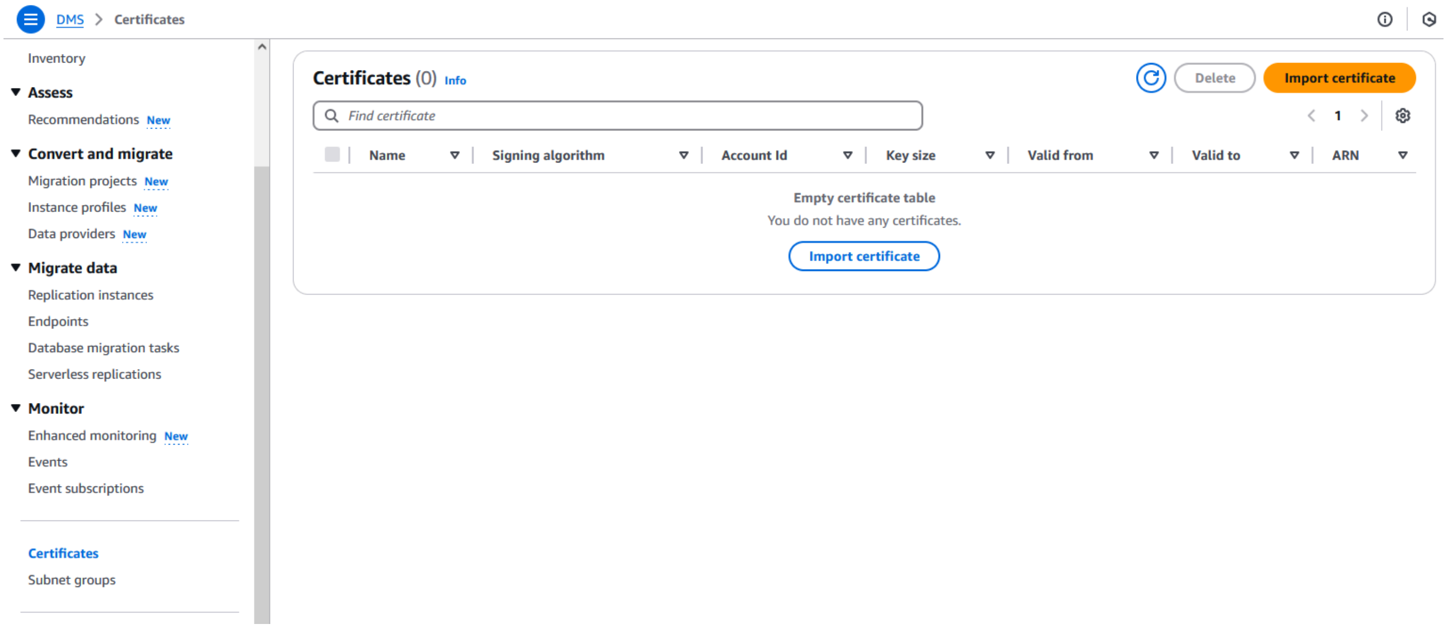
Limitations on using SSL with AWS DMS

Following are limitations on using SSL with AWS DMS:

- SSL connections to Amazon Redshift target endpoints are not 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.
- 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.
- AWS DMS does not support TLS version 1.3 for MySQL and MariaDb endpoints.

Managing certificates

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



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

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

To create an AWS DMS endpoint with SSL

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.

   If you're 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 about the permissions required for database migration, see IAM permissions needed to use AWS DMS.

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.

For Aurora Serverless as target, get the certificate mentioned in Using TLS/SSL with Aurora Serverless.

5. Create an endpoint as described in Step 2: Specify source and target endpoints

To modify an existing AWS DMS endpoint to use SSL

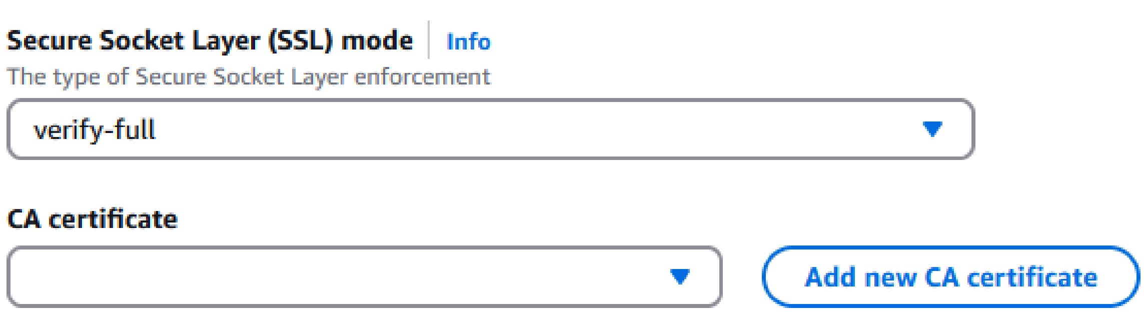
1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.

   If you're signed in as an IAM user, make sure that you 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.
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.



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 3: Create a task and migrate data.

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 needs a few additional steps. 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 open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.

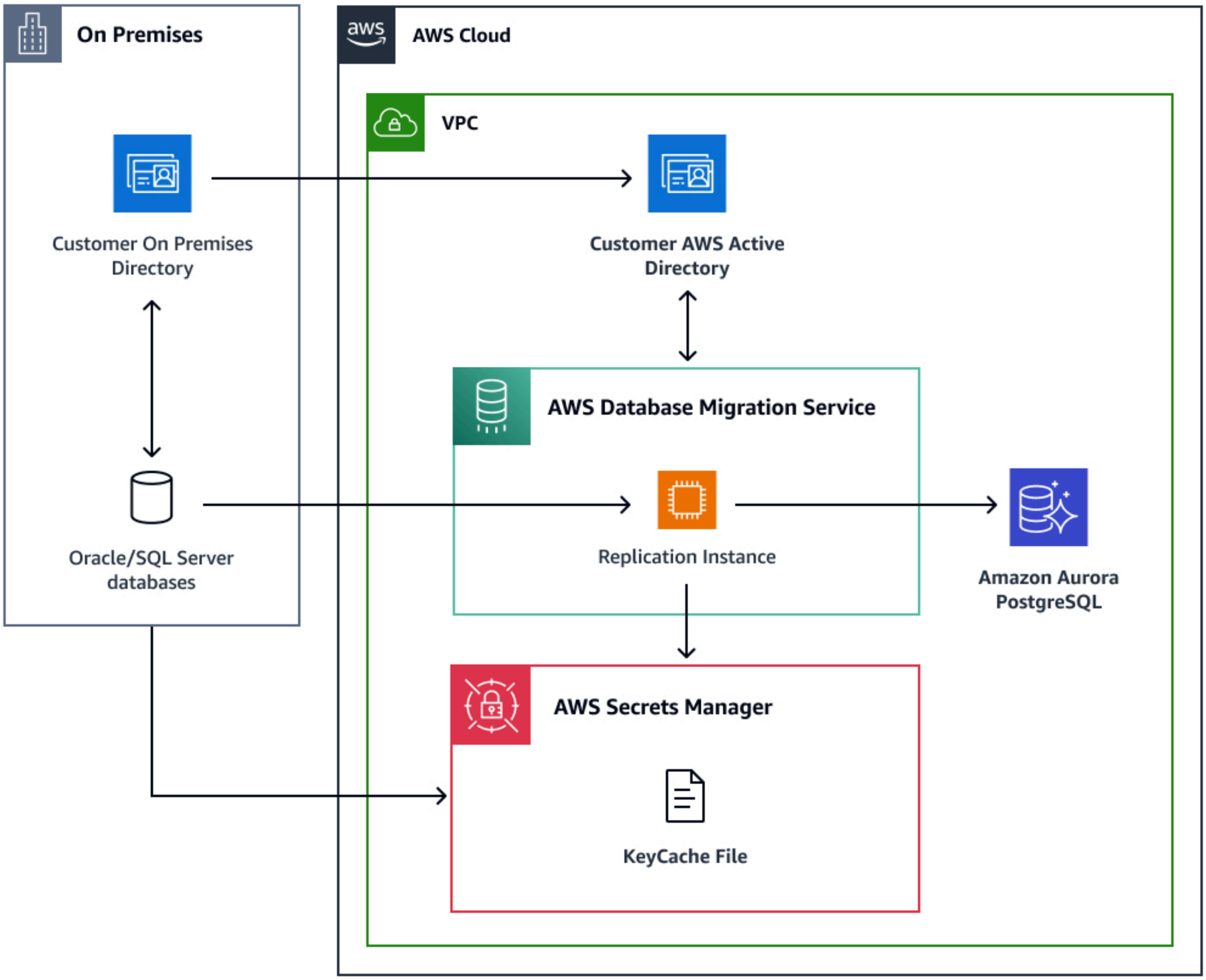
   If you're signed in as an IAM user, make sure that you have the appropriate permissions to access AWS DMS. For more information about the permissions required, see IAM permissions needed to use AWS DMS.
2. In the navigation pane, choose **Database migration tasks**.
3. Choose the task that uses the endpoint 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 **Save**.
8. Choose **Database migration tasks** from the navigation pane.
9. Choose the task that you stopped previously, and choose **Restart/Resume**.
10. Choose either **Restart** or **Resume**, depending on how you want to continue the task, and then choose **Start task**.

Using Kerberos Authentication with AWS Database Migration Service

Starting with DMS v3.5.3, you can configure your Oracle or SQL Server source endpoint to connect to your database instance using Kerberos authentication. DMS supports Directory Service for Microsoft Active Directory and Kerberos authentication. For more information about AWS-managed access to Microsoft Active Directory Services, see What is Directory Service?.

AWS DMS Kerberos Authentication Architecture Overview

The following diagram provides a high level overview of the AWS DMS Kerberos authentication workflow.



Limitations on using Kerberos authentication with AWS DMS

The following limitations apply when using Kerberos authentication with AWS DMS:

- DMS replication instances support one Kerberos `krb5.conf` file and one keycache file.
- You must update the Kerberos keycache file in Secrets Manager at least 30 minutes prior to the ticket expiring.

- A Kerberos-enabled DMS endpoint only works with a Kerberos-enabled DMS replication instance.

Prerequisites

To start, you must complete the following prerequisites from an existing Active Directory or Kerberos-authenticated host:

- Establish an Active Directory trust relationship with your on-premise AD. For more information, see [Tutorial: Create a trust relationship between your AWS Managed Microsoft AD and your self-managed Active Directory domain](#).
- Prepare a simplified version of the Kerberos `krb5.conf` configuration file. Include information about the realm, the location of the domain admin servers, and mappings of hostnames onto a Kerberos realm. You need to verify that the `krb5.conf` content is formatted with the correct mixed casing for the realms and domain realm names. For example:

```
[libdefaults]
 dns_lookup_realm = true
 dns_lookup_kdc = true
 forwardable = true
 default_realm = MYDOMAIN.ORG
[realms]
MYDOMAIN.ORG = {
kdc = mydomain.org
admin_server = mydomain.org
}
[domain_realm]
.mydomain.org = MYDOMAIN.ORG
mydomain.org = MYDOMAIN.ORG
```

- Prepare a Kerberos keycache file. The file contains a temporary Kerberos credential of the client principal information. The file does not store the client's password. Your DMS task uses this cache ticket information to get additional credentials without a password. Run the following steps on an existing Active Directory or Kerberos-authenticated host to generate a keycache file.
  - Create a Kerberos keytab file. You can generate a keytab file using the **kutil** or **ktpass** utility.

    For more information about the Microsoft **ktpass** utility, see [ktpass](#) in the *Windows Server documentation*.

    For more information about the MIT **kutil** utility, see [kutil](#) in the *MIT Kerberos Documentation*.

  - Create a Kerberos keycache file from keytab file using the **kinit** utility. For more information about the **kinit** utility, see [kinit](#) in the *MIT Kerberos Documentation*.
- Store the Kerberos keycache file in Secrets Manager using the `SecretBinary` parameter. When you upload the keycache file to Secrets Manager, DMS retrieves it, and then updates the local cache file about every 30 minutes. When the local keycache file exceeds the predefined expiration timestamp, DMS gracefully stops the task. To avoid authentication failures during an ongoing replication task, update the keycache file in Secrets Manager at least 30 minutes before the ticket expiration. For more information, see [createsecret](#) in the *Secrets Manager API Reference*. The following AWS CLI sample shows how to store the keycache file in binary format in Secrets Manager:

```
aws secretsmanager create-secret —name keycache —secret-binary fileb://keycachefile
```

- Grant an IAM role the `GetSecretValue` and `DescribeSecret` permissions to get the keycache file from Secrets Manager. Ensure that the IAM role includes the `dms-vpc-role` trust policy. For more information about the `dms-vpc-role` trust policy, see [Creating the IAM roles to use with AWS DMS](#).

The following example shows an IAM role policy with the Secrets Manager `GetSecretValue` and `DescribeSecret` permissions. The *`<keycache_secretsmanager_arn>`* value is the Keycache Secrets Manager ARN you created in the previous step.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ],
      "Resource": "*"
    }
  ]
}
```

Enabling Kerberos support on an AWS DMS replication instance

Kerberos realms are identical to domains in Windows. In order to resolve a principle realm, Kerberos relies on a Domain Name Service (DNS). When you set the `dns-name-servers` parameter, your replication instance will use your predefined custom set of DNS servers to resolve the Kerberos domain realms. Another alternative option to resolve Kerberos realm queries is to configure Amazon Route 53 on the replication instance virtual private cloud (VPC). For more information, see Route 53.

Enabling Kerberos support on a DMS replication instance using the AWS Management Console

To enable Kerberos support using the console, enter the following information in the **Kerberos authentication** section of the **Create Replication Instance** or **Modify Replication Instance** page:

- The content from your `krb5.conf` file
- The ARN of the Secrets Manager secret that contains the keycache file
- The ARN of the IAM role that has access to the secret manager ARN and permissions to retrieve the keycache file

Enabling Kerberos support on a DMS replication instance using the AWS CLI

The following AWS CLI sample call creates a private DMS replication instance with Kerberos support. The replication instance uses a custom DNS to resolve the Kerberos realm. For more information, see create-replication-instance.

```
aws dms create-replication-instance
--replication-instance-identifier my-replication-instance
--replication-instance-class dms.t2.micro
--allocated-storage 50
--vpc-security-group-ids sg-12345678
--engine-version 3.5.4
--no-auto-minor-version-upgrade
--kerberos-authentication-settings'{"KeyCacheSecretId":<secret-
id>,"KeyCacheSecretIamArn":<secret-iam-role-arn>,"Krb5FileContents":<krb5.conf file
 contents>}'
```

```
--dns-name-servers <custom dns server>
--no-publicly-accessible
```

Enabling Kerberos support on a source endpoint

Before enabling Kerberos authentication on a DMS Oracle or SQL server source endpoint, make sure you can authenticate to the source database using the Kerberos protocol from a client machine. You can use the AWS DMS Diagnostic AMI to launch an Amazon EC2 instance on the same VPC as the replication instance, and then test the kerberos authentication. For more information about the AMI, see Working with the AWS DMS diagnostic support AMI.

Using the AWS DMS console

Under **Access to endpoint database**, choose **Kerberos authentication**.

Using the AWS CLI

Specify the endpoint setting parameter and set `AuthenticationMethod` option as kerberos. For example:

Oracle

```
aws dms create-endpoint
--endpoint-identifier my-endpoint
--endpoint-type source
--engine-name oracle
--username dmsuser@MYDOMAIN.ORG
--server-name mydatabaseserver
--port 1521
--database-name mydatabase
--oracle-settings "{\"AuthenticationMethod\": \"kerberos\"}"
```

SQL Server

```
aws dms create-endpoint
--endpoint-identifier my-endpoint
--endpoint-type source
--engine-name sqlserver
--username dmsuser@MYDOMAIN.ORG
--server-name mydatabaseserver
--port 1433
```

```
--database-name mydatabase
--microsoft-sql-server-settings "{\"AuthenticationMethod\": \"kerberos\"}"
```

Testing a source endpoint

You must test the Kerberos-enabled endpoint against a Kerberos-enabled replication instance. When you don't properly confiugure the replication instance or source endpoint for Kerberos authentication, the endpoint `test-connection` action will fail, and might return Kerberos-related errors. For more information, see test-connection.

Quotas for AWS Database Migration Service

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

The maximum size of a database that AWS DMS can migrate depends on a number of factors. These include 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. Start slowly so you can get the configuration worked out, then add some complex objects. Finally, attempt a full load as a test.

Resource quotas for AWS Database Migration Service

Each AWS account has quotas for each AWS Region on the number of AWS DMS resources that can be created. After a quota for a resource has been reached, additional calls to create that resource fail with an exception.

The following table lists the AWS DMS resources and their quotas for each AWS Region.

Resource	Default quota
API request throttling	200 request maximum per second
API request refresh rate	8 requests per second
Replication instances per user account	60
Total amount of storage for a replication instance	30,000 GB
Event subscriptions per user account	60
Replication subnet groups per user account	60
Subnets per replication subnet group	60
Endpoints per user account	1,000

Resource	Default quota
Endpoints per replication instance	100
Tasks per user account	600
Tasks per replication instance	200
Certificates per user account	100
Data providers per user account	1,000
Instance profiles per user account	60
Migration projects per user account	50
DMS data collectors per user account	10
Target recommendations generated at one time	100
Number of files that DMS data collector can upload per hour	500
Homogeneous data migrations per user account	600
Homogeneous data migrations that run at one time	100
Homogeneous data migrations per migration project	10
Serverless replications	100

For more information on the API request throttling quota and refresh rate, see Understanding API request throttling.

The 30,000-GB quota for storage applies to all your AWS DMS replication instances in a given AWS Region. This storage is used to cache changes if a target can't keep up with a source, and for storing log information.

Understanding API request throttling

AWS DMS supports a varying, but maximum API request quota of 200 API calls per second. In other words, your API requests are throttled when they exceed this rate. Also, you can be limited to fewer API calls per second, depending on how long it takes AWS DMS to refresh your quota before you make another API request. This quota applies both when you make API calls directly and when they are made on your behalf as part of using the AWS DMS Management Console.

To understand how API request throttling works, it helps to imagine that AWS DMS maintains a token bucket that tracks your API requests. In this scenario, each token in the bucket allows you to make a single API call. You can have no more than 200 tokens in the bucket at any one time. When you make an API call, AWS DMS removes one token from the bucket. If you make 200 API calls in under a second, your bucket is empty and any attempt to make another API call fails. For each second that you don't make an API call, AWS DMS adds 8 tokens to the bucket, up to the 200 token maximum. This is the AWS DMS API request refresh rate. At any point after throttling, when you have tokens added to your bucket, you can make as many additional API calls as tokens available until your calls are throttled again.

If you are using the AWS CLI to run API calls that are throttled, AWS DMS returns an error like the following:

```
An error occurred (ThrottlingException) when calling the AwsDmsApiCall operation
 (reached max retries: 2): Rate exceeded
```

Here, *`AwsDmsApiCall`* is the name of the AWS DMS API operation that was throttled, for example, `DescribeTableStatistics`. You can then retry or make a different call after sufficient delay to avoid throttling.

> **Note**
>
> Unlike API request throttling managed by some other services, such as Amazon EC2, you can't order an increase in the API request throttling quotas managed by AWS DMS.

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.

For troubleshooting purposes, AWS DMS collects trace and dump files in the replication instance. You can provide these files to AWS Support should an issue occur requiring troubleshooting. By default, DMS purges trace and dump files that are older than thirty days. To opt out of trace and dump file collection, open a case with AWS Support.

Topics

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 Selecting the best size for a replication instance.

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.

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.

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.

AWS DMS does not create CloudWatch logs

If your replication task doesn't create CloudWatch logs, make sure that your account has the `dms-cloudwatch-logs-role` role. If this role is not present, do the following to create it:

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose the **Roles** tab. Choose **Create role**.
3. In the **Select type of trusted entity** section, choose **AWS service**.
4. In the **Choose a use case** section, choose **DMS**.
5. Choose **Next: Permissions**.
6. Enter **`AmazonDMSCloudWatchLogsRole`** in the search field, and check the box next to **AmazonDMSCloudWatchLogsRole**. This grants AWS DMS permissions to access CloudWatch.
7. Choose **Next: Tags**.
8. Choose **Next: Review**.
9. Enter **`dms-cloudwatch-logs-role`** for **Role name**. This name is case sensitive.
10. Choose **Create role**.

> **Note**
>
> If your account is part of AWS Organizations, verify that Service Control Policies (SCPs) are not restricting your IAM role permissions. SCPs can override and limit IAM role permissions even when they are properly configured.

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 NAT 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.

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.

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 following steps 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 -lmlocal2 tcp4-listen:database port,bind=0.0.0.0,reuseaddr,fork
 tcp4:source_database_ip_address:database_port
&
```

Use the Amazon EC2 instance IP address and the database port given preceding for the AWS DMS endpoint. Make sure that the endpoint has the security group that allows AWS DMS to access the database port. Note that the proxy needs to be running for the duration of your DMS task execution. Depending on the use case, you may need to automate the proxy setup.

Timestamps are garbled in Amazon Athena queries

If timestamps are garbled in Athena queries, use the AWS Management Console or the [ModifyEndpoint](#) action to set the `parquetTimestampInMillisecond` value for your Amazon S3 endpoint to `true`. For more information, see [S3Settings](#).

Troubleshooting issues with Oracle

Following, you can learn about troubleshooting issues specific to using AWS DMS with Oracle databases.

Topics

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 the **Endpoint settings** 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 open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
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 open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
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. The way the NUMBER type is converted can also be affected by using endpoint settings for the source Oracle endpoint. These endpoint settings are documented in Endpoint settings when using Oracle as a source for AWS DMS.

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 isn't 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`.

Table Error

`Table Error` appears in table statistics during replication if a `WHERE` clause doesn't reference a primary key column, and supplemental logging isn't used for all columns.

To fix this issue, turn on supplemental logging for all columns of the referenced table. For more information, see Setting up supplemental logging.

Error: Cannot retrieve Oracle archived Redo log destination ids

This error occurs when your Oracle source doesn't have any archive logs generated or V $ARCHIVED_LOG is empty. You can resolve the error by switching logs manually.

For an Amazon RDS database, run the following procedure to switch log files. The `switch_logfile` procedure doesn't have parameters.

```
exec rdsadmin.rdsadmin_util.switch_logfile;
```

For a self-managed Oracle source database, use the following command to force a log switch.

```
ALTER SYSTEM SWITCH LOGFILE ;
```

Evaluating read performance of Oracle redo or archive logs

If you experience performance issues with your Oracle source, you can evaluate the read performance of your Oracle redo or archive logs to find ways to improve performance. To test the redo or archive log read performance, use the AWS DMS diagnostic Amazon machine image (AMI).

You can use the AWS DMS diagnostic AMI to do the following:

- Use the bFile method to evaluate redo log file performance.
- Use the LogMiner method to evaluate redo log file performance.
- Use the PL/SQL (`dbms_lob.read`) method to evaluate redo log file performance.
- Use Single-thread to evaluate read performance on ASMFile.
- Use Multi-threads to evaluate read performance on ASMFile.
- Use Direct OS Readfile() Windows or Pread64 Linux function to evaluate the redo log file.

Then you can take remedial steps based upon the results.

To test read performance on an Oracle redo or archive log file

1. Create an AWS DMS diagnostic AMI Amazon EC2 instance and connect to it.

   For more information see, Working with the AWS DMS diagnostic AMI.

2. Run the **awsreplperf** command.

   ```
   $ awsreplperf
   ```

   The command displays the AWS DMS Oracle Read Performance Utility options.

   ```
   0. Quit
   1. Read using Bfile
   2. Read using LogMiner
   3. Read file PL/SQL (dms_lob.read)
   4. Read ASMFile Single Thread
   5. Read ASMFile Multi Thread
   6. Readfile() function
   ```

3. Select an option from the list.

4. Enter the following database connection and archive log information.

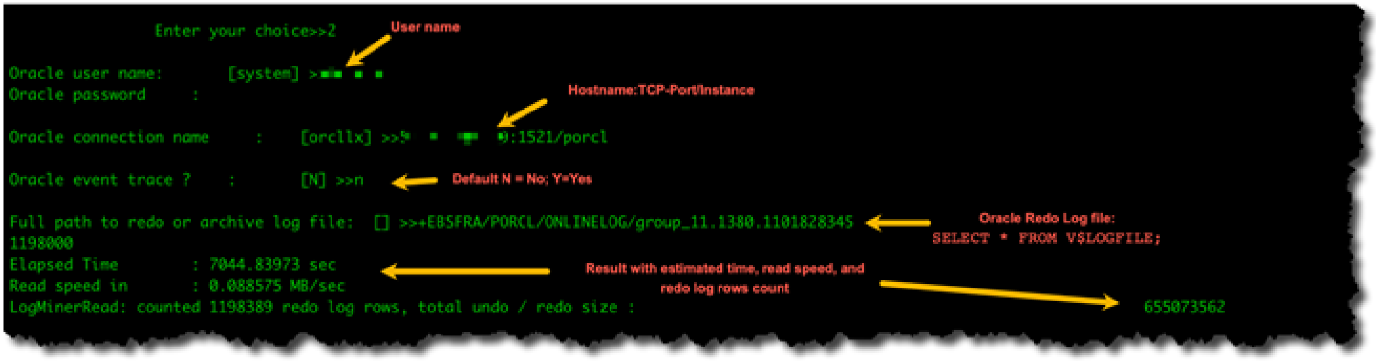
   ```
   Oracle user name [system]:
   Oracle password:

   Oracle connection name [orcllx]:
   Connection format hostname:port/instance

   Oracle event trace? [N]:
   Default N = No or Y = Yes

   Path to redo or archive log file []:
   ```

5. Examine the output displayed for relevant read performance information. For example, the following shows output that can result from selecting option number 2, **Read using LogMiner**.



6. To exit the utility, enter **0** (zero).

Next steps

- When results show that read speed is below an acceptable threshold, run the [Oracle diagnostic support script]() on the endpoint, review Wait Time, Load Profile, and IO Profile sections. Then adjust any abnormal configuration that might improve read performance. For example, if your redo log files are up to 2 GB, try increasing LOG_BUFFER to 200 MB to help improve performance.
- Review [AWS DMS Best Practices]() to make sure your DMS replication instance, task, and endpoints are configured optimally.

Failed to get LOB data

LOB (Large Object) lookup failures in AWS DMS occur under specific circumstances during data migration processes. During the full load phase, AWS DMS employs the lookup method for LOB data migration when the task is configured for FULL LOB mode. Notably, during the CDC (Change Data Capture) phase, AWS DMS consistently uses the Lookup method regardless of LOB settings.

AWS DMS first replicates rows without the LOB column, retrieves LOB data using a **SELECT** command, and executes an **UPDATE** command to replicate the LOB field on the target. This sequential INSERT and UPDATE operation characterizes the LOOKUP behavior. The LOB lookup during the CDC phase is not universally applicable across all database engines and depending on data size the tasks may replicate inline rows along with column data.

LOB Lookup process failure is a common issue that can occur during migration displaying the "Failed to get lob data, going to set to null" error message. During this failure the partial table data on the target, particularly the LOB columns appear as NULL values. Various factors can trigger these failures:

- Source row deletion occurring before DMS completes the Lookup process
- Intermittent connectivity issues disrupting lookup threads
- DMS lookup queries entering a waiting state due to source table locking scenarios

To address these LOB Lookup failures, you can do the following:

- Implement Limited LOB settings during the full load phase to eliminate lookup behavior and enhance performance.
- Reload affected tables when encountering lookup failure messages and partial data on the target.
- For issues occuring due to intermittent network or source database availability problems, restart the task to resolve all table data inconsistencies.

These steps to handling LOB Lookup failures ensures more reliable data migration and helps maintain data integrity throughout the process.

Troubleshooting issues with MySQL

Following, you can learn about troubleshooting issues specific to using AWS DMS with MySQL databases.

Topics

- Error: Codepage 1252 to UTF8 [120112] a field data conversion failed
- Indexes, Foreign Keys, or Cascade Updates or Deletes Not Migrated

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`

For information about setting MySQL system variables, see Server System Variables in the MySQL documentation.

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 open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
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 MySQL-Compatible Edition, or MariaDB endpoint.

To disable foreign keys on a target MySQL-compatible endpoint

1. Sign in to the AWS Management Console and open the AWS DMS console at https://console.aws.amazon.com/dms/v2/.
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.

"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 up 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 precreate 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)
```

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 endpoint setting.

For example, the following endpoint setting 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
```

Indexes, Foreign Keys, or Cascade Updates or Deletes Not Migrated

AWS DMS does not support migrating secondary objects such as indexes and foreign keys. To replicate changes made to child tables from a cascade update or delete operation, you need to have the triggering foreign key constraint active on the target table. To work around this limitation, create the foreign key manually on the target table. Then, either create a single task for full-load and CDC, or two separate tasks for full load and CDC, as described following:

Create a single task supporting full load and CDC

This procedure describes how to migrate foreign keys and indexes using a single task for full load and CDC.

Create a full load and CDC task

1. Manually create the tables with foreign keys and indexes on the target to match the source tables.

2. Add the following ECA to the target AWS DMS endpoint:

```
Initstmt=SET FOREIGN_KEY_CHECKS=0;
```

3. Create the AWS DMS task with `TargetTablePrepMode` set to `DO_NOTHING`.
4. Set the `Stop task after full load completes` setting to `StopTaskCachedChangesApplied`.
5. Start the task. AWS DMS stops the task automatically after it completes the full load and applies any cached changes.
6. Remove the `SET FOREIGN_KEY_CHECKS` ECA you added previously.
7. Resume the task. The task enters the CDC phase and applies ongoing changes from the source database to the target.

Create full load and CDC tasks separately

These procedures describe how to migrate foreign keys and indexes using separate tasks for full load and CDC.

Create a full load task

1. Manually create the tables with foreign keys and indexes on the target to match the source tables.
2. Add the following ECA to the target AWS DMS endpoint:

```
Initstmt=SET FOREIGN_KEY_CHECKS=0;
```

3. Create the AWS DMS task with the `TargetTablePrepMode` parameter set to `DO_NOTHING` and `EnableValidation` set to `FALSE`.
4. Start the task. AWS DMS stops the task automatically after it completes the full load and applies any cached changes.
5. Once the task completes, note the full load task start time in UTC, or binary log file name and position, to start the CDC only task. Refer to the logs to get the timestamp in UTC from the initial full load start time.

Create a CDC-only task

1. Remove the `SET FOREIGN_KEY_CHECKS` ECA you set previously.

2. Create the CDC-only task with the start position set to the full load start time noted in the previous step. Alternatively, you can use the binary log position recorded in the previous step. Set the `TargetTablePrepMode` setting to `DO_NOTHING`. Enable data validation by setting the `EnableValidation` setting to `TRUE` if needed.

3. Start the CDC-only task, and monitor the logs for errors.

> **Note**
>
> This workaround only applies to a MySQL to MySQL migration. You can't use this method with the Batch Apply feature, because Batch Apply requires that target tables don't have active foreign keys.

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

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 **Endpoint setting** statement.

```
"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 **Endpoint setting** statement. The **Endpoint setting** parameter is available in the tab of the source 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.

ReplicationSlotDiskUsage increases and restart_lsn stops moving forward during long transactions, such as ETL workloads

When logical replication is enabled, the maximum number of changes kept in memory per transaction is 4MB. After that, changes are spilled to disk. As a result `ReplicationSlotDiskUsage` increases, and `restart_lsn` doesn't advance until the transaction is completed/aborted and the rollback finishes. Since it is a long transaction, it can take a long time to rollback.

So, avoid long running transactions when logical replication is enabled. Instead, try to break the transaction into several smaller transactions.

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](#).

Sources that support views include the following.

- Oracle
- Microsoft SQL Server
- MySQL
- PostgreSQL
- IBM Db2 LUW
- SAP Adaptive Server Enterprise (ASE)

Invalid byte sequence for encoding "UTF8"

Data migration from Oracle to PostgreSQL using AWS DMS presents unique challenges due to character set encoding differences between the two databases. A significant issue arises from

Oracle's AL32UTF8 character set, which fully supports 4-byte characters, while PostgreSQL's UTF8 character set implementation lacks this capability. This disparity often leads to migration failures, particularly when dealing with tables or columns in the Oracle source that contain 4-byte characters.

During migration attempts, you may encounter error messages in both DMS task logs and PostgreSQL target database logs indicating problems with invalid UTF8 byte sequences. A typical error message "ERROR: invalid byte sequence for encoding "UTF8": 0xed 0xb0 0x86" is displayed. To resolve this issue, AWS DMS provides a solution through the "`ReplaceChars`" settings. It automatically sibtitutes or eliminates invalid characters during the migration process. This approach effectively prevents encoding-related errors without requiring modifications to the source data.

For more information, see *Character set validation and replacement* bullet point in Character substitution task settings topic.

Troubleshooting issues with Microsoft SQL Server

Following, you can learn about troubleshooting issues specific to using AWS DMS with Microsoft SQL Server databases.

Topics

Ongoing replication fails after RDS for SQL Server fails over to secondary

If a source SQL Server instance fails over to the secondary, AWS DMS ongoing replication keeps trying to connect, and continues replicating once the source is back online. However, for RDS for

SQL Server MAZ instances, under certain circumstances the secondary database owner can be set to NT AUTHORITY\SYSTEM. After a failover, this causes the DMS task to fail with the following error:

```
[SOURCE_CAPTURE  ]E:  RetCode: SQL_ERROR  SqlState: 42000 NativeError: 33009 Message:
                [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]The database
 owner SID recorded in the master
                database differs from the database owner SID recorded in database
 'rdsadmin'. You should correct
                this situation by resetting the owner of database 'rdsadmin' using the
 ALTER AUTHORIZATION statement.
                Line: 1 Column: -1 [1022502]  (ar_odbc_stmt.c:5035)
```

To fix this, follow the steps in Changing the db_owner to the rdsa account for your database, and then resume your DMS task.

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.

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 do not 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 does not 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.

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 does not have a clustered index, add a clustered index to the table (you can drop it later if you want).

Error : CDC task failed with a Bad Envelope, invalid data context/LCX code while processing a transaction

The 'Bad Envelope' error occurs when AWS DMS is unable to validate specific event types in CDC phase replication during the validation process. This error can commonly occur when resuming tasks from a specific timestamp that falls in the middle of a transaction. In such cases, the task might read a commit event without finding its corresponding 'start transaction' event, leading to an invalid transaction context, and triggering the 'Bad Envelope' error.

To resolve this issue, you must modify the SQL Server source endpoint configuration by setting the `ignoreTxnCtxValidityCheck` parameter to `true` in the Extra Connection Attribute section, before resuming the task. If the error persists after implementing this solution, submit an AWS support ticket.

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

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](#).

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'

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/7XO4FJHCVON7TYTLQ6RX3CQHDU/data_files/4/LOAD000001DF.csv" into
 table
`VOSPUSER`.`SANDBOX_SRC_FILE` CHARACTER SET UTF8 fields terminated by ','
enclosed by '"' lines terminated by '\n'( `SANDBOX_SRC_FILE_ID`,`SANDBOX_ID`,
`FILENAME`,`LOCAL_PATH`,`LINES_OF_CODE`,`INSERT_TS`,`MODIFIED_TS`,`MODIFIED_BY`,
`RECORD_VER`,`REF_GUID`,`PLATFORM_GENERATED`,`ANALYSIS_TYPE`,`SANITIZED`,`DYN_TYPE`,
`CRAWL_STATUS`,`ORIG_EXEC_UNIT_VER_ID` ) ; (provider_syntax_manager.c:2561)
```

Troubleshooting issues with SAP ASE

Following, you can learn about troubleshooting issues specific to using AWS DMS with SAP ASE databases.

Error: LOB columns have NULL values when source has a composite unique index with NULL values

When using SAP ASE as a source with tables configured with a composite unique index that allows NULL values, LOB values might not migrate during ongoing replication. This behavior is usually the result of ANSI_NULL set to 1 by default on the DMS replication instance client.

To ensure that LOB fields migrate correctly, include the Endpoint setting `'AnsiNull=0'` to the AWS DMS source endpoint for the task.

Troubleshooting issues with IBM Db2

Following, you can learn about troubleshooting issues specific to using AWS DMS with IBM Db2 databases.

Error: Resume from timestamp is not supported Task

For ongoing replication (CDC), if you plan to start replication from a specific timestamp, set the extra connection attribute `StartFromContext` to the required timestamp. For more information, see Extra Connection Attributes (ECAs) when using Db2 LUW. Setting `StartFromContext` to the required timestamp prevents the following issue:

```
Last Error Resume from timestamp is not supported Task error notification received
 from
subtask 0, thread 0 [reptask/replicationtask.c:2822] [1020455] 'Start from timestamp'
 was blocked to prevent Replicate from
scanning the log (to find the timestamp). When using IBM DB2 for LUW, 'Start from
 timestamp' is only supported if an actual
change was captured by this Replicate task earlier to the specified timestamp.
```

Table suspended a table with error "Failed to build 'where' statement"

In DMS when you try to update a record in a table that does not have a primary key, the system is unable to construct a WHERE condition and displays the following error:

```
[TARGET_APPLY ]E: Failed to build 'where' statement
```

This can occur due to multiple known issues or limitations, They are:

- Primary key column is removed using the `remove-column` transformation rule.
- Table structure has a mismatch between your source and target databases, that is primary key column existing on source but not on the target or may have been removed.
- Known limitations or missing prerequisites:
  - Supplemental logging not properly enabled on Oracle tables.
  - Oracle table created with long object names (over 30 bytes) , Hence object names could be table or column name.
  - Replication from Oracle application containers PDB.

Troubleshooting latency issues in AWS Database Migration Service

This section provides an overview of the common causes for AWS DMS task latency during the ongoing replication phase (CDC). AWS DMS replicates data asynchronously. Latency is the delay between when a change was committed on the source and when the change was replicated to the target. Latency can be caused due to misconfiguration of replication components, such as the following:

- Source endpoint or data source
- Target endpoint or data source
- Replication instances
- The network between these components

We recommend that you use a test migration as a proof of concept to gather information about your replication. You can then use this information for tuning your replication configuration to minimize latency. For information about running a proof of concept migration, see Running a proof of concept.

Topics

Types of CDC latency

This section contains types of replication latency that may occur during CDC.

Source latency

The delay, in seconds, between the commit time of the last event captured from the source endpoint, and the current system timestamp of the replication instance. You can monitor the latency between the data source and your replication instance using the `CDCLatencySource` CloudWatch metric. A high `CDCLatencySource` metric indicates that the process of capturing changes from the source is delayed. For example, if your application commits an insert to the source at 10:00, and AWS DMS consumes the change at 10:02, the `CDCLatencySource` metric is 120 seconds.

For information about CloudWatch metrics for AWS DMS, see [Replication task metrics](#).

Target latency

The delay, in seconds, between the commit time on the source of the first event waiting to commit to the target, and the current timestamp of the DMS replication instance. You can monitor the latency between commits on the data source and your data target using the `CDCLatencyTarget` CloudWatch metric. This means that `CDCLatencyTarget` includes any delays in reading from the source. As a result, `CDCLatencyTarget` is always greater than or equal to `CDCLatencySource`.

For example, if your application commits an insert to the source at 10:00, and AWS DMS consumes it at 10:02 and writes it to the target at 10:05, the `CDCLatencyTarget` metric is 300 seconds.

Common causes of CDC latency

This section contains causes of latency that your replication may experience during CDC.

Topics

- Endpoint resources
- Replication instance resources
- Network speed and bandwidth
- DMS configuration
- Replication scenarios

Endpoint resources

The following factors significantly affect replication performance and latency:

- Source and target database configurations
- Instance size
- Under-provisioned or misconfigured source or target data stores

To identify causes for latency caused by endpoint issues for AWS-hosted sources and targets, monitor the following CloudWatch metrics:

- `FreeMemory`
- `CPUUtilization`
- Throughput and I/O metrics, such as `WriteIOPS`, `WriteThroughput`, or `ReadLatency`
- Transaction volume metrics such as `CDCIncomingChanges`.

For information about monitoring CloudWatch metrics, see AWS Database Migration Service metrics.

Replication instance resources

Replication instance resources are critical for replication, and you should make sure that there are no resource bottlenecks, as they can lead to both source and target latency.

To identify resource bottlenecks for your replication instance, verify the following:

- Critical CloudWatch metrics such as CPU, Memory, I/O per second, and storage are not experiencing spikes or consistenly high values.
- Your replication instance is sized appropriately for your workload. For information about determining the correct size of a replication instance, see Selecting the best size for a replication instance.

Network speed and bandwidth

Network bandwith is a factor that affects data transmission. To analyze the network performance of your replication, do one of the following:

- Check the `ReadThroughput` and `WriteThroughput` metrics at the instance level. For information about monitoring CloudWatch metrics, see AWS Database Migration Service metrics.
- Use the AWS DMS Diagnostic Support AMI. If the Diagnostic Support AMI is not available in your region, you can download it from any supported region and copy it to your region to perform your network analysis. For information about the Diagnostic Support AMI, see Working with the AWS DMS diagnostic support AMI.

CDC in AWS DMS is single-threaded to ensure data consistency. As a result, you can determine the data volume your network can support by calculating your single-threaded data transfer rate. For example, if your task connects to its source using a 100 Mbps (megabits per second) network, your replication has a theoretical maximum bandwidth allocation of 12.5 MBps (megabytes per second). This is equal to 45 gigabits per hour. If the rate of transaction log generation on the source is greater than 45 gigabits per hour, this means that the task has CDC latency. For a 100 MBps network, these rates are theoretical maximums; other factors such as network traffic and resource overhead on the source and target reduce the actual available bandwidth.

DMS configuration

This section contains recommended replication configurations that can help reduce latency.

- **Endpoint settings**: Your source and target endpoint settings can cause your replication instance to suffer poor performance. Endpoint settings that turn on resource-intensive features will impact performance. For example, for an Oracle endpoint, disabling LogMiner and using Binary Reader improves performance, since LogMiner is resource-intensive. The following endpoing setting improves performance for an Oracle endpoint:

  ```
  useLogminerReader=N;useBfile=Y
  ```

  For more information about endpoint settings, see the documentation for your source and target endpoint engine in the Working with AWS DMS endpoints topic.
- **Task settings**: Some task settings for your particular replication scenario can cause your replication instance to suffer poor performance. For example, AWS DMS uses transactional apply mode by default (`BatchApplyEnabled=false`) for CDC for all endpoints except for Amazon Redshift. However, for sources with a large number of changes, setting `BatchApplyEnabled` to `true` may improve performance.

For more information about task settings, see Specifying task settings for AWS Database Migration Service tasks.

- **Start Position of a CDC only task**: Starting a CDC-only task from a position or timestamp in the past will start the task with increased CDC source latency. Depending on the volume of changes on the source, task latency will take time to subside.
- **LOB settings**: Large Object data types can hinder replication performance due to the way AWS DMS replicates large binary data. For more information, see the following topics:
  - Setting LOB support for source databases in an AWS DMS task
  - Migrating large binary objects (LOBs).

Replication scenarios

This section describes specific replication scenarios and how they may affect latency.

Topics

- Stopping a task for an extended period of time
- Cached changes
- Cross-region replication

Stopping a task for an extended period of time

When you stop a task, AWS DMS saves the position of the last transaction log that was read from the source. When you resume the task, DMS tries to continue reading from the same transaction log position. Resuming a task after several hours or days causes CDC source latency to increase until DMS finishes consuming the transaction backlog.

Cached changes

Cached changes are changes that your application writes to the data source while AWS DMS runs the full-load replication phase. DMS doesn't apply these changes until the full-load phase completes and the CDC phase starts. For a source with large number of transactions, cached changes take longer to apply, so source latency increases when the CDC phase starts. We recommend that you run the full-load phase when transaction volumes are low to minimize the number of cached changes.

Cross-region replication

Locating your DMS endpoints or your replication instance in different AWS regions increases network latency. This increases replication latency. For best performance, locate your source endpoint, target endpoint, and replication instance in the same AWS region.

Troubleshooting latency issues

This section contains troubleshooting steps for replication latency.

To troubleshoot latency, do the following:

- First, determine the type and amount of latency for the task. Check the task's Table Statistics section from the DMS console or CLI. If the counters are changing, then data transmission is in progress. Check the `CDCLatencySource` and`CDCLatencyTarget` metrics together to determine if there's a bottleneck during CDC.
- If high `CDCLatencySource` or `CDCLatencyTarget` metrics indicate a bottleneck in your replication, check the following:
  - If `CDCLatencySource` is high and `CDCLatencyTarget` is equal to `CDCLatencySource`, this indicates that there is a bottleneck in your source endpoint, and AWS DMS is writing data to the target smoothly. See Troubleshooting source latency issues following.
  - If `CDCLatencySource` is low and `CDCLatencyTarget` is high, this indicates that there is a bottleneck in your target endpoint, and AWS DMS is reading data from the source smoothly. See Troubleshooting target latency issues following.
  - If `CDCLatencySource` is high and `CDCLatencyTarget` is significantly higher than `CDCLatencySource`, this indicates bottlenecks on both source reads and target writes. Investigate source latency first, and then investigate target latency.

For information about monitoring DMS task metrics, see Monitoring AWS DMS tasks.

Troubleshooting source latency issues

The following topics describe replication scenarios specific to source endpoint types.

Topics

- Oracle Endpoint Troubleshooting
- MySQL Endpoint Troubleshooting
- PostgreSQL Endpoint Troubleshooting

- SQL Server Endpoint Troubleshooting

Oracle Endpoint Troubleshooting

This section contains replication scenarios specific to Oracle.

Source reading paused

AWS DMS pauses reading from an Oracle source in the following scenarios. This behavior is by design. You can investigate the causes for this using the task log. Look for messages similar to the following in the task log. For information about working with the task log, see Viewing and managing AWS DMS task logs.

- **SORTER message**: This indicates that DMS is caching transactions on the replication instance. For more information, see SORTER message in task log following.
- **Debug task logs**: If DMS interrupts the read process, your task repeatedly writes the following message to the debug task logs, without a change to the context field or timestamp:
  - **Binary reader**:

```
[SOURCE_CAPTURE  ]T:  Produce CTI event: 
context '00000020.f23ec6e5.00000002.000a.00.0000:190805.3477731.16' 
xid [00000000001e0018] timestamp '2021-07-19 06:57:55' 
thread 2  (oradcdc_oralog.c:817)
```

  - **Logminer**:

```
[SOURCE_CAPTURE  ]T:  Produce INSERT event: 
object id 1309826 context
 '000000000F2CECAA010000010005A8F500000275016C0000000000000F2CEC58' 
xid [000014e06411d996] timestamp '2021-08-12 09:20:32' thread 1
 (oracdc_reader.c:2269)
```

- AWS DMS logs the following message for every new redo or archived log operation.

```
00007298: 2021-08-13T22:00:34 [SOURCE_CAPTURE ]I: Start processing archived
 Redo log sequence 14850 thread 2 name XXXXX/XXXXX/ARCHIVELOG/2021_08_14/
thread_2_seq_14850.22977.1080547209 (oradcdc_redo.c:754)
```

  If the source has new redo or archived log operations, and AWS DMS is not writing these messages to the log, this means that the task is not processing events.

High redo generation

If your task is processing redo or archived logs, but the source latency remains high, try to identify the redo log generation rate and generation patterns. If you have a high level of redo log generation, this increases source latency, because your task reads all of the redo and archive logs in order to fetch changes related to the replicated tables.

To determine the redo generation rate, use the following queries.

- Per-day redo generation rate:

```
select trunc(COMPLETION_TIME,'DD') Day, thread#,
round(sum(BLOCKS*BLOCK_SIZE)/1024/1024/1024) GB,
count(*) Archives_Generated from v$archived_log
where completion_time > sysdate- 1
group by trunc(COMPLETION_TIME,'DD'),thread# order by 1;
```

- Per-hour redo generation rate:

```
Alter session set nls_date_format = 'DD-MON-YYYY HH24:MI:SS';
select trunc(COMPLETION_TIME,'HH') Hour,thread# ,
round(sum(BLOCKS*BLOCK_SIZE)/1024/1024) "REDO PER HOUR (MB)",
count(*) Archives from v$archived_log
where completion_time > sysdate- 1
group by trunc(COMPLETION_TIME,'HH'),thread#  order by 1 ;
```

To troubleshoot latency in this scenario, check the following:

- Check the network bandwidth and single-thread performance of your replication to ensure that your underlying network can support the source redo generation rate. For information about how network bandwidth can affect replication performance, see Network speed and bandwidth prior.
- Check if you set up supplemental logging correctly. Avoid extra logging on the source, such as enabling logging on all columns of a table. For information about setting up supplemental logging, see Setting up supplemental logging.
- Verify that you are using the correct API to read the redo or archved logs. You can use either Oracle LogMiner or AWS DMS Binary Reader. While LogMiner reads the online redo logs and archived redo log files, Binary Reader reads and parses the raw redo log files directly. As a result, Binary Reader is more performant. We recommend that you use Binary Reader if your redo log

generation is more than 10 GB/ hour. For more information, see Using Oracle LogMiner or AWS DMS Binary Reader for CDC.

- Check if you set `ArchivedLogsOnly` to `Y`. If this endpoint setting is set, AWS DMS reads from the archived redo logs. This increases source latency, because AWS DMS waits for the online redo log to be archived before reading. For more information, see ArchivedLogsOnly.
- If your Oracle source uses Automatic Storage Management (ASM), see Storing REDO on Oracle ASM when using Oracle as a source for AWS DMS for information about how to properly configure your data store. You may also be able to optimize reading performance further by using the `asmUsePLSQLArray` extra connection attrribute (ECA). For information about using `asmUsePLSQLArray`, see Endpoint settings when using Oracle as a source for AWS DMS.

MySQL Endpoint Troubleshooting

This section contains replication scenarios specific to MySQL. AWS DMS scans the MySQL binary log periodically to replicate changes. This process can increase latency in the following scenarios:

Topics

Long-running transaction on source

Since MySQL only writes committed transactions to the binary log, long-running transactions cause latency spikes proportional to the query run time.

To identify long-running transactions, use the following query, or use the slow query log:

```
SHOW FULL PROCESSLIST;
```

For information about using the slow query log, see The Slow Query Log in the MySQL documentation.

To avoid latency spikes from long-running transactions, restructure your source transactions to either reduce the query run time or increase your commit frequency.

High workload on source

Because DMS CDC is single-threaded, a large number of transactions can increase source latency. To identify if source latency is due to a heavy workload, compare the number and size of the binary logs generated during the latency period to the logs generated before the latency. To check the binary logs, and DMS CDC thread status, use the following queries:

```
SHOW BINARY LOGS;
SHOW PROCESSLIST;
```

For more information about CDC binary log dump thread states, see Replication Source Thread States .

You can determine the latency by comparing the latest binary log position generated on the source with the event DMS is currently processing. To identify the latest binary log on the source, do the following:

- Enable debug logs on the SOURCE_CAPTURE component.
- Retrieve the DMS processing binary log and position details from the the task debug logs.
- Use the following query to identify the latest binary log on the source:

  ```
  SHOW MASTER STATUS;
  ```

To further optimize performance, tune the `EventsPollInterval`. By default, DMS polls the binary log every 5 seconds, but you may improve performance by reducing this value. For more information about the `EventsPollInterval` setting, see Endpoint settings when using MySQL as a source for AWS DMS.

Binary log contention

When migrating multiple tables with a large amount of data, we recommend splitting tables into separate tasks for MySQL 5.7.2 or later. In MySQL versions 5.7.2 and later, the master dump thread creates fewer lock contentions and improves throughput. As a result, the dump thread no longer locks the binary log whenever it reads an event. This means that multiple dump threads can read the binary log file concurrently. This also means that dump threads can read the binary log while clients write to it. For more information about dump threads, see Replication Threads and the MySQL 5.7.2 release notes.

To improve replication performance for MySQL sources versions prior to 5.7.2, try consolidating tasks with CDC components.

PostgreSQL Endpoint Troubleshooting

This section contains replication scenarios specific to PostgreSQL.

Topics

Long-running transaction on source

When there are long-running transactions in the source database, such as a few thousand inserts in a single transaction, the DMS CDC event and transaction counters do not increase until the transaction is complete. This delay can cause latency issues that you can measure using the `CDCLatencyTarget` metric.

To review long-running transactions, do one of the following:

- Use the `pg_replication_slots` view. If the `restart_lsn` value isn't updating, it is likely that PostgreSQL is unable to release Write Ahead Logs (WALs) due to long-running active transactions. For information about the `pg_replication_slots` view, see pg_replication_slots in the PostgreSQL 15.4 Documentation.
- Use the following query to return a list of all active queries in the database, along with related information:

```
SELECT pid, age(clock_timestamp(), query_start), usename, query
FROM pg_stat_activity WHERE query != '<IDLE>'
AND query NOT ILIKE '%pg_stat_activity%'
ORDER BY query_start desc;
```

  In the query results, the `age` field shows the active duration of each query, which you can use to identify long-running queries.

High workload on source

If your source PostgreSQL has a high workload, check the following to reduce latency:

- You may experience high latency when using the `test_decoding` plugin while migrating a subset of tables from the source database with a high transactions per second (TPS) value. This is because the `test_decoding` plugin sends all database changes to the replication instance which DMS then filters, based on the task's table mapping. Events for tables that aren't part of the task's table mapping can increase source latency.
- Check TPS throughput using one of the following methods.
  - For Aurora PostgreSQL sources, use the `CommitThroughput` CloudWatch metric.
  - For PostgreSQL running on Amazon RDS or on-premises, use the following query using a PSQL client version 11 or higher (Press **`enter`** during the query to advance the results):

```
SELECT SUM(xact_commit)::numeric as temp_num_tx_ini FROM pg_stat_database; \gset
select pg_sleep(60);
SELECT SUM(xact_commit)::numeric as temp_num_tx_final FROM pg_stat_database; \gset
select (:temp_num_tx_final - :temp_num_tx_ini)/ 60.0 as "Transactions Per Second";
```

- To reduce latency when using the `test_decoding` plugin, consider using the `pglogical` plugin instead. Unlike the `test_decoding` plugin, the `pglogical` plugin filters write ahead log (WAL) changes at the source, and only sends relevant changes to the replication instance. For information about using the `pglogical` plugin with AWS DMS, see Configuring the pglogical plugin.

High network throughput

Your replication may have high network bandwidth use when using the `test_decoding` plugin, especially during high-volume transactions. This is because the `test_decoding` plugin processes changes, and converts them into a human-readable format that is larger than the original binary format.

To improve performance, consider using the `pglogical` plugin instead, which is a binary plugin. Unlike the `test_decoding` plugin, the `pglogical` plugin generates binary format output, resulting in compressed write ahead log (WAL) stream changes.

Spill files in Aurora PostgreSQL

In PostgreSQL version 13 and higher, the `logical_decoding_work_mem` parameter determines the memory allocation for decoding and streaming. For more information about the `logical_decoding_work_mem` parameter, see [Resource Consumption in PostgreSQL] in the [PostgreSQL 13.13 Documentation].

Logical replication accumulates changes for all transactions in memory until those transactions commit. If the amount of data stored across all transactions exceeds the amount specified by the database parameter `logical_decoding_work_mem`, then DMS spills the transaction data to disk to release memory for new decoding data.

Long running transactions, or many subtransactions, may result in DMS consuming increased logical decoding memory. This increased memory use results in DMS creating spill files on disk, which causes high source latency during replication.

To reduce the impact of an increase in the source workload, do the following:

- Reduce long-running transactions.
- Reduce the number of sub-transactions.
- Avoid performing operations that generate a large burst of log records, such as deleting or updating an entire table in a single transaction. Perform operations in smaller batches instead.

You can use the following CloudWatch metrics to monitor the workload on the source:

- `TransactionLogsDiskUsage`: The number of bytes currently occupied by the logical WAL. This value increases monotonically if logical replication slots are unable to keep up with the pace of new writes, or if any long running transactions prevent garbage collection of older files.
- `ReplicationSlotDiskUsage`: The amount of disk space the logical replication slots currently use.

You can reduce source latency by tuning the `logical_decoding_work_mem` parameter. The default value for this parameter is 64 MB. This parameter limits the amount of memory used by each logical streaming replication connection. We recommend setting the `logical_decoding_work_mem` value significantly higher than the `work_mem` value to reduce the amount of decoded changes that DMS writes to disk.

We recommend that you periodically check for spill files, particularly during periods of heavy migration activity or latency. If DMS is creating a significant number of spill files, this means that logical decoding isn't operating efficiently, which can increase latency. To mitigate this, increase the `logical_decoding_work_mem` parameter value.

You can check the current transaction overflow with the `aurora_stat_file` function. For more information, see [Adjusting working memory for logical decoding] in the *Amazon Relational Database Service Developer Guide*.

SQL Server Endpoint Troubleshooting

This section contains replication scenarios specific to SQL Server. To determine what changes to replicate from SQL server AWS DMS reads the transaction logs, and runs periodic scans on the source database. Replication latency usually results from SQL Server throttling these scans because of resource constraints. It can also result from a significant increase in the number of events written to the transaction log in a short time.

Topics

- Index rebuilds
- Large transactions
- Misconfigured MS-CDC polling interval for Amazon RDS SQL Server
- Multiple CDC tasks replicating from the same source database
- Transaction log backup processing for RDS for SQL Server

Index rebuilds

When SQL Server rebuilds a large index, it uses a single transaction. This generates a lot of events, and can use up a large amount of log space if SQL Server rebuilds several indexes at once. When this happens, you can expect brief replication spikes. If your SQL Server source has sustained log spikes, check the following:

- First, check the time period of the latency spikes using either the `CDCLatencySource` and `CDCLatencySource` CloudWatch metrics, or by checking Throughput Monitoring messages in the task logs. For information about CloudWatch metrics for AWS DMS, see Replication task metrics.

- Check if the size of the active transaction logs or log backups increased during the latency spike. Also check if a maintenance job or a rebuild ran during that time. For information about checking transaction log size, see Monitor log space use in the SQL Server technical documentation.
- Verify that your maintenance plan follows SQL server best practices. For information about SQL server maintenance best practices, see Index maintenance strategy in the SQL Server technical documentation.

To fix latency issues during index rebuilds, try the following:

- Use the BULK_LOGGED recovery model for offline rebuilds to reduce the events a task has to process.
- If possible, stop the task during index rebuilds. Or, try to schedule index rebuilds during non-peak hours to mitigate the impact of a latency spike.
- Try to identify resource bottlenecks that are slowing DMS reads, such as disk latency or I/O throughput, and address them.

Large transactions

Transactions with a lot of events, or long-running transactions, cause the transaction log to grow. This causes DMS reads to take longer, resulting in latency. This is similar to the effect index rebuilds have on replication performance.

You may have difficulty identifying this issue if you're not familiar with the typical workload on the source database. To troubleshoot this issue, do the following:

- First, identify the time that latency spiked using either the ReadThroughput and WriteThroughput CloudWatch metrics, or by checking Throughput Monitoring messages in the task logs.
- Check if there are any long-running queries on the source database during the latency spike. For information about long-running queries, see Troubleshoot slow-running queries in SQL Server in the SQL Server technical documentation.
- Check if the size of the active transaction logs or the log backups has increased. For more information, see Monitor log space use in the SQL Server technical documentation.

To fix this issue, do one of the following:

- The best fix is to restructure your transactions on the application side so that they complete quickly.
- If you can't restructure your transactions, a short-term workaround is to check for resource bottlenecks such as disk waits or CPU contention. If you find bottlenecks in your source database, you can reduce latency by increasing disk, CPU, and memory resources for source database. This reduces contention for system resources, allowing DMS queries to complete faster.

Misconfigured MS-CDC polling interval for Amazon RDS SQL Server

A misconfigured polling interval setting on Amazon RDS instances can cause the transaction log to grow. This is because replication prevents log truncation. While tasks that are running might continue replicating with minimal latency, stopping and resuming tasks, or starting CDC-only tasks, can cause task failures. These are due to timeouts while scanning the large transaction log.

To troubleshoot a misconfigured polling interval, do the following:

- Check if the active transaction log size is increasing, and if log usage is close to 100 percent. For more information, see Monitor log space use in the SQL Server technical documentation.
- Check if log truncation is delayed with a `log_reuse_wait_desc value` of `REPLICATION`. For more information, see The Transaction Log (SQL Server) in the SQL Server technical documentation.

If you find issues with any of the items in the previous list, tune the MS-CDC polling interval. For information about tuning the polling interval, see Recommended settings when using RDS for SQL Server as a source for AWS DMS.

Multiple CDC tasks replicating from the same source database

During the full load phase, we recommend splitting tables across tasks to improve performance, to separate dependent tables logically, and to mitigate the impact of a task failure. However, during the CDC phase, we recommend consolidating tasks to minimize DMS scans. During the CDC phase, each DMS task scans the transaction logs for new events several times a minute. Since each task runs independently, every task scans each transaction log individually. This increases disk and CPU usage on the source SQL Server database. As a result, a large number of tasks running in parallel can cause SQL Server to throttle DMS reads, leading to increased latency.

You may have difficulty identifying this issue if muliple tasks start gradually. The most common symptom of this issue is most task scans starting to take longer. This leads to higher latency for these scans. SQL Server prioritizes a few of the task scans, so a few of the tasks show normal latency. To troubleshoot this issue, check the `CDCLatencySource` metric for all of your tasks. If some of the tasks have an increasing `CDCLatencySource`, while a few tasks have a low `CDCLatencySource`, it is likely that SQL Server is throttling your DMS reads for some of your tasks.

If SQL Server is throttling your task reads during CDC, consolidate your tasks to minimize the number of DMS scans. The maximum number of tasks that can connect to your source database without creating contention depends on factors such as the source database capacity, the rate of transaction log growth, or the number of tables. To determine the ideal number of tasks for your replication scenario, test replication in a test environment similar to your production environment.

Transaction log backup processing for RDS for SQL Server

AWS DMS 3.5.3 and above support replicating from RDS for SQL Server log backups. Replicating events from the backup logs on RDS instances is slower than replicating events from the active transaction log. This is because DMS requests access to backups serially to ensure that it maintains the transaction sequence, and to minimize the risk of the Amazon RDS instance storage filling up. Moreover, at the Amazon RDS end, the time taken to make the backups available to DMS varies depending on the size of the log backup, and the load on the RDS for SQL Server instance.

Because of these constraints, we recommend that you set the ECA `ActivateSafeguard` to `true`. This ensures that transactions are not backed up while the DMS task is reading from the active transaction log. This setting also prevents Amazon RDS archiving transactions in the active log when DMS is reading transactions from the backup, thereby eliminating the possibility that DMS cannot catch up to the active log. Note that this may cause the active log size to grow while the task is catching up. Ensure that your instance has enough storage to keep the instance from running out of space.

For a CDC-only task replicating from RDS for SQL Server sources, use the use of native CDC start position over native CDC start time if possible. This is because DMS relies on system tables to identify the starting point for the native start position, rather than scanning individual log backups when you specify a native start time.

Troubleshooting target latency issues

This section contains scenarios that can contribute to target latency.

Topics

- [Indexing issues]
- [SORTER message in task log]
- [Database locking]
- [Slow LOB lookups]
- [Multi-AZ, audit logging and backups]

Indexing issues

During the CDC phase, AWS DMS replicates changes on the source by executing DML statements (insert, update, and delete) on the target. For heterogenous migrations using DMS, differences in index optimizations on the source and target can cause writes to the target to take longer. This results in target latency and performance issues.

To troubleshoot indexing issues, do the following. The procedures for these steps vary for different database engines.

- Monitor the query time for your target database. Comparing the query execution time on the target and source can indicate which indexes need optimization.
- Enable logging for slow-running queries.

To fix indexing issues for long-running replications, do the following:

- Tune the indexes on your source and target databases so that the query execution time is similar on the source and the target.
- Compare the secondary indexes used in DML queries for the source and the target. Make sure that DML performance on the target is comparable to or better than the source DML performance.

Note that the procedure for optimizing indexes is specific to your database engine. There is no DMS feature for tuning source and target indexes.

SORTER message in task log

If a target endpoint can't keep up with the volume of changes that AWS DMS writes to it, the task caches the changes on the replication instance. If the cache grows larger than an internal threshold, the task stops reading further changes from the source. DMS does this to prevent the replication instance from running out of storage, or the task being stuck while reading a large volume of pending events.

To troubleshoot this issue, check the CloudWatch logs for a message similar to either of the following:

```
[SORTER ]I: Reading from source is paused. Total disk usage exceeded the limit 90%
 (sorter_transaction.c:110)
[SORTER ]I: Reading from source is paused. Total storage used by swap files exceeded
 the limit 1048576000 bytes  (sorter_transaction.c:110)
```

If your logs contain a message similar to the first message, disable any trace logging for the task, and increase the replication instance storage. For information about increasing replication instance storage, see Modifying a replication instance.

If your logs contain a message similar to the second message, do the following:

- Move tables with numerous transactions or long running DML operations to a separate task, if they don't have any dependencies on other tables in the task.
- Increase the `MemoryLimitTotal` and `MemoryKeepTime` settings to hold the transaction for a longer duration in memory. This won't help if the latency is sustained, but it can help keep latency down during short bursts of transactional volume. For information about these task settings, see Change processing tuning settings.
- Evaluate if you can use batch apply for your transaction by setting `BatchApplyEnabled` to `true`. For information about the `BatchApplyEnabled` setting, see Target metadata task settings.

Database locking

If an application accesses a database that AWS DMS is using as a replication target, the application may lock a table that DMS is trying to access. This creates a lock contention. Since DMS writes changes to the target database in the order they occurred on the source, delays to writing to one table due to lock contentions create delays to writing to all tables.

To troubleshoot this issue, query the target database to check if a lock contention is blocking DMS write transactions. If the target database is blocking DMS write transactions, do one or more of the following:

- Restructure your queries to commit changes more frequently.

- Modify your lock timeout settings.
- Partition your tables to minimize lock contentions.

Note that the procedure for optimizing lock contentions is specific to your database engine. There is no DMS feature for tuning lock contentions.

Slow LOB lookups

When AWS DMS replicates a large object (LOB) column, it performs a lookup on the source just before writing changes to the target. This lookup normally doesn't cause any latency on the target, but if the source database delays the lookup due to locking, target latency may spike.

This issue is normally difficult to diagnose. To troubleshoot this issue, enable detailed debugging on the task logs, and compare the timestamps of the DMS LOB lookup calls. For information about enabling detailed debugging, see [Viewing and managing AWS DMS task logs](#).

To fix this issue, try the following:

- Improve the SELECT query performance on the source database.
- Tune the DMS LOB settings. For information about tuning the LOB settings, see [Migrating large binary objects (LOBs)](#).

Multi-AZ, audit logging and backups

For Amazon RDS targets, target latency can increase during the following:

- Backups
- After enabling multiple availability zones (multi-AZ)
- After enabling database logging, such as audit or slow query logs.

These issues are normally difficult to diagnose. To troubleshoot these issues, monitor latency for periodic spikes during Amazon RDS maintenance windows or periods of heavy database loads.

To fix these issues, try the following:

- If possible, during short term migration, disable multi-AZ, backups, or logging.
- Reschedule your maintenance windows for periods of low activity.

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  dfafd0d511477c699f96c64693ad0b1547d47e74d5c5f2f2025b790b1422e3c8
ORACLE  6c41ebcfc99518cfa8a10cb2ce8943b153b2cc7049117183d0b5de3d551bc312
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 following command 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

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 SELECT on DBA_PDBS 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('DBA_ENCRYPTED_COLUMNS','script_user','SELECT');

exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_PDBS','script_user','SELECT');

exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_UTILITY','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

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 following SQL*Plus command line.

   ```
   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 following prompt, enter the name of only one of the schemas that you want to migrate.
3. At the following prompt, enter the name of the user (*`script_user`*) that you have defined to connect to the database.
4. At the following prompt, 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.

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.

Topics

Setting up minimum permissions for an on-premises SQL Server database

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;
```

Setting up minimum permissions for an Amazon RDS SQL Server database

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;
```

SQL Server Support Scripts

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

Download the awsdms_support_collector_sql_server.sql script.

> **Note**
>
> Run this SQL Server diagnostic support script on SQL Server 2014 and higher 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 following sqlcmd command line.

```
sqlcmd -Uon-prem-user -Ppassword -SDMS-SQL17AG-N1 -y 0
-iC:\Users\admin\awsdms_support_collector_sql_server.sql -oC:\Users\admin
\DMS_Support_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.

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.

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.

For information about installing the MySQL client, see Installing MySQL Shell in the MySQL documentation. For information about using the MySQL client, see Using MySQL Shell Commands in the MySQL documentation.

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

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 following `mysql` command.

```
mysql -p -h hostname -P port -u username database-name
```

2. Run the script using the following mysql `source` command.

```
source awsdms_support_collector_MySQL.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.

> **Note**
>
> - If you already have a user account with required privileges described in Diagnostic support scripts for MySQL-compatible databases, 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 following necessary permissions to access any PostgreSQL RDBMS:

- PostgreSQL 10.x or higher – 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 following SQL commands for any PostgreSQL RDBMS based on the PostgreSQL version.

To grant account permissions to run these scripts for a PostgreSQL databases version 10.x or higher

- 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

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 higher.

You can use the following procedures 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 following psql command line.

     ```
     dbname=# \i awsdms_support_collector_postgres.sql
     ```

     At the following prompt, enter the name of only one of the schemas that you want to migrate.

     At the following prompt, enter the name of the user (*script_user*) that you have defined to connect to the database.

   - Run the following script directly from the command line. 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.

Working with the AWS DMS diagnostic support AMI

If you encounter a network-related issue when working with AWS DMS, your support engineer might need more information about your network configuration. 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 a prebuilt Amazon EC2 AMI with diagnostic tools to test your AWS DMS networking environment.

The diagnostic tests installed on the Amazon machine image (AMI) include the following:

- Virtual Private Cloud (VPC)
- Network packet loss
- Network latency
- Maximum Transmission Unit (MTU) size

Topics

> **Note**
>
> If you experience performance issues with your Oracle source, you can evaluate the read performance of your Oracle redo or archive logs to find ways to improve performance. For more information, see Evaluating read performance of Oracle redo or archive logs.

Launch a new AWS DMS diagnostic Amazon EC2 instance

In this section, you launch a new Amazon EC2 instance. For information about how to launch an Amazon EC2 instance, see Get started with Amazon EC2 Linux instances tutorial in the Amazon EC2 User Guide.

Launch an Amazon EC2 instance with the following settings:

- For **Application and OS Images (Amazon Machine Image)**, search for the **DMS-DIAG-AMI** AMI. If you are logged on to the console, you can search for the AMI with this query For the AMI ID of the AWS Diagnostic AMI in your region, see AMI IDs by region following.
- For **Instance type**, we recommend you choose **t2.micro**.
- For **Network Settings**, choose the same VPC that your replication instance uses.

After the instance is active, connect to the instance. For information about connecting to an Amazon EC2 Linux instance, see Connect to your Linux instance.

Create an IAM role

If you want to run the diagnostic tests on your replication instance using the minimum required permissions, create an IAM role that uses the following permissions policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "dms:DescribeEndpoints",
                "dms:DescribeTableStatistics",
                "dms:DescribeReplicationInstances",
                "dms:DescribeReplicationTasks",
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "*"
        }
    ]
}
```

Attach the role to a new IAM user. For information about creating IAM roles, policies, and users, see the following sections in the IAM User Guide:

- Getting Started with IAM
- Creating IAM Roles
- Creating IAM Policies

Run Diagnostic Tests

After you have created an Amazon EC2 instance and connected to it, do the following to run diagnostic tests on your replication instance.

1. Configure the AWS CLI:

```
$ aws configure
```

Provide the access credentials for the AWS user account you want to use to run the diagnostic tests. Provide the Region for your VPC and replication instance.

2. Display the available AWS DMS tasks in your Region. Replace the sample Region with your Region.

```
$ dms-report -r us-east-1 -l
```

This command displays the status of your tasks.

3. Display task endpoints and settings. Replace *<DMS-Task-ARN>* with your task Amazon Resource Name (ARN).

```
$ dms-report -t <DMS-Task-ARN>
```

This command displays the endpoints and settings of your task.

4. Run diagnostic tests. Replace *<DMS-Task-ARN>* with your task ARN.

```
$ dms-report -t <DMS-Task-ARN> -n y
```

This command displays diagnostic data about your replication instance's VPC, network packet transmission, network latency, and network Maximum Transmission Unit (MTU) size.


########
#
#
AWS DMS Diagnostic
Date: 07-13-2022
#
#
aws region: us-east-2
#
#
########
==== DMS DIAG Info ====
Perform AMI Diag EC2 VPC Check
Public IP: 3.
Private IP: 172.30.0.240
Instance ID: i-04829b2beb8214602
Instance MAC: 02:58:04:b5:52:28
Instane Type: t2.micro
Instance Sec Group: -sec-group
Instance AWS Region: us-east-2
Instance VPC Id: vpc-08ba020355d8a952e
Perform Network Packet Test
==== Network Packet Check ====
1.) Check DMS EC2 MetaData service
>>>>>Result: 10 packets transmitted, 10 packets received, 0% packet loss
Looks good with no issue. <<<<<
2.) Check Source endpoint (-postgres-dev-instance-1.cucdvzaur7nk.us-east-2.rds.amazonaws.com:5432)
>>>>>Result: 10 packets transmitted, 10 packets received, 0% packet loss
Looks good with no issue. <<<<<
3.) Check Target endpoint (-instance-1.cucdvzaur7nk.us-east-2.rds.amazonaws.com:5432)
>>>>>Result: 10 packets transmitted, 10 packets received, 0% packet loss
Looks good with no issue. <<<<<
Returns Test Results and Recommendation
==== End network packet check ====
==== Network Latency Check ====
1.) Check DMS MetaData Service
Perform Network Latency Test
>>>>>Result: round-trip min/avg/max = 0.4/0.4/0.5 ms
Looks good with no issue. <<<<<
2.) Check Source endpoint (-postgres-dev-instance-1.cucdvzaur7nk.us-east-2.rds.amazonaws.com:5432)
>>>>>Result: round-trip min/avg/max = 1.0/1.1/1.2 ms
Looks good with no issue. <<<<<
3.) Check Target endpoint (-instance-1.cucdvzaur7nk.us-east-2.rds.amazonaws.com:5432)
>>>>>Result: round-trip min/avg/max = 1.4/1.4/1.5 ms
Looks good with no issue. <<<<<
Perform Network Maximum Transmission Unit (MTU) Check
==== End network latency check ====
==== Network MTU Check ====
1.) Check DMS MetaData Service
>>>>>Result: MTU setting looks good. Local MTU (9001) matches remote MTU (9001) <<<<<
2.) Check Source endpoint (-postgres-dev-instance-1.cucdvzaur7nk.us-east-2.rds.amazonaws.com:5432)
>>>>>Result: MTU setting looks good. Local MTU (9001) matches remote MTU (9001) <<<<<
3.) Check Target endpoint (-instance-1.cucdvzaur7nk.us-east-2.rds.amazonaws.com:5432)
>>>>>Result: MTU setting looks good. Local MTU (9001) matches remote MTU (9001) <<<<<
==== End network MTU check ====

Next Steps

The following sections describe troubleshooting information based on the results of the network diagnostic tests:

VPC tests

This test verifies that the diagnostic Amazon EC2 instance is in the same VPC as the replication instance. If the diagnostic Amazon EC2 instance is not in the same VPC as your replication instance, terminate it and create it again in the correct VPC. You can't change the VPC of an Amazon EC2 instance after you create it.

Network packet loss tests

This test sends 10 packets to the following endpoints and checks for packet loss:

- The AWS DMS Amazon EC2 metadata service on port 80
- The source endpoint
- The target endpoint

All packets should arrive successfully. If any packets are lost, consult with a network engineer to determine the problem and find a solution.

Network latency tests

This test sends 10 packets to the same endpoints as the previous test, and checks for packet latency. All packets should have a latency of less than 100 milliseconds. If any packets have a latency greater than 100 milliseconds, consult with a network engineer to determine the problem and find a solution.

Maximum Transmission Unit (MTU) size tests

This test detects the MTU size by using the Traceroute tool on the same endpoints as the previous test. All of the packets in the test should have the same MTU size. If any packets have a different MTU size, consult with a system specialist to determine the problem and find a solution.

AMI IDs by region

To see a list of available DMS Diagnostic AMIs available in your AWS region, run the following AWS CLI sample.

```
aws ec2 describe-images --owners 343299325021 --filters "Name=name, Values=DMS-DIAG*"
 --query "sort_by(Images, &CreationDate)[-1].[Name, ImageId, CreationDate]" --output
 text
```

If the output shows no results, it means the DMS Diagnostic AMI is not available in your AWS region. The workaround is to follow the below steps to copy the Diagnostic AMI from another region. For more information, see Copy an AMI.

- Launch an instance in the available region.
- Create the image. The image will be owned by you.
- Copy the AMI to your region, for example, Middle East (UAE) Region.
- Launch the instance in your local region.

AWS Systems Patch Manager

AWS Systems Patch Manager automates patching for operating systems and applications on your EC2 instances.

To configure the patch manager, perform the following steps:

1. Enable Systems Manager:
    a. Attach the `AmazonSSMManagedInstanceCore` IAM policy to your EC2 instance role.
    b. Ensure SSM Agent is installed (default for Amazon Linux 2, Ubuntu 20.04+ AMIs).
2. Create a patch baseline by defining rules for critical/security updates, including schedules for patch application.
3. Schedule updates for applying patches at a defined time by using maintenance windows in SSM.
4. Verify compliance by checking patching status and compliance reports in the Systems Manager.

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 Microsoft Learn
  - Why floating-point numbers may lose precision on Microsoft Learn

Topics

- Data types for AWS Database Migration Service

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.

AWS DMS data types	Description
WSTRING	A double-byte character string.
BOOLEAN	A Boolean value.
BYTE	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. The following format is supported : YYYY:MM:DD HH:MM:SS.F(9). For Amazon S3 Select and Amazon S3 Glacier Select, the DATETIME data type format is different . For more information, see the description of the `timestamp` primitive data type in Supported Data Types of the *Amazon Simple Storage Service User Guide*.
INT1	A one-byte, signed integer.
INT2	A two-byte, signed integer.
INT4	A four-byte, signed integer.
INT8	An eight-byte, signed integer.
NUMERIC	An exact numeric value with a fixed precision and scale.
REAL4	A single-precision floating-point value.
REAL8	A double-precision floating-point value.
UINT1	A one-byte, unsigned integer.

AWS DMS data types	Description
UINT2	A two-byte, unsigned integer.
UINT4	A four-byte, unsigned integer.
UINT8	An eight-byte, unsigned integer.
BLOB	Binary large object.
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 doesn't differentiate between major and minor versions when you enable **Automatic version upgrade** for your replication instance. DMS automatically upgrades the replication instance's version during the maintenance window if the version is deprecated.

Note that to upgrade your replication instance's version manually (using the API or CLI) from version 3.4.x to 3.5.x, you must set the `AllowMajorVersionUpgrade` parameter to `true`. For information about the `AllowMajorVersionUpgrade` parameter, see [ModifyReplicationInstance](#) in the DMS API documentation.

> **Note**
>
> The current default engine version for AWS DMS is 3.5.4.

The following table shows the following dates for active DMS versions:

- The version's release date
- The date after which you can't create new instances with the version
- The date when DMS automatically updates instances of that version (the EOL date)

Version	Release date	No new instance date	EOL date
3.6.1	May 15, 2025	Apr 30, 2027	June 30, 2027
3.6.0	Dec 27, 2024	Apr 27, 2026	June 27, 2026
3.5.4	Nov 15, 2024	Sep 15, 2026	Nov 15, 2026
3.5.3	May 17, 2024	Feb 28, 2026	Apr 30, 2026
3.5.2	Oct 29, 2023	Mar 30, 2025	Apr 29, 2025

Version	Release date	No new instance date	EOL date
3.5.1	Jun 30, 2023	Nov 30, 2024	Mar 15, 2025
3.4.7	May 31, 2022	Sep 30, 2024	Oct 31, 2024
3.4.6	Nov 30, 2021	Sep 30, 2024	Oct 31, 2024

> **Note**
> The AWS DMS Version 3.6.0 is currently available in Asia Pacific (Sydney): ap-southeast-2 region.

AWS Database Migration Service 3.6.1 release notes

New features in AWS DMS 3.6.1

New feature or enhancement	Description
DMS Data Resync	AWS DMS automatically fixes data inconsistencies identified through data validation between your source and target databases . The data resync feature is supported for Oracle and SQL Server as a source and Postgre SQL as a target databases.
IAM database authentic ation for MariaDB, MySQL, and PostgreSQL	Introduced support connecting to Amazon RDS and Aurora MariaDB, MySQL, and PostgreSQL endpoints through [AWS Identity and Access Management (IAM) database authentication](). With this enhancement you can use IAM to centrally manage access to your database resources, instead of managing access individually on each database endpoint.
PostgreSQL Read Replica Support for Change Data Capture (CDC) Replication	AWS DMS supports using PostgreSQL read replicas as source endpoints for Change Data Capture (CDC) replication, available with PostgreSQL version 16.x and later, starting with AWS DMS version 3.6.1. This feature allows you to leverage read replicas

New feature or enhancement	Description
	for CDC tasks. For more information, see [Using a PostgreSQL database as an AWS DMS source].
Enhanced logging for missing target columns	Enhanced logging to inform customers about missing target table columns at default logging level. This improvement changes the logging level from VERBOSE to WARNING, making column discrepancy notifications more visible without requiring detailed debug settings.
SQL Server Binary(16) to PostgreSQL UUID migration	Added support for migrating SQL Server Binary(16) data types to PostgreSQL UUID format. This enables seamless conversion of binary GUID data to native UUID types, improving data type compatibility between SQL Server and PostgreSQL endpoints.

AWS DMS version 3.6.1 includes the following resolved issues:

Issues resolved in the DMS 3.6.1

Resolved Issue	Description
Selection rules filtering with data masking issue	Table selection rule filters now work correctly with data masking transformation rules. The fix ensures proper filtering when data masking transformations are applied.
PostgreSQL target unbound numeric handling issue	PostgreSQL target endpoints can handle unbound numeric data types through optimized memory allocation. This prevents task failures when migrating tables with large numeric values.
Amazon Redshift target LOB handling issue	Redshift target endpoints now maintain LOB data integrity during high-volume parallel migrations. The improved LOB handling

Resolved Issue	Description
	prevents data corruption when using parallel apply threads.
Transformation rules metadata issue	Fixed an issue with expressions in transform ation rules that were incorrectly applied.
Metadata table drop and recreate issue	Replication tasks now resume successfully when source tables are dropped and recreated during task stoppage. Improved metadata handling and status synchronization for recreated source tables.
SQL Server `AlwaysOn` primary replica issue	Improved primary replica detection for SQL Server sources in `AlwaysOn` configurations by handling case sensitivity variations.
MySQL source TIME data type handling issue	Resolved migration issues with TIME values containing fractions and hours exceeding 24 when `loadUsingCSV` is disabled.
Data validation NUL character handling issue	Data validation now handles NUL (0x00) characters correctly without failing. Improved validation processing for data containing NUL characters.
SQL Server data validation issue	Resolved validation comparison issues with `UNIQUEIDENTIFIER` columns for SQL Server sources.
Table pre-load validation issue	AWS DMS tasks now complete validation successfully when tables are in pre-load phase during CDC replication.
Oracle source logging issue	Oracle source error logs now display the correct SQL statement during timezone query execution. Improved logging accuracy for better troubleshooting capabilities.

Resolved Issue	Description
SQL Server driver update	SQL Server driver support is updated from ODBC 17 to ODBC 18.
DB2 empty strings CDC migration issue	Resolved an issue where empty VARCHAR values from IBM DB2 LUW source database s were incorrectly captured as NULL values instead of empty strings during Change Data Capture (CDC) operations.
Oracle LOB lookup memory leak issue	Fixed memory leak in Oracle source endpoints during LOB lookups when full LOB mode is enabled. This prevents continuous memory growth and out-of-memory failures during replication tasks with LOB data.
Redshift CDC boolean column mapping issue	Redshift target endpoints now correctly map new boolean columns added during CDC when MapBooleanAsBoolean is enabled. This maintains data type consistency wit h PostgreSQL sources instead of creating varchar columns.
PostgreSQL UUID array inline LOB migration issue	PostgreSQL source endpoints now correctly migrate UUID array data types when inline LOB mode is enabled.
Data validation nullable column diff reporting issue	Data validation now correctly reports mismatch details for nullable columns.
SQL Server to Oracle validation filtering issue	Data validation now correctly formats date filter conditions for Oracle target endpoints when using SQL Server sources.

Resolved Issue	Description
DO_NOTHING mode table creation warning message	Added warning log message when AWS DMS automatically creates target tables in DO_NOTHING mode. This improves visibility and helps users identify when tables are created despite selecting a mode that implies no automatic table creation.
SQL Server BIT to PostgreSQL SMALLINT conversion issue	Fixed SQL Server BIT to PostgreSQL SMALLINT conversion during CDC operations.
S3 cross-account bucket access validation issue	Enhanced S3 bucket ownership validation to improve security when using S3 as a migration target or source.
Redshift endpoint reconnection and initialization issue	Redshift endpoints now properly reconnect after connection termination and correctly validate credentials during initialization. This prevents "Server name must be supplied" errors and task failures during recovery.
Data validation memory leak issue	Fixed memory leak in data validation. This prevents out-of-memory failures during long-running validation tasks.
SQL Server CHAR primary key validation false positive issue	Fixed false positive MISSING_TARGET errors when validating SQL Server to PostgreSQL migrations with CHAR/NCHAR primary keys containing trailing spaces.
S3 validation Athena database conflict issue	Fixed race condition when multiple tasks share S3 endpoints by isolating Athena databases per task.

AWS Database Migration Service 3.6.0 release notes

New features in AWS DMS 3.6.0

New feature or enhancement	Description
New metadata variables for transformations	Introduced two new metadata variables for transformation rules: `$AR_M_MODIFIED_SCHEMA` and `$AR_M_MODIFIED_TABLE_NAME`. These variables allow you to create more dynamic and flexible transformations that can adapt to schema or table name changes during your migration process.
Support for LOB Column Replication in UPSERT Error-Handling Mode	Introduced support for replication of Large Object (LOB) columns when using the "No record found for applying an UPDATE: Insert the missing target record" error-handling option. With this enhancement, you can now accurately replicate LOB columns, ensuring complete and accurate data replication.

AWS DMS version 3.6.0 includes the following resolved issues:

Issues resolved in the DMS 3.6.0

Resolved Issue	Description
PostgreSQL Memory Issue	Resolved excessive memory consumption when using PostgreSQL as a source. Fixed an issue where the Write-Ahead Log (WAL) slot would continuously grow, leading to degraded performance. This update enhances the stability and efficiency of PostgreSQL migrations, especially for large databases or long-running tasks.
Oracle Character Large Object (CLOB) and Character (CHAR) issue	Addressed the issue where wide Character Large Object (CLOB) and Character (CHAR) values containing non-ASCII characters (e.g., special symbols or international characters) were not replicated correctly. This fix ensures accurate replication of large text data, reducing errors, and maintaining data consistency.

Resolved Issue	Description
Microsoft SQL Server Incorrect Latency Issue	Resolved an issue where the Microsoft SQL Server source endpoint would incorrectly report very high latency. This fix provides more accurate performance metrics, allowing you to better monitor and optimize your SQL Server migration tasks.
Microsoft SQL Server Metadata Issue	Fixed an issue with redundant calls to retrieve other tables' metadata, which resulted in degraded performance and latency after a table's metadata had been altered. This optimization improves overall task performance, especially for databases with frequent schema changes.
Microsoft SQL Server UPDATE Issue	Fixed an issue where non-standard UPDATE operations caused tasks to terminate unexpectedly without generating error messages. This fix ensures that such operations are processed correctly, avoiding task failures during complex update scenarios.
Batch Apply Large Object (LOB) issue	Fixed an issue in Batch Optimized Apply mode where Large Object (LOB) lookup would fail to find a record when a DELETE operation and an INSERT operation were combined into a single UPDATE operation. This fix improves data consistency and integrity during migrations involving LOB data and complex Data Manipulation Language (DML) operations.
Transformations Task Start Issue	Addressed an issue where tasks involving numerous transform ations would crash during startup. This fix ensures stability and reliability for tasks with complex transformation logic, enabling you to confidently execute data processing tasks.
MySQL Data Definition Language (DDL) Issue	Fixed a failure to capture special format Data Definition Language (DDL) changes during the Change Data Capture (CDC) phase. This fix ensures that all schema changes are properly replicate d, maintaining schema consistency between source and target databases throughout the migration process.
S3 cross-account bucket access validation issue	Enhanced S3 bucket ownership validation to improve security when using S3 as a migration target or source.

Resolved Issue	Description
Data validation memory leak issue	Fixed memory leak in data validation. This prevents out-of-me mory failures during long-running validation tasks.

AWS Database Migration Service 3.5.4 release notes

New features in AWS DMS 3.5.4

New feature or enhancement	Description
Data masking	Introduced data masking, allowing transforming sensitive data with options for digit randomization, masking, or hashing at the column level.
Enhanced Data Validation Performance	Introduced enhanced data validation performance, enabling faster processing for large datasets during full load and CDC migration tasks across select migration paths.
PostgreSQL source unicode issue	Fixed an issues for PostgreSQL source where degraded migration performance was observed while using filtering. Introduced disableUnicodeSourceFilter ECA to control this behavior.
Transformation support for enhanced throughput feature	Introduce support for all transformation rules for the enhanced throughput feature.
Enhanced logging for missing target columns	Enhanced logging to inform customers about missing target table columns at default logging level. This improvement changes the logging level from VERBOSE to WARNING, making column discrepancy notifications more visible without requiring detailed debug settings.
SQL Server Binary(16) to PostgreSQL UUID migration	Added support for migrating SQL Server Binary(16) data types to PostgreSQL UUID format. This enables seamless conversion of binary GUID data to native UUID types, improving data type compatibility between SQL Server and PostgreSQL endpoints.

New feature or enhancement	Description
Support for PostgreSQL 17	Introduced support for PostgreSQL version 17. For more informati on, see: • Using a PostgreSQL database as an AWS DMS source • Using a PostgreSQL database as a target for AWS Database Migration Service

AWS DMS version 3.5.4 includes the following resolved issues:

Issues resolved in the DMS 3.5.4

Resolved Issue	Description
PostgreSQL, test_deco ding issue	Fixed an issue for PostgreSQL source where certain events would not be replicated while using the test_decoding plugin.
MySQL to Redshift timestamp issue	Fixed an issue for MySQL to Redshift migrations where a timestamp column would not be defined correctly on the target.
Oracle July 2024 PSU issue	Fixed an issue for Oracle source with binary reader where DMS task would crash after applying Oracle July 2024 PSU.
MySQL secrets manager issue	Fixed an issue for MySQL endpoint where credentials would become corrupted when using secrets manager.
Amazon DocumentDB/ MongoDB data record handling issue	Fixed an issue for Amazon DocumentDB/MongoDB endpoints where some records would be sent to the target twice causing duplicate key exception and AWS DMS task failure.
Relational to noSQL migration issue	Fixed an issue for RDS for SQL Server to noSQL migrations where the document structure was incorrect due to incorrect PK handling.
Data validation issue with Oracle endpoint	Fixed an issue for Oracle source where data validation would report false positives for null or empty LOBs.

Resolved Issue	Description
Uniqueidentifier PK issue with Babelfish target.	Fixed an issue for `PG - Babelfish` target where AWS DMS task would fail while replicating tables with PK defined as uniqueide ntifier.
PostgreSQL source issue with MAZ.	Fixed an issue for PostgreSQL source where AWS DMS MAZ failover would cause a fatal AWS DMS task failure.
Column order issue	Fixed an issue where LOB data was not replicated correctly when the column order differed between the source and target.
Internal AWS DMS database contention issue	Fixed an issue for internal DMS database where the AWS DMS task would fail due to internal AWS DMS database concurency issues.
Internal AWS DMS database structure issue	Fixed an issues for internal AWS DMS database where the AWS DMS task would fail due to the lack of presence of certain internal database objects.
Oracle source data validaiton issue	Fixed an issue for Oracle source where data validation would return false positives for certain rare types of replicated events.
Data validation issue for unicode data types	Fixed an issue for the data validation feature where ceratin unicode data types were not compared properly which resulted in false positives.
Parquet target timestamp issue	Fixed an issue for parquet target where zero-timestamp was replicated as null.
Babelfish target GeoSpatia l data type issue.	Fixed an issue for Babelfish target where the GeoSpatial data type was not supported.
Amazon S3 target issue with columns being added during CDC	Fixed an issue for Amazon S3 target where new column additions were not handeled properly when the before image setting was enabled.
SQL Server 2022 CU12 issue	Fixed an issue for SQL Server source where AWS DMS could not automatically implement MS Replication prerequisites on sources using SQL Server 2022 with CU12 or above.

Resolved Issue	Description
PostgreSQL boolean issue	Fixed an issue for PostgreSQL source where boolean data type would not be migrated correctly while `MapBooleanAsBoolean` was set to `true` and pglogical plugin was used.
`TaskrecoveryTableEnabled` setting issue	Fixed an issue for the `TaskrecoveryTableEnabled` setting, where the AWS DMS task would fail upon task stop when set to true.
Data duplication with `TaskrecoveryTableEnabled` setting	Fixed an issue where some transactions would be replicated twice when the `TaskrecoveryTableEnabled` setting was enabled.
MySQL 5.5 source issue	Fixed an issue for MySQL source where AWS DMS task would fail due to inability to read the `BINLOG` from MySQL v5.5.
Data validation partition issue with corrupted data type names	Data validation now prevents memory corruption during data type processing, eliminating fallback to row-by-row validation for affected partitions.
Amazon S3 source to target migration issue	S3 source to target replication now properly handles external table objects during full load and ongoing replication.
Amazon S3 target CDC replication issue	S3 target CDC replication processes CSV format data correctly during ongoing replication phase.
S3 replication issue	Enhanced S3 directory cleanup process prevents task interruptions during migrations.
Data validation performance issue	Data validation operation now optimizes transitions between validation phases, reducing unnecessary delays.
Data validation issue with specific data types	Data validation now accurately processes unbound characters and TEXT data types, ensuring correct validation results.
PostgreSQL source issue with MAZ	PostgreSQL source replication maintains connectivity during Multi-AZ failover events, preventing task failures.

Resolved Issue	Description
Babelfish datetime validation issue	Data validation now correctly compares datetime values when using Babelfish as a target.
MySQL source column replication issue	MySQL source replication now correctly handles mid-table duplicate column additions, preventing task interruptions.
MySQL source column modification issue	MySQL source replication maintains column sequence integrity when multiple columns are added during CDC operations.
DynamoDB LOB replicati on issue	DynamoDB target replication now correctly processes LOB data during CDC, ensuring complete data transfer.
PostgreSQL boolean data validation issue	PostgreSQL source data validation now correctly interprets boolean data type mappings, producing accurate comparison results.
DocumentDB connection recovery issue	DocumentDB replication maintains data consistency during cluster connectivity interruptions and recovery.
Oracle extended VARCHAR2 data truncatio n issue	Oracle source replication preserves trailing spaces in VARCHAR2(4000) columns when extended data type support is enabled.
SQL Server secondary replica DDL handling issue	SQL Server source replication maintains connectivity during DDL operations on secondary replicas, preventing task interruptions.
Secret Manager special character handling issue	AWS Secrets Manager connection strings now support special characters while preserving security protocols.
MongoDB/Amazon DocumentDB duplicate key issue	MongoDB and Amazon DocumentDB replication prevents record duplication that previously triggered key constraint errors.
Oracle timestamp handling issue	Oracle source replication accurately processes timestamp values across various session time zone configurations.

Resolved Issue	Description
Oracle data conversion issue	Oracle source replication now handles data type conversions more robustly, preventing ORA-01460 errors.
SQL Server 2022 CDC replication issue	SQL Server sources (CU12 and above) can now automatically implement MS Replication prerequisites in AWS DMS.
Test decoding DDL handling	AWS DMS ignores DDL statements for tables not configured in table mappings, preventing unnecessary processing of source database schema changes for unmapped tables.
Columns truncation from source SQL Server Views	When migrating data from source SQL Server view to target endpoints, the column length is correctly preserved and no columns are truncated.
Metrics Recording for enhanced data validation	AWS DMS now accurately reports validated record metrics when using enhanced data validation.
DMS task resume issue after upgrade	When resuming tasks after an AWS DMS version upgrade, DMS tasks do not fail to resume due to incompatible internal file formats.
SQL Server Always On primary replica issue	Fixed an issue where AWS DMS now correctly identifies the primary replica in Microsoft SQL Server Always On Availability Groups, resolving previous case sensitivity detection errors.
PostgreSQL target numeric data handling issue	Fixed an issue where AWS DMS tasks failed when migrating unbound numeric data types to PostgreSQL target endpoints.
Data validation character handling issue	AWS DMS now correctly validates CHAR and VARCHAR data during migrations, eliminating false positive reports in validation tasks.
Amazon Redshift LOB parallel apply issue	AWS DMS prevents data corruption in Large Object (LOB) data during parellel batch operations when using Amazon Redshift as a target endpoint.

Resolved Issue	Description
Data validation with column filter issue	Validation failures are prevented and queries are successfully executed on Amazon S3 target when source column filters are applied to VARCHAR, CHAR, DATE, and DATETIME data types.
S3 validation data persistence issue	When using Amazon S3 targets, consistent data validation states are maintained throughout Change Data Capture (CDC) operation s.
String formatting validatio n issue	Fixed an issue where data validation failed due to string formattin g errors during record comparison operations.
DB2 empty strings CDC migration issue	Resolved an issue where empty VARCHAR values from IBM DB2 LUW source databases were incorrectly captured as NULL values instead of empty strings during Change Data Capture (CDC) operations.
Oracle LOB lookup memory leak issue	Fixed memory leak in Oracle source endpoints during LOB lookups when full LOB mode is enabled. This prevents continuous memory growth and out-of-memory failures during replication tasks with LOB data.
Redshift CDC boolean column mapping issue	Redshift target endpoints now correctly map new boolean columns added during CDC when MapBooleanAsBoolean is enabled. This maintains data type consistency with PostgreSQL sources instead of creating varchar columns.
PostgreSQL UUID array inline LOB migration issue	PostgreSQL source endpoints now correctly migrate UUID array data types when inline LOB mode is enabled.
Data validation nullable column diff reporting issue	Data validation now correctly reports mismatch details for nullable columns.
SQL Server to Oracle validation filtering issue	Data validation now correctly formats date filter conditions for Oracle target endpoints when using SQL Server sources.

Resolved Issue	Description
DO_NOTHING mode table creation warning message	Added warning log message when AWS DMS automatically creates target tables in DO_NOTHING mode. This improves visibility and helps users identify when tables are created despite selecting a mode that implies no automatic table creation.
SQL Server BIT to PostgreSQL SMALLINT conversion issue	Fixed SQL Server BIT to PostgreSQL SMALLINT conversion during CDC operations.
S3 cross-account bucket access validation issue	Enhanced S3 bucket ownership validation to improve security when using S3 as a migration target or source.
Redshift endpoint reconnection and initializ ation issue	Redshift endpoints now properly reconnect after connection termination and correctly validate credentials during initialization. This prevents "Server name must be supplied" errors and task failures during recovery.
Data validation memory leak issue	Fixed memory leak in data validation. This prevents out-of-me mory failures during long-running validation tasks.
SQL Server CHAR primary key validation false positive issue	Fixed false positive MISSING_TARGET errors when validating SQL Server to PostgreSQL migrations with CHAR/NCHAR primary keys containing trailing spaces.
S3 validation Athena database conflict issue	Fixed race condition when multiple tasks share S3 endpoints by isolating Athena databases per task.

AWS Database Migration Service 3.5.3 release notes

New features in AWS DMS 3.5.3

New feature or enhancement	Description
Enhanced PostgreSQL source endpoint for Babelfish support	AWS DMS has enhanced its PostgreSQL source endpoint to support Babelfish datatypes. For more information, see Using a PostgreSQL database as an AWS DMS source.
Support for S3 Parquet as a source	AWS DMS supports S3 Parquet as a source. For more information, see Using Amazon S3 as a source for AWS DMS
Support for PostgreSQL 16.x	AWS DMS supports PostgreSQL version 16.x. For more information, see Using a PostgreSQL database as an AWS DMS source and Using a PostgreSQL database as a target for AWS Database Migration Service.
Enhanced Throughput for Full-Load Oracle to Amazon Redshift Migrations	AWS DMS Serverless provides significantly improved throughput performance for full-load migrations from Oracle to Amazon Redshift. For more information, see Enhanced Throughput for Full-Load Oracle to Amazon Redshift and Amazon S3 Migrations.
Support for RDS for SQL Server Log backups	AWS DMS supports ongoing replication from transaction log backups for RDS for SQL Server sources. This feature is available only to new and modified AWS DMS source endpoints reading from RDS for SQL Server. For more information, see Setting up ongoing replication on a cloud SQL Server DB instance.
Enhanced logging for missing target columns	Enhanced logging to inform customers about missing target table columns at default logging level. This improvement changes the logging level from VERBOSE to WARNING, making column discrepancy notifications more visible without requiring detailed debug settings.

AWS DMS version 3.5.3 includes the following resolved issues:

Issues resolved in the DMS 3.5.3 release dated 17-May-2024

Resolved Issue	Description
Data Validation override function	Fixed an issue for the data validation feature where AWS DMS would not honor source filtering when a rule action was set to `override-validation-function` in table mappings.
MySQL source CDC errors	Fixed an issue for MySQL as a source where CDC migration would fail with UTF16 encoding.
Data validation collation differences	Fixed an issue for the data validation feature where AWS DMS would not properly apply the `HandleCollationDiff` task setting when column filtering was used.
Data validation task hanging.	Fixed an issue for the data validation feature where the AWS DMS task would hang with a `target` is null" error.
Task failures in PostgreSQ L to PostgreSQL replicati on.	Fixed an issue for PostgreSQL to PostgreSQL migrations where a AWS DMS task would fail while inserting LOB data into the target during CDC replication.
Data loss with PostgreSQL as a source	Fixed an issue for PostgreSQL as a source where data loss occurring in certain edge-case scenarios.
MySQL 5.5 source CDC errors	Fixed an issue for MySQL as a source where CDC replication would fail with MySQL verion 5.5.
Oracle source IOT table issue.	Fixed an issue for Oracle as a source where AWS DMS wouldn't replicate `UPDATE` statements correctly for IOT tables with supplemental logging enabled on all columns.
MySQL source LOBS	Fixed an issue for MySQL to Amazon Redshift migrations where the AWS DMS task would fail due to LOBs exceeding the maximum size allowed by Amazon Redshift.
Validation issue with `SkipLobColumns`	Fixed an issue for the data validation feature where the AWS DMS task would fail with `SkipLobColumns = true` when a primary key was on the last column in the source table.

Resolved Issue	Description
Skip validation where the unique key is `null`	Fixed an issue for the data validation feature where AWS DMS doesn't skip rows with null unique keys properly.
Data validation improvements for Oracle `COLLATE` operator.	Fixed an issue for the data validation feature where the validation would fail with a syntax error on Oracle versions prior to 12.2.
Error handling during full load	Fixed an issue for PostgreSQL as a target where the task would hang during the full-load phase after a table error caused by invalid data.
Revalidation of CDC validation-only tasks	Enhanced the data validation feature to allow revalidation on a CDC validation-only task.
S3 as a target `CdcMaxBatchInterval Out of Memory` issue	Fixed an issue for S3 as a target where the AWS DMS task would fail with an Out of Memory condition with CdcMaxBatchInterval set.
Oracle source driver	Upgraded the AWS DMS Oracle source driver from v12.2 to v19.18.
LOB truncation warning with SQL Server source	Enhanced logging for SQL Server as a source to show warnings on LOB truncation during CDC.
Oracle binary reader enhancements	Enhanced the Oracle source binary reader to support the following: • Big Endian platform • Parallel DML hints with HCC compression • Advanced Oracle Compressions with Golden Gate enabled
Skip cached changes issue after interruptions	Fixed an issue for skipping cached changes in case of an unexpecte d interruption while applying those changes.
MongoDB AWS DMS swap file issue	Fixed an issue for MongoDB source where AWS DMS task would fail to resume after AWS DMS upgrade when AWS DMS swap file was present.

Resolved Issue	Description
MySQL to Amazon Redshift JSON migration issue.	Fixed an issue for MySQL source where the JSON data type was not being hadled propely with Batch Apply enabled.
Data validation issue for SQL Server to APG migrations.	Fixed an issue for SQL Server to PostgreSQL migration where data validaiton would report false positives in certain situations.
EmptyAsNull ECA issue for Amazon Redshift target CDC	Fixed an issue for Amazon Redshift target where the `EmptyAsNull` ECA would not work correctly.
Logging enhancement for SQL Server source	Enhanced logging for SQL Server source to unclude the storage unit value.
Memory leak issue	Fixed an issue where for targets using CSV files to load data AWS DMS task was showing a memory leak.
Amazon S3 target issue	Fixed an issue for Amazon S3 target where CdcMaxBatchInterval and CdcMinFileSize were not respected when cdcInsertAndUpdate Only setting was enabled.
Logging enhancement for Kafka target	Enhanced logging for Kafka target
Logging enhancement for Oracle source.	Enhanced logging for Oracle source with binary reader to properly indicate tables being skipped due to missing primary keys.
Logging Enhancement for SQL Server source	Enhanced logging for SQL Server source in `AlwaysOn` configuration to properly indicate missing permissions.
Logging Enhancement for migrations with disabled DDL	Enhanced logging for migrations with disabled DDL replication to indicate unexpected target table structure after its modified outside of AWS DMS.
Db2 target issue	Fixed an issue for Db2 target where the task would fail when the AWS DMS status table is enabled.

Resolved Issue	Description
MongoDB / Amazon DocumentDB Secret Manger issue.	Fixed an issue for MongoDB / Amazon DocumentDB endpoints where the credentials could not be retrieved from Secret Manager which resulted in an error.
MongoDB / Amazon DocumentDB endpoint issue	Fixed an issue for MongoDB / Amazon DocumentDB where the task would fail with ParallelApply enabled while replicating certain sequence of events.
Logging enhancement for Amazon Redshift target	Enhanced logging for Amazon Redshift target to include more detailed information in default logging level.
Amazon S3 target task crash issue	Fixed an issue for Amazon S3 target where DMS task would crash after receiving alter table DDL when `GlueCatalogGeneration` is enabled.
Data validation issue	Fixed an issue for data validation feature where validation would fail on `NUL (0x00)` characters.
Babelsifh endpoint issue	Fixed an issue for babelfish endpoint where tables names with mixed case would be suspended.
Amazon S3 source issue	Fixed an issue for Amazon S3 source where files were not process due to a file name validation issue.
Db2 LUC source issue with selection rules.	Fixed an issue for Db2 LUW source where "table-type" option in selection rules was being ignored.
Amazon Redshift target missing data issue.	Fixed an issue for Amazon Redshift target where data loss would occur when `ParallelLoadThreads` was >0 under certain conditions.
Data validation enhancement for Amazon Redshift target	Enhanced the data validation feature for Amazon Redshift target to support `HandleCollationDiff` setting.

Resolved Issue	Description
Amazon S3 target data validation issue	Fixed an issue for Amazon S3 target data validation where validation would fail when there were no other columns than the PK in the table.
Data validation CloudWatc h metrics	Fixed an issue for data validation feature where the CloudWatc h metrics would be missing for validation which took a short amount of time to complete.
Data validation re-validate option unavailable	Fixed an issue for data validation feature where the re-validation option was unavailable in certain situations.
Large tranasction replicati on issue	Fixed an issue where the maximum number of events per transacti on was limited to 201,326,592 under certain conditions.
MySQL source data loss issue	Fixed an issue for MySQL to Amazon S3 migration where first DML executed after "add column" DDL would be missed resulting in data loss.
Memory leak issue in batch apply	Fixed a memory leak issue for batch apply feature which would occur under certain conditions.
DMS task start issue	Fixed an issue where AWS DMS task start woudl take a very long time and never complete.
MySQL source data validation issue.	Fixed an issue for MySQL source data validation where auto-incr emented columns were not handled properly.
Oracle endpoint Kerberos authentication issue	Fixed an issue with Oracle endpoint where connectivity would not work with Kerberos authentication.
Babelfish target uniqueide ntifier issue	Fixed an issue for Babelfish target where replication would fail for tables with primary keys defined as uniqueidentifier.
PostgreSQL souce missing data issue	Fixed an issue for PostgreSQL source where data loss would occur due to unknown events in the replication slot.

Resolved Issue	Description
Data validation issue for unicode data types.	Fixed an issue for the data validation feature where ceratin unicode data types were not compared properly which resulted in false positives.
Redshift Target Timestamp Column	Fixed an issue with ColumnType definition where timestamp column types were not set correctly
S3 Target Timestamp Conversion	Fixed an issue with S3 target parquet file format by handling zero timestamp values (these values should be converted to NULL on the target)
MySQL source and target LOB columns order	Fixed an issue with MySQL source and target migrating Lob columns. Now DMS uses the column id from the target table instead of source table when deciding to which column we need to write the LOB Data
Babelfish target Datetime data types	Fixed an issue with Babelfish target where we have restricted the fractional seconds precision for Datetime and Time types to solve rounding errors.
MySQL 5.5 task failures while reading binglog during on going replicati on (CDC).	Fixed an issue with MySQL 5.5 source, we have added retry mechanism to prevent task failure when DMS would fail to read binary log events during on going replication (CDC).
PostgreSQL Source segfault and memory leak issue while using test decoding plugin.	Fixed an issue with PostgreSQL source where certain onging replication (CDC) events failed to be parsed correctly when using the test_decoding plugin for Postgres.
DocumentDB Target parallel apply settings	Fixed an issue with DocumentDB target with parallel apply setting which was preventing the use of multiple threads while using this feature.
Oracle HCC Compression DIRECT INSERT	Fixed an issue with Oracle HCC compression DIRECT INSERT with parallel DML hint causing missing and duplicate data.

Resolved Issue	Description
Oracle Binary Reader issue with Oracle July 2024 CPU	Fixed an issue with Oracle source, DMS task with binary reader were failing due to Oracle July 2024 CPU
Babebfish Target UUID Primary Keys	Fixed an issue with Babelfish target where DMS task were failing while replicating tables with Primary Key defined as UUID.
DMS ongoin replicati on (CDC) tasks failing with TaskRecoveryTableE nabled	Fixed an issue with `TaskRecoveryTableEnabled` enabled, where DMS attempts to update the target system table `awsdms_txn_state` after the target connection is terminated.
PostgreSQL source duplicate records	Fixed an issue with PostgreSQL source where some transactions would be replicated twice when the TaskrecoveryTableEnabled setting was enabled.
DMS S3 source/target replication issue	Fixed an issue with S3 source to S3 target where DMS task was not replicating data during full load and on going replication.
DMS S3 source task failure during on going replicati on	Fixed an issue for S3 source where DMS task was seg faulting during on going replication for DMS vesion 3.5.3
DMS DB2 source CcsidMapping ECA	Fixed an issue with DB2 soruce with CcsidMapping, CCSID mappin ECA is now properly applied to task when code page is 0 and data is migrated properly
DMS Aurora PostgreSQ L to Redshift Serverless Boolean Value replication.	Fixed an issue where DMS migration from Aurora PostgreSQL to Redshift Serverless was seeing issue with Boolean value.
Data validation issue with specific data types	Data validation operation can now accurately processes unbound characters and TEXT data types, ensuring correct validation results.
PostgreSQL source issue with MAZ	PostgreSQL source replication maintains connectivity during Multi-AZ failover events, preventing task failures.

Resolved Issue	Description
Babelfish datetime validation issue	Data validation now correctly compares datetime values when using Babelfish as a target, improving cross-platform compatibi lity.
MySQL source column replication issue	MySQL source replication now correctly handles mid-table duplicate column additions, preventing task interruptions.
MySQL source column modification issue	MySQL source replication maintains column sequence integrity when multiple columns are added during CDC operations.
DynamoDB LOB replicati on issue	DynamoDB target replication now correctly processes LOB data during CDC, ensuring complete data transfer.
PostgreSQL boolean data validation issue	PostgreSQL source data validation now correctly interprets boolean data type mappings, producing accurate comparison results.
Oracle extended VARCHAR2 data truncatio n issue	Oracle source replication preserves trailing spaces in VARCHAR2(4000) columns when extended data type support is enabled.
SQL Server secondary replica DDL handling issue	SQL Server source replication maintains connectivity during DDL operations on secondary replicas, preventing task interruptions.
Secret Manager special character handling issue	AWS Secrets Manager connection strings now support special characters while preserving security protocols.
MongoDB/Amazon DocumentDB duplicate key issue	MongoDB and Amazon DocumentDB replication prevents record duplication that previously triggered key constraint errors.
Oracle timestamp handling issue	Oracle source replication accurately processes timestamp values across various session time zone configurations.
Oracle data conversion issue	Oracle source replication now handles data type conversions more robustly, preventing ORA-01460 errors and associated task failures.

Resolved Issue	Description
Test decoding DDL handling	AWS DMS ignores DDL statements for tables not configured in table mappings, preventing unnecessary processing of source database schema changes for unmapped tables.
Columns truncation from source SQL Server Views	When migrating data from source SQL Server view to target endpoints, the column length is correctly preserved and no columns are truncated.
SQL Server Always On primary replica issue	Fixed an issue where AWS DMS now correctly identifies the primary replica in Microsoft SQL Server Always On Availability Groups, resolving previous case sensitivity detection errors.
PostgreSQL target numeric data handling issue	Fixed an issue where AWS DMS tasks failed when migrating unbound numeric data types to PostgreSQL target endpoints.
Data validation character handling issue	AWS DMS now correctly validates CHAR and VARCHAR data during migrations, eliminating false positive reports in validation tasks.
Amazon Redshift LOB parallel apply issue	AWS DMS prevents data corruption in Large Object (LOB) data during parellel batch operations when using Amazon Redshift as a target endpoint.
Data validation with column filter issue	Validation failures are prevented and queries are successfully executed on Amazon S3 target when source column filters are applied to VARCHAR, CHAR, DATE, and DATETIME data types.
S3 validation data persistence issue	When using Amazon S3 targets, consistent data validation states are maintained throughout Change Data Capture (CDC) operation s.
String formatting validatio n issue	Fixed an issue where data validation failed due to string formattin g errors during record comparison operations.

Resolved Issue	Description
DB2 empty strings CDC migration issue	Resolved an issue where empty VARCHAR values from IBM DB2 LUW source databases were incorrectly captured as NULL values instead of empty strings during Change Data Capture (CDC) operations.
Oracle LOB lookup memory leak issue	Fixed memory leak in Oracle source endpoints during LOB lookups when full LOB mode is enabled. This prevents continuous memory growth and out-of-memory failures during replication tasks with LOB data.
S3 cross-account bucket access validation issue	Enhanced S3 bucket ownership validation to improve security when using S3 as a migration target or source.
Data validation memory leak issue	Fixed memory leak in data validation. This prevents out-of-memory failures during long-running validation tasks.

AWS Database Migration Service 3.5.2 release notes

New features in AWS DMS 3.5.2

New feature or enhancement	Description
Amazon Redshift data validation	AWS DMS now supports validating data in Amazon Redshift targets.
Support for Microsoft SQL Server version 2022 as a source and target.	AWS DMS now supports using Microsoft SQL Server version 2022 as a source and target.
IBM Db2 LUW as a target	AWS DMS now supports IBM Db2 LUW as a target. Using AWS DMS, you can now perform live migrations from IBM Db2 LUW to IBM Db2 LUW.

AWS DMS version 3.5.2 includes the following resolved issues:

Issues resolved in the DMS 3.5.2 maintenance release dated 29-Apr-2024

Resolved Issue	Description
IBM Db2 target segmented full load	Added support for segmented full load with IBM Db2 as a target.
Amazon Timestream as a target settings	Enhanced the handling of invalid timestamp settings and unsupported table operations for Timestream as a target.
Task crash with column filter	Fixed an issue where a task was crashing while using a filter on a column that DMS added dynamically using a transformation rule.
Logging transaction swap file reading	Added logging to show when DMS is reading from transaction swap files.
S3 as a target with `CdcInsertsAndUpdates`	Fixed an issue for S3 as a target where a task would crash when `CdcInsertsAndUpdates` is `true` and `PreserveTransactions` is `true`.
Source filter negative operators	Fixed an issue where the source filter-operator when set to a negative operator had incorrect behavior if the same column had a transformation rule defined.
Added logging for when DMS pauses reading from the source	Enhanced logging to show when DMS temporarily pauses reading from the source to improve performance.
Source filters with escaped characters	Fixed an issue for source filters where DMS applies escaped characters to newly created tables during CDC.
PostgreSQL as a target, incorrectly replicated deletes	Fixed an issue for PostgreSQL as a target where DMS replicates deletes as null values.
Oracle as a source logging improvements	Enhanced logging for Oracle as a source to remove extraneous error codes.
Improved logging of XMLTYPE limitations	Improved logging for Oracle as a source to show DMS' lack of support for full LOB mode for the `XMLTYPE` data type.

Resolved Issue	Description
MySQL data loss	Fixed an issue for MySQL as a target where corrupted column metadata could cause task crashes or data loss.
Filter applied to a new column	Fixed an issue during full load where DMS ignores a filter that a transformation rule adds to a new column.
S3 as a target: Validation issue	Fixed an issue for S3 as a target where data validation would fail while migrating multiple tables with different validation partition ing definitions.
CDC-only task crash	Fixed an issue for CDC-only tasks where the task would crash when `TaskRecoveryTableEnabled` is `true`.
MySQL to MariaDB incompatible collations	Fixed an issue for MySQL to MariaDB migrations where DMS doesn't migrate MySQL v8 tables with `tf8mb4_0900_ai_ci` collation.
Task crashes with `BatchApplyEnabled`	Fixed an issue for the Batch Apply feature where the task would fail under certain conditions.
Non UTF-8 characters in Amazon DocumentDB	Added support for non UTF-8 characters for Amazon DocumentDB endpoints.
Batch Apply task crash	Fixed an issue for the Batch Apply feature where the DMS task crashes while replicating large transactions.
Db2 transaction rollback handling	Fixed an issue for Db2 as a source where DMS would replicate an `INSERT` to the target, despite being rolled back on the source.
Validation with source filters	Fixed an issue where validation was not respecting source filters.
MongoDB AWS DMS swap file issue	Fixed an issue for MongoDB source where DMS task would fail to resume after AWS DMS upgrade when AWS DMS swap file was present.

Resolved Issue	Description
MySQL to Amazon Redshift JSON migration issue.	Fixed an issue for MySQL source where the JSON data type was not being hadled propely with Batch Apply enabled.
MySQL 5.5 source CDC errors	Fixed an issue for MySQL source where CDC replicaiton would fail with MySQL verion 5.5.
LOB migration issue	Fixed an issue for LOB migration where the AWS DMS task would crash while processing certain event types.
Data validation issue	Fixed an issue for Data Validation feature where the Validation-only task would hang on certain DDL events.
Data validation issue with filters.	Fixed an issue for data validation feature where the the `HandleCollationDiff` setting was not applied when filters were present
MySQL SourceEnum encoding issue.	Fixed an issue for MySQL source where UTF-16 encoded enum values were not migrated correctly.
LOB truncation warning with SQL Server source	Enhanced logging for SQL Server source to show warning on LOB truncation during CDC.
Data validation issue for SQL Server to APG migrations.	Fixed an issue for SQL Server to PostgreSQL migration where data validaiton would report false positives in certain situations.
`EmptyAsNull` ECA issue for Amazon Redshift target CDC	Fixed an issue for Amazon Redshift target where the `EmptyAsNull` ECA would not work correctly.
Logging enhancement for SQL Server source	Enhanced logging for SQL Server source to unclude the storage unit value.
Memory leak issue	Fixed an issue where for targets using CSV files to load data AWS DMS task was showing a memory leak.

Resolved Issue	Description
Amazon S3 target issue	Fixed an issue for Amazon S3 target where `CdcMaxBatchInterval` and `CdcMinFileSize` were not respected when `cdcInsertAndUpdateOnly` setting was enabled.
Logging enhancement for Kafka target	Enhanced logging for Kafka target
Logging enhancement for Oracle source.	Enhanced logging for Oracle source with binary reader to properly indicate tables being skipped due to missing primary keys.
Logging Enhancement for SQL Server source	Enhanced logging for SQL Server source in `AlwaysOn` configuration to properly indicate missing permissions.
Logging Enhancement for migrations with disabled DDL	Enhanced logging for migrations with disabled DDL replication to indicate unexpected target table structure after its modified outside of AWS DMS.
Db2 target issue	Fixed an issue for Db2 target where the task would fail when the DMS status table is enabled.
MongoDB / Amazon DocumentDB Secret Manger issue.	Fixed an issue for MongoDB / Amazon DocumentDB endpoints where the credentials could not be retrieved from Secret Manager which resulted in an error.
MongoDB / Amazon DocumentDB endpoint issue	Fixed an issue for MongoDB / Amazon DocumentDB where the task would fail with ParallelApply enabled while replicating certain sequence of events.
MongoDB / Amazon DocumentDB data validation issue	Fixed an issue for MongoDB / Amazon DocumentDB data validation where an error would occur after detecting data differences.
Logging enhancement for Amazon Redshift target	Enhanced logging for Amazon Redshift target to include more detailed information in default logging level.

Resolved Issue	Description
Amazon S3 target task crash issue	FIxed an issue for Amazon S3 target where AWS DMS task would crash after receiving alter table DDL when `GlueCatalogGeneration` is enabled.
Data validation issue	Fixed an issue for data validation feature where validation would fail on `NUL (0x00)` characters.
Babelsifh endpoint issue	Fixed an issue for babelfish endpoint where tables names with mixed case would be suspended.
Db2 LUC source issue with selection rules.	Fixed an issue for Db2 LUW source where "table-type" option in selection rules was being ignored.
Amazon Redshift target missing data issue.	Fixed an issue for Amazon Redshift target where data loss would occur when `ParallelLoadThreads` was >0 under certain conditions.
Data validation enhancement for Amazon Redshift target	Enhanced the data validation feature for Amazon Redshift target to support `HandleCollationDiff` setting.
Amazon S3 target data validation issue	Fixed an issue for Amazon S3 target data validation where validation would fail when there were no other columns than the PK in the table.
Data validation CloudWatch metrics	Fixed an issue for data validation feature where the CloudWatch metrics would be missing for validation which took a short amount of time to complete.
Data validation re-validate option unavailable	Fixed an issue for data validation feature where the re-validation option was unavailable in certain situations.
Large tranasction replication issue	Fixed an issue where the maximum number of events per transaction was limited to 201,326,592 under certain conditions.
Table reload issue	Fixed an issue where a reload of multiple tables was canceled when at least one of the table was invalid.

Resolved Issue	Description
MySQL source data loss issue	Fixed an issue for MySQL to S3 migration where first DML executed after "add column" DDL would be missed resulting in data loss.
Memory leak issue in batch apply	Fixed a memory leak issue for batch apply feature which would occur under certain conditions.
AWS DMS task start issue	Fixed an issue where AWS DMS task start woudl take a very long time and never complete.
PostgreSQL souce missing data issue	Fixed an issue for PostgreSQL source where data loss would occur due to unknown events in the replication slot.
MySQL source and target LOB columns order	Fixed an issue with MySQL source and target migrating Lob columns. Now DMS uses the column id from the target table instead of source table when deciding to which column we need to write the LOB Data.
MySQL 5.5 task failures while reading binglog during on going replicati on (CDC).	Fixed an issue with MySQL 5.5 Source, added retry mechanism to prevent task failure when DMS would fail to read binary log events during on-going replication (CDC).
PostgreSQL Source segfault and memory leak issue while using test decoding plugin.	Fixed an issue with PostgreSQL Source where certain onging replication (CDC) events failed to be parsed correctly when using the test_decoding plugin for Postgres.
DocumentDB Target parallel apply settings	Fixed an issue with DocumentDB target with parallel apply setting which was preventing the use of multiple threads while using this feature.
Oracle HCC Compression DIRECT INSERT	Fixed an issue with Oracle HCC compression DIRECT INSERT with parallel DML hint causing missing and duplicate data.
Oracle Binary Reader issue with Oracle July 2024 CPU	Fixed an issue with Oracle Source, DMS task with binary reader were failing due to Oracle July 2024 CPU.

Resolved Issue	Description
DMS ongoin replicati on (CDC) tasks failing with TaskRecoveryTableE nabled	Fixed an issue with `TaskRecoveryTableEnabled` enabled, where DMS attempts to update the target system table `awsdms_txn_state` after the target connection is terminated.
PostgreSQL source duplicate records	Fixed an issue with PostgreSQL source where some transactions would be replicated twice when the `TaskrecoveryTableE nabled` setting was enabled.
DMS S3 source/target replication issue	Fixed an issue with S3 source to S3 target where DMS task was not replicating data during full load and on going replication.
DMS DB2 source CcsidMapping ECA	Fixed an issue with DB2 soruce with CcsidMapping, CCSID mappin ECA is now properly applied to task when code page is 0 and data is migrated properly.
DMS Aurora PostgreSQ L to Redshift Serverless Boolean Value replication.	Fixed an issue where DMS migration from Aurora PostgreSQL to Redshift Serverless was seeing issue with Boolean value.
PostgreSQL source issue with MAZ	PostgreSQL as a source endpoints now maintain uninterrupted replication during Multi-AZ failover events, preventing task failures.
MySQL source column replication issue	MySQL as a source replication handles mid-table duplicate column additions, preventing unexpected task interruptions.
MySQL source column modification issue	MySQL as a source replication maintains column sequence integrity when multiple columns are added during CDC operation s.
PostgreSQL boolean data validation issue	PostgreSQL as a source data validation now correctly interpret s boolean data type mappings, producing accurate comparison results.

Resolved Issue	Description
Oracle extended VARCHAR2 data truncatio n issue	Fixed an issue for Oracle as a source where trailing spaces were being truncated in VARCHAR2(4000) columns when using extended data type support.
SQL Server secondary replica DDL handling issue	SQL Server source replication maintains connectivity during DDL operations on secondary replicas, preventing task interruptions.
Secret Manager special character handling issue	AWS Secrets Manager connection strings now support special characters while preserving security protocols.
MongoDB/Amazon DocumentDB duplicate key issue	MongoDB and Amazon DocumentDB replication prevents record duplication that previously triggered key constraint errors.
Oracle timestamp handling issue	Oracle source replication now accurately processes timestamp values across various session time zone configurations.
Oracle data conversion issue	Oracle source caused AWS DMS task to fail with an ORA-01460 error during data type conversion. This update improves the handling of data type conversions to prevent task failures.

AWS Database Migration Service 3.5.1 release notes

Change in Handling of Large Numeric Values for Streaming Targets

In AWS DMS version 3.5.1, there is a change in how large integer and high-precision numeric values are handled when streaming data to targets like Kafka and Kinesis. Specifically, AWS DMS changed its internal data type representation, handling these values as `INT64` instead of `INT8`. This shift can result in different data formats on the streaming endpoints, particularly when the values exceed the limits of `INT8`. Consequently, the representation of these numeric types may differ from their previous formatting when streamed to destinations like Kafka and Kinesis, potentially impacting downstream systems and processes that consume the data from these targets.

Summary of Change:

- In previous versions (e.g., 3.4.7/3.4.6), large integer values were represented as integers.

- Starting with version 3.5.1, these values may appear in scientific notation (e.g., `7.88129934789981E15`), potentially leading to precision and formatting differences.

Affected Data Types

The recent change affects the representation of several numeric types when streamed to endpoints like Kafka and Kinesis. The impacted types are:

- Large integer types (e.g., `bigint`)
- Floating-point types (`FLOAT, DOUBLE`)
- High-precision decimal types (`DECIMAL, NUMERIC`)

Affected Scenarios:

- Full load migrations to streaming targets
- Change Data Capture (CDC) to streaming targets

This change specifically impacts streaming endpoints such as Kafka and Kinesis, while non-streaming targets remain unaffected.

To mitigate this change, you can implement a data type transformation that reverts to the previous formatting, representing large numbers as integers. However, it's important to note that this workaround may not be suitable for all scenarios, as it could potentially introduce limitations or compatibility issues.

Recommendations:

- Test your specific use case in a non-production environment before deploying AWS DMS version 3.5.1 or later to identify and address any impacts of this change.
- Affected customers can implement the change-data-type transformation workaround, if applicable, to revert to the previous formatting for large numbers as integers. However, this approach may not suit all scenarios.

We are reviewing this behavior to ensure consistent data type handling across endpoints in future releases.

The following table shows the new features and enhancements introduced in AWS Database Migration Service (AWS DMS) version 3.5.1.

New feature or enhancement	Description
Support for PostgreSQL 15.x	AWS DMS version 3.5.1 supports PostgreSQL version 15.x. For more information, see Using PostgreSQL as a source and Using PostgreSQL as a target.
Support for Amazon DocumentDB Elastic Clusters with sharded collections	AWS DMS version 3.5.1 supports Amazon DocumentDB Elastic Clusters with sharded collections. For more information, see Using Amazon DocumentDB as a target for AWS Database Migration Service.
Amazon Redshift Serverles s as a Target	Support for using Amazon Amazon Redshift Serverless as a target endpoint. For more information, see Using an Amazon Redshift database as a target for AWS Database Migration Service.
Babelfish Endpoint Settings	Enhanced PostgreSQL target endpoint settings for providing Babelfish support. For more information, see Using a PostgreSQL database as a target for AWS Database Migration Service.
Oracle Source Open Transactions	AWS DMS 3.5.1 improves the methodology of handling open transactions when starting a CDC-Only task from the Start Position for an Oracle source. For more information, see `OpenTransactionWindow` in the Endpoint settings when using Oracle as a source for AWS DMS section.
Amazon Timestream as a Target	Support for using Amazon Timestream as a target endpoint. For more information, see Using Amazon Timestream as a target for AWS Database Migration Service.

AWS DMS version 3.5.1 includes the following resolved issues:

Resolved Issue	Description
Large numeric value handling change	The representation of large numeric values on streaming targets has been updated. Review the 'Handling of Large Numeric Values in Streaming Targets' documentation for details on potential impacts.
Oracle as a source growing inactive sessions	Fixed an issue for Oracle source where CDC-only tasks had continuously growing inactive sessions, resulting in the following exception: `ORA-00020: maximum number of processes exceeded on the source database`.
Replicating UPDATE changes to DocumentDB	Fixed an issue for DocumentDB as a target where UPDATE statements were not properly replicated in some scenarios.
Validation-only task	Improved error handling for the data validation feature to properly fail the task when data validation is disabled for validatio n-only tasks.
Amazon Redshift replicati on after connection termination	Fixed an issue for Amazon Redshift target where the DMS task would not retry applying changes on the target when the target has `ParallelApplyThreads` set greater than zero after connection termination, which would result in data loss.
MySQL text to mediumtex t replication	Fixed an issue for MySQL to MySQL replication of mediumtext data types with full-LOB mode.
CDC task not replicating with rotated secret	Fixed an issue for DMS tasks with `BatchApplyEnabled` set to `true` where DMS would stop replicating data after Secrets Manager rotated the password.
MongoDB/DocumentDB segmentation issue	Fixed an issue for MongoDB / DocDB source where range segmentation would not work properly when the primary key column contained a large value.
Oracle data validation of unbound numeric values	Fixed an issue for Oracle target where DMS would recognize a value of unbound data type `NUMERIC` as a `STRING` during data validation.

Resolved Issue	Description
SQL Server data validation	Fixed an issue for SQL Server endpoints where DMS data validation constructed an invalid SQL statement.
MongoDB Auto-Segementation	Improved the functionality of automatic partitioning of data when migrating documents in parallel from MongoDB as a source.
Amazon S3 Apache Parquet format	Fixed an issue so that Apache Parquet files written to S3 as a target can be viewed with Python with Apache Arrow C++.
PostgreSQL as a source DDL handling	Fixed an issue with PostgreSQL source where unsupported DDL operations were not properly ignored.
PostgreSQL `timestamp tz` data error	Fixed an issue with PostgreSQL to PostgreSQL migrations where timestamp with time zone data was not migrated properly with Batch apply enabled during CDC.
Oracle to PostgreSQL validation failure	Fixed an issue with Oracle to PostgreSQL migrations where data validation was failing for the NUMERIC(38,30) data type.
Oracle extended data type error	Fixed an issue with Oracle source where extended varchar data type was being truncated.
Combining filter operators	Fixed an issue for the column filtering functionality where the null column operator could not be combined with other types of operators.
CDC latency resulting from excessive logging.	Fixed an issue with PostgreSQL source where excessive logging of pglogical plug-in warnings was causing source CDC latency.
Bi-directional replication handling of Create Table DDL	Fixed an issue for bi-directional replication from PostgreSQL to PostgreSQL where the Create Table DDL change was not replicated properly.
CDC failure while using filters	Fixed an issue for the filtering feature where CDC replication was failing.

Resolved Issue	Description
Certificate authority hostname validation for Kafka endpoints	Enhanced the functionality of Kafka endpoints by adding the option to disable hostname validation of the certificate authority (`SslEndpointIdentificationAlgorithm`).
IBM Db2 LUW validation	Fixed an issue where Db2 LUW source date, timestamp, and time data types were not handled properly during data validation.
S3 validation	Fixed an issue with Db2 LUW to S3 migrations where the validatio n fuctionality was not handling the timestamp(0) datatype properly.
DMS task restart failure	Fixed an issue with PostgreSQL source where the AWS DMS task was failing to restart and could not consume relational events when using the pglogical plugin.
SQL Server validation of HIERARCHY data type	Fixed an issue for SQL Server source where validation of HIERARCHY data type would fail.
SQL Server strings with control characters	Fixed an issue for SQL Server soruce where strings with control characters were not replicated correctly.
Amazon Redshift with Secrets Manager	Fixed an issue with Amazon Redshift target where testing the endpoint would fail when using Secrets Manager.
MySQL ParallelL oadThreads setting inconsistency	Fixed an issue with MySQL target where the `ParallelL oadThreads` setting was not properly retained after task settings changes.
Error with PostgreSQL to Oracle data type mapping	Fixed an issue with PostgreSQL to Oracle migrations where the task would fail when replicating from data type TEXT to data type VARCHAR2(2000).
Oracle to PostgreSQL data validation	Fixed an issue with Oracle to PostgreSQL migrations where data validation reported false positives when NULL characters were replicated as SPACE characters.

Resolved Issue	Description
SQL Server source in AlwaysOn configuration	Fixed an issue with SQL Server source in AlwaysOn confuguration where the AWS DMS task would fail when the replica name didn't match the actual server name exactly.
Oracle source endpoint test failure	Fixed an issue with Oracle source where the AWS DMS endpoint connection test would fail due to insufficient priveleges while retrieving the Oracle session ID (SID).
CDC not picking up new tables	Fixed an issue with CDC-only tasks where tables created on the source after the task was started were not replicated in some cases.
Open transactions in Oracle as a source	Improved the methodology of handling open transactions when starting a CDC-Only task from the Start Position for an Oracle source.
Missing data issue	Fixed an issue of missing data when resuming a task if it was stopped after cached changes were applied (`StopTaskCachedChangesApplied` option set to true). This issue could occur rarely if AWS DMS persists cached changes to the AWS DMS replication instance disk due to a high volume of changes on the source.
Data validation issue on extended datatype	Fixed an issue for PostgreSQL to Oracle data validation where validation was failing for extended data types.
Data validation issue on inconsistent character encoding	Fixed an issue for SQL Server to PostgreSQL data validation where validation was failing when character encoding was inconsistent between source and target.
Data validation issue ORA-01455	Fixed an issue where an ORA-01455 error occurs during validation when a PostgreSQL `integer` maps to an Oracle `number(10)` .
SQL Server `IDENTITY` support	Fixed an issue for SQL Server to SQL Server data replication where migrating identity columns fails when the target column had the `IDENTITY` property.

Resolved Issue	Description
Character set issue with ALTER statements	Fixed an issue for MySQL to MySQL replication where AWS DMS changes the character set to UTF16 when migrating an `ALTER` statement during CDC.
PostgreSQL to Amazon Redshift Spatial data type support	Added support for the `spatial` datatype when migrating from PostgreSQL to Amazon Redshift.
GZIP compression of .parquet files	Fixed an issue where AWS DMS fails to generate .parquet files with GZIP compression with S3 as a target.
MongoDB/DocDB source migration	Fixed an issue where AWS DMS doesn't migrate some of the partitions from a MongoDB source.
Table statistics issue	Fixed an issue where table statistics were not shown when at least one of the tasks on the replication instance contained more than 1001 tables.
Table suspended for IBM Db2 LUW versions 10.1.0 and lower	Fixed an issue for Db2 LUW source where table migration was suspended with the error `TYPESTRINGUNITS is not valid` when the source database version is 10.1.0 or lower.
MongoDB partitioning issue	Fixed an issue for MongoDB/DocDB where one or more segments of the source partition are missing.
MongoDB partitioning issue	Fix an issue where segmentation based on a column with the NumberLong() type fails due to type conversion bug.
MongoDB partitioning issue	Improved autosegmentation performance for large data sets with MongoDB as a source.
MongoDB driver version	Downgraded the MongoDB driver to 1.20.0 to continue support for MongoDB versions 3.6 and lower.
Amazon S3 Apache Parquet timestamp datatype	Fixed an issue for Amazon S3 parquet target. AWS DMS now sets the format parameter `isAdjustedToUTC` to `true` to match the behavior in previous versions of AWS DMS.

Resolved Issue	Description
Amazon Redshift as a target copy command	Fixed an issue for Amazon Redshift as a target where the copy command failed for large tables when copying data from Amazon S3 to Amazon Redshift.
PostgreSQL geometry datatypes	Fixed an issue for PostgreSQL to PostgreSQL migrations where migration failed on large geometry datatypes.
Oracle to PostgreSQL XML	Fixed an issue where migration added an extra space on XML when replicating from Oracle to PostgreSQL.
Updating target checkpoint in supported engines	AWS DMS now updates the target checkpoint in the `awsdms_txn_state` table in the target database.
MongoDB/DocDB records sent to wrong collection	Fixed an issue for MongoDB/DocDB where data gets sent to the wrong target collection.
Oracle source new table selection with EscapeCharacter endpoint setting	Fixed an issue for Oracle source where AWS DMS was only picking up new tables for replication when the task was stopped and resumed while the `EscapeCharacter` endpoint setting was set.
CDC recovery checkpoint	Fixed an inconsistency in the CDC recovery checkpoint observed between the target datastore and AWS DMS console.
CDC validation-only tasks	Fixed an issue with CDC validation-only tasks where the task would not fail, even if all tables in the task encountered failures.
Validation behavior with source or target connection issues	Fixed an issue with data validation where AWS DMS would suspend tables on the source or target when the connection dropped.
Oracle to PostgreSQL data validation false positives	Fixed an issue with Oracle to PostgreSQL data validation where AWS DMS reported false positives. This is because the differences in the representation of source NULL characters on the target were not accounted for with text-based datatypes other than VARCHAR.

Resolved Issue	Description
Oracle to PostgreSQL data truncation	Fixed an issue with Oracle as a source and PostgreSQL as a target where AWS DMS was truncating data for NVARCHAR columns with the Oracle `NLS_NCHAR_CHARACTERSET` setting set to `AL16UTF16`.
Data validation error	Fixed an issue with data validation where an `unable to create where filter clause` error was thrown when both source filtering and an add-column transformation rule were in use.
Amazon Redshift target error handling	Fixed an issue with Amazon Redshift as a target where error handling did not function as configured when the CDC task had the `ParallelApplyThreads` task setting set to a value greater than zero.
Oracle as a source communication failure	Fixed an issue with Oracle as a source where the task remained in the `RUNNING` state, but was unable to migrate any data after a communication failure.
CDC table suspended with column filters	Fixed an issue with Full Load + CDC tasks where a table would be suspended during the CDC phase when column filters were applied.
S3 as a target data validation failure for special characters	Fixed an issue with S3 target data validation where the task would fail if the table name included a special character other than an underscore.
MongoDB source Full Load and CDC failure	Fixed an issue with MongoDB as a source where a Full Load + CDC task would fail during handling of cache events while migrating a large collection.
Upgrade issue with `BatchApplyEnabled` set to true	Fixed an issue where a task with the `BatchApplyEnabled` task setting set to true would fail after migrating from AWS DMS version 3.4.6 to 3.5.1 in some cases.

Resolved Issue	Description
SQL Server AlwaysOn source with case-sensitive collation	Fixed an issue with SQL Server AlwaysOn as a source where a task would fail with case-sensitive collation.
MySQL source task hanging	Fixed an issue with MySQL as a source where a task would hang instead of failing when the source was not properly configured.
S3 source full load task failure	Fixed an issue with S3 as a source where a task would fail on resume after upgrading from AWS DMS version 3.4.6 or 3.4.7 to version 3.5.1.
PostgreSQL source with CaptureDDLs set to false	Fixed an issue with PostgreSQL as a source where DDLs were not properly handled with the `CaptureDDLs` endpoint setting set to false.
Oracle source task crash during resume	Fixed an issue with Oracle as a source where a task would crash on resume due to incorrect data in the column name.
MySQL source LOB lookup failure	Fixed an issue with MySQL as a source where an LOB lookup would fail when the `ParallelApplyThreads` task setting was set to a value greater than zero.
SQL Server source illogical LSN error	Fixed an issue with SQL Server as a source where a task would fail with an `illogical LSN sequencing state error` error after upgrading from AWS DMS version 3.4.7 to version 3.5.1.
PostgreSQL source with pglogical	Fixed an issue with PostgreSQL as a source where a task using the pglogical plugin would fail when the task was stopped, a table was removed from selection rules, the task was resumed, and changes were made to the removed table.
Aurora MySQL incorrect recovery checkpoint.	Fixed an issue for Aurora MySQL as a source where an incorrect recovery checkpoint would be saved as a result of an Aurora failover or Aurora source stop and start.

Resolved Issue	Description
SQL Server as a source task crash.	Fixed an issue for SQL Server as a source where a task would crash when `SafeguardPolicy` was set to `RELY_ON_SQL_SERVER_REPLICATION_AGENT`.
Incorrect data type casting with MySQL as a target	Fixed an issue for MySQL as a target there where CDC replication would fail as a result of incorrect data type casting in the batch-apply phase.
Task failure with CaptureDDLs set to false for PostgreSQL as a source.	Fixed an issue for PostgreSQL as a source where a task would fail due to a DDL being treated as a DML when the `CaptureDDLs` endpoint setting was set to `false`.
MongoDB empty collection crash	Fixed an issue for MongoDB as a source where the task would crash due to an empty collection.
Amazon Redshift as a target full load task crash	Fixed an issue for Amazon Redshift as a target where a task would crash during the full load phase when the recovery checkpoint control table was enabled.
S3 to S3 - no data movement.	Fixed an issue for S3 to S3 replication where AWS DMS would not replicate the data if the `bucketFolder` was not specified.
CDC latency with `GlueCatalogGeneration` set to `true`	Fixed an issue for S3 as a target where excessive latency would occur when `GlueCatalogGeneration` was set to `true`.
Oracle as a target data truncation	Fixed an issue with Oracle as a target where AWS DMS truncates data in VARCHAR2 columns.
PostgreSQL underscore wildcard behavior	Fixed an issue for PostgreSQL as a source where the behavior of the '_' wildcard in the selection rules was not working as documented.
PostgreSQL as a source empty WAL header issue.	Fixed an issue for PostgreSQL as a source where the task would fail due to an empty WAL header received from the replication slot.

Resolved Issue	Description
MySQL or MariaDB as a source with compressed binary logs	Fixed an issue for MySQL and MariaDB as sources where a proper error message was not emitted when AWS DMS detected BINLOG compression.
S3 data validation special characters	Improved S3 data validation to handle special characters in primary and non-primary key columns.
Misleading task log entries with Amazon Redshift as a target.	Fixed an issue for Amazon Redshift as a target where misleadin g entires were present in the task log reporting batch-apply statement failures on UPDATES and DELETES.
SQL Server to S3 migration task crash.	Fixed an issue for SQL Server to S3 migrations where the task would crash while applying cached changes.
Missing data on batch-app ly errors.	Fixed an issue for the batch-apply feature where an error in applying a batch would result in missing data.
Logging enhancement for SQL Server source	Improved logging for SQL Server source to unclude the storage unit value. Enhanced logging for SQL Server source in `AlwaysOn` configuration to properly indicate missing permissions.
Logging enhancement for Kafka target	Several loggin enhancements introduced to provide better visibliti y and troubleshooting capalities for the Kafka target.
Logging enhancement for Oracle source.	Enhanced logging for Oracle source with binary reader to properly indicate tables being skipped due to missing primary keys.
Logging Enhancement for migrations with disabled DDL	Enhanced logging for migrations with disabled DDL replication to indicate unexpected target table structure after its modified outside of AWS DMS.
Logging enhancement for paused source capture.	Enhanced logging to better explain paused source capture situation.
Logging enhancement for AWS DMS swap	Enhanced logging to indicate when AWS DMS is reading from internal swap files.

Resolved Issue	Description
Logging enhancement for Amazon Redshift target	Enhanced logging for Amazon Redshift target to include more detailed information in default logging level.
Logging improvement for metadata manager	Enhanced logging to report issues with table matadata under info logging level to simplify troublshooting.
Data validation enhancement for Amazon Redshift target	Enhanced the data validation feature for Amazon Redshift target to support `HandleCollationDiff` setting.
Data validation re-validate option unavailable	Fixed an issue for data validation feature where the re-validation option was unavailable in certain situations.
Table reload issue	Fixed an issue where a reload of multiple tables was canceled when at least one of the table was invalid.
MySQL to Amazon Redshift JSON migration issue.	Fixed an issue for MySQL source where the JSON data type was not being hadled propely with Batch Apply enabled.
Column filtering issue	Fixed an issue for the column filtering feature where filters were not applied correctly to newly added columns during FL.
Db2 LUC source issue with selection rules.	Fixed an issue for Db2 LUW source where "table-type" option in selection rules was being ignored.
Data validation issue with filtered tasks.	Fixed an issue for data validation feature where filters were not respected while validating data.
LOB migration issue	Fixed an issue for LOB migration where the AWS DMS task would crash while processing certain event types.
Data validation issue	Fixed an issue for Data Validation feature where the Validation-only task would hang on certain DDL events.

Resolved Issue	Description
Data validation issue with filters	Fixed an issue for data validation feature where the the `HandleCollationDiff` setting was not applied when filters were present
MySQL SourceEnum encoding issue.	Fixed an issue for MySQL source where UTF-16 encoded enum values were not migrated correctly.
Data validation issue for SQL Server to APG migrations.	Fixed an issue for SQL Server to PostgreSQL migration where data validaiton would report false positives in certain situations.
EmptyAsNull ECA issue for Amazon Redshift target CDC	Fixed an issue for Amazon Redshift target where the EmptyAsNull ECA would not work correctly.
Memory leak issue	Fixed an issue where for targets using CSV files to load data DMS task was showing a memory leak.
Transformation rules with data validation	Fixed an issue for data validation feature where transformation rules were ignored when override rules existed.
S3 target issue	Fixed an issue for Amazon S3 target where `CdcMaxBatchInterval` and `CdcMinFileSize` were not respected when `cdcInsertAndUpdateOnly` setting was enabled.
MySQL target issue	Fixed an issue for MySQL target where corrupted column metadata could potentially cause AWS DMS task crash and/or data loss.
Data validation issue	Fixed an issue for data validation feature where the validation process would be terminated prematurely on any table supsension.
Oracle target batch apply issue	Fixed an issue for Oracle target where AWS DMS task would crash with Batch Apply enabled.

Resolved Issue	Description
Amazon S3 target data validation issue.	FIxed an issue for Amazon S3 target data validation where the taks would fail due to the Athena not storing the table names correctly.
MongoDB and Amazon DocumentDB Secret Manger issue.	Fixed an issue for MongoDB and Amazon DocumentDB endpoints where the credentials could not be retrieved from Secret Manager which resulted in an error.
Oracle data validation never completing issue.	Fixed an issue for Oracle data validation where validation of certain tables would never complete.
Data validation hanging issue	Fixed an issue for data validation feature where validation of certain tables would hang due to insufficient memory allocation.
Amazon S3 target task crash issue	Fixed an issue for Amazon S3 target where AWS DMS task would crash after receiving alter table DDL when `GlueCatalogGeneration` is enabled.
Data validation issue	Fixed an issue for data validation feature where validation would fail on `NUL (0x00)` characters.
Babelsifh endpoint issue	Fixed an issue for babelfish endpoint where tables names with mixed case would be suspended.
Amazon Redshift target missing data issue	Fixed an issue for Amazon Redshift target where data loss would occur when `ParallelLoadThreads` was >0 under certain conditions.
Amazon S3 target data validation issue	Fixed an issue for Amazon S3 target data validation where validation would fail when there were no other columns than the PK in the table.
Data validation CloudWatch metrics	Fixed an issue for data validation feature where the CloudWatch metrics would be missing for validation which took a short amount of time to complete.

Resolved Issue	Description
Memory leak issue in batch apply	Fixed a memory leak issue for batch apply feature which would occur under certain conditions.
AWS DMS task start issue	Fixed an issue where AWS DMS task start woudl take a very long time and never complete.
PostgreSQL souce missing data issue	Fixed an issue for PostgreSQL source where data loss would occur due to unknown events in the replication slot.
Amazon S3 target data loss issue	Fixed an issue for Amazon S3 target where failed LOB lookup would result in data loss.

AWS Database Migration Service 3.5.0 Beta release notes

> **Important**
>
> AWS DMS 3.5.0 is a beta version of the replication instance engine. AWS DMS supports this version the same as all previous releases. But we recommend that you test AWS DMS 3.5.0 Beta before using it for production purposes.

The following table shows the new features and enhancements introduced in AWS Database Migration Service (AWS DMS) version 3.5.0 Beta.

New feature or enhancement	Description
Time Travel for Oracle and Microsoft SQL Server	You can now use Time Travel in all AWS Regions with DMS-supported Oracle, Microsoft SQL Server, and PostgreSQL source endpoints, and DMS-supported PostgreSQL and MySQL target endpoints.
S3 validation	AWS DMS now supports validating replicated data in Amazon S3 target endpoints. For information about validating Amazon S3 target data, see Amazon S3 target data validation.

New feature or enhancement	Description
Glue Catalog Integration	AWS Glue is a service that provides simple ways to categorize data, and consists of a metadata repository known as AWS Glue Data Catalog. You can now integrate an AWS Glue Data Catalog with your Amazon S3 target endpoint and query Amazon S3 data through other AWS services such as Amazon Athena. For more information, see Using AWS Glue Data Catalog with an Amazon S3 target for AWS DMS.
Parallel apply for DocumentDB as a target	Using DocumentDB as the target with new `ParallelApply*` task settings, AWS DMS now supports a maximum of 5000 records per second during CDC replication. For more informati on, see Using Amazon DocumentDB as a target for AWS Database Migration Service.
Customer Centric Logging	You can now examine and manage task logs more effectively with AWS DMS version 3.5.0. For information about viewing and managing AWS DMS task logs, see Viewing and managing AWS DMS task logs.
SASL_PLAIN mechanism for Kafka target endpoints	You can now use SASL_PLAIN authentication to support Kafka MSK target endpoints.
Replication of XA transacti ons in MySQL	You can now use XA transactions on your MySQL DMS source. Prior to DMS 3.5.0, DML changes applied as part of XA transacti ons weren't replicated correctly.
Oracle Extended Data Types	AWS DMS now supports replication of extended data types in Oracle version 12.2 and higher.
Db2 LUW PureScale Environment	AWS DMS now supports replication from a Db2 LUW PureScale environment. This functionality is *only* supported using the **Start processing changes from source change position option**.

New feature or enhancement	Description
SQL Server source with `READ_COMMITTED_SNAPSHOT` option	When using a Microsoft SQL Server source database with the `READ_COMMITTED_SNAPSHOT` option set to `TRUE`, you can replicate DML changes correctly by setting the **forceDataRowLookup** connection attribute.

AWS DMS 3.5.0 includes the following resolved issues:

Issues resolved in AWS DMS 3.5.0 launched on 17-March-2023

Topic	Resolution	
Oracle—comparing special case for string that was converted from numeric	Fixed an issue for Oracle source where filtering rules weren't working as expected for a numeric column when data type transformation to string existed for the same column.	
On-premises SQL Server AG enhancements	Improved efficiency of connection handling with SQL Server source in **AlwaysOn** configuration by eliminating unnecessary connections to replicas that aren't used by DMS.	
SQL Server HIERARCHYID internal conversion	Fixed an issues with SQL Server Source where HIERARCHYID data type was replicated as VARCHAR(250) instead of HIERARCHYID to SQL Server target.	
S3 target move task fix	Fixed an issue when moving a task with an S3 target would take a very long time, appear frozen or never complete.	
SASL Plain mechanism of Kafka	Introduced support for SASL Plain authentication method for Kafka MSK target endpoint.	
Parallel Load/Apply fails due to _type	Fixed an issue for Opensearch 2.x target where parallel load or parallel apply would fail due to lack of support for _type parameter.	

Topic	Resolution
parameter with Opensearch 2.x	
Support table mapping filter with mixed operators	Removed a limitation where only one filter could be applied on a column.
S3, Kinesis, Kafka endpoints — alter-based lob columns migration in CDC phase	Fixed an issue for Kinesis, Kafka and S3 targets where data in LOB columns added during CDC wasn't replicated.
MongoDB driver upgrade	Upgraded the MongoDB driver to v1.23.2.
Kafka driver update	Upgraded the Kafka driver from 1.5.3 to 1.9.2.
S3 endpoint setting was not working properly	Fixed an issue for S3 target where the `AddTrailingPaddingCharacter` endpoint setting was not working when data contained the character specified as the delimiter for the S3 target.
Kinesis target task would crash	Fixed an issue for Kinesis target where a task would crash when PK value was empty and detailed debug was enabled.
When S3 targets' column names were shifted by one position	Fixed an issue for an S3 target where column names where shifted by one position when `AddColumnName` was set to `true` and `TimestampColumnName` was set to `""`.
Improved logging LOB truncation warning	Improved warning logging for LOB truncation for SQL Server source to include the select statement used to retrieve the LOB.

Topic	Resolution
Add Fatal error to avoid DMS task crashes if TDE password is wrong.	Introduced meaningful error message and eliminated task crash issue in situations where DMS task was failing with no error message due incorrect TDE password for Oracle as a source.
Allows PostgreSQL CTAS(Create table as selected) DDL's migration during CDC.	Removed limitations of DMS not being able to replicate PostgreSQL CTAS (create table as selected) DDLs during CDC.
Fix pg_logical task crash when table columns are dropped in CDC.	Fixed an issue for PostgreSQL source with S3 target where columns were misaligned on the target when support for LOBs was disabled and LOBs were present.
Fix memory leak in MySQL connection handling	Fixed an issue for MySQL source where task memory consumption was increasing continuously.
Oracle source endpoint setting – `ConvertTimestampWithZoneToUTC`	Set this attribute to `true` to convert the timestamp value of 'TIMESTAMP WITH TIME ZONE' and 'TIMESTAMP WITH LOCAL TIME ZONE' columns to UTC. By default the value of this attribute is 'false' and data is replicated using the source database timezone.
Oracle source - `DataTruncationErrorPolicy` to SUSPEND_TABLE not working	Fixed an issue for Oracle source with S3 target where tables were not suspended while the `DataTruncationErrorPolicy` task setting was set to SUSPEND_TABLE.

Topic	Resolution
SQL Server fail on long schema/table while building query clause	Fixed an issues for SQL Server source where task would fail or become unresponsive when selection rule contained comma separated list of tables.
Secret Manager authentication with MongoDB endpoint	Fixed an issue for MongoDB and DocumentDB endpoints where secret manager based authentication wasn't working.
DMS truncating the data during CDC for a multi-byte varchar column when NLS_NCHAR_CHARACTERSET is set to UTF8	Fixed an issues for Oracle source with Oracle target where data was being truncated for multi-byte VARCHAR columns with NLS_NCHAR_CHARACTERSET set to UTF8.
`filterTransactionsOfUser` ECA for Oracle LogMiner	Added an Extra Connection Attribute (ECA) `filterTransactionsOfUser` to allow DMS to ignore transactions from a specified user when replicating from Oracle using LogMiner.
SQL Server Setting recoverable error when lsn missing from backup	Fixed an issue for SQL Server where a task would not fail on missing LSN.

AWS Database Migration Service 3.4.7 release notes

The following table shows the new features and enhancements introduced in AWS Database Migration Service (AWS DMS) version 3.4.7.

New feature or enhancement	Description
Support Babelfish as a target	AWS DMS now supports Babelfish as a target. Using AWS DMS, you can now migrate live data from any AWS DMS supported source to a Babelfish, with minimal downtime. For more information, see Using Babelfish as a target for AWS Database Migration Service.
Support IBM Db2 z/OS databases as a source for full load only	AWS DMS now supports IBM Db2 z/OS databases as a source. Using AWS DMS, you can now perform live migrations from Db2 mainframes to any AWS DMS supported target. For more information, see Using IBM Db2 for z/OS databases as a source for AWS DMS.
Support SQL Server read replica as a source	AWS DMS now supports SQL Server read replica as a source. Using AWS DMS, you can now perform live migrations from SQL Server read replica to any AWS DMS supported target. For more information, see Using a Microsoft SQL Server database as a source for AWS DMS.
Support EventBridge DMS events	AWS DMS supports managing event subscriptions using EventBrid ge for DMS events. For more information, see Working with Amazon EventBridge events and notifications in AWS Database Migration Service.
Support VPC source and target endpoints	AWS DMS now supports Amazon Virtual Private Cloud (VPC) endpoints as sources and targets. AWS DMS can now connect to any AWS service with VPC endpoints when explicitly defined routes to the services are defined in their AWS DMS VPC. **Note** Upgrades to AWS DMS versions 3.4.7 and higher require that you first configure AWS DMS to use VPC endpoints

New feature or enhancement	Description
	or to use public routes. This requirement applies to source and target endpoints for Amazon S3, Amazon Kinesis Data Streams, AWS Secrets Manager, Amazon DynamoDB, Amazon Redshift, and Amazon OpenSearch Service. For more information, see Configuring VPC endpoints for AWS DMS.
New PostgreSQL version	PostgreSQL version 14.x is now supported as a source and as a target.
Support Aurora Serverless v2 as a target	AWS DMS now supports Aurora Serverless v2 as a target. Using AWS DMS, you can now perform live migrations to Aurora Serverless v2. For information about supported AWS DMS targets, see Targets for data migration.
New IBM Db2 for LUW versions	AWS DMS now supports IBM Db2 for LUW versions 11.5.6 and 11.5.7 as a source. Using AWS DMS, you can now perform live migrations from the latest versions of IBM DB2 for LUW. For information about AWS DMS sources, see Sources for data migration. For information about supported AWS DMS targets, see Targets for data migration.

AWS DMS 3.4.7 includes the following new or changed behavior and resolved issues:

- You can now use a date format from the table definition to parse a data string into a date object when using Amazon S3 as a source.
- New table statistics counters are now available: `AppliedInserts`, `AppliedDdls`, `AppliedDeletes`, and `AppliedUpdates`.

- You can now choose the default mapping type when using OpenSearch as a target.
- The new `TrimSpaceInChar` endpoint setting for Oracle, PostgreSQL, and SQLServer sources allows you to specify whether to trim data on CHAR and NCHAR data types.
- The new `ExpectedBucketOwner` endpoint setting for Amazon S3 prevents sniping when using S3 as a source or target.
- For RDS SQL Server, Azure SQL Server, and self-managed SQL Server — DMS now provides automated setup of MS-CDC on all tables selected for a migration task that are with or without a PRIMARY KEY, or with a unique index considering the enablement priority for MS-REPLICATION on self-managed SQL Server tables with PRIMARY KEY.
- Added support for replication of Oracle Partition and sub-partition DDL Operations during Oracle homogenous migrations.
- Fixed an issue where a data validation task crashes with a composite primary key while using Oracle as a source and target.
- Fixed an issue with correctly casting a varying character type to a boolean while the target column was pre-created as a boolean when using Amazon Redshift as a target.
- Fixed an issue that was causing data truncation for `varchar` data types migrated as `varchar(255)` due to a known ODBC issue when using PostgreSQL as a target.
- Fixed an issue where Parallel Hint for the DELETE operation wasn't respected with `BatchApplyEnabled` set to `true` and `BatchApplyPreserveTransaction` set to `false` when using Oracle as a target.
- The new `AddTrailingPaddingCharacter` endpoint setting for an Amazon S3 adds padding on string data when using S3 as a target.
- The new `max_statement_timeout_seconds` task setting extends the default timeout of endpoint queries. This setting is currently used by MySQL endpoint metadata queries.
- When using PostgreSQL as a target, fixed an issue where a CDC task wasn't properly utilizing the error handling task settings.
- Fixed an issue where DMS was unable to correctly identify Redis OSS mode for a Redis OSS Enterprise instance.
- Extended the support of `includeOpForFullLoad` extra connection attribute (ECA) for the S3 target parquet format.
- Introduced a new PostgreSQL endpoint setting `migrateBooleanAsBoolean`. When this setting is set to `true` for a PostgreSQL to Amazon Redshift migration, a boolean will be migrated as varchar(1). When it is set to `false`, a boolean is migrated as varchar(15), which is the default behavior.

- When using SQL Server source, fixed a migration issue with `datetime` datatype. This fix addresses the issue of inserting `Null` when precision is in milliseconds.
- For PostgresSQL source with PGLOGICAL, fixed a migration issue when using pglogical and removing a field from the source table during the CDC phase, where the value after the removed field wasn't migrated to the target table.
- Fixed a SQL Server Loopback migration issue with Bidirectional replication getting repeated records.
- Added a new ECA `mapBooleanAsBoolean` for PostgresSQL as a source. Using this extra connection attribute , you can override default data type mapping of a PostgresSQL Boolean to a RedShift Boolean data type.
- Fixed a migration issue when using SQL Server as source that addresses the ALTER DECIMAL/NUMERIC SCALE not replicating to targets.
- As of October 17, 2022, DMS 3.4.7 now supports Generation 6 Amazon EC2 instance classes for replication instances.
- As of November 25, 2022, with DMS 3.4.7 you can convert database schemas and code objects using **DMS Schema Conversion**, and discover databases in your network environment that are good candidates for migration using **DMS Fleet Advisor**.
- As of November 25, 2022, DMS Studio is retired.
- As of January 31, 2023, DMS Schema Conversion supports Aurora MySQL and Aurora PostgreSQL as a target data provider.
- As of March 6, 2023, you can generate right sized target recommendations for your source databases with DMS Fleet Advisor.
- As of March 6, 2023, AWS DMS supports the AWS managed policy that allows publishing metric data points to Amazon CloudWatch.

Issues resolved in the DMS 3.4.7 maintenance release dated 5-May-2023

Topic	Resolution
PostgreSQL source task failure	Fixed an issue for PostgreSQL source where tasks were failing when exceeding the maximum allowed DDL operations in a single event.

Topic	Resolution
PostgreSQL source data validation false positives	Fixed an issue for PostgreSQL source with Oracle target where incorrect casting of timestamp field resulted in false positive data validation errors.
MySQL source error handling	Fixed an issues for a MySQL source where the DMS task wasn't failing when the next BIN log was unavailable.
MySQL source ROTATE_EVENT logging	Improved logging for MySQL source related to ROTATE_EVENT – included the BIN log name being read.
Data validation timeout issue	Fixed an issue for Data Validation feature where the `executeTimeout` endpoint setting wasn't being respected for queries related to data validation.
PostgreSQL target parallel full load issue	Fixed an issue for PostgreSQL target where segmented (parallel) full load was failing due to "connection down" error.
DMS task move issue	Fixed an issue for S3 target where a DMS task move operation was taking a very long time or never completing.
PostgreSQL source duplicate record issue	Fixed an issue for PostgreSQL source where a DMS task would throw errors related to duplicates on the target after a task stop and resume.
Oracle target data validation false positives	Fixed an issue for Oracle target where data validation would report false positive errors due to incorrectly replicated timezone for timestamp fields.

Issues resolved in the DMS 3.4.7 maintenance release dated 22-February-2023

Topic	Resolution
SQL Server AG replicas as a source	Added support for SQL Server source in **AlwaysOn** configuration where the listener TCP port differed from replica TCP port.
Data loss with Amazon Redshift as a target	Fixed an issue for Amazon Redshift target where in some rare cases unexpected Amazon Redshift restart could have caused missing data on target.
SQL Server source safeguard support	Fixed an issue for SQL Server source where the DMS task could fail with an error indicating inability to read transaction log backups when Endpoint Setting `"SafeguardPolicy": "EXCLUSIVE_AUTOMATIC_TRUNCATION"` is specified.
Data validation task failure for Oracle as a source	Fixed an issue for Oracle source where DMS task could fail on data validation due to incorrectly identified Primary Key values.
Kinesis before image data issue	Fixed an issue for streaming targets (Kinesis, Kafka) where the `"EnableBeforeImage"` task setting was working only for character data types.
Time Travel log files	Fixed an issue for the Time Travel feature where DMS was creating zero byte time travel log files when the source is idle.

Issues resolved in the DMS 3.4.7 maintenance release dated 16-December-2022

Topic	Resolution
BatchApplyEnabled	Fixed an issue of excessive logging when BatchApplyEnabled is set to True.
New MongoDB endpoint setting–F	MongoDB endpoint setting FullLoadNoCursorTimeout specifies NoCursorTimeout for the full load cursor.

Topic	Resolution
ullLoadNoCursorTimeout	NoCursorTimeout is a MongoDB connection setting that prevents the server from closing the cursor if idle.
MongoDB—Filter function for single column segmentation	New filter function improves performance for migrating MongoDB databases using a single column for segmentation.
MongoDB to Amazon Redshift	When migrating from MongoDB to Amazon Redshift, if the MongoDB collection has binary data type, fixed an issue where DMS was not creating the target table on Amazon Redshift.
New MongoDB SocketTimeoutMS connection attribute	New MongoDB SocketTimeoutMS extra connection attribute configures the connection timeout for MongoDB clients in units of milliseconds. If the value is less than or equal to zero, then the MongoDB client default is used.
Fixed issue causing an Amazon Kinesis task to crash	When migrating to Amazon Kinesis Data Streams as a target, fixed an issue handling null values if a primary key wasn't present in the table.
Oracle NULL PK/UK data validation supported	Removed the limitation that data validation of NULL PK/UK values aren't supported.
Oracle to Amazon S3	When migrating from Oracle to Amazon S3, fixed an issue where a few records were incorrectly migrated as NULL.
Oracle Standby	When using Oracle Standby as a source, added the ability for DMS to handle open transactions.
Oracle to Oracle migration with SDO_GEOMETRY spatial data type	When migrating from Oracle to Oracle, fixed an issue where the task failed if the table had an SDO_GEOMETRY column present in the DDL.

Topic	Resolution
Oracle as source	When using Oracle as a source, fixed an issue where DMS occasionally skips an Oracle redo log sequence number.
Oracle as source—mi ssing archive/online redo logs	When using Oracle as a source, fixed an issue so that the DMS task fails when archive logs are missing.
Fixed—DMS occasionally skips Oracle standby redo log	When using Oracle as a source, fixed an issue where DMS occasionally skips an Oracle redo log sequence number.
Fixed—Oracle to Oracle spatial datatypes not replicating during CDC	When replicating from Oracle to Oracle, fixed an issue where spatial datatypes were not replicating during CDC.
Oracle as target	When using Oracle as a target, fixed an issue where the target apply was failing with an ORA-01747 error.
Amazon S3—Fixed reload table data loss	When using Amazon S3 as a target, fixed an issue where a table reload operation wasn't generating CDC files.
Fixed—SQL Server Always On context initialization in case primary server as a source	When using SQL Server Always On as a source, fixed an issue to not initialize Availability Groups (AG) if the source is primary and AlwaysOnSharedSyncedBackupI sEnabled is set to true.
Updated SQL Server endpoint setting	When a source endpoint is SQL Server Always On Availability Group and is a secondary replica, fixed an issue where the replication task fails if AlwaysOnS haredSynchedBackupsIsEnabled is set to True.

Topic	Resolution
PostgreSQL as source	Fixed an issue where CDC fails to migrate delete/up date operations on the PostgreSQL source, which was introduced in 3.4.7 in supporting mapBoolea nAsBoolean.

AWS Database Migration Service 3.4.6 release notes

The following table shows the new features and enhancements introduced in AWS Database Migration Service (AWS DMS) version 3.4.6.

New feature or enhancement	Description
AWS DMS Time Travel	AWS DMS introduces Time Travel, a feature granting customers flexibility on their logging capabilities, and enhancing their troubleshooting experience. With Time Travel, you can store and encrypt AWS DMS logs using Amazon S3, and view, download and obfuscate the logs within a certain time frame.
Support Microsoft Azure SQL Managed Instance as a source	AWS DMS now supports Microsoft Azure SQL Managed Instance as a source. Using AWS DMS, you can now perform live migration s from Microsoft Azure SQL Managed Instance to any AWS DMS supported target. For information about AWS DMS sources, see Sources for data migration. For information about supported AWS DMS targets, see Targets for data migration.
Support Google Cloud SQL for MySQL as a source	AWS DMS now supports Google Cloud SQL for MySQL as a source. Using AWS DMS, you can now perform live migrations from Google Cloud SQL for MySQL to any AWS DMS supported target.

New feature or enhancement	Description
	For information about AWS DMS sources, see Sources for data migration. For information about supported AWS DMS targets, see Targets for data migration.
Support parallel load for partitioned data to S3	AWS DMS now supports parallel load for partitioned data to Amazon S3, improving the load times for migrating partitioned data from supported database engine source data to Amazon S3. This feature creates Amazon S3 sub-folders for each partition of the table in the database source, allowing AWS DMS to run parallel processes to populate each sub-folder.
Support multiple Apache Kafka target topics in a single task	AWS DMS now supports Apache Kafka multi-topic targets with a single task. Using AWS DMS, you can now replicate multiple schemas from a single database to different Apache Kafka target topics using the same task. This eliminates the need to create multiple separate tasks in situations where many tables from the same source database need to be migrated to different Kafka target topics.

The issues resolved in AWS DMS 3.4.6 include the following:

- Fixed an issue where columns from UPDATE statements were populated to incorrect columns if the primary key column is not the first column when using Amazon S3 as a target with CSV format.
- Fixed an issue where AWS DMS tasks might crash when using the pglogical plugin with `NULL` values in `BYTEA` columns under limited LOB mode when using PostgreSQL as a source.
- Fixed an issue where AWS DMS tasks might crash when a large number of source tables are deleted when using PostgreSQL as a source.
- Improved Amazon S3 date-based folder partitioning by introducing a new Amazon S3 setting `DatePartitionTimezone` to allow partitioning on non-UTC dates.

- Supported the mapping between data type `TIMESTAMP WITH TIME ZONE` from sources to `TIMESTAMPTZ` when using Amazon Redshift as a target
- Improved the performance of CDC for tasks without wildcard selection rules when using MongoDB or Amazon DocumentDB as a source.
- Fixed an issue where schema names with underscore wildcard and length less than 8 were not captured by AWS DMS tasks when using Db2 LUW as a source.
- Fixed an issue where AWS DMS instances ran out of memory under large data volume when using OpenSearch Service as a target.
- Improved the performance of data validation by supporting full load validation only tasks.
- Fixed an issue where AWS DMS tasks failed to resume after forced failover when using Sybase as a source.
- Fixed an issue where AWS DMS sent warning `Invalid BC timestamp was encountered in column` incorrectly.

Issues resolved in the DMS 3.4.6 maintenance release include the following:

- Fixed an issue of a task crashing when bulk apply mode is enabled when using Oracle as the source and target.
- Fixed an issue so that a full load task properly uses the `ExecuteTimeout` endpoint setting with PostgreSQL as source.
- Fixed an issue with migrating Array data type columns when the task is set to limited LOB mode while using PostgreSQL as a source.
- Fixed an issue with migrating timestamps with time zone before 1970-01-01 when using PostgreSQL as a source.
- Fixed an issue where DMS was treating an empty string as null during replication when using SQL Server as a source and target.
- Fixed an issue to honor session read and write timeout endpoint settings when using MySQL source/target.
- Fixed an issue where a DMS CDC task was downloading full load related files when using Amazon S3 as a source.
- Fixed a log crashing issue when `CdcInsertsAndUpdates` and `PreserveTransactions` are both set to `true` when using Amazon S3 as a target.
- Fixed an issue where a task crashed when the ParallelApply* feature is enabled, but some tables don't have a default primary key when using Amazon Kinesis Data Streams as a source.

- Fixed an issue where an error wasn't given for an incorrect StreamArn when using Amazon Kinesis Data Streams as a source.
- Fixed an issue where a primary key value as an empty string would cause a task to crash when using OpenSearch as a target.
- Fixed an issue where too much disk space was used by data validation.

Issues resolved in the DMS 3.4.6 maintenance release dated 13-December-2022

Topic	Resolution
SAP ASE odbc driver	Fixed an issue for SAP ASE as a source so that the ODBC driver can support character sets.
Sqlserver datetime primary key bug for lob lookup	Fixed an issue for SQL Server as a source where LOB lookup was not working properly, when primary key has a datetime datatype, with precision in milliseco nds.
SQL Server to Amazon Redshift–'datetimeoffset' mapped to 'timestam ptz'	For migrations from SQL Server to Amazon Redshift, improved mapping so that the SQL Server 'datetime offset' format is mapped to the Amazon Redshift 'timestamptz' format.
Data Validation - SkipLobColumns is True	Fixed an issue where DMS task crashes when SkipLobColumns is True, there is a LOB on the source, the Primary Key is on the last column, and a data difference is detected by validation.
Data Validation with MySQL as source	Fixed an issue for MySQL as a source with data validation enabled, where a DMS task crash occurs while using a table that has a composite unique key that has null values.
MySQL as source	Fixed an issue for MySQL as a source, where a table is getting suspended with Overflow error when the columns are altered to add precision.

Topic	Resolution
Upgrade MySQL ODBC driver to 8.0.23	Fixed an issue for MySQL as a source, where the collation ‘utf8mb4_0900_bin' was incompatible with the mysql driver used by DMS.
MySQL–support DDL changes for partition ed tables	Introduced a new MySQL endpoint setting skipTable SuspensionForPartitionDdl to allow the user to skip table suspension for partition DDL changes during CDC, so that DMS can now support DDL changes for partitioned MySQL tables.
MongoDB to Amazon Redshift migration	Fixed an issue for MongoDB to Amazon Redshift migrations, where DMS fails to create the target table on Amazon Redshift if the MongoDB collection has binary data type.
Amazon Redshift target–Time Travel Segfault in Bulk Apply	Fixed an issue for Amazon Redshift as a target, where DMS task crashes with BatchApplyEnabled set to true.
Amazon Redshift as target	Fixed an issue for Amazon Redshift as a target, where with parallel-load set to type=partitions-auto, parallel segments were writing bulk CSV files to the same table directory and interfering with each other.
Amazon Redshift as target	Fixed an issue with Amazon Redshift as a target, where during CDC the target column is of type boolean while the source is of type character varying.
Amazon Redshift as target	Improved the task log to identify a DDL change that fails to replicate to Amazon Redshift as a target.
Data validation with PostgreSQL	Fixed an issue for validation with PostgreSQL, where the validation fails when boolean data types are present.

Topic	Resolution
PostgreSQL as source	Fixed an issue for PostgreSQL as a source, so that the full load uses the ExecuteTimeout field in Extra connection attributes.
PostgreSQL as source	Fixed an issue for PostgreSQL as a source, so that a task will fail if it's reading LSNs which are greater than the requested task resume LSN for more than 60 min to indicate that the is a problem with the replication slot being used.
PostgreSQL as a source–timestamptz before 1970-01-01	Fixed an issue for PostgreSQL as a source, where timestamptz before 1970-01-01 were not migrated correctly during CDC.
PostgreSQL as source	Fixed an issue for PostgreSQL as a source, where DMS was truncating character varying datatype values during CDC.
PostgreSQL as a source–resuming stopped task	Fixed an issue for PostgreSQL as a source where resuming a previously stopped task replay misses one or more transactions during CDC.
Amazon S3 as target	Fixed an issue for S3 as a target, where the resultant CSV file header is off by one column when AddColumn Name is true and TimestampColumnName is "".
Amazon S3 as source–memory usage behavior in full load phase for task	Fixed an issue for S3 as a source, where a DMS task in full load was only releasing the used memory after the entire table was loaded to the target database.
Amazon S3 as target–table reload operation	Fixed an issue for S3 as a target, where a table reload operation missed generating CDC files.

AWS Database Migration Service 3.4.5 release notes

The following table shows the new features and enhancements introduced in AWS Database Migration Service (AWS DMS) version 3.4.5.

New feature or enhancement	Description
Support for Redis OSS as a target	AWS DMS now supports Redis OSS as a target. Using AWS DMS, you can now migrate live data from any AWS DMS supported source to a Redis OSS data store, with minimal downtime. For information about AWS DMS targets, see Targets for data migration.
Support for MongoDB 4.2 and 4.4 as sources	AWS DMS now supports MongoDB 4.2 and 4.4 as sources. Using AWS DMS, you can now migrate data from MongoDB 4.2 and 4.4 clusters to any AWS DMS supported target including Amazon DocumentDB (with MongoDB compatibility), with minimal downtime. For information about AWS DMS sources, see Sources for data migration.
Support for multiple databases using MongoDB as a source	AWS DMS now supports migrating multiple databases in one task using MongoDB as a source. Using AWS DMS, you can now group multiple databases of a MongoDB cluster, and migrate them using one database migration task. You can migrate to any AWS DMS supported target, including Amazon DocumentDB (with MongoDB compatibility), with minimal downtime.
Support for automatic segmentation using MongoDB or Amazon DocumentDB (with MongoDB compatibility) as a source	AWS DMS now supports automatic segmentation using MongoDB or Amazon DocumentDB as a source. Using AWS DMS, you can configure database migration tasks to segment the collection of a MongoDB or DocumentDB cluster automatically. You can then migrate the segments in parallel to any AWS DMS supported target, including Amazon DocumentDB, with minimal downtime.
Amazon Redshift full load performance improvement	AWS DMS now supports using parallel threads when using Amazon Redshift as a target during full load. By taking advantage of the multithreaded full load task settings, you can improve

New feature or enhancement	Description
	the performance of your initial migration from any AWS DMS supported source to Amazon Redshift. For information about AWS DMS targets, see Targets for data migration.

The issues resolved in AWS DMS 3.4.5 include the following:

- Fixed an issue where data could potentially be missing or duplicated after resuming, when using PostgreSQL as a source with high transaction concurrency.
- Fixed an issue where database migration tasks fail with error **Could not find relation id ...** when using PostgreSQL as a source, with the pglogical plugin enabled.
- Fixed an issue where `VARCHAR` columns are not replicated correctly when using PostgreSQL as a source and Oracle as a target.
- Fixed an issue where delete operations are not properly captured when the primary key is not the first column in the table definition, when using PostgreSQL as a source.
- Fixed an issue where database migration tasks miss LOB updates in a special metadata setting when using MySQL as a source.
- Fixed an issue where `TIMESTAMP` columns are treated as `DATETIME` in full LOB mode when using MySQL version 8 as a source.
- Fixed an issue where database migration tasks fail when parsing `NULL DATETIME` records when using MySQL 5.6.4 and higher as a source.
- Fixed an issue where database migration tasks get stuck after encountering a **Thread is exiting** error when using Amazon Redshift as a target with parallel apply.
- Fixed an issue where data could potentially be lost, when database migration tasks disconnect with a Amazon Redshift target endpoint during batch-apply CDC.
- Improved the performance of full load by calling `ACCEPTINVCHARS` when using Amazon Redshift as a target.
- Fixed an issue where duplicated records are replicated when reverting from one-by-one mode to parallel apply mode using Amazon Redshift as a target.
- Fixed an issue where database migration tasks do not switch Amazon S3 object ownership to bucket owner with `cannedAclForObjects=bucket_owner_full_control` when using Amazon S3 as a target.

- Improved AWS DMS by supporting multiple archive destinations with ECA `additionalArchivedLogDestId` when using Oracle as a source.
- Fixed an issue where database migration tasks fail with error `OCI_INVALID_HANDLE` while updating a LOB column in full LOB mode.
- Fixed an issue where `NVARCHAR2` columns are not migrated properly during CDC when using Oracle as a source.
- Improved AWS DMS by enabling `SafeguardPolicy` when using RDS for SQL Server as a source.
- Fixed an issue where database migration tasks report error on `rdsadmin` when using a non-RDS SQL Server source.
- Fixed an issue where data validation fails with UUID as the primary key in a partition setting when using SQL Server as a source.
- Fixed an issue where full load plus CDC tasks might fail if the required LSN cannot be found in the database log when using Db2 LUW as a source.
- Improved AWS DMS by supporting custom CDC timestamps when using MongoDB as a source.
- Fixed an issue where database migration tasks get stuck when stopping, using MongoDB as a source, when the MongoDB driver errors on `endSessions`.
- Fixed an issue where AWS DMS fails to update non-primary fields when using DynamoDB as a target
- Fixed an issue where data validation reports false positive mismatches on `CLOB` and `NCLOB` columns.
- Fixed an issue where data validation fails on whitespace-only records when using Oracle as a source.
- Fixed an issue where database migration tasks crash when truncating a partitioned table.
- Fixed an issue where database migration tasks fail when creating the `awsdms_apply_exceptions` control table.
- Extended support of the `caching_sha2_password` authentication plugin when using MySQL version 8.

AWS Database Migration Service 3.4.4 release notes

The following table shows the new features and enhancements introduced in AWS DMS version 3.4.4.

New feature or enhancement	Description
Support TLS encryption and TLS or SASL authentic ation using Kafka as a target	AWS DMS now supports TLS encryption and TLS or SASL authentication using Amazon MSK and on-premises Kafka cluster as a target. For more information on using encryption and authentication for Kafka endpoints, see Connecting to Kafka using Transport Layer Security (TLS).

The issues resolved in AWS DMS 3.4.4 include the following:

- Improved AWS DMS logging on task failures when using Oracle endpoints.
- Improved AWS DMS task execution continues processing when Oracle source endpoints switch roles after Oracle Data Guard fail over.
- Improved error handling treats ORA—12561 as a recoverable error when using Oracle endpoints.
- Fixed an issue where `EMPTY_BLOB()` and `EMPTY_CLOB()` columns are migrated as null when using Oracle as a source.
- Fixed an issue where AWS DMS tasks fail to update records after add column DDL changes when using SQL Server as a source.
- Improved PostgreSQL as a source migration by supporting the `TIMESTAMP WITH TIME ZONE` data type.
- Fixed an issue where the `afterConnectScript` setting does not work during a full load when using PostgreSQL as a target.
- Introduced a new `mapUnboundedNumericAsString` setting to better handle the `NUMERIC` date type without precision and scale when using PostgreSQL endpoints.
- Fixed an issue where AWS DMS tasks fail with "0 rows affected" after stopping and resuming the task when using PostgreSQL as a source.
- Fixed an issue where AWS DMS fails to migrate the `TIMESTAMP` data type with the BC suffix when using PostgreSQL as a source.
- Fixed an issue where AWS DMS fails to migrate the `TIMESTAMP` value "±infinity" when using PostgreSQL as a source.
- Fixed an issue where empty strings are treated as `NULL` when using S3 as a source with the `csvNullValue` setting set to other values.

- Improved the `timestampColumnName` extra connection attribute in a full load with CDC to be sortable during CDC when using S3 as a target.
- Improved the handling of binary data types in hex format such as `BYTE`, `BINARY`, and `BLOB` when using S3 as a source.
- Fixed an issue where deleted records are migrated with special characters when using S3 as a target.
- Fixed an issue to handle empty key values when using Amazon DocumentDB (with MongoDB compatibility) as a target.
- Fixed an issue where AWS DMS fails to replicate `NumberDecimal` or `Decimal128` columns when using MongoDB or Amazon DocumentDB (with MongoDB compatibility) as a source.
- Fixed an issue to allow CDC tasks to retry when there is a fail over on MongoDB or Amazon DocumentDB (with MongoDB compatibility) as a source.
- Added an option to remove the hexadecimal “0x” prefix to `RAW` data type values when using Kinesis, Kafka, or OpenSearch as a target.
- Fixed an issue where validation fails on fixed length character columns when using Db2 LUW as a source.
- Fixed an issue where validation fails when only the source data type or the target data type is `FLOAT` or `DOUBLE`.
- Fixed an issue where validation fails on `NULL` characters when using Oracle as a source.
- Fixed an issue where validation fails on XML columns when using Oracle as a source.
- Fixed an issue where AWS DMS tasks crash when there are nullable columns in composite keys using MySQL as a source.
- Fixed an issue where AWS DMS fails to validate both `UNIQUEIDENTIFIER` columns from SQL Server source endpoints and UUID columns from PostgreSQL target endpoints.
- Fixed an issue where a CDC task does not use an updated source table definition after it is modified.
- Improved AWS DMS fail over to treat task failures caused by an invalid user name or password as recoverable errors.
- Fixed an issue where AWS DMS tasks fail because of missing LSNs when using RDS for SQL Server as a source.

AWS Database Migration Service 3.4.3 release notes

The following table shows the new features and enhancements introduced in AWS DMS version 3.4.3.

New feature or enhancement	Description
New Amazon DocumentDB version	Amazon DocumentDB version 4.0 is now supported as a source.
New MariaDB version	MariaDB version 10.4 is now supported as both a source and target.
Support for AWS Secrets Manager integration	You can store the database connection details (user credentials) for supported endpoints securely in AWS Secrets Manager. You can then submit the corresponding secret instead of plain-text credentials to AWS DMS when you create or modify an endpoint. AWS DMS then connects to the endpoint databases using the secret. For more information on creating secrets for AWS DMS endpoints, see Using secrets to access AWS Database Migration Service endpoints.
Larger options for C5 and R5 replication instances	You can now create the following larger replication instance sizes: C5 sizes up to 96 vCPUs and 192 GiB of memory and R5 sizes up to 96 vCPUs and 768 GiB of memory.
Amazon Redshift performance improvement	AWS DMS now supports parallel apply when using Amazon Redshift as a target to improve the performance of on-going replication. For more information, see Multithreaded task settings for Amazon Redshift.

The issues resolved in AWS DMS 3.4.3 include the following:

- Fixed an issue where commit timestamp became “1970-01-01 00:00:00” for deferred events when using Db2 LUW as a source.

- Fixed an issue where AWS DMS tasks failed with an `NVARCHAR` column as primary key when using SQL Server as a source with Full LOB mode.
- Fixed an issue of missing records during cached changes phase when using SQL Server as a source.
- Fixed an issue where records were skipped after AWS DMS tasks were resumed when using RDS for SQL Server as a source.
- Fixed an issue where AWS DMS ASSERTION logging component generates large logs for SQL Server.
- Fixed an issue where data validation failed during CDC phase due to column parsing overflow when using MySQL as a source.
- Fixed an issue where AWS DMS tasks crashed due to a segmentation fault during data validation when using PostgreSQL as a target.
- Fixed an issue where data validation failed on DOUBLE data type during CDC when using PostgreSQL as a source and a target.
- Fixed an issue where records inserted by copy command were not replicated correctly when using PostgreSQL as a source and Amazon Redshift as a target.
- Fixed a data loss issue during cached changes phase when using PostgreSQL as a source.
- Fixed an issue which could potentially cause either data loss or record duplicates when using PostgreSQL as a source.
- Fixed an issue where schemas with mixed cases failed to migrate with pglogical when using PostgreSQL as a source.
- Fixed an issue where the Last Failure Message did not contain the ORA error when using Oracle as a source.
- Fixed an issue where AWS DMS tasks failed to build UPDATE statements when using Oracle as a target.
- Fixed an issue where AWS DMS tasks did not replicate data when using Oracle 12.2 as a source with ASM and Pluggable Database configuration.
- Improved record parsing by preserving quotes to be compliant with RFC 4180 when using S3 as a source.
- Improved the handling of `timestampColumnName` so that the column from Full Load is sortable with that from CDC.
- By introducing a new endpoint setting `MessageMaxBytes`, fixed an issue where AWS DMS tasks failed when there are LOB elements larger than 1MB .

- Fixed an issue where AWS DMS tasks crashed due to a segmentation fault when using Amazon Redshift as a target.
- Improved error logging for Amazon Redshift test connection.
- Fixed an issue where AWS DMS did not transfer all documents from MongoDB to DocumentDB during Full Load.
- Fixed an issue where AWS DMS tasks reported fatal error when no tables were included in the table mapping rules.
- Fixed an issue where schemas and tables created before restarting AWS DMS tasks did not replicate to the target when using MySQL as a source.
- Fixed an issue where wildcard escape [_] cannot escape wildcard "_" in exclude rule when using MySQL as a source.
- Fixed an issue where column of data type UNSIGNED BIGINT did not replicate correctly when using MySQL as a source.

AWS Database Migration Service 3.4.2 release notes

The following table shows the new features and enhancements introduced in AWS DMS version 3.4.2.

New feature or enhancement	Description
Support for privately connecting your Amazon Virtual Private Cloud (Amazon VPC) to AWS Database Migration Service (DMS) without requiring an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection.	You can now connect to and access AWS DMS from your Amazon VPC through a VPC interface endpoint that you create. This interface endpoint allows you to isolate all network activity of your AWS DMS replication instance within the Amazon network infrastructure. By including a reference to this interface endpoint in all API calls to AWS DMS using the AWS CLI or an SDK, you ensure that all AWS DMS activity remains invisible to the public Internet. For more information, see Infrastructure security in AWS Database Migration Service.

New feature or enhancement	Description
	Note This feature is available using all supported AWS DMS engine versions.
CDC date-based folder partitioning using Amazon S3 as a target	AWS DMS now supports date-based folder partitioning when replicating data using S3 as a target. For more information, see [Using date-based folder partitioning]().

The issues resolved in AWS DMS 3.4.2 include the following:

- Added a `STATUPDATE` option when performing a migration using Amazon Redshift as a target.
- Improved validation tasks by introducing a new setting. `ValidQueryCdcDelaySecond` delays the first validation query on both source and target endpoints to help reduce resource contention when migration latency is high.
- Fixed an issue where AWS DMS took a long time to start validation tasks.
- Fixed an issue where empty records were generated when starting or stopping replication tasks using S3 as a target.
- Fixed an issue where tasks got stuck after a full load completed.
- Fixed an issue where tasks got stuck when a source table has data errors while using S3 as a source .
- Fixed an issue where tasks got stuck while starting when the user account of the source endpoint is disabled.
- Fixed an issue where tasks crashed when using PostgreSQL as a source with `REPLICA IDENTITY FULL`.
- Fixed an issue where tasks missed transactions when using PostgreSQL as a source with **pglogical** plugin.
- Fixed an issue when AWS DMS didn't delete compressed source files when using Amazon Redshift as a target.
- Fixed an issue where validation tasks reported false negatives when using MySQL as both source and target with data type `BIGINT UNSIGNED`.

- Fixed an issue where validation tasks reported false positives when using SQL Server as a source with a primary key column as CHAR type.
- Fixed an issue where AWS DMS doesn't clear target objects when using `start-replication` to start replication tasks using S3 as a target.
- Fixed several issues on data validation when using Db2 as a source.
- Fixed an issue where validation tasks got stuck when using SQL Server as a source with VARCHAR column as primary key.
- Added support for data type TIMESTAMP WITH TIMEZONE when using PostgreSQL as a source

AWS Database Migration Service 3.4.1 Beta release notes

The following table 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 Amazon 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 following table 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. **Note** 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.
- 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 OpenSearch Service (OpenSearch Service) as a target.
- Improved OpenSearch Service 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. This can result in data loss if the polling interval parameter is set incorrectly. For more information on how to setup a recommended polling interval value, see Using a Microsoft SQL Server database as a source for AWS DMS.
- 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 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 following table 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 OpenSearch Service as target	As of AWS DMS 3.3.3, use of a composite primary key is supported by OpenSearch Service targets.
Support for Oracle extended data types	Oracle extended data types for both Oracle source and targets are now supported.

New feature or enhancement	Description
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 Quotas for AWS Database Migration Service.

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.

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.

Change	Description	Date
AWS DMS added support for a new AWS managed role	AWS DMS updated `AWSDMSServerlessServiceRolePolicy` to allow DMS to create S3 buckets and put premigration assessment results into those buckets for replication tasks not related to DMS serverless.	November 5, 2025
End of support notice	End of support notice: On May 20, 2026, AWS will end support for AWS Database Migration Service Fleet Advisor. After May 20, 2026, you will no longer be able to access the AWS DMS Fleet Advisor console or AWS DMS Fleet Advisor resources. For more information, see AWS DMS Fleet Advisor end of support.	May 20, 2025
AWS DMS updated support for an AWS service-linked role	AWS DMS updated `AWSDMSServerlessServiceRolePolicy` to allow `dms:StartReplicationTaskAss`	February 14, 2025

	`essmentRun` to support running premigration assessments. AWS DMS also updated the serverless service-linked role to create S3 buckets and put the premigration assessment results into those buckets.	
AWS DMS added support for a new AWS managed policy	AWS DMS added `dms:ModifyReplicationTask` which is required by AWS DMS Serverless to call the `ModifyReplicationTask` operation to modify a replication task. AWS DMS added `dms:ModifyReplicationInstance` which is required by AWS DMS Serverless to call `ModifyReplicationInstance` operation to modify a replication instance.	January 17, 2025
AWS DMS added support for enhanced monitoring dashboard	AWS DMS now includes the enhanced monitoring dashboard that provides comprehensive visibility into critical metrics for your data migration tasks and replication instances.	September 19, 2024
AWS DMS added support for RDS IBM DB2 as a target	AWS DMS now supports using Amazon RDS IBM DB2 as a target.	December 4, 2023

AWS DMS added support for Timestream as a target.	AWS DMS now supports Timestream as a target.	November 17, 2023
AWS DMS added support for Redshift target data validation	AWS DMS now supports validating data in Redshift targets.	November 14, 2023
AWS DMS added support for four new endpoint types	AWS DMS now supports using Microsoft Azure Database for PostgreSQL, Microsoft Azure Database for MySQL, OCI MySQL Heatwave, and Google Cloud for PostgreSQL as a source.	October 26, 2023
AWS DMS added support for a new AWS service-linked role	AWS DMS now supports the AWS service-linked role `AWSServiceRoleForDMSServerless` that allows AWS DMS to create and manage resources on your behalf, such as publishing metric data points to Amazon CloudWatch.	May 22, 2023
AWS DMS added support for a new AWS managed policy	AWS DMS now supports the AWS managed policy that allows publishing serverless replication logs to CloudWatch Logs.	May 22, 2023

AWS DMS added support for a new AWS managed policy	AWS DMS now supports the AWS managed policy that allows publishing metric data points to Amazon CloudWatch. Also, AWS DMS started tracking changes for its AWS managed policies.	March 6, 2023
Support VPC source and target endpoints	AWS DMS now supports virtual private cloud (VPC) endpoints as sources and targets. AWS DMS can now connect to any AWS service with VPC endpoints when explicitly defined routes to the services are defined in their AWS DMS VPC.	June 30, 2022
Support SQL Server read replica as a source	AWS DMS now supports SQL Server read replica as a source. Using AWS DMS, you can now perform live migrations from SQL Server read replica to any AWS DMS supported target.	June 30, 2022
Support IBM Db2 z/OS databases as a source for full load only	AWS DMS now supports IBM Db2 z/OS databases as a source. Using AWS DMS, you can now perform live migrations from Db2 mainframes to any AWS DMS supported target.	June 30, 2022
Support EventBridge DMS events	AWS DMS supports managing event subscriptions using EventBridge for DMS events.	June 30, 2022

Support Babelfish as a target	AWS DMS now supports Babelfish as a target. Using AWS DMS, you can now migrate live data from any AWS DMS supported source to a Babelfish, with minimal downtime.	June 30, 2022
Support Aurora Serverless v2 as a target	AWS DMS now supports Aurora Serverless v2 as a target. Using AWS DMS, you can now perform live migrations to Aurora Serverless v2.	June 30, 2022
Getting Started Tutorial	An update for the Getting Started Tutorial for AWS DMS. The tutorial uses a MySQL database as a source and a PostgreSQL database as a target.	May 20, 2021
Support for Amazon Neptune as a target	Added support for Amazon Neptune as a target for data migration.	June 1, 2020
Support for Apache Kafka as a target	Added support for Apache Kafka as a target for data migration.	March 20, 2020
Updated security content	Updated and standardized security content as a response to customer requests.	December 20, 2019
Migrating with AWS Snowball Edge	Added support for using AWS Snowball Edge Edge to migrate large databases.	January 24, 2019

Support for Amazon DocumentDB (with MongoDB compatibility) as a target	Added support for Amazon DocumentDB (with MongoDB compatibility) as a target for data migration.	January 9, 2019
Support for Amazon OpenSearch Service and Amazon Kinesis Data Streams as targets	Added support for OpenSearch Service and Kinesis Data Streams as targets for data migration.	November 15, 2018
CDC native start support	Added support for native start points when using change data capture (CDC).	June 28, 2018
Db2 LUW support	Added support for IBM Db2 LUW as a source for data migration.	April 26, 2018
SQL Server as target support	Added support for Amazon RDS for Microsoft SQL Server as a source.	February 6, 2018

AWS Glossary

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